

Copyright is owned by the Author of the thesis. Permission is given for a copy to be downloaded by an individual for the purpose of research and private study only. The thesis may not be reproduced elsewhere without the permission of the Author.

Adding an Intelligent Component to an Existing Decision Support System

A thesis presented in partial fulfilment of the requirements for the degree of

Master of Science in Information Systems

at Massey University

Anna Elizabeth Jeffries

1996

ABSTRACT

A framework to guide the development of an intelligent component and its integration with an existing decision support system has been proposed. An initial framework was outlined, drawing concepts from the fields of decision support systems, knowledge based systems and intelligent decision support systems. This framework was applied to a problem in the domain of dairy farm management. A prototype intelligent decision support system was developed. Experiences gained during the development process enabled refinements to the framework to be made. The prototype was tested to assess the success of the framework in producing the desired results. The development framework was evaluated based on criteria drawn from relevant literature. The proposed development framework is considered to be a useful tool for intelligent decision support system development from an existing decision support system. Its success is attributed to the integration of methods and techniques drawn from a number of well established methodologies.

ACKNOWLEDGMENTS

Thanks are due to my three supervisors Mrs Elisabeth Todd, Dr Elizabeth Kemp and Mr David Gray who have all invested a lot of time in helping me complete this research report. They have provided continual support and encouragement throughout my post graduate studies.

The cooperation of the sponsor, Barry Butler of B.M. Butler Computing Ltd., has been greatly appreciated. The use of FarmTracker was a vital part of the study as was Barry's role as domain expert.

Thanks also to the staff of the Information Systems department, Faculty support staff and students from both the Information Systems and Computer Science departments, especially Dougald Mabin for his help in developing the ERD, and Andrew Jorgensen for his proof reading.

Funding received from various sources throughout my postgraduate studies has been gratefully appreciated. These sources include the Department of Information Systems and the Faculty of Science at Massey University, and the Manawatu branch of the New Zealand Federation of University Women.

TABLE OF CONTENTS

Abstract	iii
Acknowledgements	v
Chapter 1 Introduction	1
1.1 Overview	1
1.2 Definition of Terms	2
1.3 Research Methodology	3
1.4 Research Problem	4
Chapter 2 Literature Review	6
2.1 Introduction	6
2.2 Decision Support Systems	6
2.2.1 Decision Support System Architecture	8
2.2.2 Modelling and Simulation	11
2.2.3 Decision Support System Development Methodologies	13
2.2.4 Techniques for Development of DSS	21
2.3 Knowledge Based Systems	22
2.3.1 Knowledge Based System Architecture	23
2.3.3 Knowledge Representation	24
2.3.4 Knowledge Based Systems Development Methodologies	25
2.3.4 Knowledge Acquisition	31
2.3.5 Conceptual Graphs	33
2.4 Knowledge based Systems Versus Decision Support Systems	36
2.5 Intelligent Decision Support Systems	38
2.5.3 Model Management in IDSSs	40
2.5.4 Intelligent Decision Support System Architectures	41
2.5.5 Intelligent Decision Support Systems Development Methodologies	43
2.6 Decision Analysis & Influence Diagrams	51
2.7 Conclusion	54
Chapter 3 Initial Development Framework	55
3.1 Introduction	55
3.2 The Framework	55
3.2.1 Existing Decision Support System Analysis	57
3.2.2 Knowledge Acquisition for The Intelligent Component	58
3.3 Conclusion	60

Chapter 4 The Farm Management Case Study..... 62

 4.1 Introduction 62

 4.2 Case Study selection..... 63

 4.3 The Problem Domain 64

 4.4 FarmTracker 68

 4.4.1 FarmTracker Overview..... 68

 4.4.2 Is FarmTracker A Decision Support System?..... 71

 4.5 Conclusion 72

Chapter 5 Applying the Framework 74

 5.1 Introduction 74

 5.2 Existing Decision Support System Analysis 74

 5.3 Knowledge Acquisition for The Intelligent Component..... 76

 5.4 Conceptual Graphs Versus Influence Diagrams..... 83

 5.5 Conclusion 85

Chapter 6 Prototyping the Intelligent Component..... 88

 6.1 Introduction 88

 6.2 Prototype Development..... 89

 6.3 Prototype Functionality 93

 6.4 Conclusion 95

Chapter 7 The Development Framework..... 97

 7.1 Introduction 97

 7.2 Prototype Evaluation..... 97

 7.2.1 Does The Intelligent Component Meet The Defined Aims? 97

 7.2.2 Is the Prototype System a True IDSS? 99

 7.2.3 Summary 101

 7.3 Framework Evaluation 101

 7.3.1 Discussion 102

 7.4 Conclusion 111

Chapter 8 Conclusions & Future Work 112

 8.1 Future Work 114

References..... 115

Appendix A Existing DSS Analysis..... 122

 A1 Unlevelled Data Flow Diagram..... 122

 A2 Simplified Entity Relationship Diagram 123

 A3 Technical Analysis..... 124

Appendix B Knowledge Acquisition 129

 B1 The Domain Layer 129

 B1.1 Type Definitions..... 129

 B1.2 Actor Graphs 131

 B1.3 If-Then conceptual graphs 134

 B1.4 Type Lattice 137

 B1.5 Conceptual Catalogue of Concepts..... 140

 B1.6 Conceptual Catalogue of Relations..... 141

 B1.7 Influence Diagrams..... 142

 B2 Inference Layer 144

 B2.1 Diagnosis inference structure..... 144

 B2.2 Assessment inference structure 144

 B3 Task Layer..... 145

 B3.1 Diagnosis 145

 B3.2 Assessment 146

 B4 Task Model..... 147

 B5 Model of Cooperation..... 147

 B5.1 Specification of Functionality 147

Appendix C Prototype IDSS 149

 C1 Screen Shots..... 149

 C2 Rules 152

 C2.1 Rule Groups..... 158

 C3 Selected Functions 159

 C3.1 Diagnosis 159

 C3.2 Assessment 159

 C3.3 Use of the GROW Model..... 160

 C3.4 Simulated Feed Budget 161

 C3.5 Average Weekly Data to Match Targets 162

LIST OF FIGURES

Figure 1.1 - A Process for Systems Development Research (Nunamaker <i>et al.</i> , 1991)	3
Figure 2.1 - DSS Architecture (Ford, 1985).....	9
Figure 2.2 - DSS Architecture (Bonczek <i>et al.</i> , 1980)	10
Figure 2.3 - The Predesign Cycle (Keen & Scott Morton, 1978)	14
Figure 2.4 - The Design Cycle (Keen & Scott Morton, 1978)	15
Figure 2.5 - Decision research process (Stabell, 1983)	16
Figure 2.6 - Phases in building a DSS (Turban, 1995)	19
Figure 2.7 - DSS development strategies (Alter, 1994)	21
Figure 2.8 - Knowledge Based System Architecture (Adapted from (Turban, 1995))	24
Figure 2.9 - Prototyping and System Development (Duffin, 1988)	28
Figure 2.10 - Intermediate models in the KADS methodology (Wielinga <i>et al.</i> , 1991)	29
Figure 2.11 - Knowledge acquisition process (Kemp <i>et al.</i> , 1994).....	31
Figure 2.12 - Concept map for Conceptual Analysis.....	33
Figure 2.13 - Graphical representation of a basic conceptual graph	33
Figure 2.14 - Graphical representation of "not (graph 1 and not graph 2)"	35
Figure 2.15 - Example of a dataflow graph	36
Figure 2.16 - KB-DSS implementation process (Klein & Methlie, 1995)	50
Figure 2.17 - An influence diagram for a borrowing decision (Bodily, 1985).	52
Figure 2.18 - Different types of influence (Bodily, 1985).....	53
Figure 3.1 - Development framework outline	56
Figure 3.2 - The initial proposed development framework.....	61
Figure 4.1 - Feed budget report produced by FarmTracker	66
Figure 4.2 - Desirable pasture covers for dairy systems (Gray & Parker,)	67
Figure 4.3 - Overview of FarmTracker.....	68
Figure 5.1 - Example of a type definition	77
Figure 5.2 - Example of a dataflow graph	77
Figure 5.3 - Example of an If-Then graph	78
Figure 5.4 - A section of an influence diagram	78
Figure 5.5 - The Task Model	79
Figure 5.6 - Diagnosis inference structure for multiple fault diagnosis (Based on (Hickman, 1989)).	80

Figure 5.7 - Assessment inference structure (Hickman, 1989).....	80
Figure 5.8 - Part of the task layer based on the diagnosis inference structure.....	81
Figure 5.9 - Part of the task layer based on the assessment inference structure	81
Figure 5.10 - The Model of Cooperation	82
Figure 5.11 - Ideal proposed architecture of IDSS	82
Figure 5.12 - Proposed architecture of IDSS	83
Figure 5.13a - An influence diagram	83
Figure 5.13b - A data flow conceptual graph	84
Figure 5.14 - A conceptual graph showing influences.....	84
Figure 5.15 - The proposed development framework	87
Figure 6.1 - Section of object hierarchy	89
Figure 6.2 - Equivalent branch of object hierarchy and type lattice.	90
Figure 6.3 - Section of the object hierarchy.....	90
Figure 6.4 - Editor for the class Weather	91
Figure 6.5 - KAL implementation of a rule	92
Figure 6.6 - Section of Inference Browser for an example situation.....	94
Figure 6.7 - Graph displaying varying projections	94
Figure 6.8 - Graph displaying projection given changes in the system.....	95

LIST OF TABLES

Table 2.1 - The breakdown of definitions for knowledge acquisition.	32
Table 2.2 - Comparison of DSS and KBS features.....	38
Table 6.1 - Definition of prototype aims using criteria of Basili <i>et al.</i> (1986).....	88
Table 7.1 - Scenarios used to evaluate prototype	98
Table 7.2 - Summary of evaluation session	98
Table 7.3 - Selected evaluation criteria adapted from (Blair <i>et al.</i> , 1995a) and Jeffries (1994).....	103

1.1 OVERVIEW

Decision Support Systems are being used in the work place to aid managers make decisions involving semi-structured problems (Er, 1988). Knowledge Based Systems technology has also had significant success in practice, performing tasks which have previously required the expertise of a human (Duffin, 1988).

The functions of a conventional knowledge based system (KBS) are considered by some authors to be very similar to that of a decision support system (DSS) (Chang, Holsapple & Whinston, 1993; Doukidis, 1988; Ford, 1985). However, the two fields have traditionally been quite separate research and practice areas. A number of authors have acknowledged the potential for the use of knowledge based systems techniques in decision support (Beulens & Van Nunen, 1988; El-Najdawi & Sylianou, 1993; Er, 1988; Marsden & Pingry, 1993; Sprague, 1980; Turban, 1995; Turban & Watkins, 1986)

Systems that integrate DSS and KBS technologies have been described as Intelligent Decision Support Systems (IDSS), Intelligent Support Systems, Expert Decision Support Systems (EDSS), Expert Support Systems (ESS) or Knowledge Based decision support systems (KBDSS) (El-Najdawi & Sylianou, 1993). It is thought that the addition of an intelligent component to an existing conventional decision support system could prove to be a valuable technique for the development of intelligent decision support systems.

Both DSSs and KBSs have various development methodologies and techniques associated with them in the literature. There are a limited number of methodologies which exist for the development of IDSSs. Blair, Debenham and Edwards (1995a) are currently investigating and developing a methodology for the development of an IDSS. To the author's knowledge there are no methodologies which exist for the addition of an intelligent component to an existing DSS.

This study investigates a framework for the development of an IDSS from an existing DSS. KBS, DSS and IDSS methodologies have associated techniques which could be useful within such a framework, thus each group of methodologies have been studied.

An initial framework was proposed based on current literature. This framework was applied to a problem involving an existing decision support system used in the domain of dairy farm management. The initial framework has been revised in light of experiences gained during its application, and a prototype system was built to examine the feasibility of the framework. The resultant prototype system, and experiences gained during its development, provided a means for evaluation of the framework.

1.2 DEFINITION OF TERMS

There is considerable confusion within the systems development discipline as to the use of terms such as method, methodology and project life cycle; often these terms are used interchangeably (Yourdon, 1989). An attempt has been made to define these terms as they will be used throughout this report.

Gillies (1991) defines a methodology to be "...a framework for the systematic organisation of a collection of methods". Harmon and Hall (1993) when discussing development methodologies for knowledge based systems comment that "a methodology is more than a knowledge acquisition strategy; it provides a vocabulary, diagramming techniques, and a step-by-step procedure for actually constructing a system and tools for managing development projects". Avison and Fitzgerald (1995) consider an information systems methodology to be "a recommended series of steps and procedures to be followed in the course of developing an information system". Turban (1995) defines a life cycle in systems development to be "Structured approach to the development of information systems with several distinct steps". It can be seen that there is no clear distinction between the terms methodology and lifecycle; this study will use the term methodology wherever possible, as defined by Avison and Fitzgerald (1995).

Methodologies and lifecycles appear to vary in the detail of their descriptions. Avison and Fitzgerald (1995) comment that "a methodology can range from being a fully fledged product detailing every stage and task to be undertaken to being a vague outline of the basic principles in a short pamphlet". This study is attempting to propose only an initial framework for a methodology; the author believes its detail does not warrant it being called a methodology. Thus the term "framework" will be considered an outline to a methodology; something that, with further investigation and addition of detail, could be considered a methodology.

1.3 RESEARCH METHODOLOGY

The research methodology used was that of Systems Development Research (Nunamaker, Chen & Purdin, 1991). Systems Development as a research strategy is considered to consist of five stages, shown in the Figure 1.1 from Nunamaker *et al* (1991).

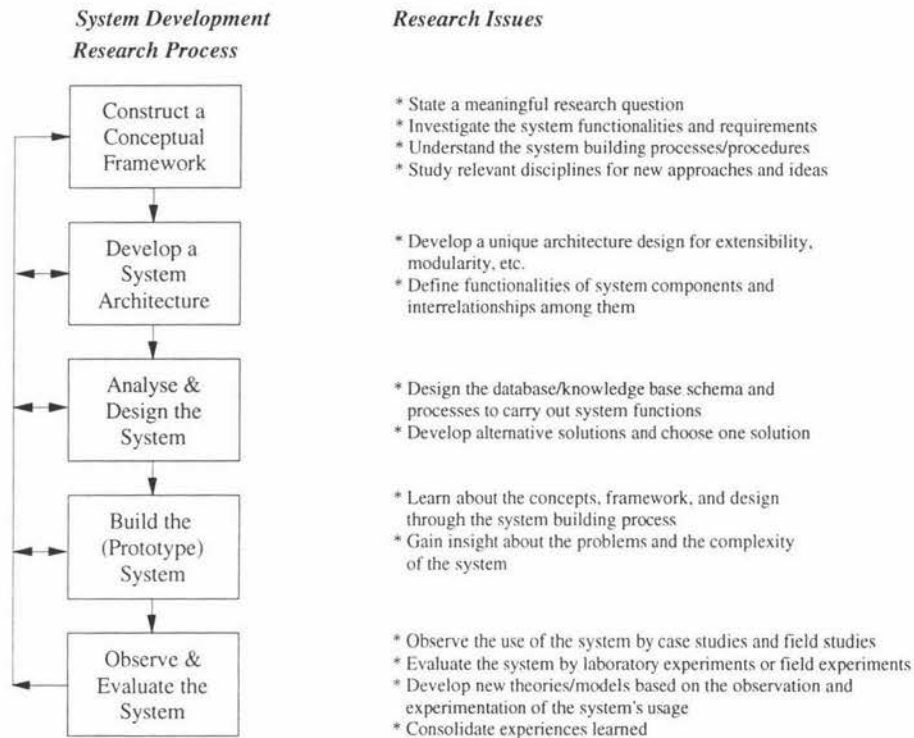


Figure 1.1 - A Process for Systems Development Research (Nunamaker *et al.*, 1991)

The first step involves stating a clear research problem, which provides a focus for the research throughout the development process. From here an appropriate conceptual framework can be established drawing from literature in various related disciplines.

The second step involves investigating appropriate architectures for the system, this stage must state objectives and define associated required functionality's of the resulting system. These first two steps provide a basis for the third step which involves the analysis and design of the system.

From here a prototype system is developed as a "proof-of-concept" to demonstrate feasibility of the proposal. The development in itself is a learning process; the difficulties and constraints encountered during the development process can be used to evaluate the concepts and theories initially proposed. The resulting prototype can also be evaluated based on the defined aims of the research.

It is considered to be very important to use other research methodologies to support systems development, as the development on its own is not generally considered to be serious information systems research. Therefore, the integration of techniques such as evaluation and experimentation is critical to validate the development process. Nunamaker *et al.* (1991) emphasise that building a system in itself does not constitute research, however the "synthesis and expression of new technologies and new concepts in a tangible product can act as both the fulfilment of the contributing basic research and as an impetus to continuing research" (Nunamaker *et al.*, 1991).

1.4 RESEARCH PROBLEM

As indicated by Nunamaker *et al.* (1991) it is necessary to provide a clear definition of the research problem before continuing. Its aim is to direct the development process, and is stated as follows.

To investigate and propose a framework for the development of an intelligent component to be integrated with an existing decision support system.

In order to investigate this research problem, the Nunamaker *et al.* process for systems development research was followed. Chapters in the report correspond generally with the steps in the process. Chapter 2 reviews literature related to the research problem with the aim of investigating methodologies and concepts which could be utilised in the resulting framework. Related research areas are reviewed including those of decision support systems, knowledge based systems and intelligent decision support systems.

Chapter 3 collates what is considered appropriate concepts and theories from the literature into an initial development framework. Chapter 4 provides an overview of the domain chosen to test the framework (dairy farm management), and describes the decision support system which is to be studied.

The development framework was applied to the problem domain; this process is described in Chapter 5. A revised framework is proposed based on experiences gained during its application. The resultant models were then used for the development of a prototype IDSS, which is discussed in Chapter 6.

The documented experiences and the resultant prototype IDSS, provided a basis for the evaluation of the framework in Chapter 7. The evaluation has taken two stand points.

Firstly, the prototype was evaluated based on the aims determined before its development relating specifically to the problem domain, which required involvement from the domain expert. This evaluation took the proof-by-demonstration approach (Nunamaker *et al.*, 1991); the framework was evaluated using the prototype to prove it was feasible and could create the desired results. Secondly, the framework was evaluated based on experiences gained during its application in relation to established criteria drawn from the literature.

Chapter 8 concludes the study, discussing the extent to which the stated objectives were met. Areas for associated future work, and extensions to this study are also outlined.

2.1 INTRODUCTION

This chapter reviews the literature associated with decision support systems, knowledge based systems and intelligent decision support systems. Within each discipline, relevant terms are defined, and architectures for the systems are investigated. Development methodologies for each are also studied, and any techniques considered relevant are highlighted and discussed.

2.2 DECISION SUPPORT SYSTEMS

Keen and Scott Morton (1978) were amongst the early researchers of decision support systems, and proposed the definition for a decision support system (DSS) to be:

"The application of available and suitable computer-based technology to help improve the effectiveness of managerial decision making in semistructured tasks" (Keen & Scott Morton, 1978)

Sprague (1980) describes the early characterisation of a decision support system to be:

"interactive computer based systems, which help decision makers utilise data and models to solve unstructured problems" (Sprague, 1980)

Sprague (1980) feels that this definition is too restrictive, resulting in very few systems which can completely satisfy it. Broader definitions too, have their problems due to the varied interpretations of them by people from different backgrounds. Doukidis (1988) believes that DSSs are most effective when dealing with semistructured tasks where "the system provides mathematical/analytical tools and models to evaluate the situation and the user adds his experience and judgement to make a decision". Similarly, Klein and Methlie (1995) define a DSS to be:

"A computer program that provides information in a given domain of application by means of analytical decision models and access to databases, in order to support a decision maker in making decisions effectively in complex and ill-structured (non-programmable) tasks" (Klein & Methlie, 1995)

Er (1988) faults these definitions by questioning the meaning of some of the terms used, such as "semistructured tasks". Klein and Methlie (1995) also see the definition of this term as a key concept in DSS. Thus it seems necessary to define the terms semistructured and unstructured tasks in order to define DSSs. Er (1988) discusses three types of problems to be solved by management; structured, semistructured and

unstructured, between which there are not clear boundaries. The classification of problems can depend on the existence of methods for solving them (Er, 1988), and thus the definition of structured versus unstructured problems varies with decision makers (Marsden & Pingry, 1993).

Marsden and Pingry (1993) define the term structured problem as "If a problem can be adequately represented by a decision model which is solvable, then we will say that the problem is structured", their corresponding definition of an unstructured problem, is any problem that does not satisfy the definition for a structured problem. However, this definition of structured depends on the interpretation of 'adequately represent' and 'solvable'.

Klein and Methlie (1995) discuss the structure of problems in a similar way, as follows:

"Structured problems are routine and repetitive, because they are unambiguous (since each such problem has a single solution method). A less structured problem has more alternative solution methods, and the solutions may not be equivalent. A completely 'unstructured' problem has unknown solution methods or too many solution methods to evaluate effectively" (Klein & Methlie, 1995)

Associated with this definition Klein and Methlie (1995) list characteristics of semistructured problem situations to be:

- "The preferences, judgments, intuition and experience of the decision maker are essential"
- "The search for a solution implies a mixture of:
 - search for information
 - formalisation, or problem definition and structuring (system modelling)
 - computation
 - data manipulation"
- "The sequence of the above operations is not known in advance since:
 - it can be a function of data
 - it can be modified, given partial results
 - it can be a function of the user preferences"
- "Criteria for the decision are numerous, in conflict, and highly dependent on the perception of the user"
- "The solution must be achieved in limited time"
- "The problem evolves rapidly" (Klein & Methlie, 1995)

Turban (1995) uses the following definitions for the different types of decisions:

structured decisions	"Standard or repetitive decision situations for which solution techniques are already available"
unstructured decisions	"Complex decisions for which no standard solutions exist"
semistructured decisions	"Decision's in which some aspects of the problem are structured and others are unstructured"

Turbans (1995) definitions will be used throughout this study as they are consistent with the other authors quoted, however he separates them into three clear definitions for the different types of decisions.

Due to the difficulty in defining DSSs and associated terms, Sprague (1980) sees it more appropriate to provide characteristics of a DSS in order to develop an understanding for the term, rather than attempt to provide a definition. This method of definition has been adopted by a number of authors in the field, and the following is a collation of characteristics of a DSS from various authors:

- DSSs are interactive, computer based information systems (El-Najdawi & Sylianou, 1993; Mittra, 1986; Sprague, 1980)
- DSSs specifically focus on features which make them easy to use by non computer people (Mittra, 1986; Sprague, 1980; Turban, 1995)
- DSSs tend to be aimed at the less well structured, under specified problems that upper level managers typically face; (Alter, 1980; El-Najdawi & Sylianou, 1993; Mittra, 1986; Sprague, 1980; Turban, 1995)
- DSSs support but do not replace upper-level managers in decision making (Alter, 1980; Er, 1988; Turban, 1995)
- DSSs attempt to combine the use of models or analytic techniques with traditional data access and retrieval functions; (El-Najdawi & Sylianou, 1993; Mittra, 1986; Sprague, 1980; Turban, 1995)
- DSSs rely on simulation in cases where an analytic optimising model cannot be solved (Mittra, 1986)
- DSSs use statistical analysis to collect data and to predict trends (Mittra, 1986)
- DSSs emphasise flexibility and adaptability to accommodate changes in the environment and the decision making approach of the user. (Alter, 1980; El-Najdawi & Sylianou, 1993; Mittra, 1986; Sprague, 1980; Turban, 1995)

2.2.1 DECISION SUPPORT SYSTEM ARCHITECTURE

Sprague (1980) considers DSS software to have three sets of capabilities; database management software (DBMS), model based management software (MBMS) and software for managing the interface between the user and the system - the dialogue generation and management software (DGMS) (Figure 2.1). The division of a DSS into these three components is supported by a number of authors (Beulens & Van Nunen, 1988; El-Najdawi & Sylianou, 1993; Ford, 1985; Marsden & Pingry, 1993).

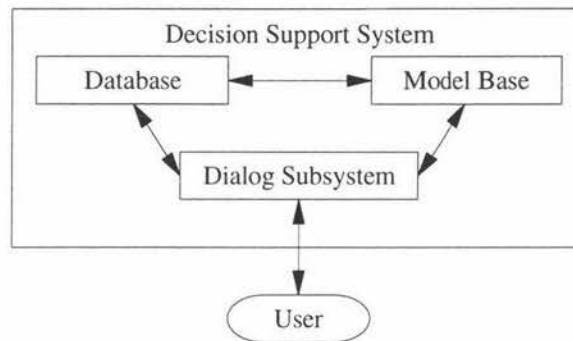


Figure 2.1 - DSS Architecture (Ford, 1985)

Sprague (1980) identifies technical capabilities each subsystem must possess if they are to be combined to become a DSS. The data subsystem should, according to Sprague (1980) possess, the following capabilities:

- the ability to combine data sources
- the ability to add and delete data efficiently
- the ability to represent data structures in a way the user understands
- the ability to allow the user to experiment with alternatives based on personal judgement and unofficial data
- the ability to manage this variety of data

According to Chang, Holsapple and Whinston (1993) model management is concerned with "computer-based means for representing and processing models", aiming to increase the productivity of decision makers, DSS developers and modelling experts. A model base is considered in the Sprague and Carlson (1982) framework as a collection of procedures that can be executed to analyse data. The model subsystem should, according to Sprague (1980) possess the following capabilities:

- the ability to quickly and easily create new models
- the ability to organise and maintain a range of models, suitable for all levels of management
- the ability to integrate models with appropriate links to data;
- the ability to manage the model base with similar functions to database management

The user System interface should, according to Sprague (1980) possess the following capabilities:

- the ability to deal with different user styles
- the ability to interact with the user in a variety of media
- the ability to present data in various formats and media;
- the ability to provide flexible support given the users' understanding of the situation

Bonczek, Holsapple and Whinston (1980) propose a framework for DSSs which differs from the popular framework of Sprague (1980). This framework too, has three major components; the language system, a knowledge system and a problem processing system (PPS) (Figure 2.2). The first two are systems of representation and the third is a software system.

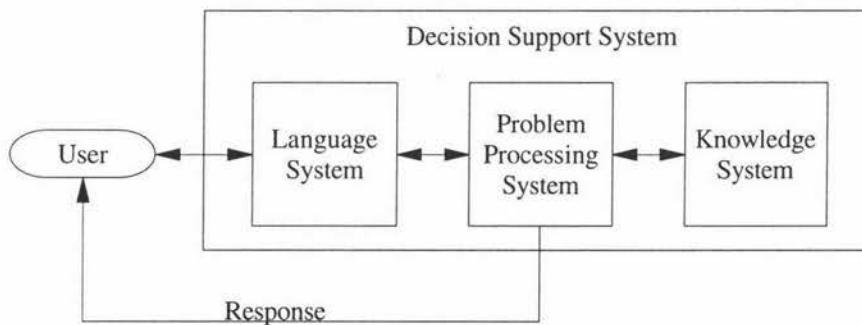


Figure 2.2 - DSS Architecture (Bonczek *et al.*, 1980)

The PPS has the function of mediating between the expressions of knowledge in the knowledge system and expressions of problems in the language system (Bonczek *et al.*, 1980). Chang *et al.* (1993) describe the language and knowledge system as fuel for the activities of the problem processor. "The language system is comprised of all requests that the PPS can act on with respect to the current contents of DSS's knowledge system. The knowledge system is comprised of knowledge that the PPS can use in generating responses to user requests" (Chang *et al.*, 1993).

The PPS is considered to have one or more of the seven abilities required for decision making, as defined by Bonczek, Holsapple and Whinston (1980) those being: Power, Perception, Design, Analysis, Valuation, Organisation and Adaptation. Power is described as being the ability to exercise some power or authoritative force, perception is the ability to collect information and design is the ability to formulate models. Power and perception are related to valuation as values are based on available information and available powers. Analysis is described as a continuing adjustment between design and perception and results in beliefs, expectations and facts. Organisation results in the execution of plans and is a continuing adjustment between design and power. Adaptation is a continuing adjustment between the other six abilities, and involves problem recognition, constrained by the other six abilities. Not all of these can be accounted for in the processor, such as power, as the processor has no intrinsic power or authority. Thus the processor supports rather than makes the decisions.

The knowledge system contains the body of knowledge about a problem domain. There are a variety of representation schemes which can be adopted in the knowledge system, and various types of knowledge can be represented; procedural, descriptive or meta knowledge (which is used to govern the use of programs and data, such as reasoning knowledge (eg in the form of rules)). This allows for powerful and flexible DSSs possibilities to emerge. Chang *et al.* (1993) fit the Sprague and Carlson framework into the Bonczek *et al.* framework by describing the knowledge system as consisting of exclusively a data base representing descriptive knowledge and a model base representing procedural knowledge.

This flexible framework is aimed at bringing together different types of systems to offer a unified context for research and study (Chang *et al.*, 1993).

2.2.2 MODELLING AND SIMULATION

It is generally considered a major characteristic of a DSS to be the inclusion of a modelling capability. Mittra (1986) defines a mathematical model to be "a representation of a real-life situation by means of variables and equations or inequalities". Jeffers (1982) has a similar definition of a model; "a formal expression of the relationship between defined entities in physical or mathematical terms". Models are used to simplify complex relationships between entities, Jeffers (1982) sees them as useful "because they reduce ambiguity and because they describe complexity with the maximum parsimony".

Chang, Holsapple and Whinston (1993) discuss the wide variation in the definition of a model within the DSS field. Chang *et al.*, (1993) after thorough review of associated literature, consider the earliest and predominant view of models to be "procedures, automated algorithms whereby data can be analysed in response to stated problems". The second view commonly taken of models is to treat them as "data that are to be analysed by a procedure" (Chang *et al.*, 1993). The third way Chang *et al.* consider 'model' to be interpreted is regarding it to be a problem. Model building in this context is considered to involve developing a set of equations that are to be 'solved'.

Another area which needs definition is descriptive and normative models where descriptive models "describe things as they are" whereas normative models prescribe how a system should operate (Turban, 1995). These type of models are used in development methodologies such as (Keen & Scott Morton, 1978) and (Stabell, 1983).

The types of models mentioned above are each solved explicitly using mathematical techniques, and the solution gives the values to be assigned to the variables defined in the model (Mittra, 1986). Mittra (1986) gives three limitations to the use of mathematical models:

1. An explicit solution is not possible because the model may be so complex
2. The time span of a model may not match the required time span, for example if the model extends over six months then the solution leaves implicit the plan for week to week operations.
3. A model contains a certain amount of uncertainty as they rely on past historical data.

Mittra (1986) sees the simulation technique as useful for when mathematical models are inadequate due to the above limitations. Watson and Blackstone (1981) describe a simulation model as "a mathematical model that describes the behaviour of a system over time". Simulation builds an experimental model of a system in analytic terms, and evaluates alternatives by performing simulated runs of the model, rather than attempting to solve the model explicitly as mathematical models do (Mittra, 1986). Simulation attempts to answer "what if" questions so that an analyst is able to make inferences about the possible behaviour of the real world system.

Turban (1995) discusses simulation modelling describing it as "a technique for conducting experiments (such as "what if") with a digital computer on a model of a management system". Turban considers simulation to be not strictly a type of model; as it imitates rather than represents reality. He describes it as a descriptive rather than a normative tool; as there is no automatic search for an optimal solution, rather it predicts the characteristics of a given system under different circumstances. Simulation is usually called for only when the problem under investigation is too complex to be treated by numerical optimisation techniques like linear programming.

Mittra (1986) warns of the limitations of simulation due to its inclusion of uncertain events, resulting in approximations subject to statistical error. A large number of simulation runs are also necessary, and as they can involve complex calculations, simulation may become an expensive exercise. Simulation does not generate an optimal solution but compares alternative solutions.

Mittra (1986) outlines the following objectives for building a simulation model, where one or more must be considered:

- "To describe a current system"
- "To explore a hypothetical system"

- "To design an improved system"
(Mittra, 1986)

There has been extensive research into the development and use of modelling and simulation for analysing biological, physical and economic components of agricultural systems. These complex models have been used successfully by researchers, however their use by groups such as farmers and extension agents is limited (Jones, Jones & Everett, 1987). Jones (1989) comments that "many well developed models were not being fielded partially because they were difficult to correctly parameterize on the front end and were difficult to interpret on the back end. Model developers certainly have the expertise to initialise and interpret their models, but no simple mechanism for transferring their capabilities".

One of the major problems is the limitations to the generality of many of these models, a problem which requires the results to be interpreted in the context of each particular application (Jones *et al.*, 1987). Jones *et al.* (1987) see a potential solution to this problem to be to "couple heuristic context knowledge to mathematical models in a computerised decision system that takes advantage of model capabilities for specific applications".

2.2.3 DECISION SUPPORT SYSTEM DEVELOPMENT METHODOLOGIES

Keen and Scott Morton (1978) proposed one of the earliest DSS methodologies. This includes a predesign cycle (Figure 2.3) which is cycled through at least twice before the more formal system design is begun. This cycle involves the development of normative models which define the potential range of designs for an information system. Klein and Methlie (1995) describe the normative approach as prescribing optimal behaviour, that is, how the decision should be made. The descriptive model defines a decision situation as it stands, it is concerned with understanding how people actually behave when solving problems and making decisions (Klein & Methlie, 1995). The difference between the two models is considered the degree of change, and is a measure of both pay off and difficulty of implementation.

More than one normative model can generally be developed for a non structured decision as there is no one best solution. The choice of normative models should be made by management who should recognise the trade off between risk and return. This is seen as the key aspect of the design strategy, as management can decide how much risk they believe to be feasible. Keen and Scott Morton (1978) place a lot of emphasis on the

predesign cycle and the importance of the managers opinions and understanding of the decision situation as an input to the design process. They feel the design team needs to understand in detail the existing decision process and potential ways of improving it.

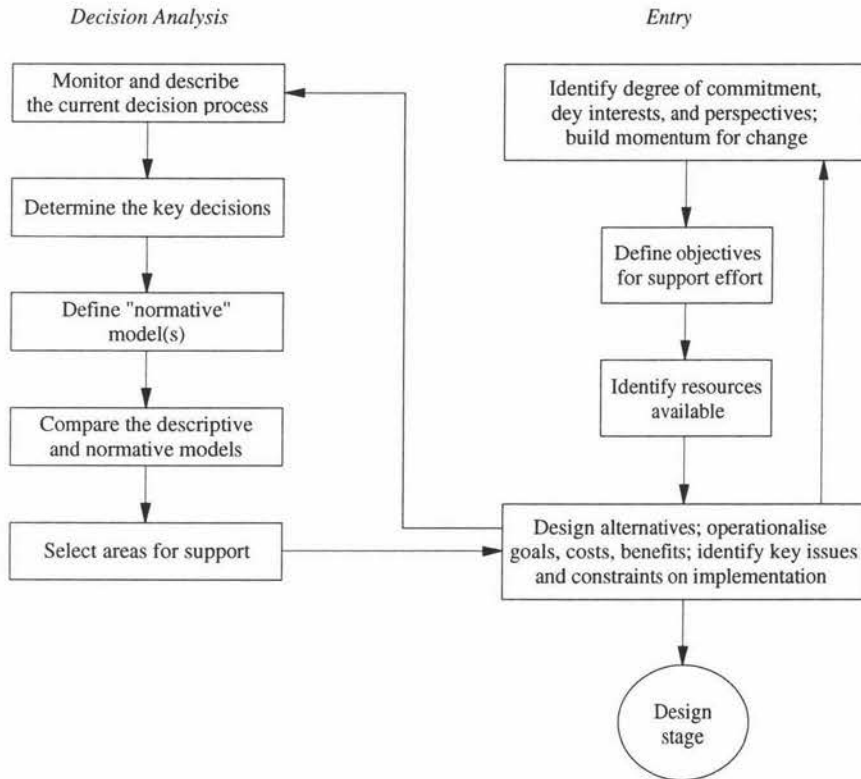


Figure 2.3 - The Predesign Cycle (Keen & Scott Morton, 1978)

The design stage involves three separate areas: the user design, defined in terms of imperatives; the interface or driver which links them; and the database management design. The imperatives answer the question of "What does the system do?" from the users perspective, for example "find", "extrapolate" (Keen & Scott Morton, 1978). The complete design cycle is shown in Figure 2.4. Arinze (1992) considers this approach to be process or operator driven due to the verbs or imperatives which are identified and subsequently transformed into DSS routines.

Keen and Scott Morton (1978) assume an evolutionary approach to design and see this as an important feature. The first stage is to design and deliver a system that is seen as useable and useful now, but the interface software should be flexible enough to allow rapid extension and addition of routines. This requires the users to be involved in the design process, and it is them who initiate another iteration of the design process for the addition of new features to the new DSS. Evolutionary design does not, therefore, mean the design of a first cut system to get it working and then work on improving it. It is

based on the assumption that the users will learn with the use of the system and extend the design.

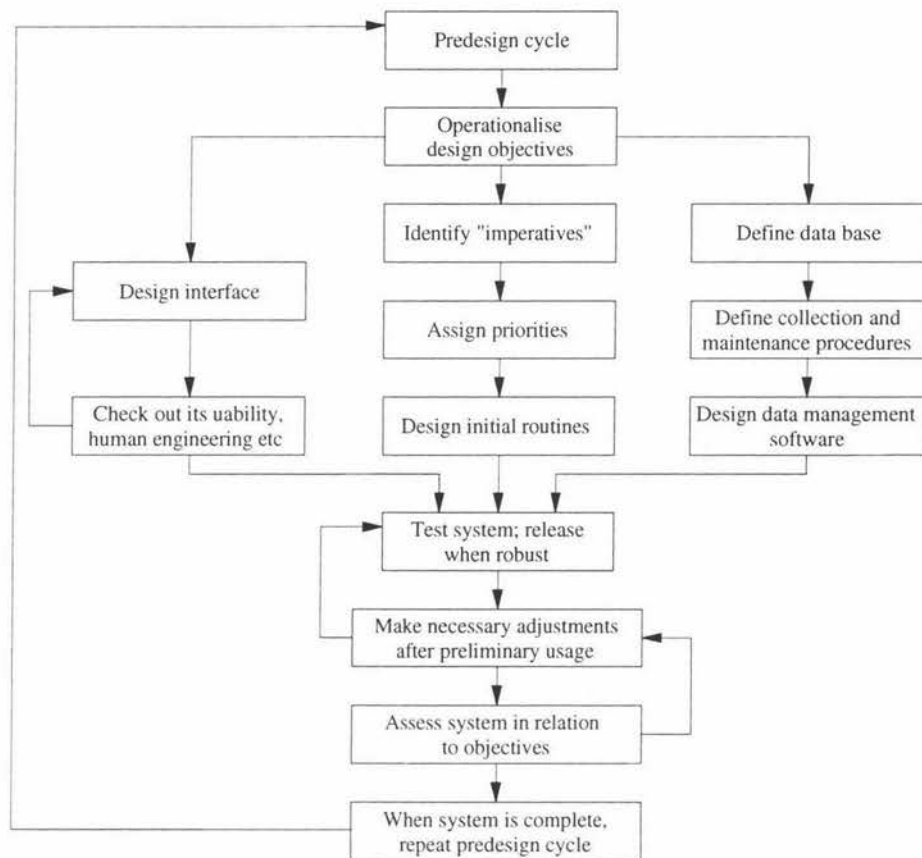


Figure 2.4 - The Design Cycle (Keen & Scott Morton, 1978)

Keen and Scott Morton (1978) summarise their strategy with the following key features:

- "1. The use of decision research to describe the decision process, define key decisions, and identify areas for decision support
2. The use of normative models to define multiple design alternatives and to help position a DSS in terms of degree of change, pay offs, and likely implementation problems.
3. The focus on system usage and objectives as determining structure and technical design
4. The separation of interface and imperatives to allow flexibility and evolution.
5. The inseparability of design and implementation" (Keen & Scott Morton, 1978)

In contrast to Keen and Scott Morton's (1978) process-driven design approach, that of Stabell (1983) is decision driven, similar to the pre-design cycle in Keen and Scott Morton's approach (Keen and Scott Morton's Pre-design cycle is based on earlier work of Stabell's). Stabell's "Decision research" process is shown in Figure 2.5. Decision

research is defined by Stabell (1983) as "a broad conceptual basis and a variety of measurement procedures for describing and evaluating decision-making behaviour in unstructured decision situations" and is considered to consist of the following three activities:

1. Collecting data on current decision-making using techniques, such as interviews, observation, questionnaires, and historical records;
2. Establishing a coherent description of the current decision process;
3. Specifying a norm for how decisions should be made" Stabell (1983)

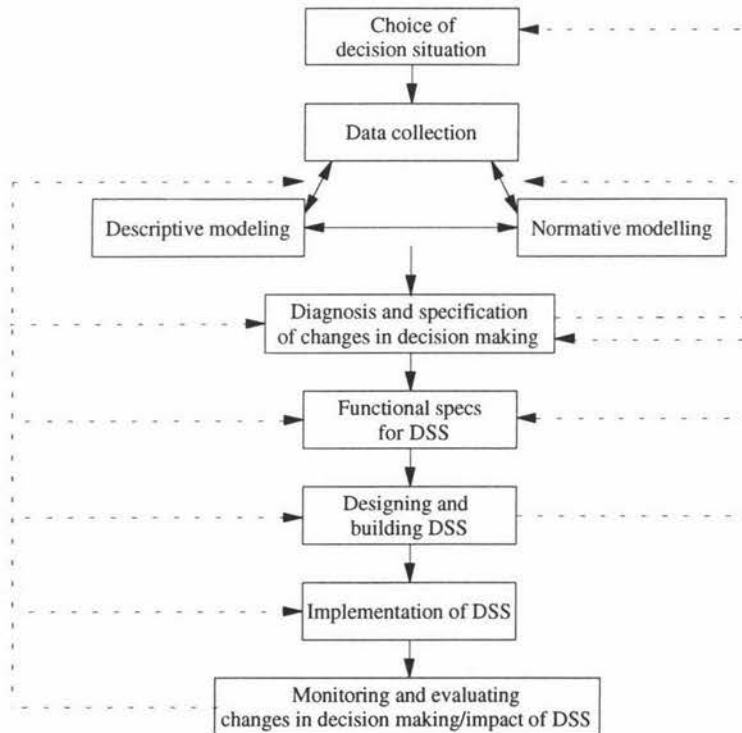


Figure 2.5 - Decision research process (Stabell, 1983)

Other authors have based their approaches on Simon's IDC model of the decision process, described by Klein and Methlie (1995) as:

1. searching for conditions that require a decision - the *intelligence* activity
2. Inventing, developing and analysing possible alternatives - the *design* activity
3. Selecting one of the alternatives - the *choice* activity

However Stabell (1983) feels this approach is not suitable for distinguishing between description and norm because it fits the decision process. Stabell believes a framework should identify "how managerial decision making in organisations, in a descriptive sense, differs from effective decision behaviour" (ie the descriptive differs from the norm).

Meador, Guyote and Rosenfeld (1986) proposed a four-stage process for developing a large scale DSS. This is outlined as follows:

1. **Decision Support analysis**
 - structured interview
 - decision analysis
 - data analysis
 - technical analysis
 - conceptual DSS orientation
 - plans and prioritization
2. **DSS software evaluation and selection**
 - identification of candidate vendors
 - feature analysis
 - benchmarks
 - external site surveys
3. **Prototype development**
 - scoping of prototype
 - project evaluation criteria
 - detailed design
 - system construction
 - testing
 - demonstration
 - evaluation
4. **Operational deployment and support**
 - functional orientation
 - operational training
 - deployment
 - maintenance

Meador *et al* (1986) discuss the use of "Decision Support Analysis" as the first step in the DSS development. Decision support analysis involves structured interviews with management, decision analysis, technical analysis and management orientation. The purpose of the interview is to allow managers to identify their critical needs, objectives and priorities. It is seen as important to use interview checklists to focus the interviews, and these are developed in consultation with one or two key user managers to ensure all important areas are covered.

Following the structured interviews, decision analysis takes place, which in this context involves the development of a conceptual framework to guide the identification of DSS opportunities, system design, project management, and communication of priorities between users and system developers. This type of decision analysis consists of three tasks: business area analysis, description of logical functional flow and specification of detailed decision areas.

Business area analysis "studies representative business units to determine their decision support functional requirements" (Meador *et al.*, 1986). The unique business needs of each unit is identified along with the objectives, functions, shared data and reports/analysis needed by each unit.

The second stage in decision analysis is converting the business area specifications into functional flow diagrams, which involves "hierarchical decomposition of the decision making activities of the business areas" (Meador *et al.*, 1986). One methodology recommended for this purpose is the Structured Analysis and Design Technique. The last step in decision analysis is the identification and classification of decisions. Once identified they can be prioritised for DSS development.

Data analysis involves the identification and description of the classes of data used by the functions using the functional flow diagrams. The final stage is technical analysis which translates the needs identified previously into a proposed system design with technical requirements for hardware and software. The results of the Decision Support Analysis guide the DSS development through the rest of the stages, outlined above.

Meador *et al.* (1986) discuss six characteristics a DSS methodology should exhibit. These include having a minimal elapsed time prior to prototype development so that the users can see a concrete system. Due to this rapid development, they feel that analysis may often be incomplete, thus it is necessary for the method to quickly focus on the highest priority applications, and on the functional requirements that require most detailed analysis. It is thought that a methodology should evolve with the DSS, "being used to discover initial DSS opportunities, establish initial functional requirements, and evaluate existing systems to identify directions for further growth" (Meador *et al.*, 1986). Meador *et al.* (1986) feel that user involvement is an important part of a methodology, and there should be an orientation toward managerial users and their decision making activity. It is thought that the methodology should capture managers' decision processes and should establish priorities to improve these processes.

Turban (1993; 1995) integrates the work of Meador *et al.* (1986) with that of Keen and Scott Morton (1978) to give the development process illustrated in Figure 2.6. The steps involve the following activities:

1. Planning - Needs assessment, problem diagnosis, objectives of the DSS defined, key decisions of the DSS determined
2. Research - Identification of a relevant approach for addressing user needs and available resources
3. Analysis - Define normative models for key decisions.
4. Design - can be divided into parts corresponding to the major components of a DSS:
 - Database and its management
 - Model base and its management
 - Dialogue subsystem

Turban's latest publication (1995) includes the design of the knowledge component. Appropriate software tools or generators are selected.

5. Construction - The technical implementation of the design
6. Implementation - Testing, evaluation, demonstration, orientation, training, and deployment
7. Maintenance and Documentation - Planning for ongoing support
8. Adaptation - Recycling through the above steps on a regular basis to respond to changing user needs.

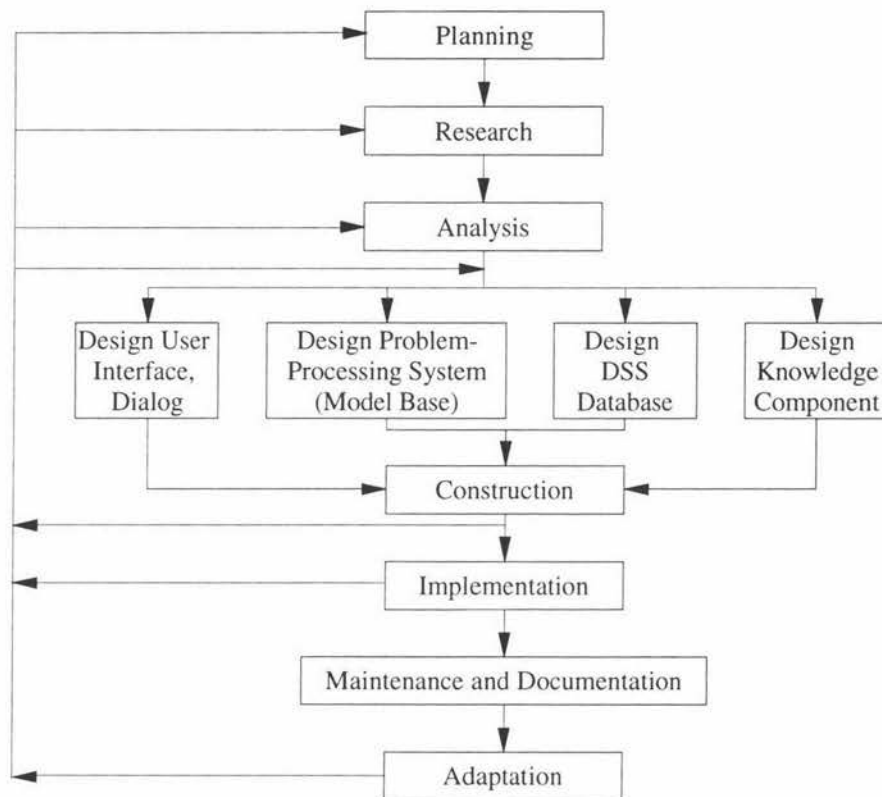


Figure 2.6 - Phases in building a DSS (Turban, 1995)

Turban (1995) associates with these phases the iterative approach to design which is also referred to as the evolutionary or prototyping approach and involves the following four activities:

1. Select a subproblem
2. Develop a small but useable system
3. Evaluate the system continuously
4. Refine, expand and modify the system in cycles

This process is repeated a number of times until a relatively stable and comprehensive system evolves. Turban sees the advantages of the iterative process to be: a short development time, short user reaction time (feedback from user) and improved users' understanding of the system, its information needs, and its capabilities.

Sprague and Carlson (1982) propose a framework for DSSs analysis and design aimed at identifying the characteristics and capabilities that a specific DSS needs to have. This approach is based on four user-oriented entities, which from the users point of view provide the capabilities of a DSS. These are:

Representations to help conceptualise and communicate the problem or decision situation

Operations to analyse and manipulate those representations (similar to Keen and Scott Morton's imperatives (Keen & Scott Morton, 1978) according to Arinze (Arinze, 1992))

Memory aids to assist the user in linking the representations and operations

Control mechanisms to handle and use the entire system

This approach is called the ROMC approach. Arinze (1992) considers this to be one of the most comprehensive DSS methodologies to date. The approach is a tool for organising and conducting systems analysis in DSS (Sprague & Carlson, 1982). ROMC is process independent and is based on five characteristics of decision making, discussed by Sprague and Carlson and summarised by Turban (1995) as follows:

1. Decision makers prefer to use graphical conceptualisation's to describe situations
2. The three decision making phases (intelligence, design, and choice) can be applied to DSS analysis
3. Memory aids are useful in decision making and should be provided by a DSS
4. The DSS should help decision makers use and develop their own styles, skills, and knowledge.
5. The decision maker expects to have personal control over the support system.

Arinze (1992) feels the ROMC methodology results in flexible DSSs, but believes "much of the burden of linking models and data is thrust upon the user". He also points out the absence of normative models which is present in other methodologies.

Sprague (1980) sees the necessity for analysis and design techniques for the development of DSSs to be different to the traditional approaches. He believes "A DSS needs to be built with short, rapid feedback from users to ensure that development is proceeding correctly. It must be developed to permit change quickly and easily". Iterative design involves the combination of analysis, design, construction and implementation into a single step which is iteratively repeated. The manager and builder work together tackling one subproblem at a time and after a short period of use, the system is evaluated, modified and incrementally expanded. The cycle is repeated three to six times until a relatively stable system which supports a number of tasks is produced (Sprague, 1980).

Weitzel and Kerschberg (1989) discuss the nature of DSSs as having ill-structured problems which deems the design-first-implement-later approach to be inappropriate. Prototyping is seen by them as a suitable means of developing DSSs due to the difficulty in specifying correct, complete and unambiguous requirements.

Alter (1994) suggests the use of which ever development strategy is most cost effective and least risk prone, however, recognises the support in the field for evolutionary

approaches in developing DSSs. The four system development methods Alter (1994) considers appropriate for DSS's are outlined in Figure 2.7.

1. **Phased Method** - Complete the following stages in sequence with formal documentation and complete user and MIS sign off after each stage:
Figure out what is going on in the organisation
Define system scope and objectives
Define format, content, and user control of inputs, calculations and outputs
Design and program the computer system
Test
Install
Revise
2. **Evolutionary method** - Iterate through the following steps as rapidly as possible and as many times as necessary:
Try to understand the problem
Build a partial solution
Obtain user feedback about the usefulness and completeness of the partial solution
Revise accordingly
3. **Incremental method**
Start with an existing computer system
Identify its shortcomings
Create additional reports, analytical tools, extract versions of the data base, or whatever else is needed to improve decision making
Use the revised system and review periodically
4. **Turnkey method**
Identify a generic problem for which a computerised solution exists which has been developed elsewhere
Install that solution or a slightly modified version of it
Identify shortcoming
Either change the organisation to fit the computer system or vice versa

Figure 2.7 - DSS development strategies (Alter, 1994)

Alter (1994) outlines the benefits of the evolutionary method as being the high degree of user interaction, with the user receiving benefits early. He sees this method as good when only partial solutions are understood at the outset, and the progress is quick. However, he also sees high costs such as the large amount of user time required, the high skill needed from the system designer, the difficulty in maintenance due to the likely lack of standardisation and documentation, and the possible need to redesign and reprogram for efficiency after the concept is demonstrated.

2.2.4 TECHNIQUES FOR DEVELOPMENT OF DSS

Atkinson and Arnott (1995) investigated the tools and techniques used in the development of DSSs. They defined techniques to be "procedures that can guide a systems analyst while performing a given task, in analysis activities...[they] can be manual or computer based instructions". A survey of practitioners involved in the development of DSSs, revealed that the most popular technique used was the data flow diagram (DFD), a contradiction to Goodwin and Wright's (1991) claim the most common

graphical technique to describe a decision situation is a decision tree (Atkinson & Arnott, 1995). Entity relationship diagrams (ERD) were also found to be a popular technique, showing that practitioners are using conventional techniques for the development of DSSs. This is surprising as many authors have claimed the necessity for alternative techniques to model a decision structure, such as influence diagrams and cognitive maps, neither of which were used by practitioners in Atkinson and Arnott's survey. Prototyping also had much less support than expected with only 3.6% of respondents applying this technique.

2.3 KNOWLEDGE BASED SYSTEMS

Conventional software systems solve problems by performing numerical calculations, solving equations and using optimisation algorithms to process large amounts of data. In contrast, knowledge based systems are generally considered to perform tasks typically solved by human experts in a particular domain. They do so by using rules-of-thumb, or heuristic's, to manipulate symbolic descriptions about the knowledge they are given (Harmon & King, 1985).

Jackson (1990) defines an expert system to be:

"a computer program that represents and reasons with knowledge of some specialist subject with a view to solving problems or giving advice".

Similarly Turban (1995) defines an expert system to be:

"a computer system that applies reasoning methodologies on knowledge in a specific domain in order to render advice or recommendations, much like a human expert"

To define knowledge based systems more accurately, a list of capabilities which one possesses has been compiled from various authors in the field. This list outlines capabilities which distinguish a knowledge based system from a conventional system.

1. A knowledge based system simulates human reasoning about a problem domain (Gillies, 1991; Hayes-Roth, Waterman & Lenat, 1983; Jackson, 1990)
2. A knowledge based system performs reasoning on representations of human knowledge (Jackson, 1990)
3. A knowledge based system solves problems by heuristic or approximate methods (Jackson, 1990; Turban, 1995)
4. A knowledge based system manipulates symbols which represent objects (Jackson, 1990)

5. A knowledge based system is capable of explaining and justifying solutions or recommendations by reasoning about their own inference processes (Gillies, 1991; Hayes-Roth *et al.*, 1983; Jackson, 1990; Turban, 1995)
6. A knowledge based system solves problems that generally fall into one of the following categories: interpretation, prediction, diagnosis, debugging, design, planning, monitoring, repair, instruction, and control (Hayes-Roth *et al.*, 1983)

The ability of knowledge based systems to widely distribute scarce expertise with perpetual accessibility and consistent answers, are seen as some of their major advantages over human problem solvers (Gonzalez & Dankel, 1993). They provide explanation facilities and easy modification, as well as the ability to deal with incomplete data; features which cannot be achieved with conventional software systems (Gonzalez & Dankel, 1993). However, knowledge based systems also have disadvantages such as their knowledge being limited to the domain of expertise and their lack of common sense (Gonzalez & Dankel, 1993).

It becomes obvious when studying the literature that the two phrases "knowledge based systems" and "expert systems" are often used synonymously (Gonzalez & Dankel, 1993). Similarly, Harmon and King (1985) describe the phrases "knowledge systems" and "expert systems" to be synonymous. Harmon and King (1985) comment that "The popular press and various software entrepreneurs have already used the term "expert system" in so many ways...that it now lacks any precise meaning". Other authors consider expert systems to be specialised instances of knowledge based systems (Evans, Mondor & Flaten, 1989; Gillies, 1991), however this view is not as well supported in the literature. The term knowledge based system rather than expert system will be used wherever possible throughout this report.

2.3.1 KNOWLEDGE BASED SYSTEM ARCHITECTURE

A knowledge based system is generally considered to consist of a knowledge base, an inference engine, a user interface and an explanation subsystem (Harmon and King, 1985; Turban, 1995). The knowledge base contains the knowledge required to understand, formulate and solve problems. This includes facts, such as the problem situation and theory of the problem area and heuristic's which direct the use of the knowledge to solve specific problems (Turban, 1995). The inference engine contains the inference strategies and controls for manipulating the facts and rules (Harmon & King, 1985).

The explanation subsystem aims to explain the behaviour of the knowledge based system, and the user interface provides a means for the user to communicate with the system. A

blackboard is also included in some knowledge based systems, and it is an area of working memory used for description of a current problem, it can include a scheduling system, which continually examines the pending actions and chooses the one to try next. (Turban, 1995, Harmon & King, 1985). Figure 2.8 summarises the architecture of a knowledge based system.

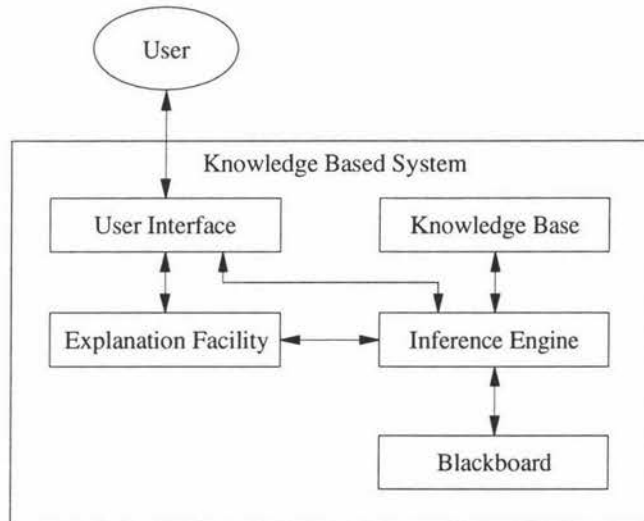


Figure 2.8 - Knowledge Based System Architecture (Adapted from (Turban, 1995))

2.3.3 KNOWLEDGE REPRESENTATION

Representation of knowledge within a knowledge based system is a major issue in the field, and should achieve the following objectives:

1. It should capture generalisations
2. It should be understandable
3. It should be able to be easily modified
4. It should be able to be used in a variety of situations
5. It should be able to be used to narrow the range of possibilities to be considered

(Mehandjiska, 1993).

One method of knowledge representation is that of production rules, developed by Newell and Simon (1972). The knowledge is presented in the form of condition-action pairs, such as "IF this condition (or premise or antecedent) occurs, THEN some action (or result, or conclusion, or consequence) will (or should) occur" (Turban, 1995). Turban (1995) discusses the two types of rules in production systems; declarative rules (which state facts and relationships about a problem), and procedural rules (which advise on how to solve a problem, given certain facts are known). Each rule in a knowledge base represents a part of the knowledge independently, which makes modifications and maintenance relatively easy (Turban, 1995). Production rules are considered to be a

good representation of the way human experts organise their knowledge and they can be expressed at varying levels of generality. However, problems can arise due to the large numbers of rules required in some cases (Turban, 1995).

Another method of organising knowledge uses principles of object-oriented systems. This involves the packaging of both data and procedures into structures (called objects) related by an inheritance mechanism (Mehandjiska, 1993). Subclasses can inherit procedures and data from their superclasses, and the objects can communicate with each other via messages. Mehandjiska and Page (1993), investigated the use of the object-oriented approach to expert systems development, and outlined three methods of applying the object-oriented approach to knowledge representation:

- using a static form of an object
- extending the object-oriented paradigm to include rules
- pure object-oriented knowledge representation

Harmon and Hall (1993) discuss "hybrid systems" combining rules and object oriented techniques, making it easier to build large KBS and making them easier to maintain. Payne and McArthur (1990) discuss similar techniques, as do Klein and Methlie (1995) who suggest the combination of rule inferencing and frame or object stored knowledge. Turban (1995) also sees advantages of using multiple representations as he feels knowledge representation should be able to support the tasks of acquiring and retrieving knowledge, as well as subsequent reasoning; three tasks which are difficult to achieve with a single representation scheme. Multiple knowledge representation allows different subtasks to have appropriate representations. However, problems may arise in translating information between them. Turban (1995) believes a successful combination of representation methods is that of production rules and frames, utilising the advantages of both representations.

2.3.4 KNOWLEDGE BASED SYSTEMS DEVELOPMENT METHODOLOGIES

Harmon and Hall (1993) discuss two recognised methods of developing knowledge based systems; rapid prototyping and model based development. Rapid prototyping involved "creating a set of rules to solve a specific case and then exercising the system to see if it could indeed solve similar cases with the rules it now had in its knowledge base". This is seen by Avison and Fitzgerald (1995) as the prevailing approach to expert system development, which has evolved from the trial and error approach of early expert systems. However, for large knowledge based systems this approach is found to be too

slow, and lacks an overview and a strategy which allows a manager to divide the work up (Harmon & Hall, 1993).

Model based approaches to system development have proved to be promising for overcoming these prototyping problems, by providing formal models of the problem type. One of these methodologies is the KADS methodology, which defines some 20 problem types and provides detailed instructions for analysing and designing modules for each type of problem (Harmon & Hall, 1993).

PROTOTYPING KNOWLEDGE BASED SYSTEM DEVELOPMENT METHODOLOGIES

Hayes-Roth *et al.* (1983) present a methodology for building expert systems, the five steps involved with very brief descriptions are outlined below:

1. Identification: Determining problem characteristics
2. Conceptualisation: Finding concepts to represent knowledge
3. Formalisation: Designing structures to organise knowledge
4. Implementation: Formulating rules that embody knowledge
5. Testing: Validating rules that embody knowledge

Hayes-Roth *et al.* (1983) emphasise the fact that these stages are not clear-cut, well-defined or independent, they consider the steps to roughly define the knowledge acquisition process. Associated with these steps is an evolutionary or incremental development technique, where a prototype is developed which solves the initial subproblem, which is then improved upon following further iterations of the process.

Harmon & King (1985) outline two methodologies for building knowledge systems; one for small and one for large systems. Both involve the development of a prototype; the more detailed process for the development of large knowledge and associated subtasks is outlined in below:

1. Selection of an appropriate problem
 - Identifying a problem domain and a specific task
 - Finding an expert willing to contribute expertise
 - Identifying a tentative approach to the problem
 - Analysing the costs and benefits of the effort
 - Preparing a specific development plan
2. Development of a prototype system
 - Learning about the domain and the task
 - Specifying performance criteria
 - Selecting an expert system building tool
 - Developing an initial implementation
 - Testing the implementation with case studies
 - Developing a detailed design for a complete expert system

3. Development of a complete expert system
 - Implementing the core structure of the complete system
 - Expanding the knowledge base
 - Tailoring the user interface
 - Monitoring the system's performance
4. Evaluation of the system
5. Integration of the system
 - Arranging for technology transfer
 - Interfacing the system with other data bases, instruments, or other hardware to enhance the speed or friendliness of the system
6. Maintenance of the system

The sequence of these steps is not entirely fixed; it is provided only to give an overview of how an ideal project might proceed (Harmon & King, 1985). It is interesting to note that the prototype is developed into the resulting system, that is, it is evolutionary prototyping as discussed in Section 2.2.3.

Another knowledge based systems methodology is that outlined by Duffin (1988): GEMINI (Government Expert system Methodology INitiative). This is not considered a prototyping methodology in the same way as the above methodologies, as it utilises prototyping in a very controlled manner. According to Duffin, GEMINI is intended to "provide a complete method for the activities involved in the development of KBS software from the inception of a project through to, and including, the process of physical design". Duffin states that GEMINI is intended to be used in conjunction with conventional software development methods because of the possible need for KBS developments to be closely integrated with conventional software systems. One of the precepts of GEMINI is the compatibility of the methodology with data processing approaches so that a single feasibility study can identify both the KBS and conventional components of a business solution, and subsequent stages can work alongside each other.

Due to the undisciplined use of prototyping and incremental development, without a clear understanding of the objectives and success criteria to which they conform, the steps involved in GEMINI are carefully controlled. It utilises a technique called a project review which is returned to after each phase in the systems development (feasibility, requirements analysis, system modelling, logical design and physical design). The project review provides the "opportunity for re-evaluation of the technical, business and organisational viability of the proposed system in the light of the accumulated information available at that point" (Duffin, 1988).

GEMINI adopts a controlled approach to prototyping; in contrast to an implementation driven and evolutionary prototyping approach. Prototyping is considered to be an

important technique, as it is useful for communication and feedback between the developer, users and experts. According to Duffin (1988), prototypes "must have an explicitly defined aim and success criteria before it is designed, implemented and used". The prototype development is considered to have its own lifecycle consisting of definition of aims, design of prototype, implementation, use of prototype and evaluation. The evaluation feeds back gained experiences into the real system development life cycle. Duffin's proposed lifecycle is shown in Figure 2.9.

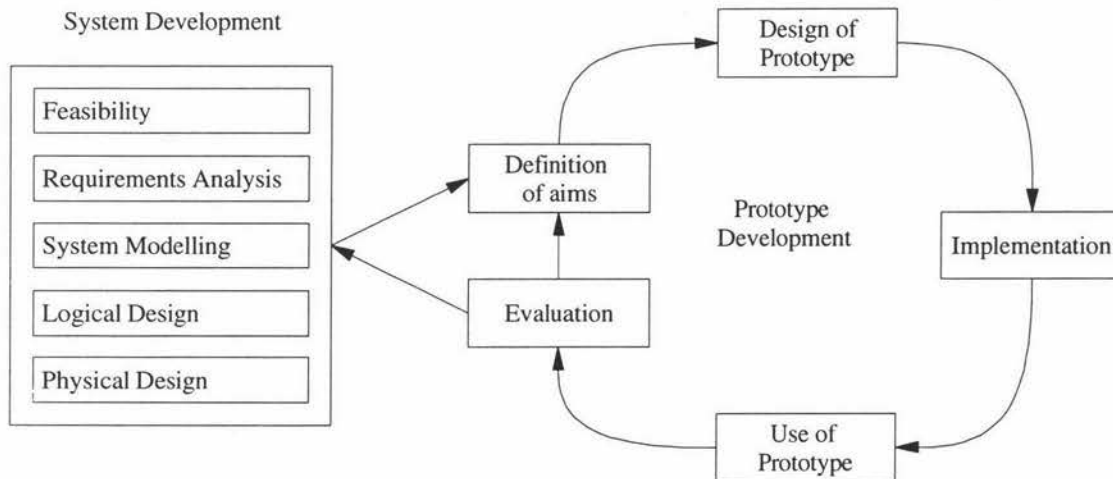


Figure 2.9 - Prototyping and System Development (Duffin, 1988)

MODEL BASED KNOWLEDGE BASED SYSTEM DEVELOPMENT METHODOLOGIES

The KADS methodology (Wielinga, Schreiber & Breuker, 1991) concentrates on the analysis phase of the knowledge based systems development. KADS attempts to provide a good understanding of the knowledge through a modelling approach. With modelling, the behaviour of a real world situation is mapped to an artefact description. Karbach, Linster and Voss (1990) explain that with KADS "The analysis of knowledge should be model driven as early as possible, as this makes the process more efficient and more proficient".

The various models in the KADS methodology are outlined in Figure 2.10. The following is a description of these models, as defined by Wielinga *et al.* (1991).

- The Organisational model provides an analysis of the socio-organisational environment in which the KBS will function. It also attempts to predict how the knowledge base system will influence the organisation.
- The Application model defines what problem the system is to solve in the organisation and what the function of the system will be in the organisation.

- The Task model specifies how the function of the system is achieved through the tasks and subtasks that the system will perform. These subtasks are the starting point for the next two models. Each task has an input and output specification, where the input is the information used to achieve a goal represented by the output. The task environment determines constraints on how the task is performed in the domain, this influences the scope and nature of the model of expertise and the model of cooperation.
- The Model of Cooperation provides a decomposition of the tasks in the task model into a number of primitive tasks. It specifies which agents carry out which tasks, and specifies the functionality of those subtasks which require cooperation.
- The Model of Expertise is the major activity in the building of a KBS. It specifies the problem solving expertise required to perform the problem solving tasks assigned to the system.
- The Conceptual model is a combined, implementation-independent, model of both expertise and cooperation.
- The Design model is a description of the computational and representational techniques that are not present in the conceptual model.

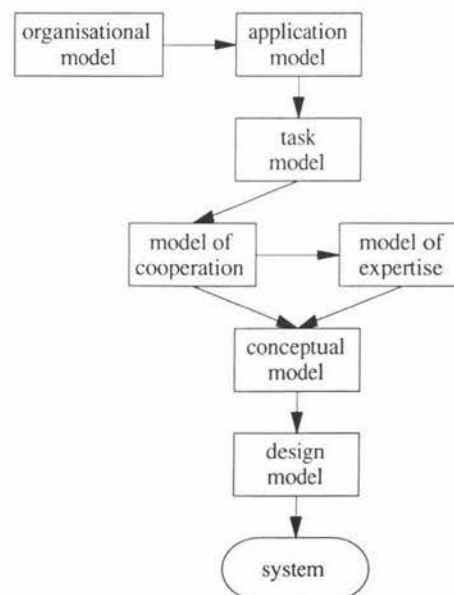


Figure 2.10 - Intermediate models in the KADS methodology (Wielinga *et al.*, 1991)

Traditionally the modelling of knowledge based systems has been "implementation-oriented" (Hickman, 1989). This causes problems, as very different systems are expected to fit into the same basic implementation structures. The level of abstraction in the KADS model of expertise ensures an implementation independent model of the experts knowledge. The abstraction also makes the description more general, facilitating maintenance (Hickman, 1989). Knowledge at this level can be distinguished as either domain or control knowledge. Four layers are distinguished in the KADS model of expertise, between which there is limited interaction. The four layers are:

- The Domain Layer embodies the conceptualisation of a domain for a particular application. This level is independent of the task layer as it is task neutral, thus there is nothing on this layer to control the use of the knowledge. This approach is an attempt to gain a flexible and reusable representation of domain knowledge. The ontological primitives used to describe domain theories are concepts, properties of the concepts, relation between concepts, and relation between property expressions
- The Inference Layer categorises the basic reasoning steps into inference structures. Included in the inference layer is an input/output specification and a reference to the domain knowledge that it uses. The inference structure is only an abstract description of the inference making, it does not show how these inferences are controlled during the task.
- The Task Layer contains knowledge about how the knowledge sources are combined for use in the problem solving processes. Tasks only refer to inferences and not explicitly to domain knowledge. There are three types of subtasks: primitive problem solving tasks (specified in the inference layer), composite problem solving tasks (specified in the task layer) and transfer tasks (require interaction with an external agent). There are four types of transfer tasks which are: obtain, present, receive and provide.
- The Strategic Layer contains the knowledge that enables an expert to choose the appropriate task structures and goals to solve a particular problem

The first two layers describe what can be known and inferred, but say nothing about how such knowledge is actually applied to reason toward desired conclusions (Hickman, 1989). The last three layers are regarded as a constraint on the domain layer and are domain independent. The middle two layers can be described by generic models of expertise called interpretation models.

A number of interpretation models for various problem solving activities are contained within the KADS library of interpretation models. One or more of these existing models can be used as a starting point to guide the data acquisition process. If a suitable model does not exist, one must be constructed from scratch by the knowledge engineer. Existing interpretation models include problem solving activities such as monitoring, diagnosis and prediction,

Kemp, Todd, da Silva & Gray (1994) propose an adaptation to the KADS methodology as a knowledge acquisition process. The resultant models are the task model and a three layered model of expertise (omitting the strategic layer). The process is illustrated in Figure 2.11, where the elicitation process obtains domain information from a source; analysis breaks down the results of the elicitation phase into concepts, attributes and relationships; the interpretation phase uses the domain knowledge to select interpretation templates from the KADS library; and modelling involves developing the task model and the model of expertise.

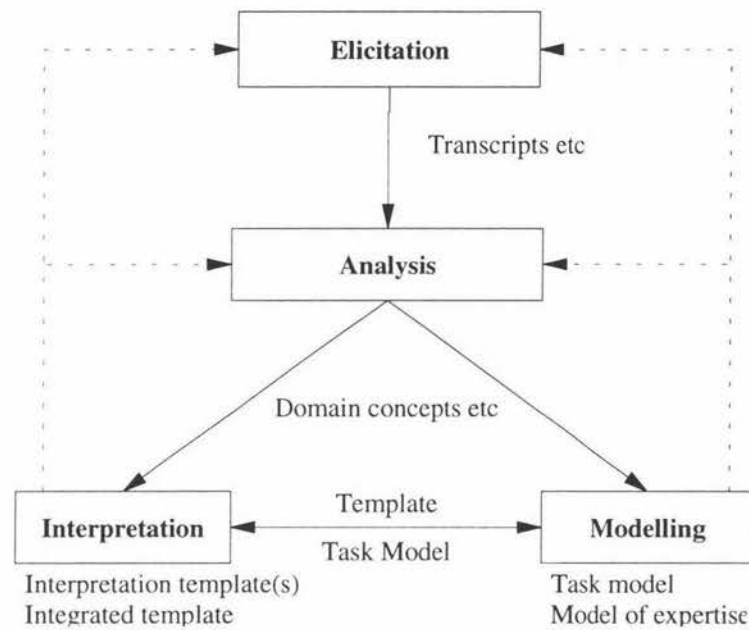


Figure 2.11 - Knowledge acquisition process (Kemp *et al.*, 1994)

2.3.4 KNOWLEDGE ACQUISITION

Knowledge acquisition is seen by Sowa (1992) as "...the process of eliciting, analysing, and formalising the interconnected patterns of thought underlying some subject matter". Wielinga *et al.* (1991) define knowledge acquisition as involving the activities of "eliciting the knowledge..., interpreting the elicited data using some conceptual framework, and formalising the conceptualisation's in such a way that the program can use the knowledge". The formalising stage in this definition has a different meaning to Sowa's (1992) use of the word. Sowa considers the formalising of the knowledge to be translating the modelled knowledge into a language of artificial intelligence. However, Wielinga *et al.*'s formalising goes no further than modelling the knowledge, their interpretation stage involves the interpretation models of the KADS methodology which guides the modelling process.

In the work by Kemp *et al.* (1994), knowledge acquisition using their adapted KADS methodology is considered to involve the stages of elicitation, analysis, interpretation and modelling (Figure 2.11). This definition is very similar to Wielinga *et al.*'s definition which excludes the translation of the knowledge into the pre-encoded form. Kemp *et al.*'s use of the word analysing is different to that of Sowa's, as Sowa's analysis includes interpreting and modelling of the data. Table 2.1 summarises the definitions discussed, with each column containing terms used equivalently by the different authors.

Sowa (1992)	eliciting			analysing	formalising
Wielinga <i>et al</i> (1991)	eliciting		interpreting	formalising	
Kemp <i>et al</i> (1994)	eliciting	analysing	interpreting	modelling	

Table 2.1 - The breakdown of definitions for knowledge acquisition.

CONCEPTUAL ANALYSIS

Conceptual analysis is the central activity in knowledge acquisition according to Sowa (1992), and needs to be integrated with the elicitation and formalisation processes. It is designed to elicit the tacit assumptions that are so obvious that no one thinks of them consciously. Regoczei and Hirst (1988) also describe conceptual analysis as explication, which involves "organising and systematising the knowledge, separating important from less important concepts, checking the knowledge for completeness of coverage, and above all making sure that both informant and analyst are talking and thinking about the same thing".

Sowa (1984) and Regoczei and Hirst (1988) consider the first step in conceptual analysis to be brainstorming, or the "free association with the goal of listing everything related to a concept" (Sowa, 1984), two methods of achieving this is by drawing Buzan's Mind Maps (Buzan & Buzan, 1993) or Novak and Gowin's concept maps (Novak & Gowin, 1984). These help in the identification and definition of the domain of discourse appropriate to the problem. These maps are a way of quickly and naturally representing a person's understanding of a domain. They aid in the loosening of inhibitions often present when trying to elicit knowledge from an expert, and are intended to be an informal notation for quickly capturing relationships (Sowa, 1992).

The more formal notation of conceptual graphs can then be used as a system of logic for analysing distinctions in detail and representing the concept maps more precisely (Sowa, 1992).

Sowa adapted a checklist developed by Sloman (1978) for conceptual analysis and applied some of the ideas to conceptual graphs. This guides the process of developing conceptual graphs.

- Instances - imagine every possible way in which the concept can be used; experiment with using the instance in different contexts.
- Type hierarchy - divide the list of instances into subtypes and group them according to different criteria in as many ways as possible; consider all supertypes; find synonyms, antonyms and related words; identify what criteria distinguish them.

- Canonical Graphs - class the concept as a natural type, a role type, a comparative type or an evaluative type; ensure the inclusion of links which are necessary for the class.
- Definitions - extend or complete dictionary definitions; look for general principles; if it is a technical concept then create a type definition.
- Schemata - create schema for concepts which cannot be defined by necessary and sufficient conditions showing how it is related to other concepts.

The Sowa-Sloman checklist follows the creation of the mind or concept maps. The maps can generally be divided into small sections which can be converted into conceptual graphs. The conceptual graph notation should help guide the process by providing formal ways of representing the information. Figure 2.12 shows a concept map for conceptual analysis adapted from Sowa (1992).

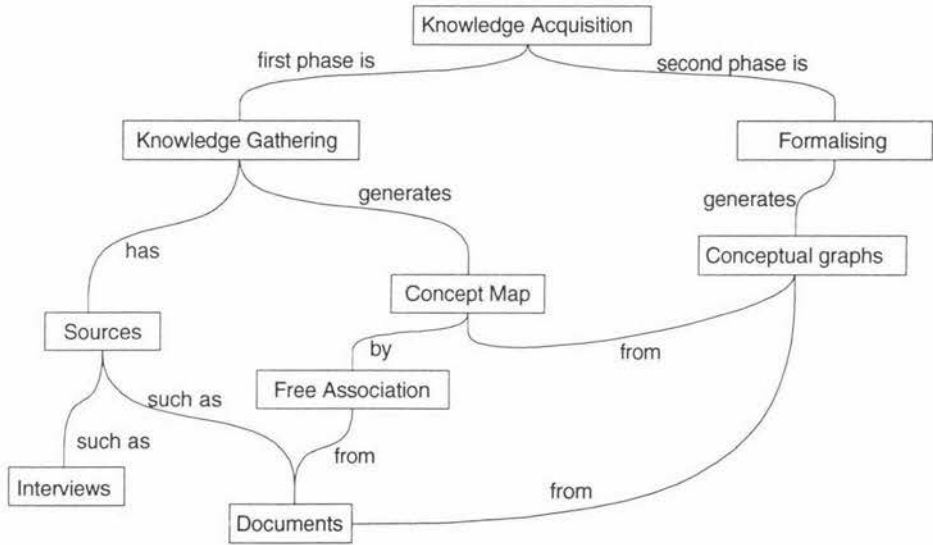


Figure 2.12 - Concept map for Conceptual Analysis.

2.3.5 CONCEPTUAL GRAPHS

A conceptual Graph is defined by Sowa (1984) as "...a finite, connected, bipartite graph". The two kinds of nodes that can exist in a conceptual graph are concepts and conceptual relations. Each conceptual relation has one or more arcs, each of which must be linked to some concept, if a relation has n arcs, it is said to be n-adic. A conceptual graph by itself may form a conceptual graph, but every arc of every relation must be linked to some concept. The graphs are represented as shown in Figure 2.13.



Figure 2.13 - Graphical representation of a basic conceptual graph

The graph in Figure 2.13 is read as "the relation of concept 1 is concept 2". Sowa (1992) sees the graphical representation as a good way of recording associations, he claims that graphs allow a change of focus at any time, so any node in a graph can become a new centre for growing a new pattern of relations. The graphs can also be written in textual form, with square brackets used around concepts, and rounded brackets around relations, for example:

$$[\text{CONCEPT1}] \rightarrow (\text{RELATION}) \rightarrow [\text{CONCEPT2}]$$

The function *type* maps concepts into a set *T*, whose elements are called type labels. The denotation of type *t*, written δt , is the set of all entities that are instances of any concept of type *t*. The types can be ordered into a subtype/supertype hierarchy called the type hierarchy. The type hierarchy tends to form a type lattice as some types have more than one immediate supertype. Biébow *et al.* (1993) describes this lattice as an "IsAKindOf" hierarchy, which allows multiple inheritance.

Generally common nouns correspond to type labels, they are generic terms that specify some attribute of the entity they are referring to. They apply equally well to any entity with that attribute. Proper nouns instantiate these type labels. The concept box is divided into two parts, the type label is followed by a colon and then the referent to that type. For example the concept [PERSON:#123] refers to the individual person number 123, and #123 is called the individual marker. Generic concepts have an asterisk as its referent, which refers to any individual of the concept, for example [COW:*) reads "a cow" or "some cow". Abstract concepts can have referents, for example [HAPPY:#234] refers to a particular instance of happiness. A referent can also refer to a particular mass of a substance, for example [WATER:#345] refers to a particular mass of water, where 345 is the identifier for the particular sample of water.

To represent sets in conceptual graphs, the referent field in a concept may contain a list of names or individual markers within braces. For example, [CAT:{Amber, Jasper, Blake}], indicates that all three of the individuals in the braces conform to the type CAT. To represent sets of different types, a type label which is general enough to enable all members to conform to it must be used. For completely unrelated types, the type label *T* would have to be used, as the universal type can have anything as its referent.

Sowa (1984) discusses four types of plural noun phrases to enable the handling of different relationships with set concepts. They are:

- A collective set, is when all elements participate in some relationship together. The individuals are separated by commas. $\{i_1 \dots i_n\}$

- In a distributive set, each element of the set satisfies the relationship individually. The set is preceded by the prefix Dist. $\text{Dist}\{i_1 \dots i_n\}$
- In a disjunctive set, it is known that one of the individuals participates in the relationship, however which one is not known. The individuals in the set are separated by a vertical line. $\{i_1 \mid \dots \mid i_n\}$
- A respective set has an ordered list of individuals which correspond respectively to another ordered list with which it is related. The set is within angle brackets and is preceded with the prefix Resp. $\text{Resp}\langle i_1 \dots i_n \rangle$

If a concept v , is within the context of concept u , then u dominates v . Peirce logic leads to the ability of inferences to be made with conceptual graphs. The statement "if graph 1, then graph 2" can be read as "if graph 1 can project into any graphs in the knowledge base, then Graph 2 can be asserted" (Polovina & Heaton, 1992). Logically this if/then statement can be written as:

not (graph 1 and not graph 2) which is the general rule of modus ponens.

This can be written in the box notation for easier understanding, as in Figure 2.14. The domination of graph 1 over graph 2 is obvious in this notation. Polovina and Heaton (1992) simplify the logic by explaining that "Any graph that projects into a graph that dominates it may be "rubbed-out" or deiterated". If graph 1 can be projected into the knowledge base (ie. if it is true), then graph one can be removed leaving a double negated ring around graph two, to give not(not graph 2), which cancels out and asserts 2 to be true.

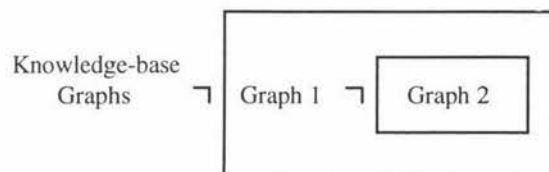


Figure 2.14 - Graphical representation of "not (graph 1 and not graph 2)"

Figure 2.14 shows that if $\neg[\text{graph 2}]$ was found to exist in the data base, thus asserting graph 2 to be false, the entire statement $\neg[\text{graph 2}]$ can be deiterated, leaving $\neg[\text{graph 1}]$, thus if graph 2 is found to be false, graph 1 is false. This is the general inference rule of modus tollens explained by Polovina and Heaton (1992) as "if the consequence of an if/then rule is false, then so is its antecedent". These rules of inference ensure truth in conceptual graphs, which the formation rules do not.

A dataflow graph is a type of conceptual graph used for computation which include concepts, and a set of nodes called actors (Sowa, 1984). An actor is a process that responds to messages by performing some function and then generating messages that it

passes to other actors (Sowa, 1984). An example of a dataflow graph is shown in Figure 2.15.

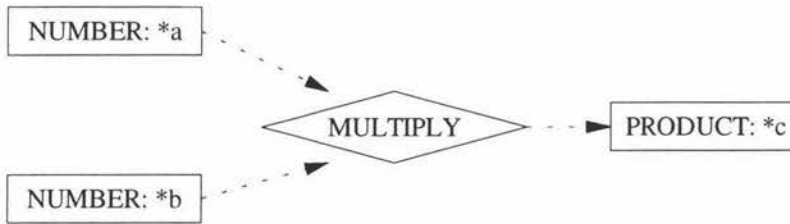


Figure 2.15 - Example of a dataflow graph

For further discussion of conceptual graphs see Jeffries, Todd and Kemp (1995b) and Sowa (1984).

2.4 KNOWLEDGE BASED SYSTEMS VERSUS DECISION SUPPORT SYSTEMS

Some authors argue that DSSs and Expert Systems (ESs) have very similar goals, Doukidis (1988) investigated this claim by surveying 67 ESs to see if they employed DSS concepts, which he defined as:

1. The role of the DSS, addressing the following issues; semistructured task, support decision making, effectiveness, multi-objective, multi-domain, individual and group decision.
2. Design features
3. Components
4. Prescriptive and Descriptive View
5. Phases of Decision
6. Level of Organisational features

Doukidis (1988) found that ESs address semistructured problems and they support users to improve the effectiveness of their work, thus have similar aims to DSSs. Ford (1985) also investigated the differences between DSSs and ESs, and felt, like Doukidis, that the fundamental goal of the two types of systems is basically the same: to improve the quality of decisions.

DSSs are seen by Ford (1985) as having more of a support role, providing access to data and models relevant and applicable to that decision, whereas ESs provide a conclusion or decision to the non-expert user that are more likely to be correct than that user could provide. Ford (1985) discusses the different types of users of DSSs and ESs, where DSS users are mainly middle and upper level management, who are often the decision maker

who helped design that system. ESs however, have a user group who are principally scientists or researchers in a narrow field of study, and because they are built to "clone" the knowledge of particular experts for use by non-experts, the users are not involved in the development of the ESs. This is debatable, however, as methodologies such as Duffin's (1988) GEMINI methodology encourage user involvement, also some expert systems are developed for use by the expert.

Doukidis (1988), Ford (1985) and Turban and Watkins (1986) all found that ESs are less flexible, supporting only rigidly defined goals under a well-bounded domain. Thus there was a lack of user control in the ESs in comparison to DSS's.

Doukidis (1988) found that ESs are more common in providing information and advice for operational activities, whereas DSSs are mainly for strategic planning and management control. Although the two were found to have similar aims, Doukidis points out that they achieve them in completely different ways, with the main differences being the boundary of the problem space, and the way to tackle problems.

Ford (1985) discusses the similarity of development methodologies for DSSs and ESs, both generally using an iterative or prototyping approach, however he points out that the development of an ES is generally more time consuming than that of a DSS.

With the relaxed framework of Bonczek *et al.* (1980) (Section 2.2.1) a conventional knowledge based system can be considered a decision support entity where the "knowledge system is comprised of rule sets and state variables, whose language system consists of requests for advice and explanation, and whose problem processing system is an inference engine capable of dealing with such requests and carrying out deductions based on knowledge system contents" (Chang *et al.*, 1993). Bonczek *et al.* (1980) fit MYCIN (a well known expert system) into their framework. In contrast, a KBS cannot be considered a DSS under the Sprague and Carlson framework as it does not employ data base or model base management.

Table 2.2 is collated from the work of El-Najdawi and Sylianou, 1993; Richards and McDonald, 1995 and Turban, 1995, and summarises the differences between DSS and KBS.

Dimension	DSS	KBS
Objective	Support decision maker	Capture, transfer and make available expertise
Who makes the recommendations?	The human and/or the system	The system
Means to achieve objective	Provide interactive access to data and models	Provide models of the experts thought processes
Major query direction	Human queries the machine	Machine queries the human
Manipulation method	Numerical; quantitative; what-if analysis	Symbolic; heuristic
Application	Semistructured; <i>ad hoc</i> broad problems	Problem domain dependent; unstructured problems; narrow domain problems; repetitive
Content of database	Factual knowledge	Procedural and factual knowledge
Reasoning capability	No	Yes, limited
Explanation capability	Limited	Yes

Table 2.2 - Comparison of DSS and KBS features

2.5 INTELLIGENT DECISION SUPPORT SYSTEMS

Turban (1995) believes the addition of a knowledge management component to a DSS "can provide the required expertise for solving some aspects of the problem and/or providing knowledge that can enhance the operation of the other DSS components". El-Najdawi *et al.* (1993) too, can see potential in the integration of DSSs and ESs technologies, viewing the two tools as complimentary, with the comment that "In contrast to the predominantly quantitative focus of DSSs, ESs use qualitative inferences procedures and heuristics".

Although many authors discuss the integration of ES and DSS techniques producing what are generally referred to as intelligent decision support systems (IDSS), only some of the authors attempt to define IDSSs. Turban (1995) defines an IDSS to be:

"a DSS that includes one or more components of expert systems or other AI technologies. This component makes the DSS behave in a better (more "intelligent") manner" (Turban, 1995)

Gottinger and Weimann (1992) define an IDSS as:

"an interactive tool for decision making for well-structured (or well-structurable) decision and planning situations that uses expert system techniques as well as specific decision models to make it a model-based expert system" (Gottinger & Weimann, 1992)

This is a clear definition of the term. However confusion arises as it discusses well-structured decision situations, a contradiction to Klein and Methlie's (1995) definition which describes an IDSS to be:

"a computer information system that provides information and methodological knowledge (domain knowledge and decision methodology knowledge) by means of analytical decision models (systems and users), and access to data bases and knowledge bases to support a decision maker in making decisions effectively in complex and ill-structured tasks" (Klein & Methlie, 1995)

The use of the terms structured and semistructured tasks was discussed in section 2.2, as this also became an apparent problem when attempting to define DSSs. Blair (1995) considers both of the above definitions, but summarises by defining IDSSs as attempting to solve semistructured problems. The author agrees with this decision. It is felt that IDSSs aim to solve the same type of problems as DSS, however they also provide knowledge based system capabilities.

Blair *et al.* (1995a) use the terms intelligent decision system (IDS) and intelligent decision support system synonymously. Holtzman (1989) defines an intelligent decision system to be a:

"decision system that delivers expert-level decision analysis assistance. As part of this assistance, the IDS may provide access to a substantial knowledge base in the domain of the decision", where a decision system is defined as "a system that makes recommendations for action...typically implemented on a computer" (Holtzman, 1989)

McGovern, Samson and Wirth (1991) see intelligent decision systems to be an extension of expert systems, defining them as:

"a class of expert systems with an inference engine based on the application of the axioms of decision theory" (McGovern *et al.*, 1991)

Klein and Methlie (1995) discuss intelligent decision systems as a particular type of knowledge based decision support system. They consider an IDS to be a tool which makes the skill of expert decision analysts available by using the expert system technology to provide both domain knowledge and methodological knowledge. These systems utilise decision classes, discussed further in Section 2.5.5. Klein and Methlie (1995) feel that the IDS concept fits within their IDSS framework, with a domain-specific knowledge base and a decision analysis methodology knowledge base which helps the decision maker to formulate and assess influence diagrams for the problem. An IDS has the ability to use the decision analysis methodology to elicit important features of a given situation. In contrast, the normative power of traditional expert systems are limited to the situations represented within the existing knowledge base.

This study is investigating intelligent decision support systems as defined by Turban (1995) above, rather than the more specific intelligent decision systems.

DSSs were defined in Section 2.2 by outlining the capabilities which such a system should possess. Blair (1995) outlines an equivalent list for IDSSs, dividing the capabilities into three perspectives. The list is outlined below with reference to other authors who support the capability. As highlighted in Turban's definition, there need be only one expert system capability for the system to be considered an IDSS, thus all capabilities listed may not be met by any one IDSS.

External Perspective

- An IDSS addresses semistructured problems (Blair, 1995, Klein & Methlie, 1995)
- An IDSS may deal with uncertain or incomplete information (Blair, 1995)
- An IDSS may deal with decisions which involve risks (Blair, 1995)
- An IDSS provides normative power to the decision maker (Blair, 1995, Klein & Methlie, 1995)

System Perspective

- An IDSS is dependent on multiple internal and external information/knowledge sources (Blair, 1995, Klein & Methlie, 1995)
- The IDSS utilises and manages the use of models (Beck & Jones, 1989, Moser, 1986, Plant & Stone, 1991, Klein & Methlie, 1995)

Internal Perspective

- An IDSS is based on several disciplines (eg DSS, databases, KBS etc) (Blair 1995, Klein & Methlie, 1995)

2.5.3 MODEL MANAGEMENT IN IDSS

The combination of KBS techniques and model management is an area of interest among authors within the field. Beck and Jones (1989) regard expert systems to be prescriptive tools, whereas simulations are predictive, thus see the two tools as complementary.

Moser (1986) proposes the creation of an integrated DSS developed by combining the ability of simulation to predict the values of complex sets of variables over time, with the reasoning ability of expert systems. This allows the analysis of the simulation output. The system includes a knowledge base editor and a model editor, which helps the user create knowledge bases or simulation models that conform to the syntax requirements of

the system. The system also contains a model library which has a collection of user-created simulation models to which the system has access during a consultation session. The complex results of the simulation can be analysed by the expert system component, enabling recommendations to be made to non-expert users.

Plant and Stone (1991) regard simulation models as demons, when used with knowledge based systems, which are external functions or programs called by the inference engine to perform a specific function. This idea is also suggested by Fedorowicz and Williams (1986) who discuss an IDSS issuing a call to the appropriate model when all necessary data and parameter settings have been supplied. With the ability of expert system shells to call or access other programs or software packages, these demon functions are made possible. Plant and Stone (1991) suggest the use of expert systems technology for determining when to run a simulation model, how to initialise it, and how to interpret its results. Any missing or unavailable information could be replaced with expert guesses or appropriate default values.

2.5.4 INTELLIGENT DECISION SUPPORT SYSTEM ARCHITECTURES

There appear to be two approaches to conceptualising the architecture of an IDSS. The first considers the IDSS to be a conventional DSS with an added intelligent component. This view is supported by Turban (1995) who utilises the commonly used architecture of Sprague (1980), integrating the intelligent component with the database management software (DBMS), the model based management software (MBMS), and the dialogue generation and management software (DGMS). This results in four subsystems which interact in a variety of ways to form an IDSS.

Fedorowicz and Williams (1986) however adapt the Bonczek *et al.*, (1980) framework by including both "non-procedural knowledge represented with conventional data base techniques, plus procedural knowledge represented with artificial intelligence techniques such as formal logic clauses, production rules, semantic nets, and frames" within the knowledge system (Chang *et al.*, 1993). The knowledge system also includes models utilised by the system. This results in a very integrated system with all information being manipulated and stored within the knowledge system. Holtzman's (1989) architecture is similar to this, consisting of a general-purpose inference engine, a set of data structures, a corresponding set of specialised procedures, and a user interface. This too results in a very integrated system where the intelligence appears to be inseparable from the conventional features.

As this study is concerned with adding an intelligent component to an existing system, it seems that the first method of integration is more plausible as the intelligent component is considered a separate part of the system. Turban (1995) and Turban and Watkins (1986) outline a number of different proposals for this method of integration, these include:

1. Expert System integration into various DSS components; usually considered a one way integration; the ES supports the DSS
 - i. ES interaction with DBMS. Either the ES is used to improve the structure, use, and maintenance of the DBMS, or the DSS provides the ES with essential data.
 - ii. ES interaction with model base. The ES is an intelligent agent for the model base and its management.
 - iii. ES interaction with the interface. The ES provides, for example, a natural language interface; explanation capabilities to the DSS; manipulation of symbolic information; tutoring to the user.
 - iv. ES as a consultant to the model builder. The ES gives advice on how to structure a DSS; how to conduct a feasibility study; or how to adapt the DSS.
 - v. ES integration with the user. The ES provides advice to the user regarding the type of problem or environmental conditions; possible implementation problems; how to use the DSS output; or to provide advice on which DSS to use.
2. Expert System as a separate component of the DSS
 - i. ES output as input to a DSS. The ES is used to determine the importance of the problem or to identify the problem, then the problem is transferred to a DSS for possible solution.
 - ii. DSS output as input to ES. The ES interprets the results of the DSS analysis.
 - iii. Feedback. The output from the ES goes to a DSS, and the output from the DSS is fed back to the original ES.
3. Sharing in the decision making process. The ES is separate from the DSS, where the DSS performs the usual functions, and the ES carries out strategy formulation, the only component of the decision making process as proposed by Meador *et al* (1986) which requires judgement and thus requires an ES. The expert system will be a completely separate system (loose integration) but may share the database and perhaps use some of the capabilities of the model base.
4. Generating alternative solutions. Reitman (1982) discusses the possibility for using ES techniques for developing alternative solutions for evaluation by the DSS - as most DSS's only help users evaluate and choose among potential courses of action rather than suggest the alternatives.

5. A unified approach. The ES is placed between the data and the models with the function of integrating the two components in an intelligent manner

2.5.5 INTELLIGENT DECISION SUPPORT SYSTEMS DEVELOPMENT METHODOLOGIES

Blair, Debenham & Edwards (1995b) carried out an extensive study into IDSS development methodologies. This consisted of three stages, firstly they were involved in the development of an IDSS within a large telecommunications company. This highlighted areas where key methodological problems arose. These problems are outlined below, and fall into three categories:

- Requirements analysis
 1. The requirements were difficult to specify, decompose and abstract
 2. The requirements were difficult to understand because more than one model was used
 3. The inter-disciplinary requirements were difficult to analyse and model
- Reuse analysis
 1. Existing software components were difficult to find, understand and cost
 2. Existing software components were difficult to integrate into the IDSS
- Conceptual design
 1. The conceptual model was difficult to specify, abstract and decompose
 2. The conceptual model was difficult to understand, there was no uniform model
 3. There was little support for normalising the conceptual model

Having completed this initial study, Blair *et al.* (1995b) surveyed practitioners in the field to investigate methodologies used. The results of the survey are summarised into the following limitations and benefits:

- Limitations
 1. Requirements analysis, conceptual design and reuse analysis were the least supported phases
 2. The used methods used were drawn from various sources
- Benefits
 1. Some useful diagrams were found to be influence diagrams, structured grammar networks for knowledge acquisition
 2. Expert system based methodologies (Goul & Tonge, 1987) were said to be good for rapid prototyping
 3. KADS was found to be helpful in formalising the design process

Following the survey Blair *et al.* (1995a; 1995b) investigated IDSS methodologies, comparing five recently published formal methodologies. They found that none of the

methodologies investigated provided support for the entire IDSS development process. Each of these methodologies are briefly discussed.

Goul and Tonge (1987) propose an expert-based systems methodology based on DSS design techniques. They draw the concept of adaptive design from DSS techniques, thus suggest prototyping, with user responses to the prototype. The ROMC paradigm is the basis for the approach, with the focus being on the representation (how an IDSS is structured and presented to the user), memory aids (implied by the system and selected by the expert eg dictionary, scratchpad), operations (features intrinsic to the system that cannot be changed by the user) and control mechanisms (selections a user can make during system use). The expert is allowed, within some limits, to choose these features which are the basis for constructing the system.

MEDESS (Van Weelderen & Sol, 1993) is a methodology for designing expert support systems, which are considered by Van Weelderen and Sol (1993) to be synonymous to IDSSs. The methodology is based on the understanding that computers should support rather than replace experts. MEDESS aims to help the designer to pay attention to understanding the problem situation as well as to design a new, improved situation. There are four questions that need to be addressed by the designer - why, what, how and with what. The why problem is described in terms of organisations, organisational structures, problem classes, individuals and tasks. The what problem is described in terms of tasks which are characterised by the information needed to solve the problem, the resulting information, the expert's problem-solving behaviour, and the support structure of an associated ESS. A task is divided into subtasks, and coordination tasks which control the sequence the subtasks are executed. The how problem focuses on how information is grouped and processed (implementation independent), and the with what problem concerns the design of hardware and software.

This methodology appears to have some similarities to the structure of the KADS methodology, with the organisational model being similar to the why problem and the task model being similar to the what problem. The how phase is similar to the development of the conceptual model, and the with what problem and the KADS design model appear to have similar aims.

Angehrn and Luthi (1990) propose the Visual Interactive Modelling (VIM) IDSS methodology. This methodology is based on two principles, the first being "Useability prior to functionality", a user centred approach which aims at designing a flexible, useable working environment the user can interact with effectively. The user interface of

the IDSS is designed before the functions are analysed and specified. The second principle is that of "active cooperation", where the system takes on the role of adviser and facilitator.

Blair *et al.* (1995a) interpret the VIM methodology to include the analysis of the decision context of the IDSS, the determination of an appropriate language and form of communication between the IDSS and decision makers, and "to identify notions, concepts and operations which are familiar to the decision maker and that correspond to his/her knowledge and experience" (Blair *et al.*, 1995a). From there, the fundamental constructs that support the visual interactive environment of the IDSS can be identified (Blair *et al.*, 1995a).

McGovern *et al.* (1991) propose a methodology for the knowledge acquisition phase of IDS development, which results in an influence diagram representation of the decision problem. Five steps are followed for the production of a "first cut" influence diagram from the expression of the problem in textual form. The steps are:

1. "Write a description of the problem in simple declarative sentences"
2. "Identify and isolate decision elements"
3. "Establish relationships between decision elements"
4. "Draw influence diagram"
5. "Add goal node"

(McGovern *et al.*, 1991)

From here each of the nodes is examined to check for clarity and expand them if necessary. Blair *et al.* (1995a) found this methodology to be the least supportive in developing an IDSS, deeming it simplistic and lacking in guidelines on how to break down the complexity of gathering the knowledge for a difficult decision. This is to be expected as it is intended to be a methodology for knowledge acquisition rather than a complete methodology for the development of an IDS.

Holtzman (1989) outlines the "decision analysis cycle" intended to be a means of implementing decision analysis. Blair *et al.* (1995a) consider this to be a methodology for the development of IDSS. This is questionable however, as it goes no further than decision analysis. It appears to be very related to Intelligent Decision System rather than Intelligent Decision Support System development. The decision analysis cycle is divided into two parts; deterministic attention-focusing method which results in a deterministic decision making model and probabilistic decision method, which adds probability measures to the decision model. This is based on decision theory and enables the uncertainty of the decision maker to be modelled.

The idea of decision classes is also introduced by Holtzman (1989), which involves grouping similar decisions together into classes, and associating a collective decision class analysis. This enables individual decisions within a class to be analysed by creating an instance of the collective analysis. This can reduce the expense and time typical of decision analysis as decisions within a class should share a significant amount of structure. Analysing a class of decisions occurs at a higher level of abstraction than analysing a single decision as knowledge pertaining to a particular decision situation is omitted.

The idea of decision classes seems very similar to the principles of the interpretation models in the KADS library. These are generic models which provide a guide to the knowledge acquisition process for collection of domain knowledge as well as a structure to be instantiated and ordered in the task layer.

Following their comparison of the above mentioned IDSS methodologies, Blair (1995) found the following benefits and limitations:

- Benefits:
 1. Influence diagrams are considered a useful tool for modelling requirements
 2. The VIM approach helps address issues of designing user interface
 3. The IDS Methodology supports the modelling decision makers risks
- Limitations:
 1. The methodologies do not address most of the limitations from the project and survey
 2. The methodologies provide little support for the later design phases
 3. The methodologies do not exploit existing KBS and DSS design methodologies

Based on the problems identified in the Blair (1995) and Blair *et al.*, (1995a; 1995b) study, Blair *et al.* (1995b) propose an initial IDSS design methodology. The methodology consists of a lifecycle model for sequencing development phases in the methodology. A development phase is performed by the use of a method which is defined as a "systematic process, technique, or model of inquiry, used to aid in the creation of a satisfactory deliverable" (Blair *et al.*, 1995b). A suitable lifecycle is chosen by the developer of the IDSS which suits their project, such as the waterfall lifecycle or an iterative development process. The development phases, in the order which they occur, are as follows:

1. Requirements Analysis. Results in the "requirements specification model" which is an implementation independent description and functional specification of the problem, along with the external behaviour of the proposed IDSS. This involves constructing a high level context diagram which is decomposed resulting in a fairly complete

requirements specification model. Developers should search for existing components which could be reused in the IDSS.

2. Conceptual Design. Results in a "conceptual model" which is an implementation independent description and non-functional specification of how the IDSS will address the specified requirements. Where non-functional is defined as being "not in the form of what should be stored and what should be deduced" (Blair *et al.*, 1995b). This uses the results of phase one which is refined and specified at a low level with enough detail for programmers to implement it. The interconnections with existing software components are then modelled.
3. External Design. Results in an "external model" which contains implementation details as to what is required to solve the problem.
4. Internal Design. Results in an "internal model" which specifies what is stored and deduced by the system, it is a complete specification for the programmer.
5. Physical Design. Results in a "physical model" which is the actual implementation of the system, and involves using the internal model and a specified implementation platform.

Blair *et al* (1995b) does not examine the methodologies of Turban (1995), Gillies (1991) or Klein *et al.* (1995). Turban (1995) expands his earlier DSS methodology by adding the design of a knowledge component to his conventional development process, thus making it pass as an IDSS development methodology. This is in line with his proposed architectures, keeping the intelligent component separate to the rest of the system.

Gillies (1991) proposes three strategies for the development of integrated systems (integration of conventional and knowledge based systems). These strategies are:

1. Structured integrated expert system (IES) methodology

This is recommended if "the system has been conceived as an information system in the first instance based upon algorithmic solutions, and if the benefits sought from integration are greater flexibility and the ability to handle problems in terms of knowledge and symbolic reasoning" (Gillies, 1991). This methodology is an extension of the traditional waterfall lifecycle, and the stages are as follows:

- i. Knowledge elicitation for requirements analysis. The scope of the problem is defined, involving communication between the domain expert and the knowledge engineer.
- ii. Structuring and Software Design. Produces a modular structure for the system, resulting in a data flow diagram and then a structure chart.
- iii. Task allocation. The tasks are allocated to either the information system, the expert system or the user, or the task may be considered a compound task.
- iv. Implementation and unit testing - The conventional modules can be represented using conventional structured code. The knowledge based modules can be

represented in an AI environment such as a PC based shell, which may be able to be converted into an algorithmic approach later.

- v. Synthesis and Integration - The modules are integrated; the difficulty of this phase depends on the compatibility of the modules.
- vi. System Testing and Installation
- vii. Operation and Maintenance

2. The data-centred approach

This approach is suitable when a data base management system is envisaged, but extra features such as data sorting and extraction are also desired. Gillies states that "Many organisations are now facing serious problems in terms of an 'information overload'. By designing an expert system to run on top of the database, the data extracted may be first sorted by the computer, in order to present the user with more useful information" (Gillies, 1991). This approach is divided into design and implementation stages.

The design stage is divided into ES design and database design. The first design stage is that of the database using a recognised methodology, the resultant data model (such as an entity relationship model) is used as a starting point for knowledge elicitation for the expert system. Gillies states that "the knowledge required from the elicitation process is concerned with how the client makes use of the data in the database and what knowledge he uses in the process". A decision tree may be derived. If the expert system was stand alone then the answers to the questions would come from the user, but in this case, they may be drawn directly from the database (Gillies, 1991).

One method of implementation is having the two systems separate, linked by data exchange; the expert system replaces the user input with automated input. Gillies (Gillies, 1991) comments that this will not be producing a truly integrated system but may provide the best solution. He warns of problems in maintaining integrity between the two systems. Gillies outlines the use of object-oriented programming techniques highlighting its advantage in combining both knowledge and data records. The other implementation path outlined by Gillies is that of PROLOG environments.

3. The Prototyping and porting (ProP) approach

This approach has a number of deliverables in a sequential development process. The first deliverable is a prototype system "which must handle all the functionality of the problem, but does not have to conform to the full specification in terms of speed performance and method of implementation" (Gillies, 1991). Typically, it is built within an AI toolkit environment. The purpose of the prototype is to prove that a system is feasible, and it is developed using traditional expert system development lines.

The intermediate system is then developed and is a complete rewrite of the prototype in a transparent well-structure language such as C or Pascal (Gillies, 1991). The intermediate may end up as the final system, or further development may be necessary to come up with the final system.

Gillies (1991) suggests the adaptation of any of the above strategies to fit the problem as he feels the "ill-defined nature of expert system problems does not lend itself to the rigid application of a methodology".

Klein and Methlie (1995) propose a methodology for the implementation of a knowledge based decision support system (KB-DSS) which is outlined in Figure 2.16. The first two steps are involve becoming familiar with the users goals and defining the problem. Step three involves understanding and modelling the actual decision processes. This is important in assessing what improvements are feasible. This step may utilise influence diagrams to represent the decision situation. The normative point of view is modelled in step four, which can be elicited from one or more experts. An important task in this step is the definition of how various resources are utilised by the expert such as models and data.

Step five utilises the descriptive and normative decision processes to come up with a feasible decision process to be implemented in the KB-DSS. Which part of the decision process to support is defined in step six. The functional analysis step involves defining the main functions and the overall architecture of the system, deciding which components and resources are needed by the KB-DSS.

The design and implementation phase is divided into six tasks

1. data analysis and modelling
2. form definition and input verification
3. decision model design and testing
4. report definition
5. knowledge base modelling and testing
6. overall user interface design and global application logic definition

In performing task six Klein and Methlie (1995) comment that "...in our experience, it is very important to have the possibility of demonstrating simplified knowledge bases to the expert in order to help him or her structure his or her knowledge. Running such a simplified knowledge base helps very much the expert to define his or her ideas". This idea seems very similar to the principles of the VIM (Angehrn & Luthi, 1990) approach.

The final three steps involve testing the KB-DSS, educating users and monitoring and evaluating the performance of the system.

Klein and Methlie (1995) suggests an evolutionary design strategy which involves implementing a first version of the system which may only include a simple interface, a simple decision model, a first data file and a knowledge base containing only a few rules. Such a system helps users define their needs for data, interfaces, decision models, reports, input forms and type of assistance requested from the intelligent component. The ability to utilise an evolutionary design process depends on the flexibility of the development environment used.

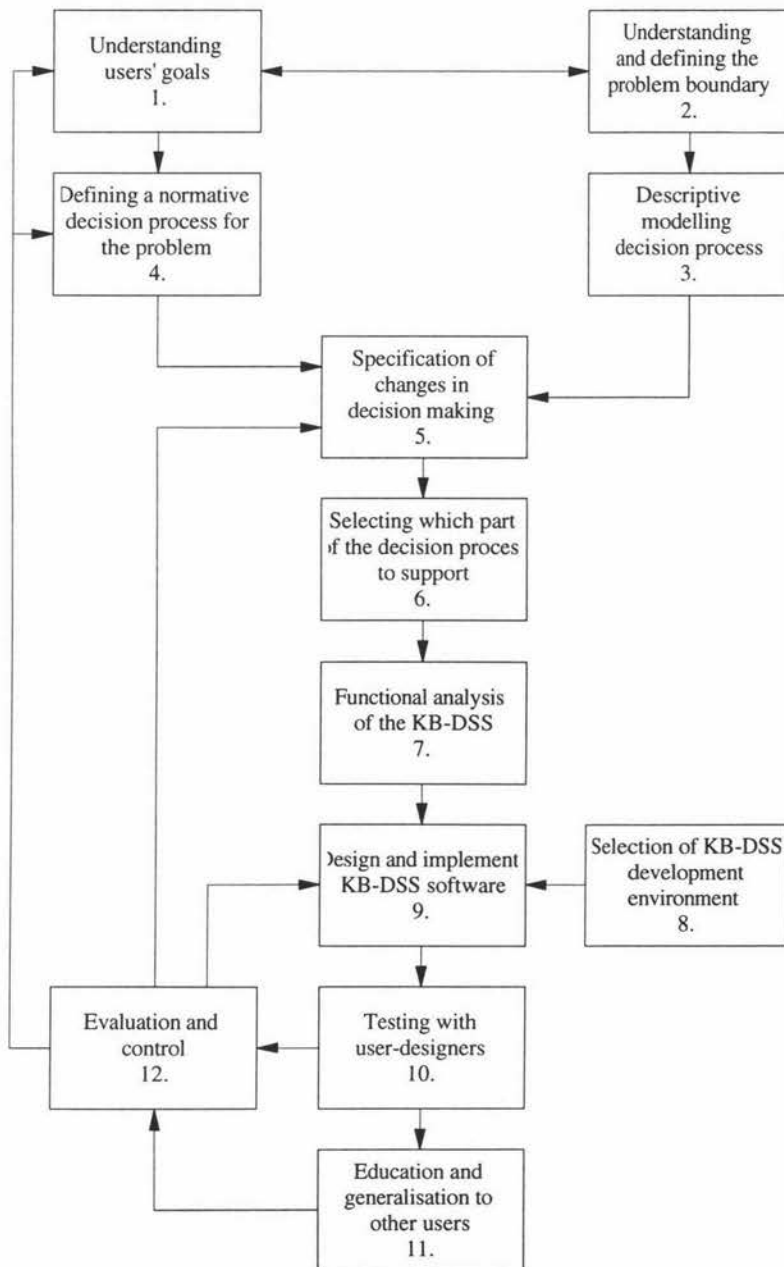


Figure 2.16 - KB-DSS implementation process (Klein & Methlie, 1995)

2.6 DECISION ANALYSIS & INFLUENCE DIAGRAMS

Decision analysis and influence diagrams have been discussed by a number of authors for use when developing both DSS and IDSS (Arnott, 1992; Holtzman, 1989; McGovern *et al.*, 1991; Stabell, 1983). The discipline of decision analysis combines aspects of systems analysis and statistical theory, and results in knowledge that can deal practically and generally with the logic of making choices in complex, dynamic, and uncertain situations (Holtzman, 1989).

Smith (1988) sees decision analysis as a means of organising a client's information into a coherent picture of the problem, so that the best course of action can be taken given the client's stated beliefs and objectives. The reasons for why the chosen course of action is optimal should be able to be communicated, and a framework of the decision situation, which can be critically appraised and modified, should be developed. Similarly, Arnott (1992) considers the outputs of decision research to be "a detailed description of the decision process that is understandable to both user and analyst". Arnott (1992) feels this should incorporate both a normative and descriptive view of the decision process. Decision analysis is seen as playing an integrator role between the decision participants.

Decision trees are the best known representation language for decision problems, making each controllable and uncertain variable of a choice problem explicit (Holtzman, 1989; Smith, 1988). However, Holtzman (1989) points out a number of drawbacks of decision trees including their inability to exploit independence relations, and their necessity to be symmetrical if information other than an optimal policy recommendation is required. Furthermore, they grow exponentially and so anything other than a small problem becomes impractical. Holtzman (1989) sees significant theoretical and practical advantages of influence diagrams over decision trees. Influence diagrams are an effective communication language that promote efficient, goal-directed generation of decision models (Holtzman, 1989). Arnott (1992) considers influence diagramming to be a useful tool for describing decisions prior to DSS construction.

Gottinger and Weimann (1992) discuss the use of influence diagrams as a means of decision model based representation. Influence diagrams are network depictions of decision situations, and have previously been used to elicit and communicate the structure of decision problems. More recently, their use in providing a complete mathematical description of a decision problem, and as representations for computation has been investigated (Gottinger & Weimann, 1992). Turban (1995) describes influence diagrams as a graphical representation of a model which provides a visual

communication to the model builder and as a framework for expressing the exact nature of the relationship within the model. Shachter (1986) claims that influence diagrams are "intuitive enough to communicate with decision makers and experts and, at the same time, precise enough for normative analysis".

O'Donnell (1995) warns of the use of influence diagrams for knowledge acquisition in the way decision analysts such as Holtzman (1989) and Shachter (1986) use them. O'Donnell (1995) believes decision analysts impose an unnatural normative structure on the influence diagram, and sees them as a more valuable tool when used descriptively, in the way Bodily (1985) uses them. It is this method of using influence diagrams which is seen as valuable in knowledge acquisition - representing the decision process of an expert in a descriptive manner.

The notation of influence diagrams varies between authors. That of Bodily (1985) will be discussed, as it is this notation which is supported by O'Donnell (1995) and O'Donnell and Watson (1994).

An influence diagram is made up of nodes and links where the node represents a variable or decision alternative, and the links show some type of influence. Decision nodes are usually represented by a rectangle or a square and denote choices or alternatives facing the decision maker. Intermediate variables are any variables necessary to link decision variables to outcomes, and are represented as circles. Attributes or outcome variables are those used by a decision maker to measure performance, and are represented by ovals. Figure 2.17 shows an example of an influence diagram from Bodily (1985). Random variables, which model uncertainty, are noted with a tilde (~) above the variable, and any variable influenced by a random variable is also a random variable.

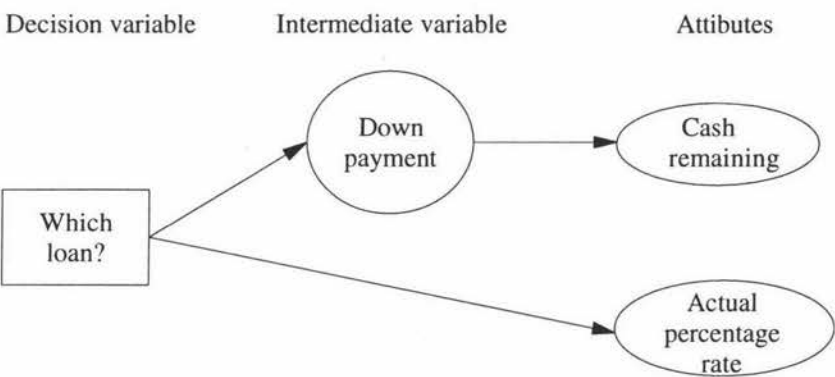


Figure 2.17 - An influence diagram for a borrowing decision (Bodily, 1985).

There are three types of influence arrows. A certain influence is represented by a single straight arrow, and is used when one node will, without question, affect the value of the other node in the relationship. An uncertain influence is represented by an arrow with a sharp bend in it, and is used when one node will affect another, but it is uncertain by how much. A double straight arrow signifies a preference dependency, which indicates that the decision maker's preference for an attribute is influenced by the level of the predecessor attribute. A preference variable indicates an influence on the desirability of the influenced variable, and not its level. Examples of each type of influence are shown in Figure 2.18. For a more detailed description of influence diagrams the reader is referred to Bodily (1985).

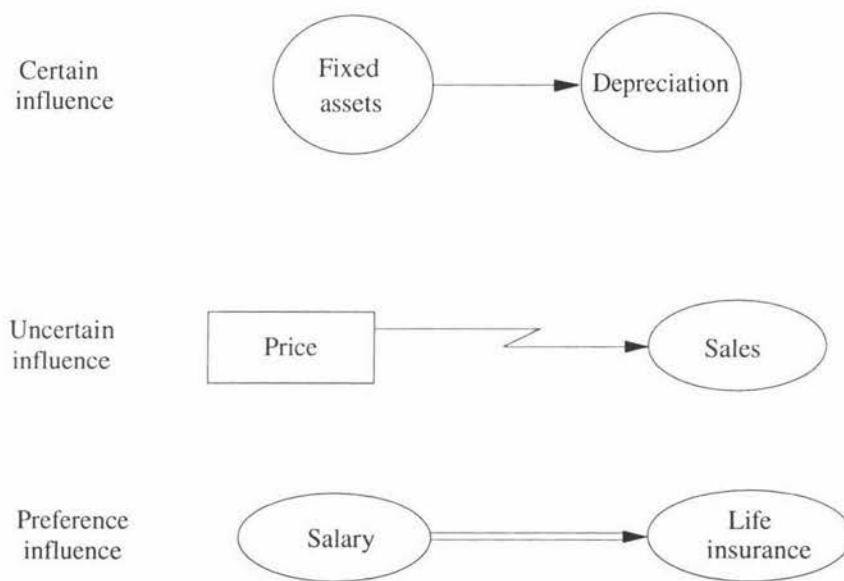


Figure 2.18 - Different types of influence (Bodily, 1985).

According to Holtzman (1989), a full description of a decision problem requires that the diagram contains at least one decision node directly or indirectly influencing a variable node and that consistent, detailed specifications exist for each node in the diagram. An influence diagram which is structurally complete, but has not been specified in detail is said to be defined at the level of structure. A completely developed and specified diagram is said to be defined at the level of function and number, and a well formed diagram must be consistent between all levels. Well formed influence diagrams can be manipulated by performing four operations; reversing an influence, merging two nodes, splitting a node and removing a node.

2.7 CONCLUSION

This chapter firstly investigated the decision support systems literature, to provide an overview of their capabilities and development approaches. Similarly, the knowledge based systems literature has been briefly reviewed. The combination of the two types of systems has been discussed by a number of authors, and the resultant intelligent decision support systems is a growing area of research. The associated literature has been studied to discover existing methodologies and techniques for the development of an IDSS. Finally, decision analysis and influence diagrams have been discussed, as it was discovered these are closely related to DSS and IDSS development.

Having completed a thorough review of relevant literature, it is clear that, to the author's knowledge, there are no methodologies for the development of an IDSS from an existing DSS. Chapter 3 will propose a framework for this type of project, based on the concepts discussed in this chapter.

3.1 INTRODUCTION

This chapter outlines the initial proposed framework for guiding the development of an IDSS from an existing DSS. It might be thought that an intelligent component developed for interaction with an existing DSS could be developed quite separately from the DSS using a conventional KBS methodology. However, the intelligent component will differ from a normal KBS because of its interaction with an existing DSS.

DSS, KBS and IDSS methodologies have all been investigated because principles from each could be used within a methodology for developing an intelligent component for integration with an existing DSS. An initial framework was developed from reviewing the literature, combining what appeared to be appropriate techniques.

3.2 THE FRAMEWORK

The aim of the framework is to provide a means of developing and integrating an intelligent component with an existing decision support system. The proposed framework is based on the data centred approach of Gillies (1991) (Section 2.5.5), which separates the design of the conventional system and the knowledge based system components. Similarly, Turban's (1995) (Section 2.2.3) methodology separates the design of the intelligent component from that of other components. Gillies considers the conventional systems design to be the first step in the knowledge acquisition process, as it familiarises the developer with data needs of the problem.

The methodologies which integrate the design of the knowledge based and conventional features of an IDSS (such as Goul & Tonge, 1987 and Blair *et al* 1995b) were regarded as inappropriate in a situation where the DSS already exists. This is because complete analysis and design of conventional features is not necessary, as they are already implemented in the DSS.

The first step in the proposed framework involves gaining an understanding of the existing DSSs capabilities and associated data. The existing DSS may already be fully documented and modelled, which would provide a useful means for the knowledge engineer to familiarise him/herself with the data and functions of the system. However, if

models did not exist, or were out of date, the system would have to be fully analysed. The intelligent component could then be designed in light of existing capabilities.

A prototyping lifecycle has not been suggested within the Gillies (1991) methodology. However, prototyping and evolutionary design are considered important aspects of many KBS, DSS and IDSS methodologies (Keen & Scott Morton, 1978; Alter, 1994; Goul & Tonge, 1987). The very structured approach to prototyping proposed by Duffin (1988) (Section 2.3.4) has been suggested for use within the proposed framework. A prototype is seen as a useful communication tool between the users, the expert and the developers. Duffin (1988) suggests clearly defining the aims of the prototype before it is developed, and evaluation of the prototype is required before continuing development.

The concept of "useability prior to functionality", suggested within the Visual Interactive Modelling (VIM) IDSS methodology (Angehrn & Luthi, 1990), was also seen as suitable for inclusion in the framework. Functionality could be simplified in early versions of a prototype (if one is developed) so that the prototype system could be implemented quickly to aid in communication with the user or the expert. The available tool for this case study is KAPPA-PC which offers a very flexible graphical user interface facility, well suited to the VIM and structured prototyping concepts.

An initial framework is proposed for guiding the development of an IDSS from an existing DSS. This is summarised in Figure 3.1. The framework is based on Gillies (1991) data centred approach, as the existing DSS will dictate much of the structure of the final IDSS. A prototyping process can be utilised within the framework, incorporating the ideas of VIM.

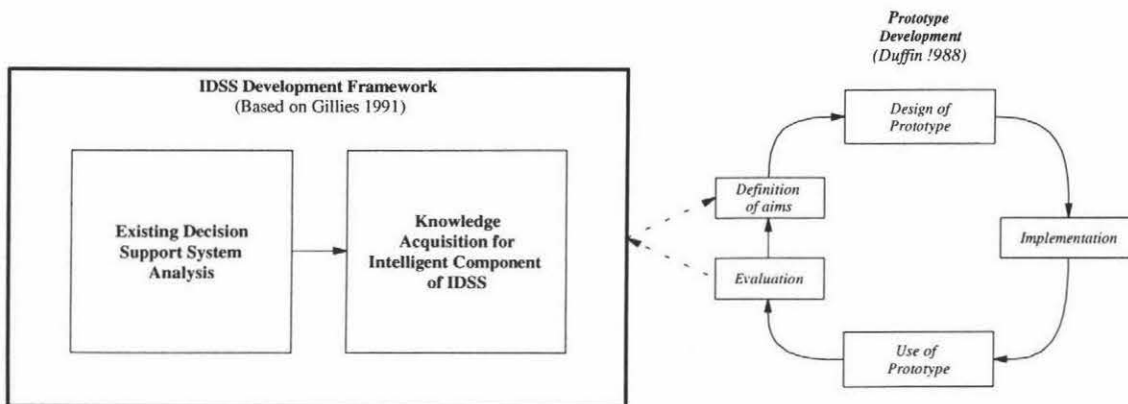


Figure 3.1 - Development framework outline

Methods from the reviewed literature were chosen, and modified where appropriate, to achieve the desired tasks within the proposed framework. The first was for analysis of the existing DSS, and the second involved the knowledge acquisition process. These sub tasks are outlined below.

3.2.1 EXISTING DECISION SUPPORT SYSTEM ANALYSIS

Meador *et al.*'s (1986) Decision Support Analysis (Section 2.2.3) provides a basis for the analysis of the existing decision support system. It is intended to be the first stage in the development of a large scale DSS, however it has been modified as follows for use in analysis of an existing decision support system.

The structured interviews are intended by Meador *et al.* (1986) to allow the systems analyst to become aware of areas within the business which require decision support. In this situation they provide a good understanding of the decision situation which is supported by the system, thus establishing a good basis for the decision analysis stage, and also the knowledge acquisition process.

The decision analysis stage is described in Section 2.2.3, and aims to identify the functional requirements of the system. Three steps are involved in decision analysis: business area analysis, description of logical functional flow and specification of detailed decision areas. The functionality of the existing DSS is seen as all that needs to be analysed to understand its decision capabilities. It was not seen as necessary to go to the depth of identifying decisions, as it is not the intention of the framework to change the existing systems method of aiding the decision maker. The business area analysis involves studying the entire organisation, whereas this framework is only concerned with the existing system. Thus both the business area analysis and the identification and classification of decisions were omitted from this framework, and the decision analysis stage has been renamed "functional analysis".

Functional analysis involves the development of functional flow diagrams, which are a standard means of representing the functionality of the existing DSS. Data flow diagrams are a suitable tool for this activity, due to their popularity amongst DSS developers, as discussed in Section 2.2.4. If data flow diagrams already exist for the DSS, it is not necessary to develop them again. It is necessary, however, for the knowledge engineer to be familiar with them, and ensure they are up to date. Studying or developing such diagrams provides the knowledge engineer with a detailed understanding of the purpose and capabilities of the existing system.

The data analysis stage of decision support analysis results in a data model such as an entity relationship diagram, which should be consistent with the data described in the functional model. The development of such a model for an existing system enables the knowledge engineer to become familiar with the data involved in the system, an important aspect for the integration with a KBS component. Once again, if a current data model already existed, the redevelopment is not necessary.

The technical analysis stage of decision support analysis, is seen as necessary when the DSS exists, so as to understand the options for integration with a KBS component. This requires the investigation into features such as systems compatibility and existing file structures, and could dictate the hardware and software suitable for the development of the additional component.

3.2.2 KNOWLEDGE ACQUISITION FOR THE INTELLIGENT COMPONENT

The adaptation of the KADS methodology from Kemp *et al.* (1994) (Section 2.3.4) was seen as suitable for combining with decision support approaches as it concentrates on the knowledge acquisition process.

An additional step has been included within the knowledge acquisition process which involves defining the aims of the intelligent component. Similar steps are included within the IDSS methodologies of Blair *et al.*, (1995b), Klein and Methlie (1995) and the IES methodology of Gillies (1991). The step aims to define the scope and functionality of the intelligent component, and requires communication between the knowledge engineer and the domain expert.

The first step in Kemp *et al.*'s (1994) process is knowledge elicitation, which aims to obtain information related to the problem domain. This is most likely to take the form of interviews, and could overlap the content of the initial interviews in the analysis of the existing DSS, and the discussions defining the aims of the intelligent component.

The analysis phase of the methodology involves the identification and representation of the domain knowledge into concepts, attributes and relationships. Knowledge acquisition and decision analysis could be viewed as similar as both are attempting to elicit and represent a clients conceptualisation and activities in a problem situation. Thus methods of decision research were investigated for inclusion into the analysis phase. A method of knowledge acquisition and a method of decision research which appear to

complement one another were chosen to make up the analysis phase. These are outlined below.

Sowa's conceptual analysis provides a complete and consistent means of representing information in the form of conceptual graphs. As suggested by Jeffries, Todd and Kemp (1995a), they could be a suitable means for representing information within the structure of the KADS model of expertise.

Influence diagrams have been recommended by a number of authors for decision analysis (Arnott, 1992; Gottinger & Weimann, 1992; Holtzman, 1989). McGovern *et al.* (1991) recommend the use of influence diagrams for knowledge acquisition in IDSSs. Thus influence diagrams appear to be a suitable tool for incorporating into the analysis phase of knowledge acquisition. A five step methodology for analysing text to develop influence diagrams is proposed by McGovern *et al.* (1991) (Section 2.5.5), which fits into the overall framework, as interview data from earlier stages can be analysed.

The interpretation stage of Kemp *et al.*'s methodology uses knowledge of the domain to select interpretation templates from the KADS library. Once identified, these can be used to guide the continuing knowledge acquisition process, and the development of the model of expertise. The task model is seen as a useful tool in identifying the interpretation templates, thus the two processes of interpretation and modelling are related and are likely to occur simultaneously.

The resultant models of Kemp *et al.*'s knowledge acquisition process are the task model and the model of expertise. It is envisaged that the conceptual graphs and influence diagrams created in the analysis phase will fit into the domain layer within the model of expertise.

The KADS model of cooperation (Breuker & de Greef, 1993) is also seen as useful in an IDSS situation, as its aim is to describe further the tasks in the task model. This involves assigning each task to either the user or the system, and describing the functionality of these tasks. Gillies (1991) mentions the use a model of cooperation in integrated systems. In addition to the standard specifications, he suggests the cooperation between information and knowledge processing be examined and modelled to achieve full integration. Thus, the model of cooperation could be modified slightly for IDSSs, by assigning tasks to the KBS, the DSS or the user. This is similar to the task allocation step in the IES methodology of Gillies (1991), where tasks are assigned to the

conventional system, the expert system or the user. Tasks which are performed by more than one party are considered by Gillies (1991) to be compound tasks.

The model of cooperation provides a useful means of representing the interaction and cooperation between the different components of the IDSS. Its development requires information from the functional and data models, and the technical analysis of the existing DSS. These are required to understand the DSSs current capabilities, available data and technical constraints on interaction. The creation of the model of cooperation is included in the modelling phase. The model of cooperation is intended to describe the link between the different components of the proposed IDSS.

The detailed initial proposed framework is outlined in Figure 3.2.

3.3 CONCLUSION

This chapter has proposed an initial framework for the development of an IDSS from an existing DSS. The framework provides the basis for the remainder of the study; it is to be applied to a real problem to assess its feasibility and highlight areas for improvement. Chapter 4 discusses the problem domain and the existing DSS which was chosen to test the framework. The steps involved in applying the framework to the domain are documented in Chapter 5, along with relevant examples. A prototype intelligent component was developed to aid in evaluating the framework, and details of the development and functionality of the prototype are outlined in Chapter 6.

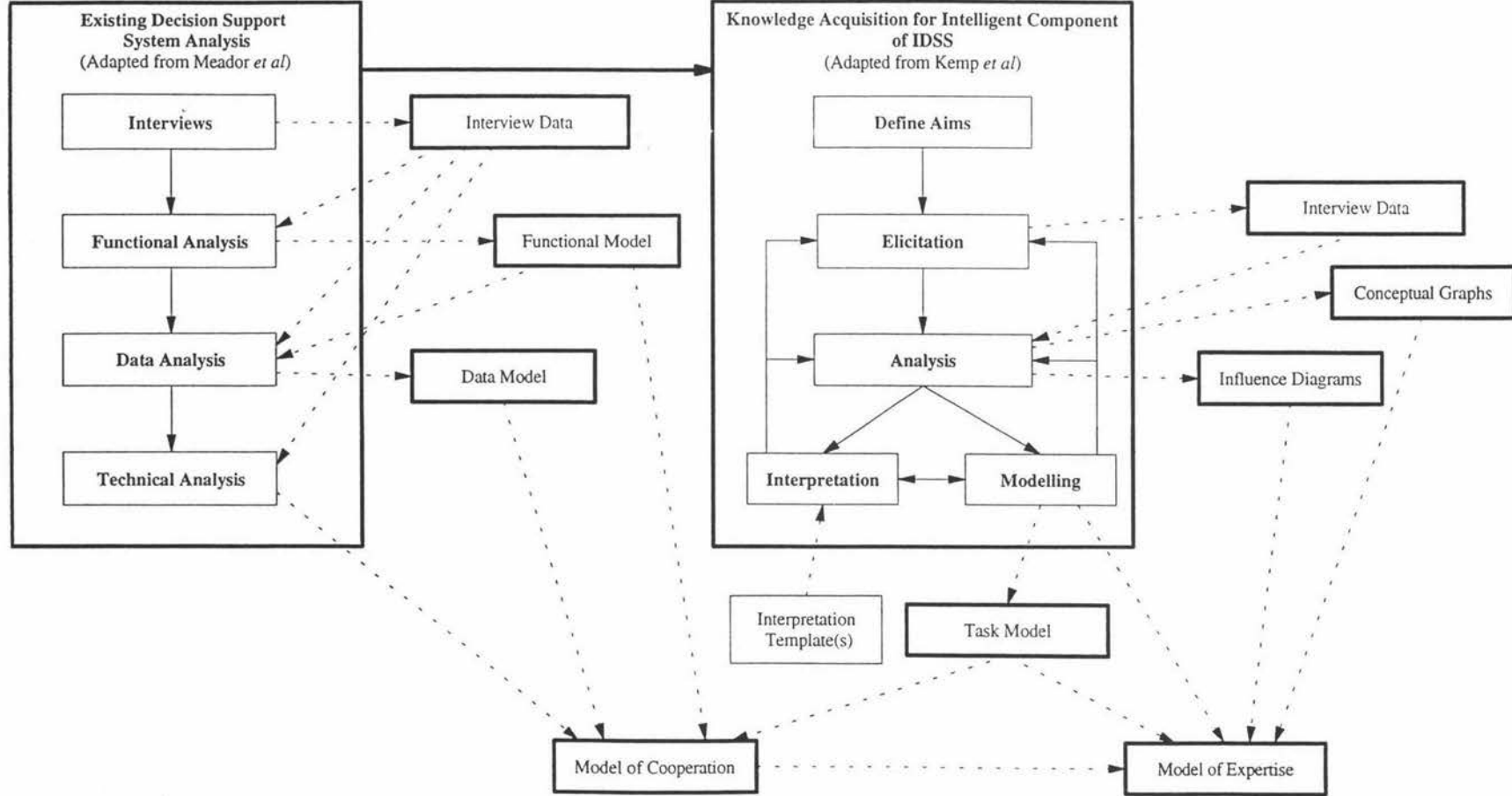


Figure 3.2 - The initial proposed development framework

4.1 INTRODUCTION

Decision support systems have been found to be useful within the agricultural domain by a number of authors (Bennett, 1992; Lal, Jones, Peart & Shoup, 1992; Mainland, 1994). Agricultural decision support systems available on the New Zealand market include FarmTracker, DairyMan, UDDER and StockPol¹. These systems aid in the management of farm data. However, they do not possess capabilities for fully interpreting the information presented to the farmer.

The use of knowledge based systems technology in the agricultural domain has also been investigated by a number of authors (Beck & Jones, 1989; Jones *et al.*, 1987; Jones, 1989; Plant & Stone, 1991). Current areas of application include, for example, diagnosis of diseases in crops and planning for fertilisation or irrigation (Plant & Stone, 1991). Jones *et al.* (1987) describe a decision aid which links an economic model with a database of insecticide attributes and rules of thumb from an expert, which together make a recommendation for applying insecticide to a crop. Such use of simulation models in conjunction with knowledge based systems techniques is also discussed by Plant and Stone (1991) and Jones (1989). It is felt that the use of models is restricted to users who are very familiar with the specific capabilities and limitations of the models. The combination of models with heuristics is seen as a way of increasing the useability of the models (Jones *et al.*, 1987).

Interpretation of large quantities of data, such as the outputs from simulation models is also seen as a potential use of knowledge based systems in agriculture (Plant & Stone, 1991). Plant and Stone (1991) comment that "Databases can overwhelm the farmer, who must be expert in so many disciplines to achieve success. Farmers cannot examine all the information relevant to their decisions; there isn't time and they do not have the expertise".

This combination of knowledge based system and decision support system techniques is a growing area of research within the agricultural field. However, there is limited work on the improvement of existing agricultural DSSs by integrating them with an intelligent component. It is this improvement of existing DSSs which the framework proposed in

¹Much of this chapter has been developed following personal communications with David Gray, 1996

Chapter 3 is directed towards. The agricultural domain, and one of its existing DSSs, is seen as a suitable testing ground for the framework.

The selection of the farm management tool to be the case for application of the development framework is discussed in Section 4.2. Relevant aspects of the problem domain are outlined in Section 4.3. Section 4.4 provides an overview of the chosen system, to give a basic understanding of its current capabilities. Finally, the case is discussed in relation to the list of capabilities of a DSS compiled in Chapter 2.

4.2 CASE STUDY SELECTION

"FarmTracker" (developed by B.M. Butler Computing Limited) is a decision support tool which was designed to help farmers monitor and plan their farm situation. Relevant data is entered into the system, and can be manipulated and presented to the farmer in a variety of ways. FarmTracker includes feed budgeting facilities (a planning tool) with associated simulation models of pasture growth and stock intake and performance.

Although FarmTracker aids farmers in collating, manipulating and displaying summaries of data for planning purposes, it does not show farmers how to use this information. The system does not automatically develop feasible plans or evaluate a plans success. The farmer is not advised on possible reasons for not achieving targets and the system does not help the farmer come up with a new plan that minimises deviation from targets.

An intelligent component integrated with FarmTracker may be able to assist the farmer in developing and evaluating management plans. FarmTracker alone cannot provide this type of support due to its lack of heuristic reasoning abilities.

FarmTracker's developer has a close association with Massey University, and was willing to become involved in the study. B.M. Butler Computing Limited is a locally based firm making its involvement in the study convenient. The developer of the system is also considered an expert in the domain of dairy farm pasture management (Butler, 1986), making him a suitable candidate for domain expert. He is also an expert user of the FarmTracker system. Throughout the rest of this study the FarmTracker developer will be referred to as the sponsor, taking the role of domain expert, DSS developer and expert user.

Farm management is a large and complex domain. Due to the time constraints imposed on the study, it was decided that the research would focus on pasture management during part of the dairy farming year. The sponsor requested that the study address the area of early spring pasture management. He considered this to be a critical period of the year, having a large affect on milk production, with the farmer looking towards having his stock in good condition at mating. This is a tightly defined management segment, which is relatively well understood by domain experts.

4.3 THE PROBLEM DOMAIN

A dairy farm is a complex system involving humans, animals, pastures and soils. One of the aims of effective dairy farm management is the achievement of high levels of profitability and production without jeopardising the health of the animals and avoiding adverse effects on the environment. The domain is complex because it is a biological system with a number of factors affecting it, this makes it difficult to predict the systems behaviour. There is a considerable amount of data which can be collected to aid farmers in their decision making, such as pasture cover levels, pasture growth rates, milk production levels, weather information, cow condition, the liveweight of stock, input costs and market information. A "knowledgeable" farmer is able to utilise this data for tactical decision making by using systems models and rules of thumb to predict future events and act accordingly.

The management of the early spring period is considered critical for achieving high levels of production and profitability. An important aim during early spring is to fully feed the cows during early lactation. This directly results in high short term milk production, and also effects milk production in the longer term. Feeding cows well at this time also ensures the cows are in good condition at mating (mid spring) which results in good reproductive performance.

Correct tactical management throughout early spring is necessary in order to fully feed cows during this period. It is also important to ensure adequate pasture cover and supplements are on hand, and cows are in good condition, at the time of calving (the start of the "early spring" period). These aims are often difficult to achieve because pasture growth is weather dependant and therefore highly variable. Therefore it is often necessary to implement contingency plans to minimise the associated impact of this highly variable feed supply. Experienced managers plan for an average year, and build in contingency plans which are designed to minimise the impact of variable pasture growth rates.

Tactical management is used to "control" the dairy system and minimise the impact of variability of feed supply. This involves a cyclical process of planning, implementation, monitoring and evaluation (Todd, Gray, Lockhart & Parker, 1993). The farmer develops a short term (2-3 month) plan from calving until feed supply exceeds feed demand during September or October. This plan includes pre-determined targets for animal intakes, production levels, cow condition, pasture growth rates, pasture cover levels, the use of feed supplements (timing, quantity and type used) and the number of cows on hand at any point in time. The plan is implemented and the farmer then monitors the system to identify any deviations from the plan. If deviations are identified, the system is then evaluated to identify reasons for the deviation. A new plan is then developed to minimise the impact of the deviation. This process requires the farmer to decide upon the best course of action from a large set of alternative solutions.

Knowledgeable farmers often use a planning tool known as a feed budget to help them manage the early spring period. A feed budget balances feed demand and feed supply. It quantifies the relationship between pasture growth and other feed sources and animal feed requirements over time, and is used to identify potential feed supply problems.

Feed budgets can be used to analyse "what if" scenarios and develop the best plan for the early spring period. For example, the farmer can experiment with various combinations of feeding supplements, applying Nitrogen, reducing cow intakes or stock numbers. The plan also documents important targets (such as pasture cover levels and production levels) at weekly or two-weekly intervals, which can be used for control purposes. These targets can then be compared to actual levels of performance to identify potential problems. Figure 4.1 shows a feed budget summary report produced by FarmTracker for a trial budget.

A feed budget can be summarised by the following formula, where t_0 is the start of the period, and t_1 is some point in time in the future.

$$\text{Average pasture cover } t_1 = \text{Average pasture cover } t_0 + \text{Feed supply} - \text{feed demand}$$

Average pasture cover is a measure of the average level of pasture on a farm at any time, and is measured in kilograms of dry matter per hectare (kgDM/ha). Pasture cover is often estimated by eye assessment. Alternatively, pasture plate meters can be used which measure pasture height, and associated equations convert this measurement to kgDM/ha. More accurate methods such as cutting, drying and weighing a measured area, provide a means of calibrating the commonly used methods.

Feed budget Summary for Trial budget 3		
Total Area used in Budget : 89.8 ha		
	Total kgDM	kgDM/ha
Initial Pasture Cover 01/05/95	195922	2182
SUPPLY		
Pasture	487195	5426
Conservation	23745	264
Supplementary Feed	22648	252
	-----	-----
Total Supply	486097	5414
DEMAND		
	402811	4486
	-----	-----
Final Pasture Cover 31/10/95	278888	3106
Final Target Cover	197538	2200
FEED SURPLUS		
	81350	906

Figure 4.1 - Feed budget report produced by FarmTracker

Average pasture cover will increase if feed supply exceeds feed demand and decrease if feed demand exceeds feed supply. In order to fully feed cows during early spring it is necessary to ensure the pasture cover average does not fall below a certain minimum (approximately 1800 kgDM/ha in the Manawatu). This level enables cows to be fully fed without jeopardising pasture growth rates. However, problems can also arise if pasture cover goes beyond a certain critical maximum during spring due to the increase in dead matter in the sward resulting in decreased pasture quality.

Feed budgeting is necessary to ensure pasture covers remain within the critical maximum and minimum levels. Guidelines to these levels are gained from experience on farms; levels found on a Massey University farm is shown in Figure 4.2.

FEED SUPPLY

Pasture is the major source of feed on a New Zealand dairy farm supplying some 80 - 90% of the feed used on a dairy farm. Pasture production is a function of environmental and management factors. Simulation modelling shows that the most important determinants are rainfall (soil moisture), temperature (soil and air), evapotranspiration (wind), pastures species and soil fertility (Gray & Parker, 1992). Pasture growth rates can be increased to a certain extent by the tactical application of nitrogen fertiliser which is considered a form of feed supplement.

The other components of feed supply are supplementary feeds such as hay, silage, crops, grain and meal. These may be made on the farm or bought in. This is a more expensive method of feeding and so should only be considered when the pasture supply is limited

(Holmes & Wilson, 1987). The farmer can manipulate feed supply by applying nitrogen fertiliser to improve pasture growth rates, or by using some form of supplementary feed.

FEED DEMAND

The feed demand on a property consists of two components; feed eaten by livestock and feed harvested as hay or silage. Only the former is important during the early spring period. The amount of feed eaten by livestock depends on the number of animals (stocking rate), the size and age of the animals and their physiological state (non-pregnant, pregnant, lactating). Feed requirements of the herd can be calculated, given certain performance and reproductive targets. The farmer can control feed demand by; (1) manipulating stocking rate through buying or selling stock, or grazing stock off the farm, and (2) controlling how much feed the animals receive.

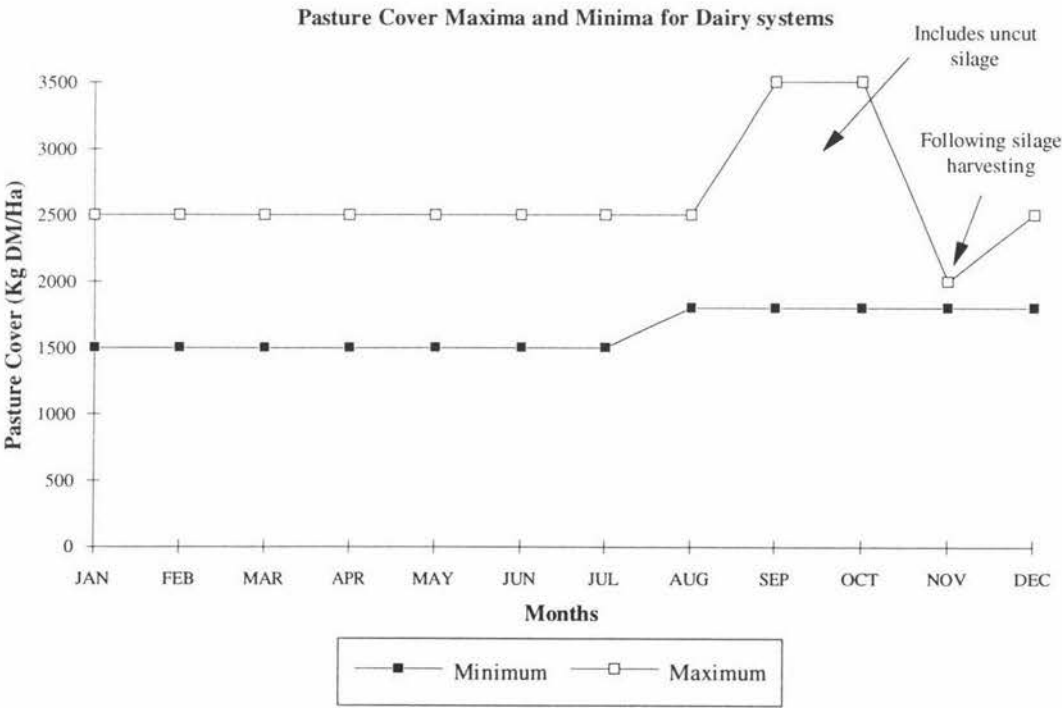


Figure 4.2 - Desirable pasture covers for dairy systems (Gray & Parker,)

4.4 FARMTRACKER

4.4.1 FARMTRACKER OVERVIEW

FarmTracker is a decision support tool used by farmers for planning and control of their pastoral system. Therefore, the DSS has a planning tool - a feed budget, which allows farmers to develop plans and evaluate the outcome using simulation models for cow intake and pasture growth rate. Relevant historical data is recorded by the system, and can be displayed and manipulated by the farmer, and used within the feed budget. FarmTracker is divided into a number of modules each dealing with a particular aspect of the farm. These modules are farm/weather, paddocks, feed, stock and maps. Figure 4.3 provides an overview of the FarmTracker system, this is followed by a discussion of each module.

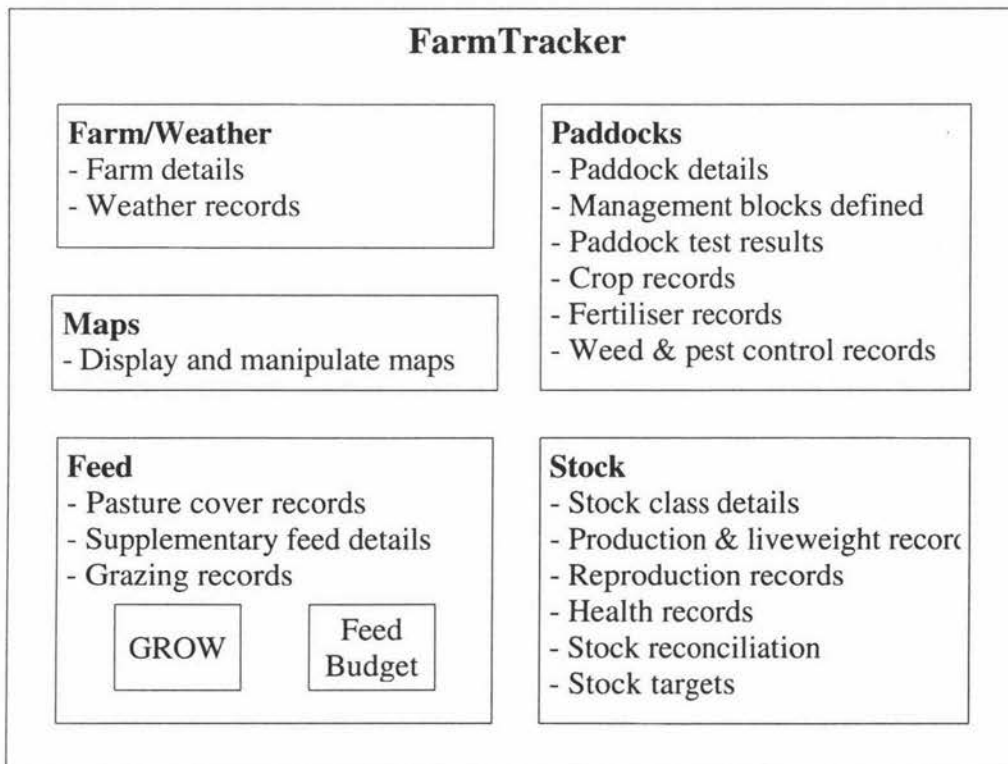


Figure 4.3 - Overview of FarmTracker

The **Farm/Weather** module manages information regarding the farm such as its area, climate and location. Enterprises such as dairy, sheep or cattle for a specific farm can be selected from a master list. This determines default attributes within other modules. Weather data can be entered in half monthly periods, and includes annual and average weather information which can be used to predict future pasture growth rates or analyse actual pasture growth rates.

Information about each paddock is recorded, stored and manipulated within the **Paddocks** module. The data stored includes paddock area, paddock name and pasture types. Management blocks can be defined as groups of paddocks. Each paddock can have a number of events associated with it and these can be classed into the following different types of events: Fertiliser/Paddock tests, Renewal/Forage Crop or Weed/Pest Control.

The **Stock** module organises all stock information. Only herd details rather than individual animal records are recorded. Various stock classes can be created (such as mixed age cows or heifers), and attributes about each class, such as birth year, breed and stock units, can be recorded. Within each stock class, a number of different mobs can be defined. Events relating to each stock class or mob within a stock class can be recorded, and these can be broken down into more specific event groups. "Production/Liveweight" events include records of milk production, condition scores and liveweights for a particular stock class. "Reproduction" events include calvings, matings and pregnancy test results. Vaccination records and incidents of disease are recorded as "Health" events. A running tally of stock numbers is recorded within the stock reconciliation, which is linked to stock events such as deaths and sales.

Livestock performance targets are recorded for each stock class. These include targets for liveweight and condition score, stock numbers, production levels, reproductive performance levels, and the amount of supplements to be fed to each stock class. The options available are dependant on the stock class. Various reports relating to each stock class and their targets can be created, such as target intake graphs and target liveweight gain graphs. These targets are linked to the feed budget.

The **Feed** module manages information about the pasture cover levels and available supplements on the farm. Details of pasture cover levels at different dates are recorded for each paddock. The type of data entered is dependant on the instrument used to measure pasture cover, and the associated pasture cover equation. Target pasture cover data can be entered, and then compared with actual pasture cover levels. Various reports and graphs can be produced relating to pasture cover. Maps can also be produced showing the differing pasture cover levels across the farm. Grazing records can be entered detailing the location of all stock on the farm. This information is used to produce various reports summarising the grazing records of particular stock, or of particular paddocks, including a map showing the location of the stock. All transactions which occur relating to supplements (such as buying, selling, feeding out) can be

recorded. This enables a record of the quantities of supplements on hand to be maintained.

GROW is a simulation model, accessed through the feed module, which is used to calculate pasture growth rates from weather, paddock and pasture type information. The location of the farm is entered along with slope, aspect and soil and pasture types. An appropriate weather station can be chosen, which has associated monthly rainfall and temperature defaults for the region. These can be adjusted for a specific farm or actual weather data from within the farm/weather module can be used. The GROW model calculates potential and actual pasture growth rates for the specified site conditions and weather data. The potential pasture cover is calculated based on optimum management of the pasture, and actual pasture cover predictions take into account reductions in pasture cover due to management. The calculated pasture growth rates can be saved for use in a feed budget.

Feed budgets can be constructed for the farm over a defined period. The user specifies values for each of the variables involved in the feed budget and feed supply and feed demand is calculated by the system. The expected average pasture cover levels through time are calculated for the specified plan. Feed supply is calculated using pasture growth rates (which can be estimated using the GROW model), supplementary feeding events, and paddock events such as the addition of nitrogen or the making of hay. Feed demand is calculated using stock details (whether the targets are to be measured in liveweight, liveweight gain or condition score), stock numbers (a separate tool to stock reconciliation, as these are target rather than actual numbers), production and reproduction details. Reports can be generated to view intakes required at the defined stocking rate given the production and reproduction targets.

When running a feed budget there is an option to change stocking rate for the existing feed supply, this is entered as a percentage, and changes the numbers of each stock class by the given percentage. There is also the option of entering initial pasture cover for the farm which defaults to the most recent actual pasture cover entered.

The results of the feed budget can be summarised in a number of different reports, such as a feed budget summary (Figure 4.1), a feed supply and feed demand graph and pasture cover graphs. The feed budget can be saved and compared to other saved feed budgets.

The **Mapping** facility can be accessed from other modules and enables various types of data to be viewed graphically such as pasture/crop types or stock locations across the

farm. The map is digitised from an aerial photograph. Overlay maps can be created and may include information such as water supply or temporary fencing. Paddock areas can be calculated automatically by counting the number of pixel's within areas.

4.4.2 IS FARMTRACKER A DECISION SUPPORT SYSTEM?

In Chapter 2, a list of capabilities which a decision support system is considered to possess was compiled from various authors. As the FarmTracker system is to be used to help test the proposed framework, it is necessary to ensure that it is in fact a decision support system. FarmTracker is therefore discussed in relation to each capability of a DSS.

- **DSS are interactive, computer based information systems**

The user has the ability to interact easily with the system. For example, it provides a means of performing "what if" feed budgets, with the user varying the farm situation.

- **DSS specifically focus on features which make them easy to use by non computer people**

The FarmTracker system is divided into logical modules with which the farmer is familiar, such as stock, feed, and paddocks. The mapping function enables the farmers to easily see, for example, pasture cover patterns across the farm, and there are numerous graphs and reports which summarise the farm situation.

- **DSS tend to be aimed at the less well structured, under specified problems that upper level managers typically face**

FarmTracker organises data for farmers to make decisions regarding future actions. These type of decisions require the intuition, preferences, judgement and experience of the farmer. The decisions involve considering numerous criteria, and must be made within a limited time frame. Such factors make the problems semistructured according to the criteria of Klein and Methlie (1995) outlined in Section 2.2.

- **DSS support but do not replace upper level managers in decision making**

The system allows the storage and manipulation of a wide range of data related to farm decision making. The final decisions, however, are left to the user of the system. The system does not attempt to identify or suggest solutions to problem situations. For example, in developing a feed budget, the farmer is required to decide factors such as stock numbers, feeding levels, timing of events and the level of feed supplements used. The farmer must also decide whether the resultant plan is feasible. Once the plan has

been implemented, the farmer must evaluate its success; the system does not recognise problems or suggest solutions.

- **DSS attempt to combine the use of models or analytic techniques with traditional data access and retrieval functions**

FarmTracker allows data to be stored, retrieved and manipulated easily by the farmer. This data is integrated with the GROW simulation model. The feed budgeting facility is an analytic tool which brings together the GROW model, stock intake models and specified targets.

- **DSS rely on simulation in cases where an analytic optimising model cannot be solved**

Pasture growth rates cannot be predicted by any completely certain means, and thus are predicted using the simulation model GROW.

- **DSS use statistical analysis to collect data and to predict trends**

Statistical summaries are used by the system to calculate average pasture covers, and average intakes. The models used within FarmTracker, such as the GROW model are also based on statistical analysis.

- **DSS emphasise flexibility and adaptability to accommodate changes in the environment and the decision making approach of the user.**

There is an extensive choice of reports, graphs and management options from which the user can choose. Users are also able to define their own graphs. The system allows many types of farm enterprises to be recorded, such as dairy or sheep, as well as the ability to record data relating to multiple enterprises on a single farm.

4.5 CONCLUSION

Agricultural DSSs which aid farmers in organising their data are widely used throughout New Zealand. However, they not attempt to interpret the information or suggest actions the farmer should take. Various authors within the agricultural community have discussed the potential for combining DSS and KBS techniques to help farmers make decisions. It seems an obvious step to attempt to improve existing DSSs by integrating them with KBS techniques. Thus the agricultural domain, and an associated existing DSS, is seen as a suitable candidate to aid in evaluating the proposed framework for the development of an intelligent component to be integrated with an existing DSS.

FarmTracker is the DSS chosen for examining the development framework. Its capabilities have been outlined and discussed in relation to a list of DSS capabilities. It was found that FarmTracker exhibited all of the qualities of a DSS, and thus is a suitable system to be used within this study.

5.1 INTRODUCTION

A framework for the development of an intelligent component to be integrated with an existing decision support system was proposed in Chapter 3. To assess its feasibility, the framework was applied to a problem related to the farm management decision support system, FarmTracker, outlined in Chapter 4. The framework was put to use by the researcher who took the role of knowledge engineer. This chapter describes the application of the framework; each step is outlined, and examples are included where appropriate. For further examples related to the case study, the reader is referred to Appendix A and B of this report. The framework has been refined in light of experiences gained during its use, and a final framework is presented in Section 5.5.

5.2 EXISTING DECISION SUPPORT SYSTEM ANALYSIS

The first stage in the proposed framework is the analysis of the existing decision support system. Interviews with the sponsor took place to enable the knowledge engineer to become familiar with the problem domain and FarmTracker. It was suggested by the sponsor that the most efficient way of learning the systems capabilities was to collect a full data set and attempt to manipulate and use the data using FarmTracker. This proved to be a valuable means of understanding the type of data which needed to be collected, and gave the knowledge engineer an opportunity to use the system in a real situation. System documentation and domain specific literature were also used to develop an understanding of FarmTracker and dairy farm management. Much of the experience and information gained could be considered to be the first stage in knowledge elicitation, giving the knowledge engineer an understanding of the problem domain.

An initial functional model was developed using data from interviews, experience from using the system and system documentation. This model proved to be a helpful means of directing further interviews as it was adjusted and improved. This process helped the knowledge engineer gain a good understanding of the functionality of the system. Initially, a levelled data flow diagram (DFD) was developed, with high level processes being broken down into more detailed processes at lower levels. However, this proved to be confusing as so much interaction between the high level processes occurred. The problem could have been that the system had not been developed in this structured

manner. It was found that an unlevelled diagram (Appendix A1) was a more successful communication tool, which was more easily understood by the two parties. Gane and Sarson (1979) suggest such an approach, calling it the expansion approach, arguing that users get lost in levelled DFDs, whereas expansion improves readability (Whitten, Bentley & Barlow, 1994).

A data model was developed for the system using an entity relationship diagram (Appendix A2). However, this did not model how the system was dealing with the data, rather it represented the data modellers conceptualisation of the information. A more useful method would have been to represent the actual data files to clarify what data was available for use by the added intelligent component. This would have been consistent with the functional model, and provided a more helpful reference to the existing structure of the data. This was not possible, however, due to confidentiality issues. The sponsor was reluctant to reveal the actual file structures used by the DSS.

From here, a technical analysis took place, in the form of semistructured interviews. As the actual file structures used by the system were unknown, it was necessary to create dummy ASCII files (Appendix A3). These were considered by the sponsor to be similar to the actual files used and produced by FarmTracker.

The tool available for development of the prototype intelligent component was KAPPA-PC, a Windows based knowledge based systems shell, which supports rule based reasoning and object oriented programming techniques.

The model to be utilised by the intelligent component was the GROW model, which is a pasture growth rate prediction model. During the technical analysis stage it was necessary to obtain structures for the input and output files to this model. Experimentation also had to take place to see how the model could be run from the KAPPA-PC/Windows environment as GROW is a DOS based model.

The feed budgeting functions were another of the existing systems features which could have been utilised by the intelligent component. However, these functions are embedded within FarmTracker, rather than being a separate module as GROW is. This made the use of the feed budgeting functions technically difficult, and beyond the scope of the case study. This highlights one of the shortfalls of developing an IDSS using an existing DSS; the structure of the DSS may limit the efficiency and functionality of the resulting system. A simplified feed budget was included within the intelligent component to temporarily overcome this problem.

5.3 KNOWLEDGE ACQUISITION FOR THE INTELLIGENT COMPONENT

The first step in the knowledge acquisition phase involved the definition of aims of the intelligent component. The sponsor envisaged a system which would help farmers interpret the information presented to them by the FarmTracker system. This was tentatively considered to consist of the following tasks:

- Identify shortfalls in production
- Determine reasons for the shortfalls
- Investigate position in two weeks time
- Investigate possible courses of action given the state of the farm
- Suggest feasible options

The sponsor understood this was a research case study and so could not expect a fully functional system suitable for use by the decision makers themselves at this stage. His aim was to see if such an addition to his system was feasible, and to see how such an addition could be undertaken. A system which successfully performed the above tasks, if only in a very primitive manner, would prove that with further development, a system to aid farmers is achievable.

The involvement of a group of users was not within the scope of the study in both the sponsors and the authors view. Addressing user issues such as interface design was not within the objectives of the study. These issues have not been overlooked, rather they would be investigated following this study into the feasibility of developing an intelligent component for integration with an existing DSS.

The knowledge elicitation process took the form of interviews; much of the familiarisation with the domain had already been achieved from earlier interviews when analysing the existing system. The domain knowledge was modelled using conceptual graphs to precisely define concepts and represent rules, and influence diagrams which represented the decision processes of the expert (Appendix B1). Interview data provided the starting point for both conceptual analysis and McGovern's text analysis (Sowa, 1992; McGovern *et al.*, 1991). Initial models provided direction for further interviews; the models were modified and extended accordingly.

Three types of conceptual graphs were developed; type definitions, dataflow graphs and If-Then graphs. The type definitions provide a concise description of the concepts by differentiating it from its supertype (Figure 5.1). Each of the concepts are also

represented within the type lattice, and defined in the conceptual catalogue (Appendix A1).

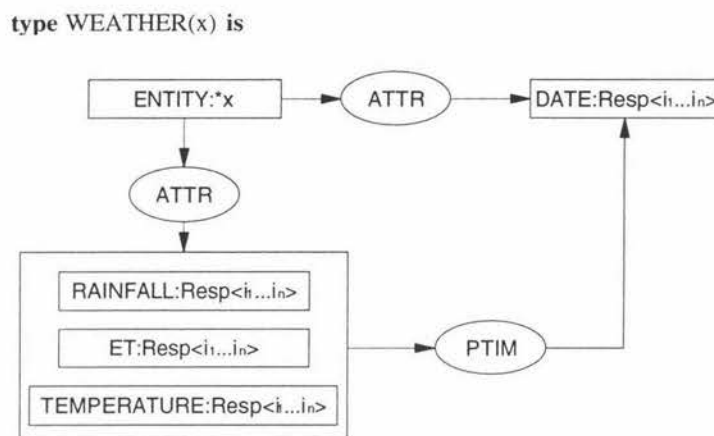


Figure 5.1 - Example of a type definition which reads - "The type weather has attributes which include a set of rainfall, evapotranspiration and temperature data which correspond respectively to dates in a set which is also an attribute of weather"

Sowa's dataflow graphs were used to clarify the mathematical relationships which exist between the various concepts (Figure 5.2).

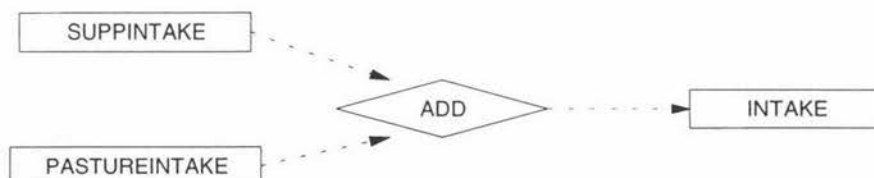


Figure 5.2 - Example of a dataflow graph which shows that the addition of supplement and pasture intake gives total intake.

If-Then graphs were used to represent the rule type relationships which existed (Figure 5.3). These could later be converted directly into rules within the knowledge based systems shell when developing the prototype.

The first two types of graphs did not appear to represent exclusively information for the intelligent component of the IDSS, as many of the relationships already exist within the DSS. For example, due to technical restrictions in reusing the feed budgeting module from within the intelligent component, a simplified feed budgeting function had to be used to duplicate some functions. Thus relationships which exist in the DSS had to also be represented in the new component, for example, Figure 5.2.

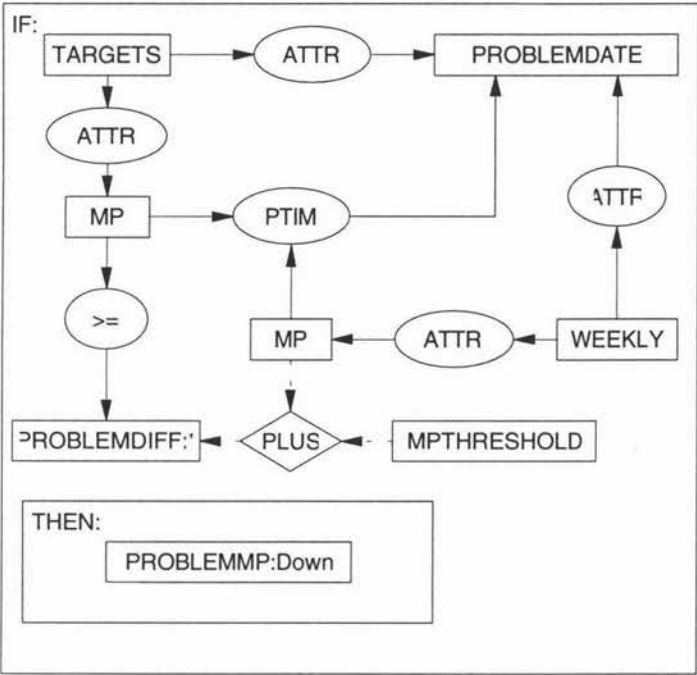


Figure 5.3 - Example of an If-Then graph which reads - "If target milk production on the problem date is greater than or equal to the weekly milk production plus the stated threshold on that date, then milk production is considered to be down"

Influence diagrams were developed to represent the decision process of the expert which the system was attempting to emulate (Figure 5.4). These seemed to adequately represent important information, however their advantages over a conceptual graph representation is in question and warrants further investigation (Section 5.4).

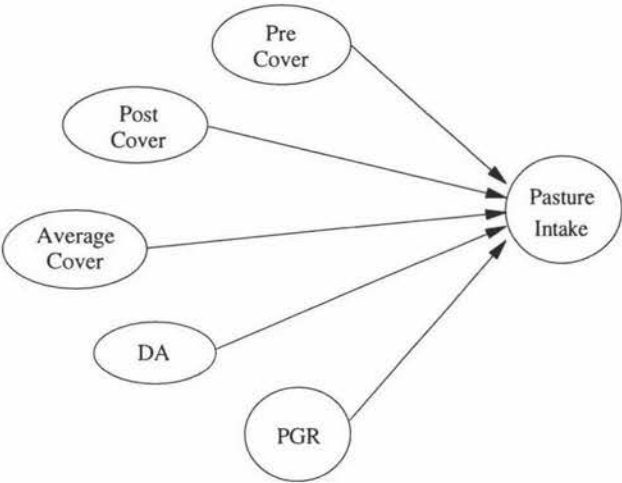


Figure 5.4 - A section of an influence diagram which shows that pasture intake is influenced by pre and post grazing pasture cover, average cover, daily allocation and pasture growth rates

The development of the task model (Figure 5.5) followed, giving an overall structure to the functions of the proposed system. This was developed and modified in discussion with the sponsor. The task model does not contain strict ordering, although generally the tasks are performed from left to right.

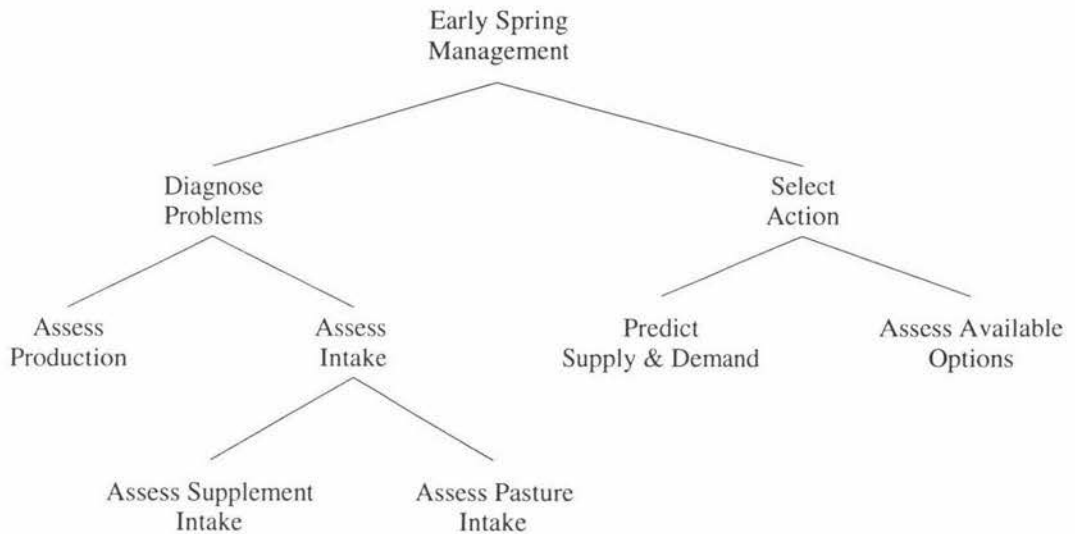
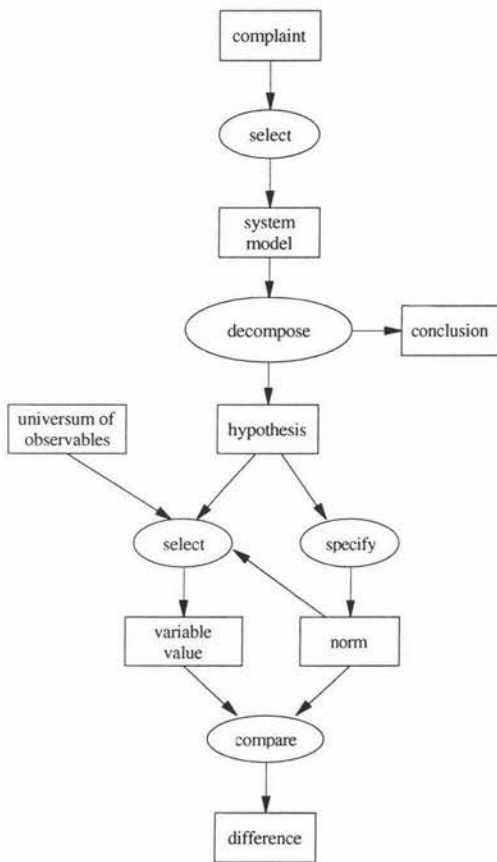


Figure 5.5 - The Task Model

The task environment is described as follows:

- The KBS is not a physical part of the farming system
- The KBS contains no sensors, but receives historical data from files produced by FarmTracker
- The KBS can manipulate data to output to an ASCII file which can be used as input to the GROW model which the KBS can execute
- The KBS users will be farmers who have a general understanding of the farming system

The task model provided the basis for choosing the interpretation models from the KADS library. The functionality of the tasks indicated which type of inference structures would be suitable. The task model could basically be divided into two sections. The first half suited the diagnosis model as the functionality involved diagnosing, and investigating reasons for, a production shortfall. The structure was modified to cope with the diagnosis of multiple faults as suggested by Hickman (1989), thus it describes exhaustive diagnosis (Figure 5.6).

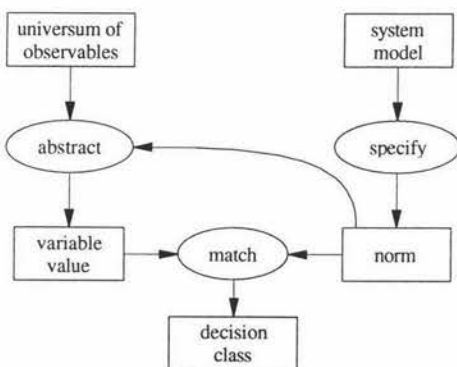


1. select system model on the basis of the complaint;
2. decompose system model into hypothesis;
3. for **each** attribute of **each** hypothesis:
 - a. select a variable and find its value;
 - b. specify a norm from the hypothesis comparable to the variable;
 - c. compare the norm to the variable value;

if the difference exceeds tolerance the current hypothesis becomes the new system model and recurse (from2), else disregard the current hypothesis and go on to the next until all hypothesis have been tested.

Figure 5.6 - Diagnosis inference structure for multiple fault diagnosis (Based on (Hickman, 1989)).

The second part of the task model involved evaluating options given the current and expected farm situation. This requires firstly a prediction of the pasture cover in the near future, and if this is unacceptable the elimination of unsuitable options given diagnosed problems and the current situation. The inference structure most suitable for this situation was found to be assessment (Figure 5.7). The prediction is a conventional function, simply calling the feed budget module, thus the assessment inference is only required for evaluating available options should there be an unfavourable prediction.



1. Specify a norm from the system model;
2. abstract from universum of observables taking the norm into account, to give the variable value in question
3. match the norm to the abstract case description and produce decision class

Figure 5.7 - Assessment inference structure (Hickman, 1989)

This is a backward chaining inference structure, as the norm is used to abstract the variable value. The various options are known; which ones are suitable depends upon the state of the observables.

Each of the inference structures had to be instantiated with domain specific information to form the task layer (Figure 5.8, 5.9, Appendix B3.1). This involved combining information represented in conceptual graphs and influence diagrams (the domain layer) with the inference structures (the inference layer). The task layer differs from the inference structure in that it is instantiated and it specifies order.

Diagnose Problem

To diagnose problems affecting intake do

select part of system model related to production complaint (eg intake relationships, intake targets)

decompose system model into relevant components (eg pasture intake, supplement intake)

Repeat for each component

specify norm for that component from the system model (eg target pasture intake)

select relevant observable based on norm (eg actual pasture intake)

compare the norm and observable to give the difference

 if the difference exceeds tolerance then

 if the component is atomic then add to the problem list

 else that component becomes the system model for further decomposition

 else disregard the current hypothesis and go on to next (eg supplement intake)

Figure 5.8 - Part of the task layer based on the diagnosis inference structure

Assess Available Options

To assess feasible options available to the farmer given the current farm situation do

Repeat for each available option in system model

specify norm from system model (eg specify tolerance for Olsen P levels from system model)

abstract from actual values the value equivalent to the specified norm (eg get actual Olsen P levels)

match actual and specified norm values to produce decision (eg match actual Olsen P and threshold Olsen P to give Olsen P decision)

Figure 5.9 - Part of the task layer based on the assessment inference structure

The conceptual graphs, influence diagrams, inference structures and task layer combined to form the initial model of expertise.

The task model in conjunction with the technical analysis, functional and data models provided the basis for developing the model of cooperation (Figure 5.10). This involves assigning each of the tasks in the task model to either the KBS, the DSS or the user, and describing the functionality of the tasks (Appendix B5.1). The technical analysis is necessary to understand the interactive ability of each of the components. The functional model is used to understand the current capabilities of the system, and the data model outlines what data is available for use by the intelligent component.

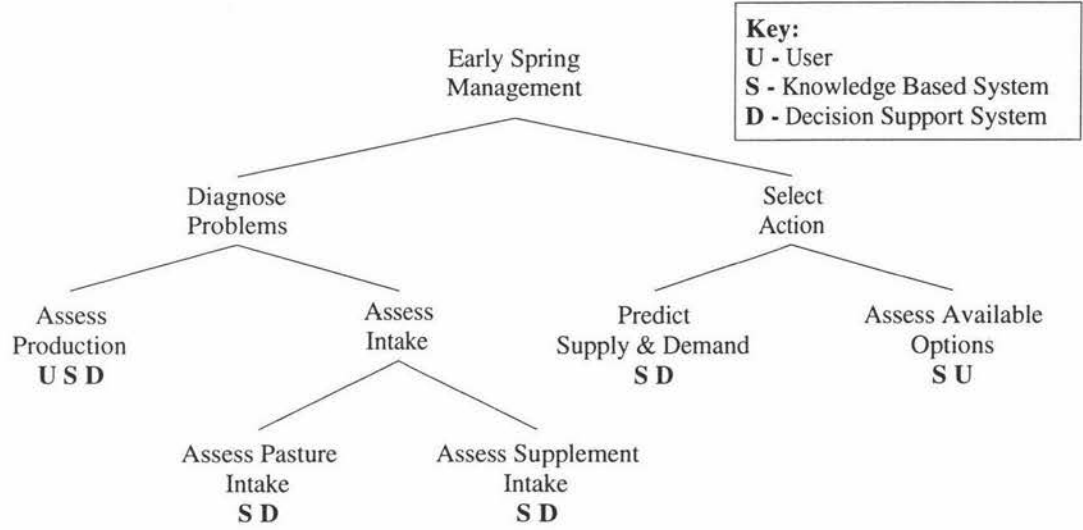


Figure 5.10 - The Model of Cooperation

The model of cooperation dictated the proposed architecture of the system, which was Turban's (1995) sharing of the decision making process (Section 2.5.4). This architecture was chosen to correspond with the nature of the problem; the proposed role of the intelligent component is to suggest options available to the farmer given the current farm situation. This requires access to current and historic farm and weather data, and use of the GROW model. Turban suggests a loose integration between the components which was envisaged as suitable given the difference in platforms between the existing DSS and the proposed intelligent component. Ideally the feed budgeting model could also have been utilised by the intelligent component (Figure 5.11), however technical constraints outlined above required the inclusion of a simulated feed budgeting model (Figure 5.12).

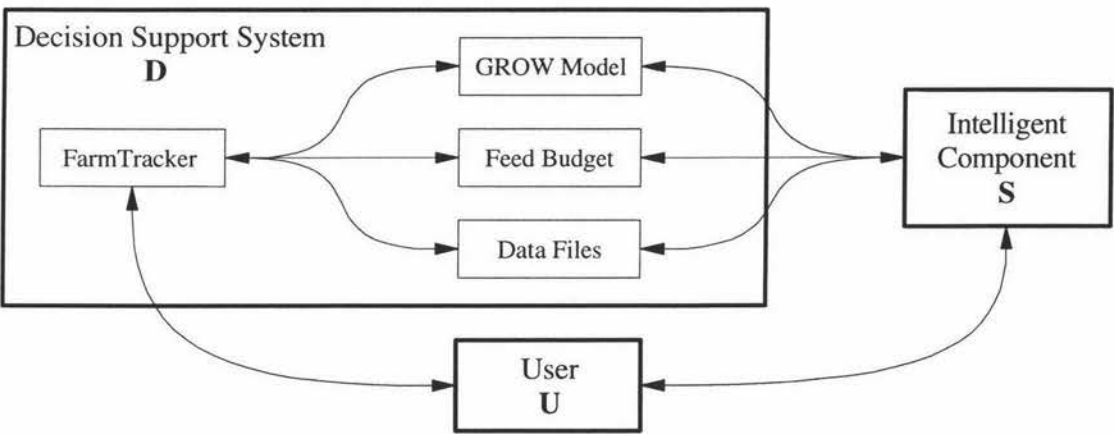


Figure 5.11 - Ideal proposed architecture of IDSS

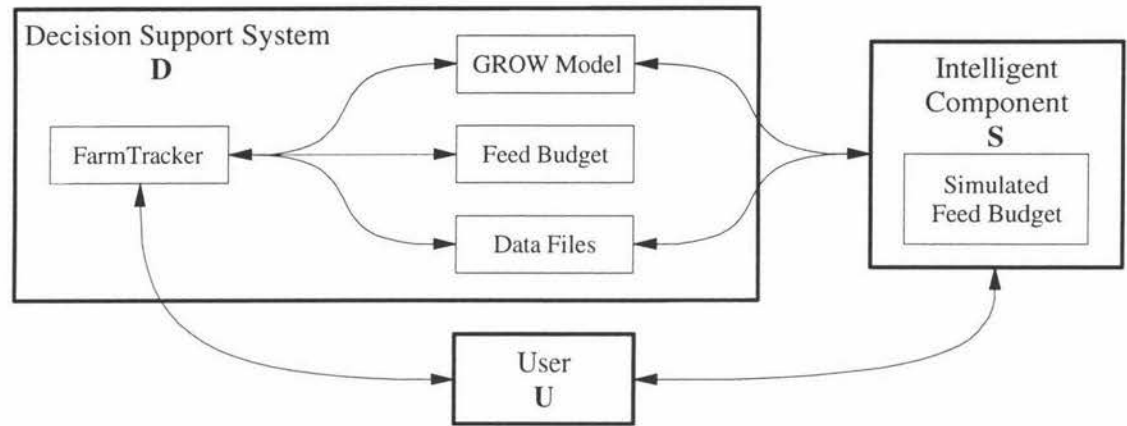


Figure 5.12 - Proposed architecture of IDSS

5.4 CONCEPTUAL GRAPHS VERSUS INFLUENCE DIAGRAMS

Influence diagrams and conceptual graphs were both proposed in the initial framework as a means of representing domain information. Both methods were used in the development of the prototype, and it was found that there was a potential overlap in the abilities of the two tools. Influence diagrams were used to represent decision processes, a task which could be achieved equally well by conceptual graphs.

The visual representation in influence diagrams does not allow the specification of the nature of the relationship between nodes, but simply shows which variables are of interest when determining the value of a particular node (O'Donnell & Watson, 1994). It is this shortfall which appears to distinguish influence diagrams from conceptual graphs. Conceptual graphs have the ability to explicitly represent relationships between concepts. Mathematical relationships can be expressed using dataflow graphs (a type of conceptual graph) as illustrated in figure 5.13a and b.

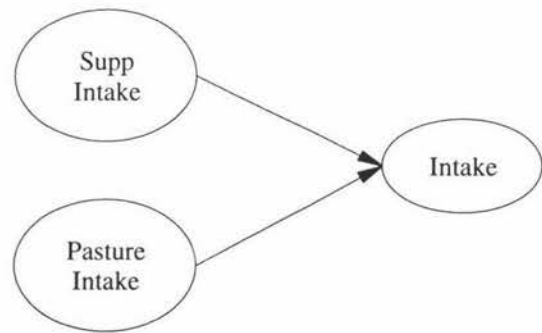


Figure 5.13a - An influence diagram which shows that pasture intake and supplement intake have some influence on the value of intake

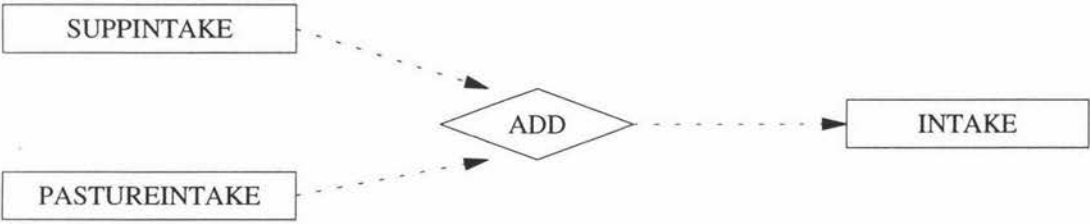


Figure 5.13b - A conceptual graph which reads "Supplement intake plus pasture intake gives the value of intake"

This example illustrates that although the influence diagram shows the relationship exists, the conceptual graph represents the relationship explicitly. In a situation where the relationships are not as explicit, such as those expected in a KBS, conceptual graphs can fulfil the same role as influence diagrams by showing that an influence relationship exists, for example, Figure 5.14 shows a conceptual graph equivalent to the influence diagram of Figure 5.4.

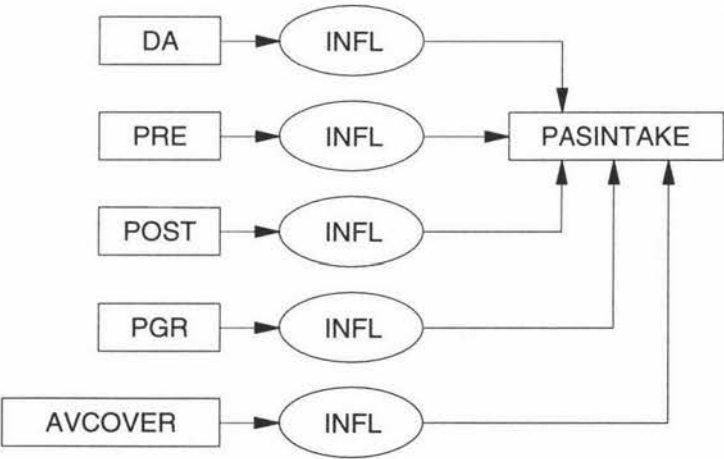


Figure 5.14 - A conceptual graph which shows that pasture intake is influenced by pre and post grazing pasture cover, average cover, daily allocation and pasture growth rates

The details of this relationship may require the representation of heuristics such as rules of thumb, which can be represented using another conceptual graph technique - an If-Then structure. Admittedly, the conceptual graph in Figure 5.14 contains more nodes than Figure 5.4. However, each of the nodes in the conceptual graph are tightly defined within the conceptual catalog, type definitions and type lattice. The use of one notation for modelling all domain knowledge provides a tidy and consistent representation.

Different types of influence arrows such as uncertain and preference arrows, could be represented within the conceptual graph notation by defining appropriate relations within the conceptual catalogue. Random variables could be represented by conceptual graphs as a proposition acting on a concept.

Another perceived disadvantage of influence diagrams is that their notation varies between authors, this variation is confusing and makes it difficult to produce consistent diagrams. This problem seems to be due to the lack of a seminal text to outline the notation. In comparison, Sowa's 1984 conceptual structures text provides a baseline to which all extensions to conceptual graphs can work from.

It appears that conceptual graphs can represent as much information as influence diagrams, with the advantage being the ability to be more explicit in representing the relationships. Conceptual graphs utilise only two types of nodes which can be explicitly defined to achieve the same meaning of the various nodes and symbols in influence diagrams. This also enables conceptual graphs to represent other information such as production rules. Therefore it seems sensible to remove influence diagrams from the proposed framework, as conceptual graphs can take their place. This simplifies the framework and standardises the resultant model.

5.5 CONCLUSIONS

The development framework has been described in relation to the case study. Overall, the framework was found to be successful. A slightly modified framework is presented in Figure 5.15 (compare with Figure 3.2) in light of experiences gained.

The modifications include the omission of influence diagrams for knowledge acquisition due to conceptual graphs having similar capabilities. The initial phase of the analysis of the existing system has been changed from "Interviews" to "Information Gathering" as it is felt that this is a better description of the phase. Gaining an understanding of the existing system involved more than just interviewing; system documentation and domain literature was studied and experience was gained by gathering data and using the system. There has also been an additional link included between the results of the information gathering phase and the elicitation phase. It was felt that the information gained was useful in elicitation, even if just implicitly through experiences gained by the knowledge engineer.

Some problems arose due to the inability to use the actual files produced by the DSS. Had the study progressed further than experimental stages, the actual file structures would have been revealed, or the existing system would have been modified to produce files equivalent to the dummy files.

The development process showed the difficulty of modelling and developing the components of a system so separately, as much of the information is required by both components. It was found that much of the intelligent component requirements were in fact conventional functions necessary to manipulate the data from the DSS into a suitable form for rule based reasoning. It is envisaged that this inefficiency could be avoided in a situation where all components are developed together and designed for integration, that is, when an IDSS is developed from scratch. It is inevitable, that when an intelligent component is developed separately, that a certain amount of repetition is necessary.

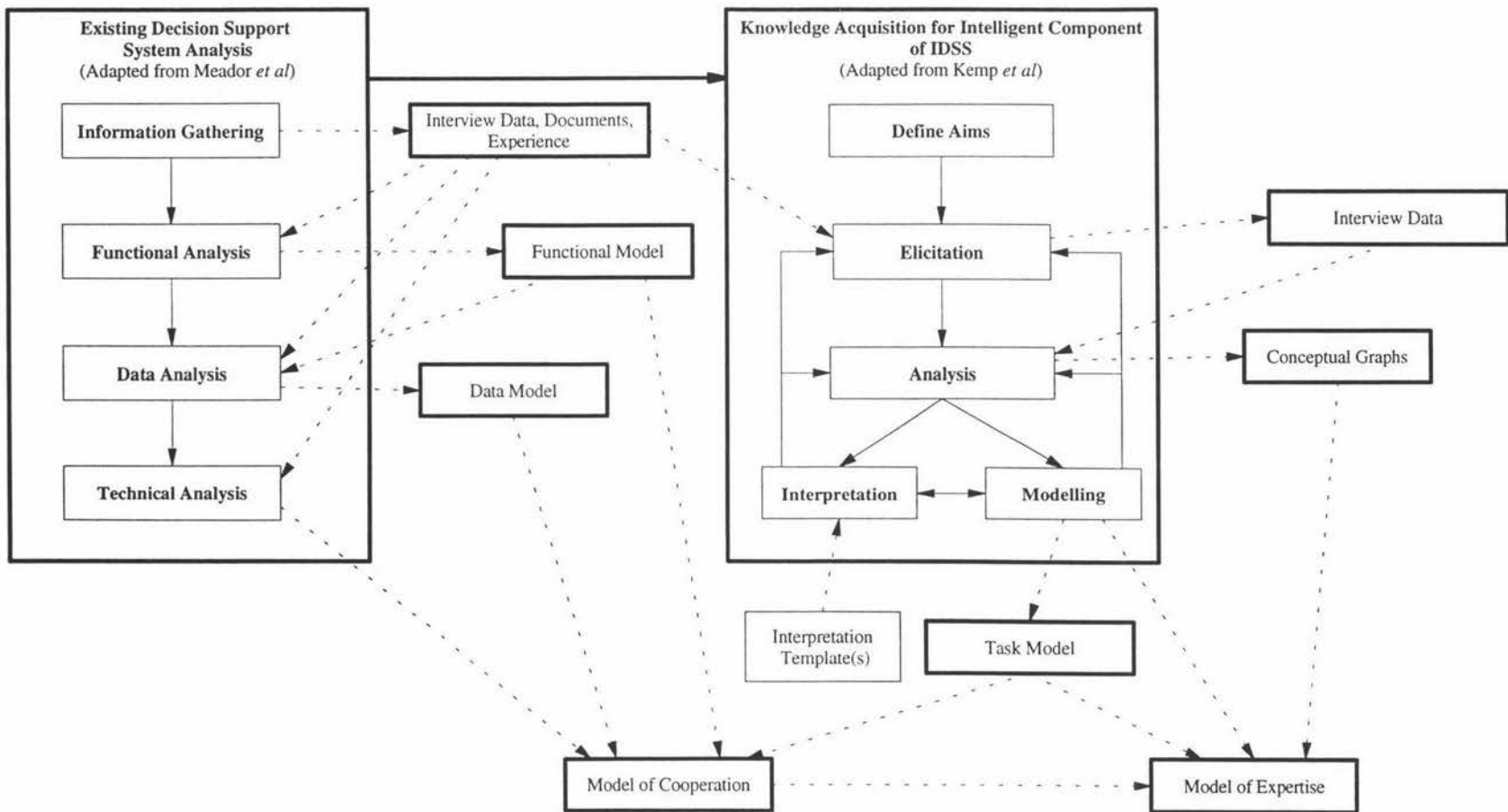


Figure 5.15 - The proposed development framework

6.1 INTRODUCTION

Having completed both the analysis of the existing DSS and knowledge acquisition for an intelligent component, a prototype could be developed. It was required within the prototyping process of Duffin (1988) that a clear definition of aims be specified before developing a prototype system. These could include, for example, testing functionality or completeness of models. In this study, however, the prototype was developed as a means of validating the framework.

Basili, Selby and Hutchens (1986) propose a framework for experimentation in software engineering, the definition phase of which has been utilised for definition of aims of the prototype. Basili *et al.*'s (1986) framework is made up of six definition elements; motivation, object, purpose, perspective, domain and scope, each of which is defined before an experiment begins. The motivation may be, for example, "to understand, assess or improve the effect of a certain technology" (Basili *et al.*, 1986). The object of the study is the primary entity being examined. The perspective is defined so as to allow correct interpretation of the purpose. For example, a study examining software quality would include correctness if from the perspective of a developer or reliability if from the perspective of the customer (Basili *et al.*, 1986). The domain is defined in terms of the programmers (such as experience, size of team) and the program (such as size, complexity). The defined domains determines the scope of the study. The definition of the prototype system, using Basili *et al.*'s definition criteria is shown in Table 6.1.

<i>Definition Element</i>	
Motivation	To validate a framework for the development of an intelligent component to be integrated with an existing DSS
Object	Development framework
Purpose	To evaluate the validity of the framework
Perspective	Developer
Domain:Programmer	As the framework is applied by a single developer
Domain:Program	A single intelligent module for an existing DSS
Scope	Single project

Table 6.1 - Definition of prototype aims using criteria of Basili *et al.* (1986)

Section 6.2 describes the process of converting the models developed in Chapter 5, into the prototype system. Section 6.3 outlines the functionality of the resultant system. For further details of the prototype, the reader is referred to Appendix C of this report.

6.2 PROTOTYPE DEVELOPMENT

KAPPA-PC, an object oriented knowledge based system shell, was used to develop the prototype intelligent component. Information is represented in KAPPA-PC by objects, which together form an object hierarchy supporting inheritance. A section of the object hierarchy for the prototype is shown in Figure 6.1.

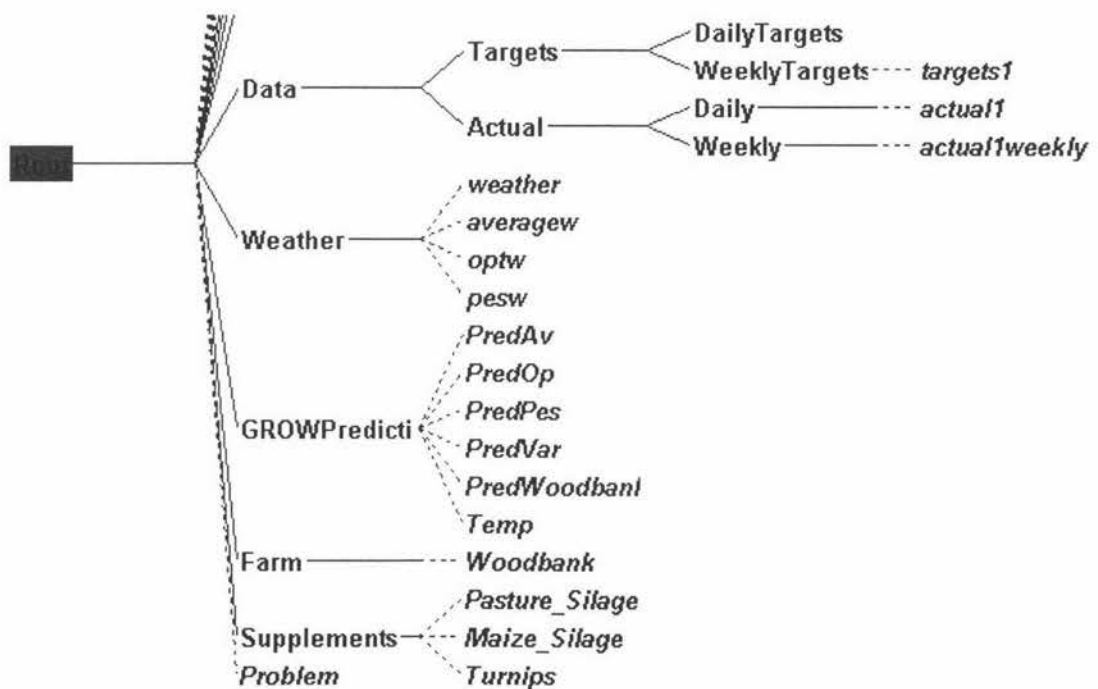


Figure 6.1 - Section of object hierarchy

It was found that the object hierarchy within the KAPPA-PC environment and the type lattice developed with conceptual graphs could be considered equivalent. This view is supported by Lukose (1993) who considers there to be a one-to-one relationship between the notion of class in the object-oriented formalism and the notion of type in the conceptual graph formalism. Both hierarchies support inheritance. This made for quick and easy implementation of the data structures required for the prototype. For example Figure 6.2 illustrates an equivalent branch in both the type lattice and the object hierarchy.

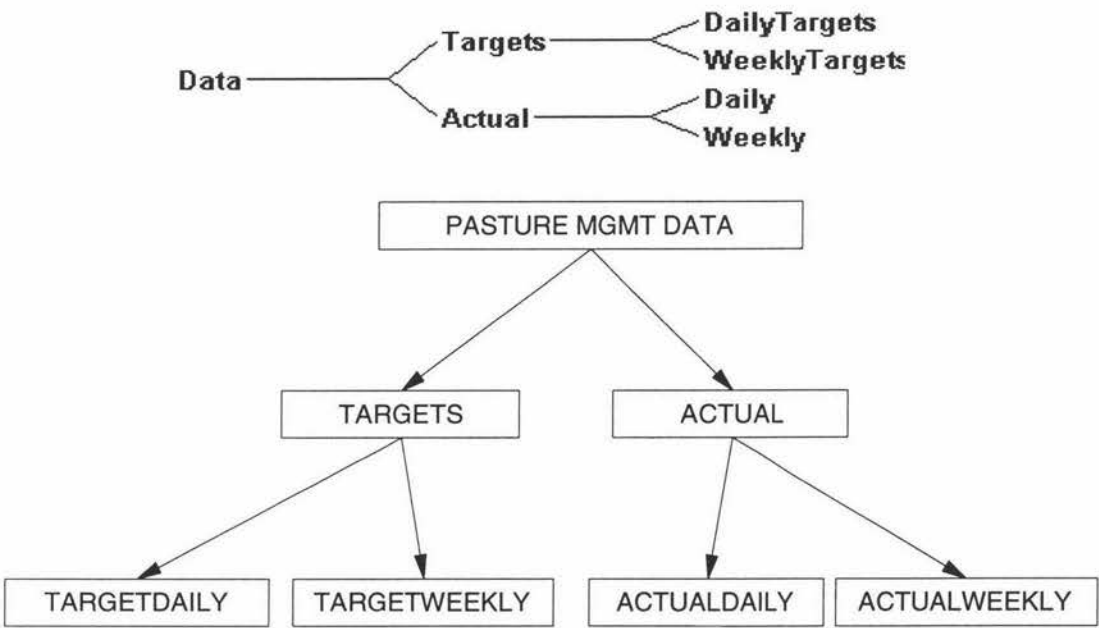


Figure 6.2 - Equivalent branch of object hierarchy and type lattice.

The intelligent component has the ability to access the following ASCII files, which were deemed to be equivalent to files produced by FarmTracker by the sponsor:

- A farm file containing details relating to the farm in question such as current Olsen P levels, soil type, total area
- A weather file containing weather details recorded on the farm over the past year
- Weather files based on historic data containing details for a variety of weather situations such as optimistic, average or pessimistic
- A file containing actual values for milk production, pre and post grazing pasture covers, average pasture covers, stock intakes, the number of cows grazed on what area and total area on that date
- A target file containing target values for each of the variables in the actual file, obtained from industry standards or entered by the farmer from within FarmTracker
- A file with details of supplements available

(For a more detailed description of the ASCII files, see Appendix A3).

The intelligent component reads the appropriate files and creates instances on the KAPPA-PC object hierarchy for the information. For example, the class weather has instances (*italicised*) for different weather scenarios, as illustrated by the section of the object hierarchy in Figure 6.3.

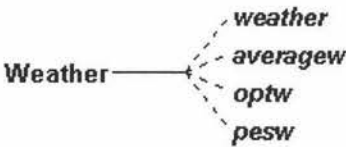


Figure 6.3 - Section of the object hierarchy

Attributes in the conceptual graph model translated to slots in the prototype. For example the attributes of the type weather, represented in the conceptual graph in Figure 5.1, are slots within the weather class, which are inherited by each of the instances in Figure 6.3. The KAPPA-PC class editor for the class Weather is shown in Figure 6.4. The slots in this case are lists, which correspond to the respective sets in the conceptual graph.

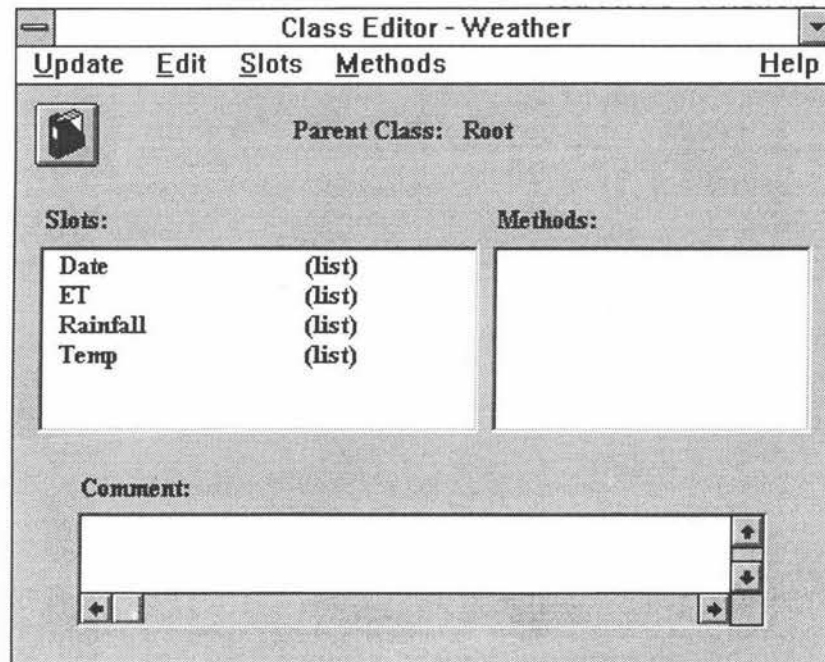


Figure 6.4 - Editor for the class Weather

Once read in, the data can be manipulated into suitable forms. For example, daily milk production levels were averaged to give a weekly value so that they could be compared to a weekly milk production target (Appendix C3.5). Conventional data manipulations were represented using dataflow conceptual graphs. These could be implemented in the prototype as either functions, or methods within an object which are triggered when the data is required. Generally functions were used in the prototype. In hindsight, however, it would appear to be preferable to use methods in keeping with KAPPA-PC's object oriented approach. Thus the data manipulation would be nested within the objects and kept separate from the reasoning processes. These methods could then be considered an intermediate component. In an ideal situation, with true integration between FarmTracker and the intelligent component, the data read by the intelligent component would already be in a suitable form, and thus the methods would not be necessary.

Many of the functions were simplified, following the principles of Visual Interactive Modelling (VIM) (Angehrn & Luthi, 1990) (Section 2.5.5). The sponsor was comfortable with this approach as he was keen to gain an overview of the capabilities of

the intelligent component before the domain details were implemented. KAPPA-PC is a suitable platform for such an approach encouraging modular programming, and supporting a very easy to use and manipulate graphical user interface. This enabled the prototype to be developed within the time frame of this study and for feedback from the expert to be gained.

The If-Then conceptual graphs could be directly implemented as rules in the prototype. For example, Figure 6.5 shows the rule represented by the conceptual graph in Figure 5.3, instantiated, and implemented with KAPPA-PC's high level descriptive language KAL.

```

MakeRule( TSDown,
  ( GetNthElem( Global:actualfileweekly:PastOnlyTS,
    GetElemPos( Global:targetfileweekly:Date, Problem:Date ) )
    < ( GetNthElem( Global:targetfileweekly:TS,
      GetElemPos( Global:targetfileweekly:Date, Problem:Date ) )
        - 0.05 ) ) And ( Problem:Units #= TotalSolids ),
  {
    PostMessage( "Total milk solids is down on target" );
    Problem:MP = Down;
  } );

```

Figure 6.5 - KAL implementation of a rule

The task model outlined the high level tasks which the intelligent component was to contain. This aided the organisation of rules into two groups, and directed the overall programming effort. The first group of rules included those for diagnosis, which were triggered by a forward chaining mechanism (Appendix C3.1). The second group of rules were the assessment rules, which were executed using backward chaining (Appendix C3.2). The task layer defined the control structure which orders and instantiates each grouping.

The interface which was developed attempted to emulate that of the existing DSS. This was clumsy due to one being a DOS application and the other a Windows based system. Ideally the IDSS should appear to the user as if it were a single system rather than separate components. This would require at least some modification to the existing system such as a means of executing the intelligent component. This, however, was beyond the scope of the study and the aims of the prototype.

6.3 PROTOTYPE FUNCTIONALITY

Upon request by the user the intelligent component firstly reads in information for a particular farm. Some of the information can be viewed at this point in graphical form, for example milk production information.

The process of diagnosis can then be initiated by the user. The user chooses the week to be evaluated and the unit of milk production to be used (total solids or milk fat). Forward chaining is initiated by asserting the variable containing milk production units, using a breadth first mechanism on the set of diagnosis rules. Breadth first forward chaining was used as this is an exhaustive strategy which corresponds to the multiple fault diagnosis in the inference structure. A class "Problem" containing slots for different variables, is used to record the results of the diagnosis and continue the forward chaining. This acts somewhat like a blackboard, recording details of the current problem (Turban, 1995). This class could be modified to include scheduling, should it be required at a later date.

Figure 6.6 shows part of the inference browser for an example of the forward chaining process. The rule names are italicised, and the variables which assert the rules are in normal print. It is helpful to highlight at this point that cow production levels are measured using either milk fat content or total solids content of the milk. The cows intake has a direct influence on milk production levels.

In the example, the variable containing the chosen units is asserted, thus triggering rules with this in their premise. Each of the milk production rules are tested (TSDown, MFDown and MPGood²), and the MFDown rule is found to be true. MFDown's consequence assigns a value to the variable Problem:MP, thus asserting it. This then triggers the next level of rules which contain the MP variable. In this example, two rules were found to be true, one being PasIntakeDown³ which compares actual and target pasture intakes. From here, variables which effect the intake are investigated. In some cases pasture growth rates (PGR's) are found to be below target, triggering the execution of the GROW model in order for further comparisons to be made using current weather conditions (Appendix C3.3). This provides the system with three scenarios for comparison and investigation; the calculated actual PGR, the target PGR, and the PGR which would have been expected given the recorded weather conditions.

²MF = Milk Fat, TS = Total Solids, MP = Milk Production

³PasIntake = Pasture Intake

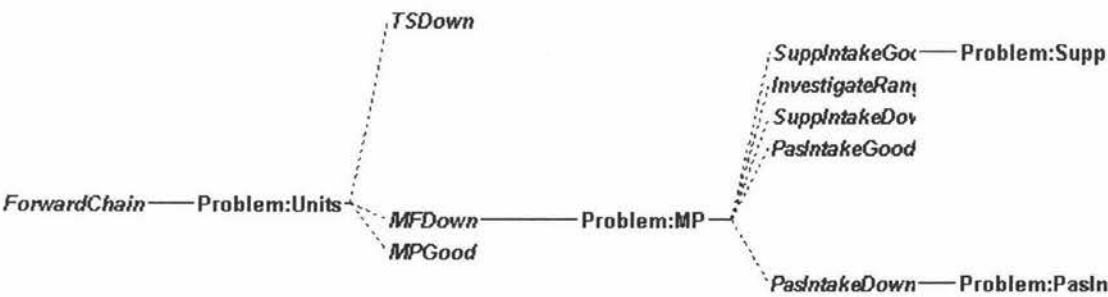


Figure 6.6 - Section of Inference Browser for an example situation

The GROW model requires an ASCII file to be created containing weather information along with details of the farm which is obtained from the instance "farm" (originally from the farm file). Once this is created the model can be executed which requires a DOS shell to be opened. The GROW model can only simulate one situation at a time, and so must be executed a number of times if various weather situations are to be investigated. The GROW model creates an output ASCII file which can then be input to the intelligent component (Appendix C3.3).

Once the basic diagnosis is complete, a simplified feed budget (See Figure 5.12 & Appendix C3.4) calculates various projections on pasture cover given the current situation and various weather scenarios. This also utilises the GROW model to predict pasture growth rates. A graph displaying the various projections is created (Figure 6.7).

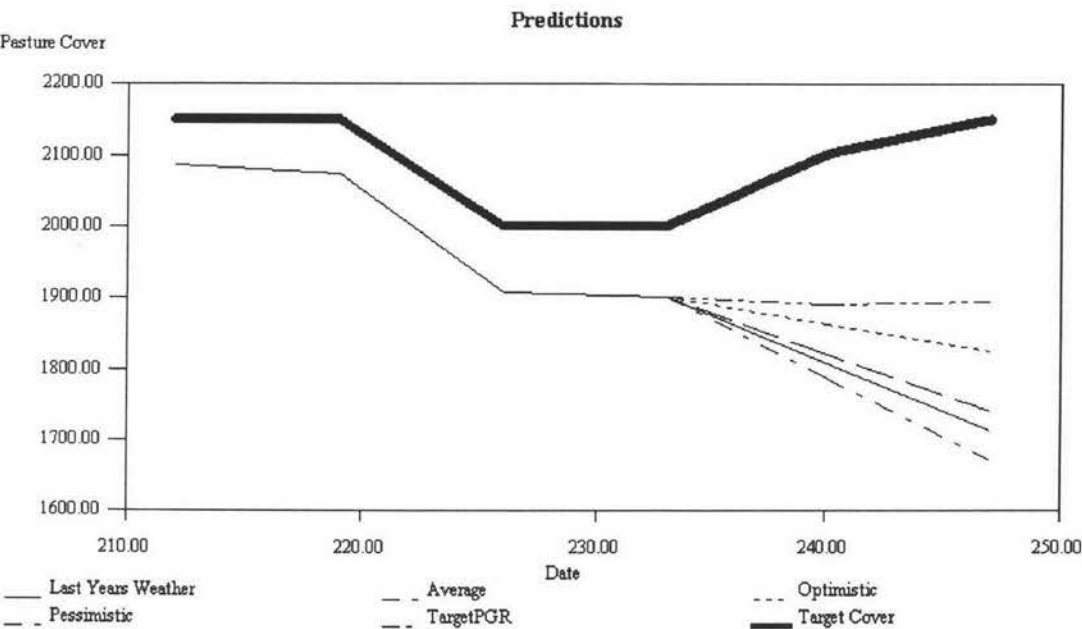


Figure 6.7 - Graph displaying varying projections

The user now has the opportunity to evaluate options the system has selected as appropriate. This selection process utilises backward chaining (the assessment inference structure, Figure 5.7), with the goal of investigating every available option (rather than stopping when a single solution has been found). At present at most four options are available to the user. The structure of the rule base, however, allows further options to be included, by increasing the number of rules within the assessment rule grouping.

The user is then able to manipulate these options and perform a "what-if" analysis by running a feed budget given a new scenario. A graph is displayed showing the predicted pasture cover given the chosen changes to the farming system, along with the target pasture cover levels (Figure 6.8). This last step can be repeated any number of times, experimenting with various combinations of the feasible options.

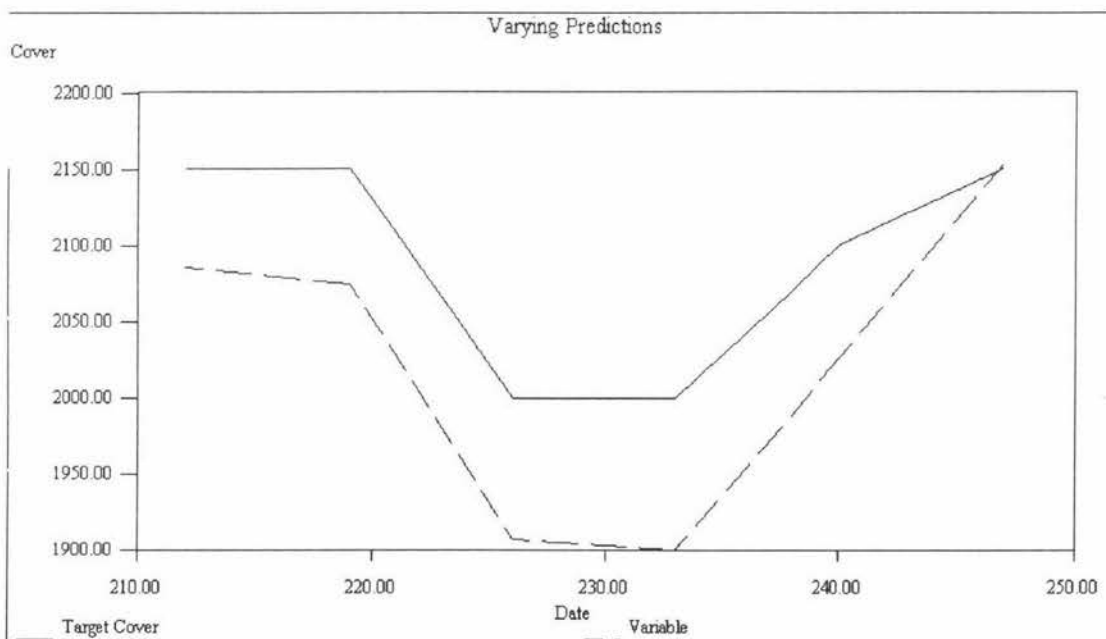


Figure 6.8 - Graph displaying projection given changes in the system

6.4 CONCLUSION

KAPPA-PC provided a flexible environment for the development of the prototype intelligent component. It allowed various decision support capabilities to be implemented, such as the inclusion of "what-if" analysis facilities.

The object oriented data structures were well suited for implementing a conceptual graph model due to its hierarchical nature. The variety of rule based reasoning methods allowed a suitable one to be chosen for each of the inference structures. The graphical user interface facility provided by KAPPA-PC enabled an easy to use interface to be developed, and data could be displayed clearly to the user.

The prototype system provides empirical evidence to aid in evaluation of the development framework. This is discussed in Chapter 7.

7.1 INTRODUCTION

This chapter evaluates the framework proposed for the development of an intelligent component to be integrated with an existing DSS (Figure 5.15). Its success has been tested using defined aims and evaluation criteria. The prototype IDSS provides empirical evidence useful for evaluation, this is discussed in Section 7.2. In Section 7.3 criteria from the literature on the validation of methodologies are compiled. These criteria have been used to evaluate the framework based on experiences gained during application of the framework and development of the prototype IDSS.

7.2 PROTOTYPE EVALUATION

Gillies (1991) comments that "a development strategy stands and falls by the quality of the resulting solutions, and the effort required to produce them, and not by any intrinsic merit". Thus the prototype IDSS (FarmTracker integrated with the intelligent component) was evaluated in order to test the development framework; this was achieved by investigating two questions. Firstly, was the intelligent component capable of fulfilling the defined aims in the view of the sponsor, and secondly, is the prototype IDSS a true IDSS?

7.2.1 DOES THE INTELLIGENT COMPONENT MEET THE DEFINED AIMS?

At the beginning of the knowledge acquisition process the aims of the intelligent component were defined. These were developed in light of the fact that this was an exploratory study, investigating the feasibility of linking an intelligent component with an existing DSS. The aims of the intelligent component were defined in Section 5.3 to be:

- Identify shortfalls in production
- Determine reasons for the shortfalls
- Investigate position in two weeks time
- Investigate possible courses of action given the state of the farm
- Suggest feasible options

These aims provide a basis for evaluating the success of the intelligent component. Each of the features were investigated in consultation with the sponsor, to see if they were included, were correct and were extendable.

The level of detail with which they were achieved was not viewed as important, but the capability of extending the intelligent component to provide more detail was. The extendability issue could not be judged by use alone, so the knowledge engineer's opinion was also taken into account. Extendability includes both that of the existing features, and the ability for additional features to be included in the intelligent component.

In the evaluation session the knowledge engineer showed the sponsor the use of the intelligent component given a variety of farm scenarios. Each of the five steps were investigated separately. For example, in testing the first aim, a scenario where production was above target was compared to situations where production was below target. This enabled the sponsor to see the capabilities and results of the intelligent component. Each of the scenarios are described in Table 7.1.

Prototype function	Scenario 1	Scenario 2	Scenario 3	Scenario 4
Identify shortfalls in production	Milk Fat above target	Milk Fat below target	Milk Fat below target	Total solids below target
Determine reasons for shortfall		Pre grazing pasture cover below target	Average pasture cover below target	PGR & DA below target
Predict pasture cover in two weeks		Predicted pasture cover above target	Predicted pasture cover below target	Predicted pasture cover below target
Investigate possible courses of action and suggest feasible options	Suggest no action	Suggest no action	Change SR Apply Fert	Change SR Apply N Feed Supps Apply Fert

Key: SR = Stocking Rate, N = Nitrogen, Supps = Supplementary Feed, Fert = Fertiliser, DA = Daily Allocation

Table 7.1 - Scenarios used to evaluate prototype

The results of the session are summarised in Table 7.2, with a brief description of the functionality which satisfies the aim. For a more detailed description of the prototype functionality, see Section 6.3.

Capability	Achieved?	Description
Identify shortfalls in production	✓	Production levels are compared to target and the user is informed if they are within the threshold
Determine reasons for the shortfalls	✓	If production is below target, diagnosis rules are triggered, flagging variables which are below target
Investigate position in two weeks time	✓	The pasture cover level in two weeks is predicted using current weather patterns and pasture covers
Investigate possible courses of action	✓	If predicted pasture cover is not acceptable to the system, then assessment rules determine feasible actions
Suggest feasible options	✓	Feasible courses of action are presented to the user, which they can then use in 'what-if' analysis

Table 7.2 - Summary of evaluation session

Although some of the results were simplistic, such as the feed budget prediction, this was not seen as a problem by the sponsor. He understood the reasons for using a basic feed budget model, and could see the advantages of integrating the feed budget component of FarmTracker with an intelligent component. Similarly, the rule sets were unrealistically small. For example, the intelligent component only dealt with four options for the farmer. The number of options could easily be increased with the addition of more rules. The intelligent component's inference structure allows its extension into a fuller system suitable for testing with potential users such as farmers.

The sponsor suggested further extensions to the functionality of the intelligent component. He felt that costing each of the scenarios would be a useful function, as well as conversion of fertiliser levels into physical quantities of fertiliser necessary to achieve those levels. These are seen as decision support capabilities rather than intelligent features, and could either be added to the intelligent component, in the form of functions, or be achieved within the FarmTracker system if true integration existed.

7.2.2 IS THE PROTOTYPE SYSTEM A TRUE IDSS?

In Section 2.5 a list of capabilities which an IDSS should possess was compiled. Examining this list in relation to the developed prototype IDSS provides the test to ascertain whether the prototype has the characteristics of an IDSS. If the prototype is considered an IDSS, this would provide further evidence of the success of the framework. The list of capabilities has been divided into three categories: external, system and internal perspectives.

External Perspective

- **An IDSS addresses semistructured problems**

Klein and Methlie (1995) outlined a number of features of semistructured problems which provide a basis for determining if the problem addressed by the prototype is in fact semistructured. These are outlined in section 2.2, and include the necessity for the decision to require the preferences, judgments, intuition and experience of the decision maker. These features are required by the prototype, for example, Olsen Phosphate threshold levels are determined by the expert decision maker based on rules of thumb and experience. These are then used in assessing whether applying fertiliser is a feasible course of action. When the system deals with a problem, it has to structure and search information for computation, consider the decision criteria and make the required trade off's between them. These features all constitute a semistructured problem.

- **An IDSS may deal with uncertain or incomplete information**

The dependence of the farming system on weather makes the prediction of future trends very uncertain. The prototype system deals with weather data and uses the GROW model to predict various patterns in pasture growth, as well as testing current situations to see if they can be attributed to weather.

- **An IDSS may deal with decisions which involve risks**

The decisions of the farmer have a very definite financial risk. The environment and livestock condition are an important factor, and causing adverse effects on them must be avoided.

- **An IDSS provides normative power to the decision maker**

The IDSS aids the farmer in developing a management plan given the current situation on the farm and heuristics represented in the intelligent component. This allows domain expertise to be utilised by the farmer; the experts decision process is considered normative behaviour, as it describes how a decision should be made.

System Perspective

- **An IDSS is dependent on multiple internal and external information/knowledge sources**

The prototype IDSS utilises an historic database containing details about prior states of the farm such as production levels, pasture cover and weather records. Stored information regarding the current state of the farm, and target levels are also utilised. This information is manipulated by the intelligent component until it is in a form suitable for heuristic analysis. The feed budgeting and GROW models from within FarmTracker are used by the intelligent component.

- **The IDSS utilises and manages the use of models**

The GROW model is used by the prototype IDSS as a means of predicting pasture growth rates. The intelligent component manipulates the data required for prediction into a suitable form, and calls the execution of the GROW model. The results of the simulation can then be read by the intelligent component for use in decision making. A feed budget model is also used to predict the outcome of a particular plan. This is able to interact with the GROW and animal intake models.

Internal Perspective

- **The IDSS is based on several disciplines (eg DSS, databases, KBS etc)**

The IDSS combines the existing DSS and its associated data files and models with rule based reasoning techniques. The intelligent component included a certain amount of conventional features, such as data manipulation and presentation. This

was necessary to prepare the data for use by the knowledge base, as well as to provide a useful user interface.

7.2.3 SUMMARY

As discussed by Gillies (1991), the quality of the results of a development strategy are a good indicator of its success. Thus this section has evaluated first the intelligent component and then the prototype IDSS developed using the proposed framework. The aims of the sponsor were discussed in relation to the intelligent component, and it was found that these were achieved. The desired characteristics of an IDSS have also been discussed and each of these characteristics have been met by the prototype in some way, enabling it to be considered an IDSS.

Having validated the success of the prototype, the framework can now be evaluated from the perspective of the knowledge engineer.

7.3 FRAMEWORK EVALUATION

The proposed framework needs to be evaluated to see whether it has the power to assist in the addition of an intelligent component to an existing DSS. Criteria for judging the framework had to be developed to provide a basis for drawing conclusions. The proposed framework is intended to be a step towards a complete methodology, but as yet does not contain adequate detail. The search for criteria for evaluation focused on that of methodologies, as it is a methodology to which the framework aspires.

Two evaluation approaches have been investigated; that of Jeffries (1994) and Blair *et al.* (1995a). Relevant aspects of each were seen as applicable for evaluation of a framework for the development of an intelligent component to be integrated with an existing DSS. A set of criteria for this study has been compiled drawing from both of the evaluation approaches.

Jeffries (1994) discussed criteria from relevant literature (Alenmang & Rothenfluh, 1992; Guida & Tasso, 1989; Rothenberg, 1989; Nosek & Roth, 1990; Teague & Pidgeon, 1985; Sloman, 1978) for evaluating knowledge acquisition methodologies. As these criteria are concerned with development techniques associated with knowledge based system concepts, most are seen as applicable for evaluating the framework.

Blair *et al.* (1995a) developed a rigorous approach for comparing IDSS design methodologies. Each phase in his proposed development methodology (Section 2.5.5)

has an associated set of criteria in question form. There are also criteria for software reuse and interface design outlined within the framework. The approach Blair *et al.* (1995a) uses appears to stem from limitations and benefits identified in existing IDSS methodologies (Section 2.5.5). It draws from literature relating to KBS, DSS and software engineering methodologies and is intended for the evaluation of complete methodologies.

This study is not proposing a complete methodology, thus only a selection of the criteria have been utilised. Much of the Blair *et al.* (1995a) work is specifically for Intelligent Decision Systems (IDS) rather than the more general Intelligent Decision Support Systems, thus these IDS criteria have been omitted. The sections involving the decision maker were also excluded because this study does not involve the decision maker. This was not within the aim of the project as defined in Chapter 1, the aim is to propose a means of adding a working intelligent component, rather than implementing a system immediately ready for use by the decision maker. This also makes the interface design phase of the Blair *et al.*'s (1995a) evaluation framework inappropriate.

Many of the criteria from both authors appear to be investigating similar aspects of methodologies. The Jeffries (1994) criteria are quite general, whereas those of Blair *et al.* (1995a) are very specific. In order to prevent repetition during evaluation which is to be in discussion form, similar criteria have been grouped together, and the wording of some of the criteria has been modified to be specific to this study. Table 7.3 summarises the selected criteria which will form the basis for discussion in Section 7.3.1.

7.3.1 DISCUSSION

The selected criteria are used as a basis for discussion of evaluation of the proposed framework. Many of these criteria require experience in using the methodology in order for them to be tested. Much of the discussion is related to experiences gained during the development process. Readers are referred to the framework summary in Figure 5.15.

1. A methodology should be focused, especially emphasising those tasks which are typical of IDSS analysis such as domain and problem analysis and knowledge modelling. (Jeffries, 1994)

The overall framework is very focused with the aim of creating a model of expertise and model of cooperation. Conventional analysis of the existing DSS provides the knowledge engineer with an understanding of the domain and the systems capabilities. The data and functional models developed provide a good grounding for the knowledge acquisition phase to begin.

1.	A methodology should be focused, especially emphasising those tasks which are typical of IDSS analysis such as domain and problem analysis and knowledge modelling.
2.	A methodology should include a procedure for eliciting knowledge from the domain expert A methodology should include a procedure for the inclusion of knowledge from knowledge sources of various types (domain experts, documentation, experimentation, observation and induction)?
3.	A methodology should be as definite as possible, there should be a clear purpose associated with the methodology, and the techniques should be explicitly specified and not left to the intuitions of individuals. A methodology should be practical, it should be easy to teach and apply and comprehensible by all parties involved in the knowledge acquisition process.
4.	A methodology should be open, it should support the use and integration of existing techniques and tools aimed at specific aspects or phases of the expert system development process (eg knowledge acquisition tools, project management tools, knowledge base verification and refinement techniques) and promote the development of new ones. A methodology should include a graphical model to represent the conceptual design
5.	A methodology should be explorative, it should allow both system specification and design to proceed incrementally, experimenting with alternative problem solving approaches.
6.	A methodology should be structured and modular, it should support as far as possible (hierarchical) work decomposition into elementary components, and enable generalisation through the structures hierarchical chaining. A methodology should include techniques to abstract and decompose the requirements A methodology should include methods to abstract and decompose the conceptual design of the IDSS
7.	A methodology's design language should be rich in the sense that it allows the expression of different kinds of knowledge by different language primitives
8.	A methodology should be rich in heuristic power, the concepts, assumptions, symbolisms and transformation procedures should be such as to make the detection of gaps and errors, the design of problem solving strategies, the recognition of relevant evidence, easily manageable.
9.	A methodology should include techniques to normalise the conceptual model
10.	A methodology should have the ability to be adapted to various applications, it should be general, flexible, maintainable and extendable. A methodology should allow the knowledge domain be expressed independently from its use
11.	A methodology should include a uniform approach to gather the inter-disciplinary requirements of the IDSS A methodology should include models to represent the inter-disciplinary requirements graphically A methodology should include languages for describing the inter-disciplinary requirements
12.	A methodology should include techniques for searching the organisation for potential existing software components A methodology should include techniques which help developers to understand what existing software components do A methodology should support the interconnection of existing and hand coded software components within the IDSS

Table 7.3 - Selected evaluation criteria adapted from (Blair et al., 1995a) and Jeffries (1994)

The "Analysis" phase of the framework (Figure 5.15) concentrates on domain analysis, working from interview data. The resultant conceptual graphs form the domain layer in the model of expertise. The creation of the task model involved problem analysis, to determine the tasks which were involved. This continued with the creation of the model of cooperation, the choice of interpretation templates, and their instantiation for the task layer. The entire process was directed by the aim of creating the resultant models; modelling is therefore an important aspect of the framework.

Well defined methods (such as KADS and conceptual analysis) were utilised, providing a sound basis for the framework. The conceptual graph notation ensures a consistent model is created, representing a wide range of information suitable for use within the domain layer of the model of expertise. The interpretation templates in the KADS library provided direction by suggesting various alternatives to the problem solving tasks. The model of expertise brought the information together into a well structured organisation.

2. A methodology should include a procedure for eliciting knowledge from the domain expert (Blair *et al.*, 1995a)

A methodology should include a procedure for the inclusion of knowledge from knowledge sources of various types (domain experts, documentation, experimentation, observation and induction)? (Blair *et al.*, 1995a)

The "Information Gathering" stage (Figure 5.15) in the analysis of the existing DSS involved collecting information from a number of sources, including interviews with the sponsor, system documentation and use of the system. The interviews with the sponsor enabled the knowledge engineer to gain an understanding of FarmTracker and the problem domain. System documentation provided an overview of the capabilities of the DSS. The collection of a complete data set for use with the system was helpful, giving the knowledge engineer both a deeper understanding of the domain, and the opportunity to experiment with the DSS.

The models created during the Functional and Data analysis phases, and the experience gained during these phases were a useful source of information for the knowledge acquisition process. Knowledge acquisition interviews with the sponsor provided additional data. The sponsor was involved in most of the knowledge acquisition phases by providing feedback as to the correctness of models, and expanding on them with further information.

3. **A methodology should be as definite as possible, there should be a clear purpose associated with the methodology, and the techniques should be explicitly specified and not left to the intuitions of individuals.** (Jeffries, 1994)

A methodology should be practical, it should be easy to teach and apply and comprehensible by all parties involved in the knowledge acquisition process. (Jeffries, 1994)

The purpose of the framework is to provide a guide for the analysis and design of an intelligent component to be integrated with an existing DSS. The framework includes methods which are generally well defined in their associated literature.

The data and functional models used (data flow diagrams and entity relationship diagrams) are each recognised standards in their discipline and have their notations defined by a number of authors (for example Yourdon, 1989). The use of unlevelled data flow diagrams made the functionality more comprehensible by the sponsor (Section 5.2). The newer techniques used (conceptual analysis and KADS) each had some problems associated with them due to the lack of complete examples.

Although the process of application of conceptual graphs (during the analysis phase of the knowledge acquisition) is fairly unstructured, the resultant models are very readable. The graphical notation makes for easy and clear reading. When presented with the models, the sponsor quickly grasped the concepts and was able to read the graphs with relative ease.

The apparent ease of reading conceptual graphs seems to be due to their direct mapping to and from natural language without loss of information. It seems that once the basic conceptual graph notation has been understood, it can be applied to any domain resulting in an easy to read representation.

The type lattice is a clear and reasonably self explanatory method of showing the hierarchy of types, which can be followed and checked by the sponsor. This ensures a mutual understanding of the meaning of the concepts has been reached between the sponsor and the knowledge engineer.

Although conceptual graphs are easy to read, their complexity is apparent when trying to learn or teach the notation. The author found that for a person with a basic background in knowledge representation, a detailed study of the literature was required before the main ideas had been grasped. This can be quite time consuming, especially if one wishes to master all the techniques of the notation, as it is quite extensive in its capabilities.

Parts of the model of expertise are not particularly easy to read and understand for a person who has little understanding of the methodology. The domain layer is fairly straight forward, represented by conceptual graphs. The inference structures however, take a fair amount of study before they are clearly understood. The choice of the structures, however, is a task for the knowledge engineer and their instantiation and ordering for the task layer clarifies their meaning for other parties to understand.

- 4. A methodology should be open, it should support the use and integration of existing techniques and tools aimed at specific aspects or phases of the expert system development process (eg knowledge acquisition tools, project management tools, knowledge base verification and refinement techniques) and promote the development of new ones. (Jeffries, 1994)**

A methodology should include a graphical model to represent the conceptual design (Blair *et al.*, 1995a)

The proposed framework certainly utilises existing techniques. Standard tools are used for the functional and data analysis of the existing DSS. Conceptual graphs are integrated with the KADS model of expertise to produce a comprehensive representation of the domain knowledge. Thus tools which specialise in various aspects of the lifecycle have been utilised.

The modelling tools utilised within the analysis of the existing system (data flow diagrams (DFDs) and entity relationship diagrams (ERDs)) are standard tools drawn from the systems analysis discipline. These tools were also found to be popular in practice for developing decision support systems (Atkinson & Arnott, 1995) (Section 2.2.4). Their use for modelling an existing DSS was found to be successful, the only unusual deviation from the norm being the use of unlevelled DFDs (Section 5.2). The ERD was not found to be completely useful, however this was because it was not based on existing files due to confidentiality issues (Section 5.2).

Data flow diagrams and entity relationship diagrams both have graphical standards for display. These standards are supported by a number of CASE tools, the Visible Analyst tool was used for the creation of these diagrams, the use of such tools aids in the standardisation of the resultant models.

5. A methodology should be explorative, it should allow both system specification and design to proceed incrementally, experimenting with alternative problem solving approaches. (Jeffries, 1994)

The model of expertise moves on to the early stages of design by providing a task layer which defines the control structure for how the intelligent component utilises the information in the inference and domain layers. The purpose of KADS is to represent the problem solving capabilities of a future system which involves specifying the early stages of design of the system.

Due to the separation of knowledge in the model of expertise, different control structures can be applied to the same domain knowledge, this reuseability of the domain layer is seen as one of the major advantages of the KADS methodology. The inference structures (Figure 5.6 and 5.7) have no implied direction, thus they can be used in different ways such as forward or backward chaining. This allows experimentation with different problem solving approaches within the same inference structure.

The model of expertise can be used to explore different methods of implementation, such as a variety of expert system shells or programming languages. It was found that the conceptual graph formalism lent itself very well to being implemented in the object oriented environment of KAPPA-PC. This straight forward mapping allowed the prototype to be developed relatively easily and quickly.

6. A methodology should be structured and modular, it should support as far as possible (hierarchical) work decomposition into elementary components, and enable generalisation through the structures hierarchical chaining. (Jeffries, 1994)

A methodology should include techniques to abstract and decompose the requirements (Blair *et al.*, 1995a)

A methodology should include methods to abstract and decompose the conceptual design of the IDSS (Blair *et al.*, 1995a)

The data flow diagrams developed in the "Functional analysis" phase (Figure 5.15) can be organised in a layered manner, with higher layers generalising lower layers. This approach, however, was found to be inappropriate in this case as an unlevelled diagram was a more effective communication tool (Section 5.2).

The conceptual graph notation is definitely based on structural inheritance networks, enabling generalisation and specialisation of the graphs and concepts. The type lattice provides the basis for this generalisation to take place. Relations can also form a hierarchy with some relations being defined in terms of more primitive ones.

The KADS models do not have this type of hierarchical chaining, but do decompose the knowledge into elementary components. The task model breaks the problem into tasks and subtasks, with the primitive tasks being allocated to particular components for execution within the model of cooperation.

The domain layer is quite separate from the other layers in the model of expertise, and is used to instantiate the inference structures in the task layer. There are several advantages of separating domain and inference knowledge, this enables the use of the same domain knowledge a number of times, knowledge redundancy is also prevented because domain knowledge that is used in more than one inference is specified only once.

The framework does not have a means of abstracting and decomposing the requirements of the IDSS. The requirements definition differs from a typical IDSS methodology, as much of the functionality is determined by the structure of the existing DSS. Thus, aims of the intelligent component are not defined until mid way through the development framework at the beginning of the knowledge acquisition phase, following the analysis of the existing DSS. There is no specified method for requirements definition within the proposed framework.

7. A methodology's design language should be rich in the sense that it allows the expression of different kinds of knowledge by different language primitives (Blair *et al.*, 1995a)

The KADS methodology separates different kinds of knowledge within the model of expertise. The domain knowledge is represented by conceptual graphs and the task layer represents the control knowledge. The overall framework also separates models of the existing DSS from that of the intelligent component. The two sets of models are linked by the model of cooperation.

8. A methodology should be rich in heuristic power, the concepts, assumptions, symbolisms and transformation procedures should be such as to make the detection of gaps and errors, the design of problem solving strategies, the recognition of relevant evidence, easily manageable. (Jeffries, 1994)

The KADS interpretation models provide a great deal of power to the methodology by acting as templates for the inference of a proposed system. These interpretation models ensure that the relevant domain knowledge is used at the inference layer. The interpretation models direct the development of the task layer which contains the problem solving strategies of the proposed system. The templates integrate the layers in the model of expertise helping to ensure that no gaps or errors are left in the model. The

overall structure of the model of expertise, with the separate layers for the different types of knowledge makes the representation manageable. These advantages of the KADS methodology results from its top-down approach to knowledge acquisition.

Conceptual graphs have the ability to represent as much information as predicate logic which is generally considered to be a rigorous representation mechanism. The division of concepts into type and referent fields allows a vast amount of information to be formalised. The conceptual relations provide an efficient way of representing the relationships between concepts. The formation rules associated with the notation ensure that all models created are consistent and represented correctly.

9. A methodology should include techniques to normalise the conceptual model
(Blair *et al.*, 1995a)

Relations are said to be normalised if all underlying domains contain only atomic values (Date, 1995). It is felt that with the association of conceptual graphs with first order predicate logic, the conceptual graph model can be considered normalised. The representation is grounded in logic and the notation, if applied correctly ensures a correct model. The ERD created in the "Data Analysis" phase (Figure 5.15 and Appendix A) for the existing DSS can be normalised to ensure a consistent model is created.

10. A methodology should have the ability to be adapted to various applications, it should be general, flexible, maintainable and extendable. (Jeffries, 1994)

A methodology should allow the knowledge domain be expressed independently from its use (Blair *et al.*, 1995a)

Stages in the development framework have been defined, and suitable methods have been suggested. However, alternative or additional methods could be used enabling the substitution of methods with which the knowledge engineer is familiar. For example, the domain layer in the model of expertise could be modelled using semantic networks rather than conceptual graphs. This makes the framework very flexible. Some of the proposed methods are also inherently flexible. For example conceptual graphs have the ability to represent anything that can be expressed in natural language.

The proposed framework has only been investigated using a single case study. It is thought, however, that the framework would lend itself to other situations where a decision support system exists and where intelligent capabilities would add value to the system. The functional and data models should be able to be created for any existing DSS.

The layering in the KADS model of expertise allows reuseability of the domain layer as the knowledge is represented independently from its use. The addition of information to the domain layer allows the model to be extendable. The inference structures aim to generally model a wide range of problem situations, and each can be extended or modified to cope with specific situations.

11. A methodology should include a uniform approach to gather the inter-disciplinary requirements of the IDSS (Blair *et al.*, 1995a)

A methodology should include models to represent the inter-disciplinary requirements graphically (Blair *et al.*, 1995a)

A methodology should include languages for describing the inter-disciplinary requirements (Blair *et al.*, 1995a)

The term "inter-disciplinary" has been taken to mean the interaction between the conventional decision support system and the intelligent component. The model of cooperation has the purpose of representing the link between the two types of information. This is a graphical representation which follows the convention outlined in the KADS methodology, with the additional ability to assign tasks to the DSS. The model of cooperation is developed using the task model and by assigning each low level subtask to particular components in the proposed IDSS. This requires information outlining the capabilities of each component, which can be gathered from the data and functional models and the model of expertise.

Although there is a graphical model for representing the inter-disciplinary requirements, the development framework does not define a language for this purpose.

12. A methodology should include techniques for searching the organisation for potential existing software components (Blair *et al.*, 1995a)

A methodology should include techniques which help developers to understand what existing software components do (Blair *et al.*, 1995a)

A methodology should support the interconnection of existing and hand coded software components within the IDSS (Blair *et al.*, 1995a)

The aim of the analysis of the existing DSS is to gain an understanding its capabilities. This allows for reuse of some of the components in the DSS and minimises repetition in functionality within the intelligent component. The development of data and functional models define the data requirements and functionality of the DSS.

The overall intention of the framework is to allow for the integration of existing software with an intelligent component. The model of cooperation is used to represent the link between the components.

7.4 CONCLUSION

It has been shown that the intelligent component developed meets the needs of the sponsor, and that the overall system can be considered an IDSS. It is felt that this validates the proposed framework's ability to develop the desired results.

The views and experiences gained in developing the prototype IDSS have also been discussed in relation to criteria outlining desired attributes of a methodology. The functional and data models developed in the framework were found to be a useful means of representing the capabilities and associated data of the existing DSS. The KADS methodology provided a solid foundation for the knowledge acquisition phase of the framework. Conceptual graphs were successfully integrated with the KADS model of expertise, ensuring a rich representation of the domain knowledge. The KADS library of interpretation models guided the knowledge engineer in representing control knowledge. The model of cooperation provided the link between the existing DSS and the proposed intelligent component.

Overall, the framework was found to be a thorough and useful tool for the development of an intelligent component that has to be integrated with an existing DSS to form an IDSS. The framework brings together components from existing, well defined methodologies in a structured manner. The integration of a variety of methods allows the different aspects of an IDSS to be rigorously modelled. Drawing appropriate techniques from established methodologies enables the framework to fulfill many of the requirements of an IDSS methodology.

The aim of the study was to investigate and propose a framework for the development of an intelligent component to be integrated with an existing decision support system. Nunamaker *et al's* (1991) systems development research methodology (Figure 1.1) provided a guide for the overall research effort and each of the five stages in the process were successfully completed.

A thorough review of the literature associated with DSS, KBS and IDSS development revealed that although methodologies exist for each of these there did not appear to be a methodology aimed at improving existing DSSs by adding an intelligent component. Thus an initial development framework was proposed to deal with such a situation, drawing on concepts from related disciplines.

Conventional DSSs are widely used by farmers throughout New Zealand. These systems aid the farmer in organising their data, but do not attempt to interpret the data or suggest feasible courses of action. Simulation models are incorporated within some of these DSSs which can be utilised for planning purposes. These often require a considerable amount of expertise to ensure they are being used correctly, and the results interpreted accurately. The combination of KBS techniques with DSS capabilities to form an IDSS is seen as a means of providing a more powerful tool for the farmer.

The integration of an intelligent component with an existing agricultural DSS would seem an efficient and cost effective method for creating an IDSS. Thus the initial development framework was applied to the domain of dairy farm management based on an existing DSS, FarmTracker. A prototype IDSS was developed with the intention of testing the frameworks feasibility. The framework was modified in light this experience.

The final step in Nunamaker *et al's* (1991) process (observation and evaluation) was discussed in Chapter 7. The success of the intelligent component was investigated by observing its functionality and assessing whether it achieved its defined aims. The overall IDSS was evaluated in light of capabilities which such a system should possess. The experiences gained during development were discussed in relation to criteria established for evaluating an IDSS development framework. The overall findings suggest that the proposed framework, summarised in Figure 5.15, is a feasible approach to developing an intelligent component for integration with an existing DSS.

The framework is divided into two strands, firstly the analysis of the existing system, and then the knowledge acquisition process for the intelligent component. The framework suggests the use of prototyping for testing specific aspects of the IDSS, as opposed to its more common use as an iterative or evolutionary lifecycle.

The analysis of the existing DSS utilises existing functional and data modelling tools. These were found to provide a clear representation of the DSS, and their development enables the capabilities and data requirements of the existing system to be understood by the knowledge engineer. This provides a sound foundation for the knowledge acquisition process to begin. The framework takes into account the data and capabilities of the existing DSS when structuring the intelligent component.

The knowledge acquisition process is based on the KADS methodology. A three layered KADS model of expertise provides an organised structure for representing the knowledge. Conceptual graphs were found to be a suitable representation mechanism for the domain layer, providing a complete and consistent model representing a wide range of knowledge. The inference structures from the KADS library provide a considerable amount of power to the framework, aiding the knowledge engineer in developing the inference layer. A task model breaks the functions of the intelligent component into subtasks, which aids in the selection of appropriate inference structures. The task layer represents the control knowledge which instantiates and specifies order to the inference layer.

A modified KADS model of cooperation was seen as a key model in the framework, providing a link between the conventional and intelligent components. This is based on the task model and defines which component is to carry out each subtask.

The knowledge based system shell, KAPPA-PC, was used to develop the prototype. The object oriented environment lent itself well to the conceptual graph representation allowing some aspects of the conceptual model to be directly implemented as equivalent object oriented structures. For example, the type lattice maps to the object hierarchy.

The study was limited by time constraints. This meant user issues were not investigated. These are of obvious importance when developing an IDSS. Time constraints also restricted the scope of the case study, with the prototype intelligent component investigating only a portion of the problem domain. Technical difficulties required a simplified feed budget to be used within the prototype, rather than using the feed budget module within FarmTracker. Confidentiality issues also limited the study, requiring

dummy files to be created to provide the link between the DSS and the intelligent component. This meant a truly integrated IDSS could not be developed.

By proposing a development framework and proving its feasibility by demonstration, the aims of the study have been achieved. Overall, the success of the framework is attributed to its use and modification of existing, well established methods. The framework draws concepts from DSS, KBS and IDSS research. The resulting integrated set of methods is well suited to a situation where an existing DSS could be enhanced by KBS techniques.

8.1 FUTURE WORK

The most obvious extension to the study is the development of the framework into a complete methodology. This would require more detail to be defined at each of the steps. For example, defining a more detailed approach to the definition of aims phase. A complete methodology could be evaluated rigorously as all aspects of development should have been dealt with.

Further studies should also involve consultation with users. This could result in a more functional system for the average user. Interface issues could also be explored.

Further investigation into the link between the conceptual graph notation and the object oriented formalism could prove valuable. Development of the prototype using KAPPA-PC indicated there is a fairly direct mapping between the two representations; one which could be used for the rapid development of systems.

Within the domain of dairy farm management, the prototype IDSS could be developed further, as the domain would benefit from the use of Intelligent Decision Support Systems. This would require additional knowledge acquisition and true integration with FarmTracker. The involvement of typical users in the development would be necessary.

REFERENCES

- Alter, S. (1980). Decision Support Systems: Current Practice and Continuing Challenges. Reading: Addison-Wesley.
- Alter, S. (1994). Transforming DSS Jargon into Principles for DSS Success. In P. Gray (Eds.), Decision Support and Executive Information Systems. Englewood Cliffs: Prentice-Hall.
- Angehrn, A. A., & Luthi, H.-J. (1990). Intelligent Decision Support Systems: A Visual Interactive Approach. Interfaces, 20(6), 17-28.
- Arinze, B. (1992). A user enquiry model for DSS requirements analysis: a framework and case study. International Journal of Man-Machine Studies, 37, 241-264.
- Arnott, D. R. (1992). A Debiasing Methodology for Decision Support Systems Development (Working paper 6/92): Monash University.
- Atkinson, J., & Arnott, D. (1995). Tools and Techniques used in Systems Analysis for Decision Support Systems (Working Paper 22/95): Monash University.
- Avison, D. E., & Fitzgerald, G. (1995). Information Systems Development: Methodologies, Techniques and Tools. (2 ed.). London: McGraw-Hill Book Company.
- Basili, V., Selby, R., & Hutchens, D. (1986). Experimentation in Software Engineering. IEEE Transactions on Software Engineering, 12(7), 733-743.
- Beck, H. W., & Jones, J. W. (1989). Simulation and Artificial Intelligence. In J. R. Barrett & D. D. Jones (Eds.), Knowledge Engineering in Agriculture. Michigan: American Society of Agricultural Engineers.
- Bennett, R. (1992). Case-Study of a Simple Decision Support System to Aid Livestock Disease Control Decisions. Agricultural Systems, 38, 111-129.
- Beulens, A. J. M., & Van Nunen, J. A. E. E. (1988). The Use of Expert System Technology in DSS. Decision Support Systems, 4, 421-431.
- Biebow, B., & Chaty, G. (1993). A Comparison between Conceptual Graphs and KL-ONE. In G. Mineau, B. Moulin, & J. F. Sowa (Eds.), Conceptual Graphs for Knowledge Representation (pp. 75-89). Germany: Springer-Verlag.
- Blair, A. (1995). A Comparative study of Methodologies for Designing IDSSs [Conference Seminar]. Canberra.
- Blair, A., Debenham, J., & Edwards, J. (1995a). A Comparative Study of Formal Methodologies for Designing IDSSs. Paper presented at the Eight Australian Joint Conference on Artificial Intelligence, Canberra, Australia.

- Blair, A., Debenham, J., & Edwards, J. (1995b). Methodologies for Designing IDSSs: Problems and Solutions. Paper presented at the First Australian Workshop on Intelligent Decision Support Systems, Canberra.
- Bodily, S. E. (1985). Modern Decision Making. New York: McGraw-Hill, Inc.
- Bonczek, R. H., Holsapple, C., & Whinston, A. (1980). Future Directions for Developing Decision Support Systems. Decision Support Systems, 11, 616-631.
- Breuker, J., & de Greef, P. (1993). Modelling System-User Cooperation in KADS. In G. Schreiber, B. Wielinga, & J. Breuker (Eds.), KADS A Principled Approach to Knowledge-Based System Development (pp. 47-70). Cambridge: Academic Press.
- Butler, B. M. (1986). The effect of grazing intensity and frequency during spring and early summer on the sward characteristics of a ryegrass-white clover pasture. Unpublished Masters thesis, Massey University.
- Buzan, T., & Buzan, B. (1993). The Mind Map Book. London: BBC Books.
- Chang, A.-M., Holsapple, C., & Whinston, A. (1993). Model management issues and directions. Decision Support Systems, 9, 19-37.
- Date, C. J. (1995). An Introduction to database systems. (6 ed.). Reading: Addison-Wesley Publishing Company.
- Doukidis, G. (1988). Decision Support System Concepts in Expert Systems: An Empirical Study. Decision Support Systems, 4, 345-354.
- Duffin, P. H. (1988). GEMINI: Government Expert system Methodology INitiative : Central Computer and Telecommunications Agency, London.
- El-Najdawi, M. K., & Sylianou, A. C. (1993). Expert Support Systems: Integrating AI Technologies. Communications of the ACM, 36(12), 55-65.
- Er, M. C. (1988). Decision Support Systems: A Summary, Problems, and Future Trends. Decision Support Systems, 4, 355-363.
- Evans, M., Mondor, R., & Flaten, D. (1989). Expert Systems and Farm Management. Canadian Journal of Agricultural Economics, 37, 639-666.
- Fedorowicz, J., Williams, G. B. (1986). Representing Modeling Knowledge in an Intelligent Decision Support System. Decision Support Systems, 2, 3-14.
- Ford, F. N. (1985). Decision Support Systems and Expert Systems: A Comparison. Information and Management, 8, 21-26.
- Gane, C., & Sarson, T. (1979). Structured Systems Analysis: Tools and Techniques. Englewood Cliffs: Prentice Hall.
- Gillies, A. (1991). The Integration of Expert Systems into Mainstream Software. Cornwall: Chapman and Hall.

- Gonzalez, A., & Dankel, D. (1993). The Engineering of Knowledge-based Systems: Theory and Practice. Englewood Cliffs: Prentice-Hall.
- Goodwin, P., & Wright, G. (1991). Decision Analysis for Managerial Judgement. New York: Wiley & Sons, Inc.
- Gottinger, H. W., & Weimann, P. (1992). Intelligent decision support systems. Decision Support Systems, 8, 317-332.
- Goul, M., & Tonge, F. (1987). Project IPMA: Applying Decision Support System Design Principles to Building Expert-based Systems. Decision Sciences, 18, 448-467.
- Gray, D. I., & Parker, W. J. (1992). The Planning, Implementation and Control of Pastoral Based Systems : Agricultural and Horticultural Systems Management Department, Massey University.
- Guida, G., Tasso, T. (1989). Building Expert Systems: From Life Cycle to Development Methodology. In G. Guida, C. Tasso (Eds.), Topics in Expert Systems Design: Methodologies and Tools (pp 3-26). Amsterdam: North Holland.
- Harmon, P., & Hall, C. (1993). Intelligent Software Systems Development: An IS Manager's Guide. New York: John Wiley and Sons, Inc.
- Harmon, P., & King, D. (1985). Expert Systems: Artificial Intelligence in Business. New York: John Wiley & Sons.
- Hayes-Roth, F., Waterman, D., & Lenat, D. (1983). An Overview of Expert Systems. In Hayes-Roth, Waterman, & Lenat (Eds.), Building Expert Systems . Reading: Addison-Wesley Publishing Company.
- Hickman, F. R. (1989). Analysis for knowledge-based systems: a practical guide to the KADS methodology. Chichester: Ellis Horwood Limited.
- Holmes, C. W., & Wilson, G. F. (1987). Milk Production From Pasture. (2 ed.). Wellington: Butterworths.
- Holtzman, S. (1989). Intelligent Decision Systems. Reading: Addison-Wesley Publishing Company, Inc.
- Jackson, P. (1990). Introduction to Expert Systems. (2 ed.). Wokingham: Addison-Wesley.
- Jeffers, J. N. R. (1982). Modelling. London: Chapman and Hall.
- Jeffries, A. E. (1994). An Investigation into the use of Conceptual Graphs for Analysing and Representing Knowledge. Unpublished Honours Dissertation, Massey University.

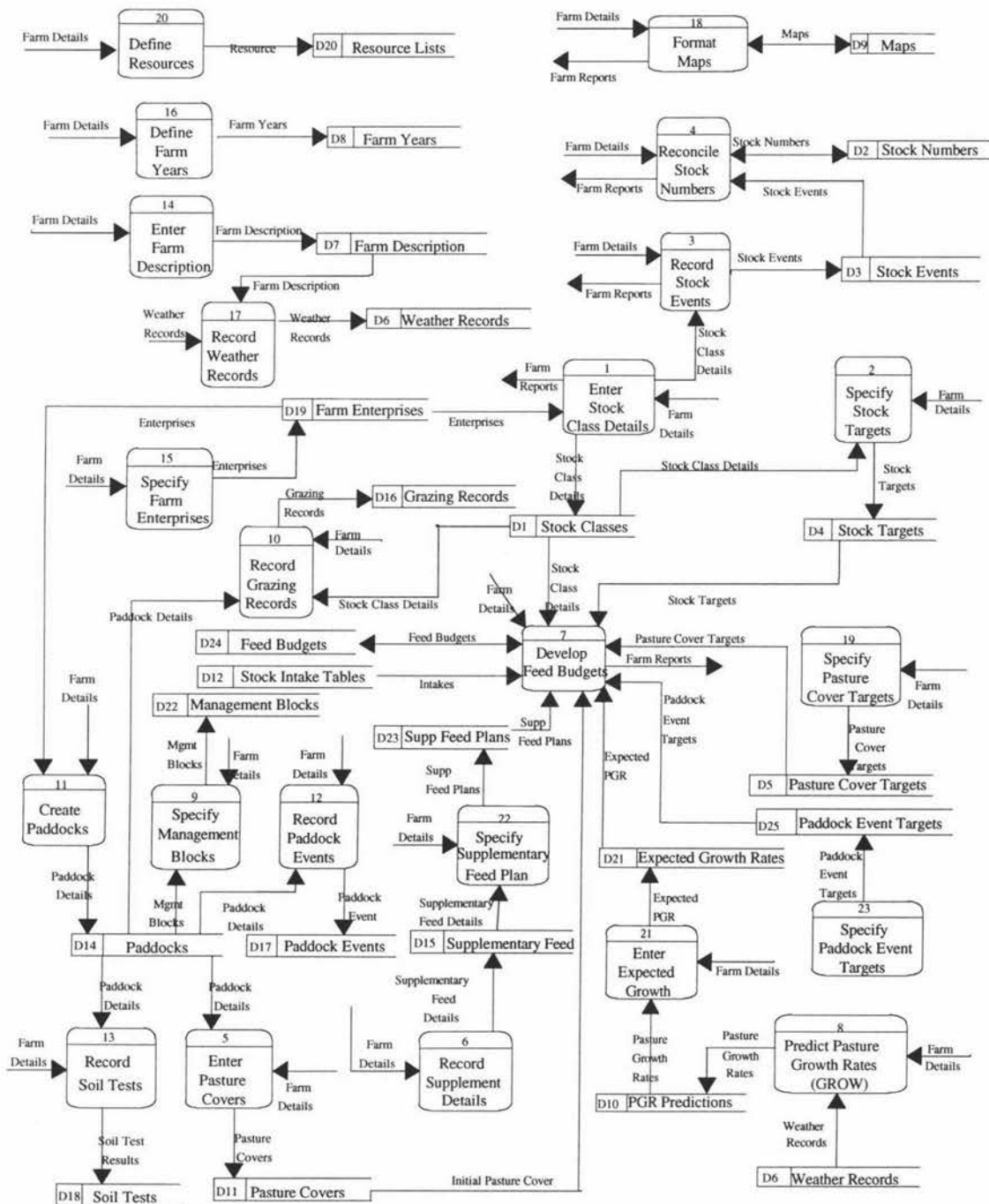
- Jeffries, A. E., Todd, E. G., & Kemp, E. A. (1995a). Comparison of KADS and Conceptual Analysis. Paper presented at the Eighth Australian Conference on Artificial Intelligence, Canberra.
- Jeffries, A. E., Todd, E. G., & Kemp, E. A. (1995b). A Tutorial on Conceptual Graphs (Technical Report 1/95): Massey University.
- Jones, J. W., Jones, P., & Everett, P. A. (1987). Combining Expert Systems and Agricultural Models: A Case Study. Transactions of the ASAE, 30(5), 1308-1314.
- Jones, P. (1989). Agricultural Applications of Expert Systems Concepts. Agricultural Systems, 31, 3-18.
- Karbach, W., Linster, M., & Voss, A. (1990). Model of Problem-Solving: One label - One idea? In B. Wielinga, J. Boose, B. Gaines, G. Schreiber, & M. van Someren (Eds.), Current Trends in Knowledge Acquisition. London: Ellis Horwood.
- Keen, P., & Scott Morton, M. (1978). Decision Support Systems: An Organizational Perspective. Reading: Addison-Wesley.
- Kemp, E. A., Todd, E. G., daSilva, A., & Gray, D. I. (1994,). Knowledge acquisition applied to farmer decision making. Paper presented at the SPICIS.
- Klein, M., & Methlie, L. (1995). Knowledge-based Decision Support Systems: With Applications in Business. (2 ed.). Chichester: John Wiley & Sons.
- Lal, H., Jones, J., Peart, R., & Shoup, W. (1992). FARMSYS - A Whole-Farm Machinery Management Decision Support System. Agricultural Systems, 38, 257-273.
- Lukose, D. (1993). Executable Conceptual Structures. In G. Mineau, B. Moulin, & J. F. Sowa (Eds.), Conceptual Graphs for Knowledge Representation. New York: Springer-Verlag.
- Mainland, D. (1994). A Decision Support System for Dairy Farmers and Advisors. Agricultural Systems, 45, 217-231.
- Marsden, J., & Pingry, D. (1993). Theory of Decision Support Systems portfolio evaluation. Decision Support Systems, 9, 183-199.
- McGovern, J., Samson, D., & Wirth, A. (1991). Knowledge acquisition for intelligent decision systems. Decision Support Systems, 7, 263-272.
- Meador, C. L., Guyote, M. J., & Rosenfeld, W. L. (1986). Decision Support Planning and Analysis: The Problems of Getting Large-Scale DSS Started. MIS Quarterly(June), 158-177.
- Mehandjiska, D. S. (1993). Knowledge Representation. Lecture presented at Massey University, Palmerston North, New Zealand.

- Mehandjiska, D. S., & Page, D. (1993). Object-Orientated Development of Expert Systems. Paper presented at the ANNES, Dunedin.
- Mittra, S. S. (1986). Decision Support Systems: Tools and Techniques. New York: John Wiley & Sons.
- Moser, J. G. (1986). Integration of artificial intelligence and simulation in a comprehensive decision-support system. Simulation, 47(6), 223-229.
- Newell, A., & Simon, H. (1972). Human Problem Solving. Englewood Cliffs: Prentice-Hall.
- Nosek, J., Roth, I. (1990). A comparison of formal knowledge representation schemes as communication tools: predicate logic vs semantic network. International Journal of Man Machine Studies, 33, 227-239.
- Novak, J. D., & Gowin, D. B. (1984). Learning How To Learn. Cambridge: Cambridge University Press.
- Nunamaker, J., Chen, M., & Purdin, T. (1991). Systems Development in Information Systems Research. Journal of Management Information Systems, 7(3), 89-106.
- O'Donnell (1995, September 24). Re:IDEdit [e-mail to Anna Jeffries], [Online]. Available e-mail: podonnel@fcit.monash.edu.au.
- O'Donnell, P., & Watson, J. (1994). IDEdit: An Influence Diagram Editor for Macintosh (Working Paper 5/94): Monash University.
- Payne, E. C., & McArthur, R. C. (1990). Developing Expert Systems: A Knowledge Engineer's Handbook for Rules & Objects. New York: John Wiley & Sons.
- Plant, R. E., & Stone, N. D. (1991). Knowledge-Based Systems in Agriculture. New York: McGraw-Hill.
- Polovina, S., & Heaton, J. (1992). An Introduction to Conceptual Graphs. AI Expert(May), 36-43.
- Regoczei, S., & Hirst, G. (1988). Knowledge acquisition as knowledge explication by conceptual analysis (Technical Report CSRI-205): Computer Systems Research Institute, University of Toronto.
- Reitman, W. (1982). Applying Artificial Intelligence to Decision Support: Where Do good Alternatives Come From? In Ginzberg, Reitman, & Stohr (Eds.), Decision Support Systems. New York: North-Holland Publishing Company.
- Richards, D., & McDonald, C. (1995,). Adapting Expert Systems to Behave Like Decision Support Systems. Paper presented at the First Australian Workshop on Intelligent Decision Support Systems, Canberra.

- Rothenberg, J. (1989). The Nature of Modelling. In L. Widman, K. Loparo, N. Nielson (Eds.), Artificial Intelligence, Simulation, and Modeling (pp. 75 - 92). New York: Wiley.
- Schacter, R. D. (1986). Evaluating Influence Diagrams. Operations Research, 34(6), 871-882.
- Sloman, A. (1978). The Computer Revolution in Philosophy. Hassocks: John Spiers.
- Smith, J. Q. (1988). Decision Analysis: A Bayesian approach. London: Chapman and Hall Ltd.
- Sowa, J. F. (1984). Conceptual Structures: Information Processing in Mind and Machine. (1 ed.). Reading: Addison-Wesley Publishing Company, Inc.
- Sowa, J. F. (1992). Conceptual Analysis as a Basis for Knowledge Acquisition. In R. R. Hoffman (Eds.), The psychology of expertise: cognitive research and empirical AI (pp. 80-96). New York: Springer-Verlay.
- Sprague, R., & Carlson, E. (1982). Building effective decision support systems. Englewood Cliffs: Prentice-Hall.
- Sprague, R. H. (1980). A Framework for the Development of Decision Support Systems. MIS Quarterly, 4(4), 1-26.
- Stabell, C. B. (1983). A Decision-oriented Approach to Building DSS. In J. L. Bennett (Eds.), Building Decision Support Systems. Menlo Park: Addison-Wesley Publishing Company.
- Teague, C., Pidgeon, C. (1985). Structured Analysis Methods for Computer Information Systems. California: Science Research Associates.
- Todd, E. G., Gray, D. I., Lockhart, J. C., & Parker, W. J. (1993,). An expert system to aid dairy farmers. Paper presented at the 13th New Zealand Computer Society Conference, New Zealand.
- Turban. (1993). Decision Support and Expert Systems: Management Support Systems.
- Turban, E. (1995). Decision Support and Expert Systems: Management Support Systems. (4 ed.). New Jersey: Prentice Hall.
- Turban, E., & Watkins, P. (1986). Integrating Expert Systems and Decision Support Systems. MIS Quarterly(June), 121-136.
- Van Weelderen, J., & Sol, H. (1993). MEDSS:A Methodology for Designing Expert Support Systems. Interfaces, 23(3), 51-61.
- Watson, H., & Blackstone, J. (1981). Computer Simulation. (2 ed.). New York: John Wiley & Sons.

- Weitzel, J. R., & Kerschberg, L. (1989). Developing Knowledge-Based Systems: Reorganizing the System Development Life Cycle. Communications of the ACM, 32(4), 482-488.
- Whitten, J. L., Bentley, L. D., & Barlow, V. M. (1994). Systems Analysis and Design Methods. (3 ed.). Burr Ridge: Irwin.
- Wielinga, B. J., Schreiber, A. T., & Breuker, J. A. (1991). KADS: A Modelling Approach to Knowledge Engineering (P5248 KADS-II): University of Amsterdam.
- Yourdon, E. (1989). Modern Structured Analysis. Englewood Cliffs: Prentice-Hall.

A1 UNLEVELLED DATA FLOW DIAGRAM



A3 TECHNICAL ANALYSIS

File structures for dummy ASCII Files, all are tab delimited.

"Farm" Data File

Attribute	Description	Unit	Example
Name	Farm Name		Woodbank
Farmer	Farmer Name		B. D. Speedy
Area	Total Farm Area	hectares	100
TotMk	Total number of milking cows		100
NumPad	Number of Paddocks		30
MaxCows	Maximum number of cows		120
OlsenP	Olsen phosphate level	µg/ml	24
Pott	Pottasium level	me/100g	5.5
S	Sulphur level	µg/g	10
pH	pH of the soil		5.8
N%	Percentage of nitrogen in the soil	%	5
SType	Soil Type		Clay
SDepth	Soil Depth (deep, mod, shallow)		deep
WHC	Water Holding Capacity	mm H ₂ O/10cm soil depth	10
Weight	Average live weight of cows	kg	400
CS	Average condition score		4.5

"Weather" Data File

Attribute	Description	Unit	Example
Date	Week start date	Julian	212
Rfall	Total week rainfall	mm	75
Temp	Weeks average temperature	°celcius	12
ET	Evapotranspiration	mm	5

"Actual" Data File

Attribute	Description	Unit	Example
Date	Date	Julian	212
TS	Total Solids	kg/cow	0.49
MF	Milk Fat	kg/cow	0.21
PadNum	Paddock number		5
Pre	Pre grazing pasture cover	kg DM/ha	2500
Post	Post grazing pasture cover	kg DM/ha	1900
Av	Average pasture cover	kg DM/ha	2200
PasInt	Pasture intake	kg/cow	13
SuppInt	Supplement intake	kg/cow	5
NumMilk	Number of cows milking		90
NumDry	Number of cows dry		12
DA	Daily area allocated to cows	hectares	1
Area	Total available area	hectares	90

"Targets" Data File

Attribute	Description	Unit	Example
Date	Week start date	Julian	212
TS	Total solids	kg/cow/day	0.49
MF	Milk fat	kg/cow/day	0.21
Pre	Pre grazing cover	kg DM/ha	2500
Post	Post grazing cover	kg DM/ha	1900
PasInt	Pasture intake	kg/DM/cow/day	13
SuppInt	Supplement intake	kg/DM/cow/day	5
PGR	Pasture growth rate	kg DM/ha/day	11
NumMilk	Number of cows milking		90
NumDry	Number of cows dry		12
DA	Daily area allocated to cows	hectares	1
Area	Total available area	hectares	90

"Supplements" Data File

Attribute	Description	Units	Example
Name	Supplement name		Hay
FeedingUnit	Unit fed to animals		Bales
KgperUnit	Kilograms per unit	kg	20
FreshDry	Fresh or dry feed		F
DMPercent	Percentage of feed which is dry matter	%	30
MJMEperKgDM	Mega joules of energy per kilogram of dry matter	MJME/ KgDM	10
CrudeProtein	Crude protein content	g/kgDM	15
Costperunit	Cost per Unit	\$	5

All records start on the 1st of July in any year

- | | |
|-----|---|
| 00, | Farm Name, Altitude, Latitude, Distance from Coast, Run Hidden (Y/N), Periods
(12 or 24 in 02-04 records) |
| 01, | Rainfall (12 or 24 periods) |
| 02, | Paddock No, Irrigation (12 or 24 periods) |
| 03, | Air Temperature (12 or 24 periods) |
| 04, | Evapotranspiration (12 or 24 periods) |
| 05, | Percentage change in Rainfall, Irrigation and Temperature |
| 06 | Paddock No, Slope, Aspect, Soil Type, 14 or 28 day grazing, Soil Name, Soil
Depth, Maximum Water Holding Capacity, Initial Water Holding Capacity, Soil
OlsenP, |
| 07, | Paddock No, Pasture Name, (HF, LF, WC), % Content
<i>(up to 5 pasture types and composition)</i> |
| 99 | End of file |

(HF = High fertility grasses, LF = Low fertility grasses, WC = White and other clovers)

Example:

```
00,KappaPrediction,0,0,0,Yes,24
01,52.5,49,59.5,66.5,70,66.5,84,73.5,94.5,94.5,105,105,122.5,112,140,126,
98,105,84,105,87.5,84,56,56
02,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
03,5.11,3.92,5.95,5.67,7.42,6.23,8.82,7.14,10.01,9.73,12.11,11.9,13.02,13.02,
13.65,13.02,9.94,11.41,8.12,10.22,7.77,7.84,5.67,5.74
04,3.5,6.3,9.1,8.4,12.6,10.5,23.8,14.7,32.2,24.5,32.9,32.2,35.7,37.8,32.9,
41.3,16.8,25.9,13.3,18.2,9.8,7.7,6.3,3.5
05,0,0,0
06,1,Flat,Sunny,Clay,14,Clay,Deep,120,120,24
07,1,Ryegrass,HF,90,Whiteclover,WC,10
99
```

All records start on the 1st of July in any year

- | | |
|-----|---|
| 00, | Farm Name, Altitude, Latitude, Distance from Coast, Run Hidden (Y/N), Periods
(12 or 24 in 02-04 records) |
| 01, | Rainfall (12 or 24 periods) |
| 02, | Paddock No, Irrigation (12 or 24 periods) |
| 03, | Air Temperature (12 or 24 periods) |
| 04, | Evapotranspiration (12 or 24 periods) |
| 05, | Percentage change in Rainfall, Irrigation and Temperature |
| 06 | Paddock No, Slope, Aspect, Soil Type, 14 or 28 day grazing, Soil Name, Soil
Depth, Maximum Water Holding Capacity, Initial Water Holding Capacity, Soil
OlsenP, |
| 07, | Paddock No, Pasture Name, (HF, LF, WC, PA, KI), % Content
<i>(up to 5 pasture types and composition)</i> |
| 51, | Predicted rainfall (12 periods) |
| 52, | Predicted temperature (12 periods) |
| 53, | Predicted evapotranspiration (12 periods) |
| 54, | Paddock No, Predicted growth rate (1st, 11th and 21st of each month = 36
periods) |
| 99 | End of file |

Example:

[illegible]

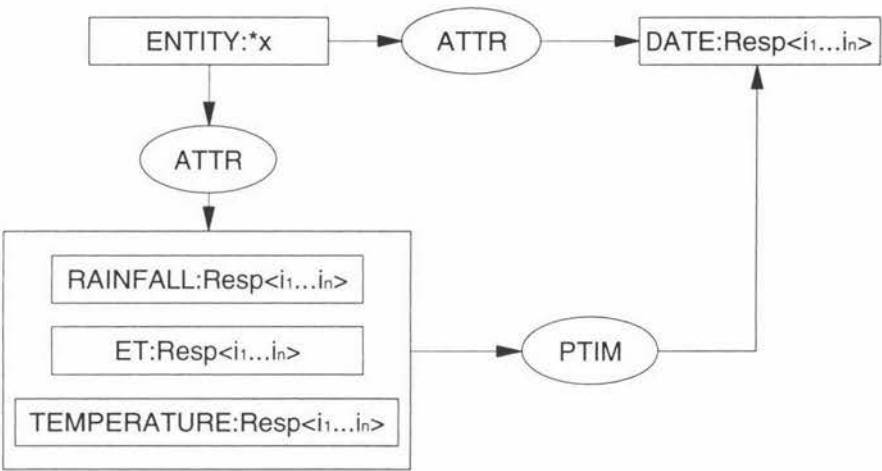
APPENDIX B	KNOWLEDGE ACQUISITION
------------	-----------------------

This appendix is not intended to provide a complete model of the problem domain. Examples of each type of representation have been included to aid in the understanding of the text.

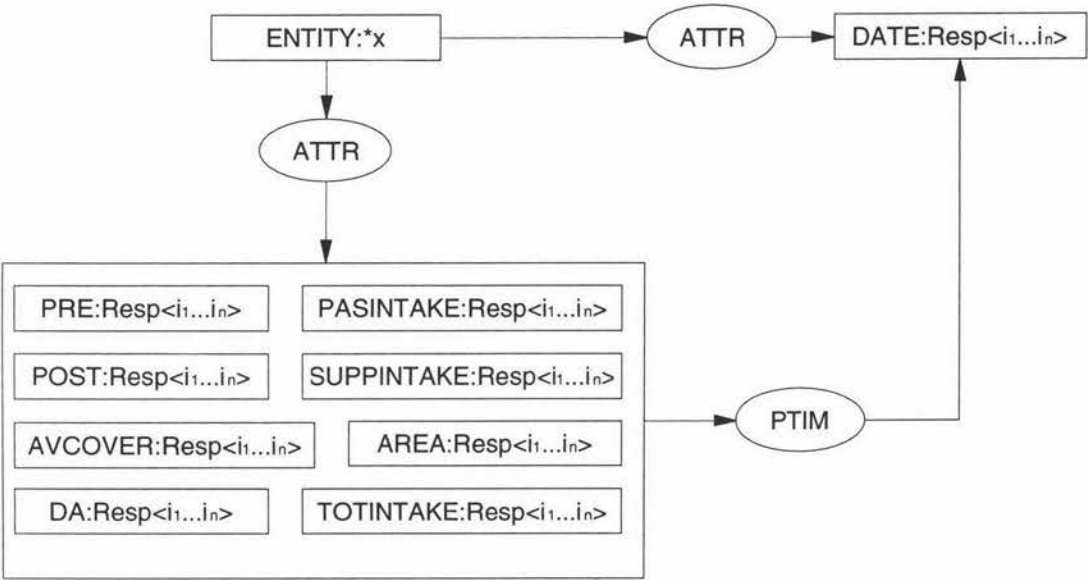
B1 THE DOMAIN LAYER

B1.1 TYPE DEFINITIONS

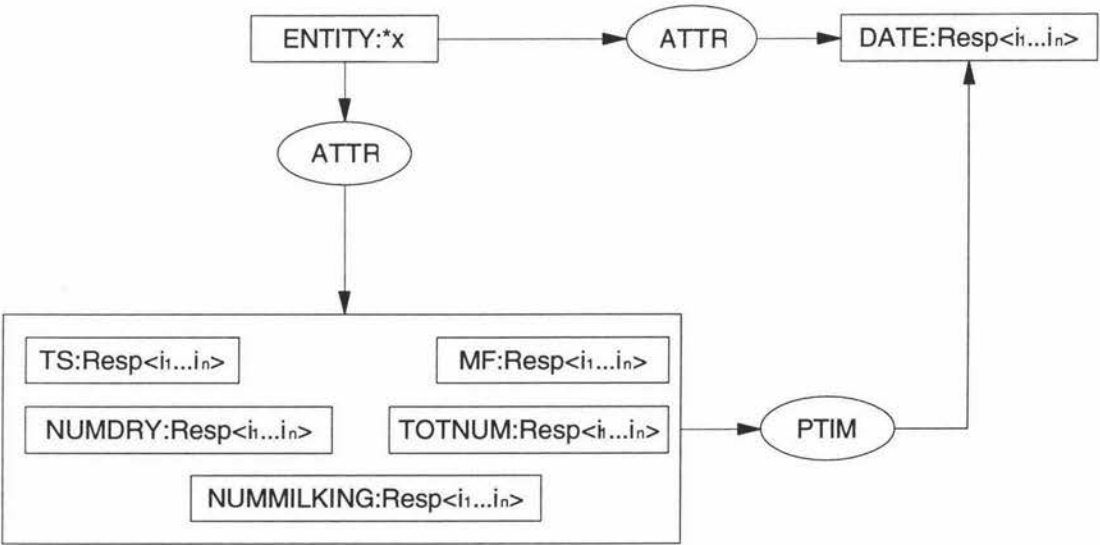
type WEATHER(x) is



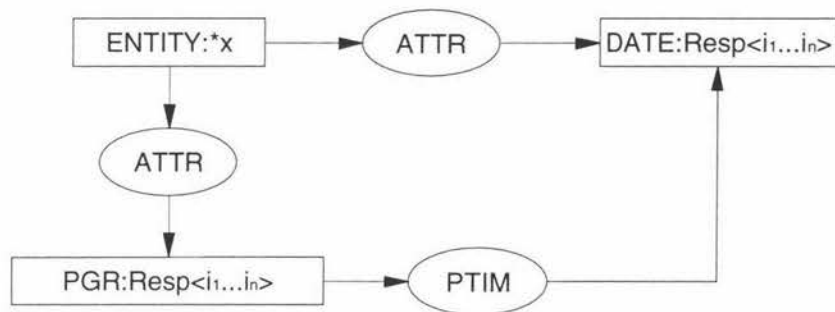
type PASTURE MANAGEMENT DATA(x) is



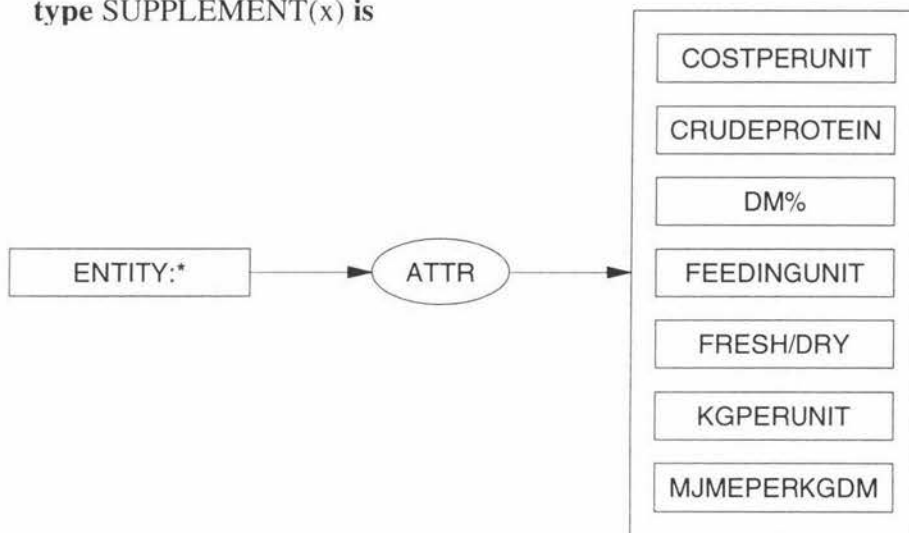
type MILK PRODUCTION DATA(x) is



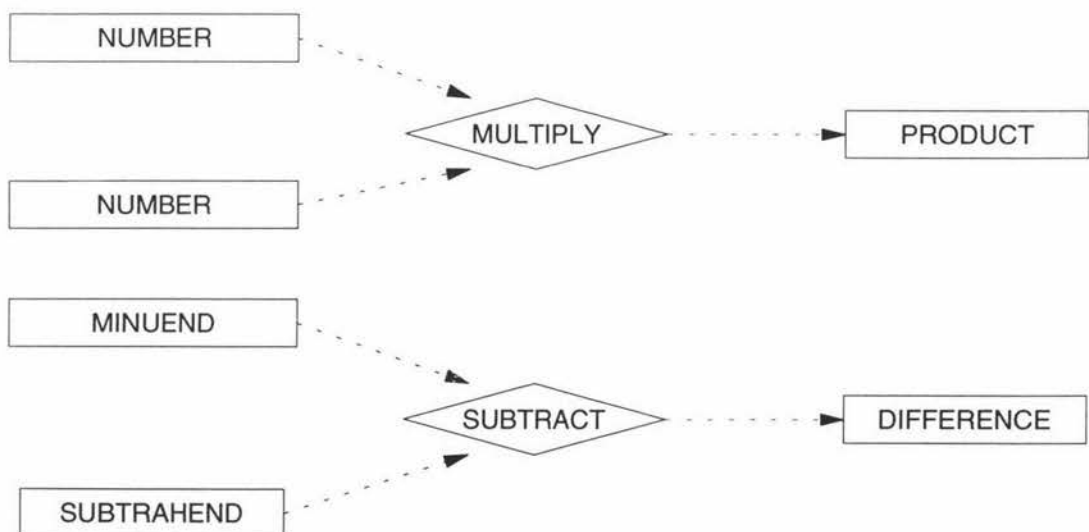
type PGR PREDICTION(x) **is**

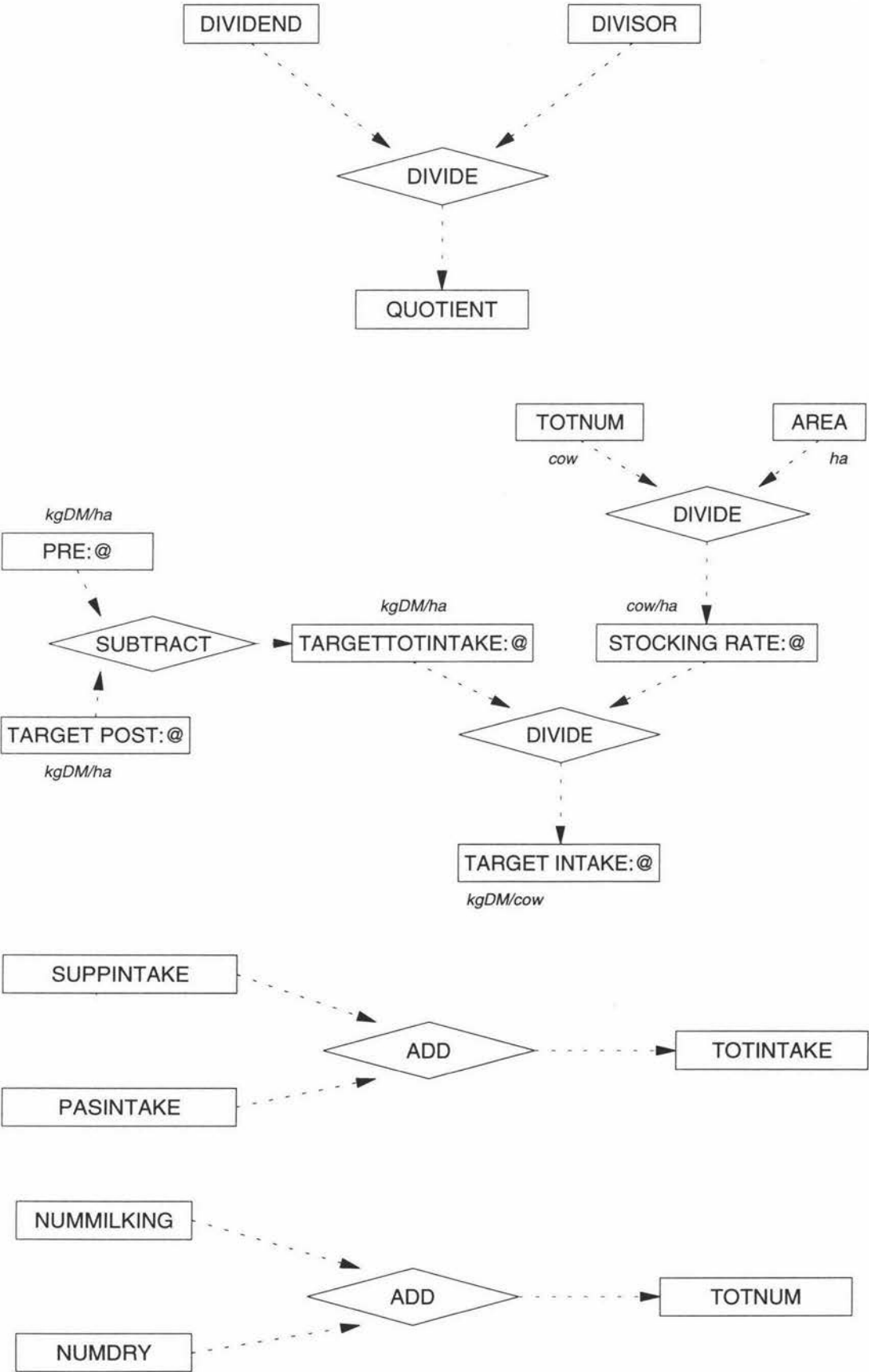


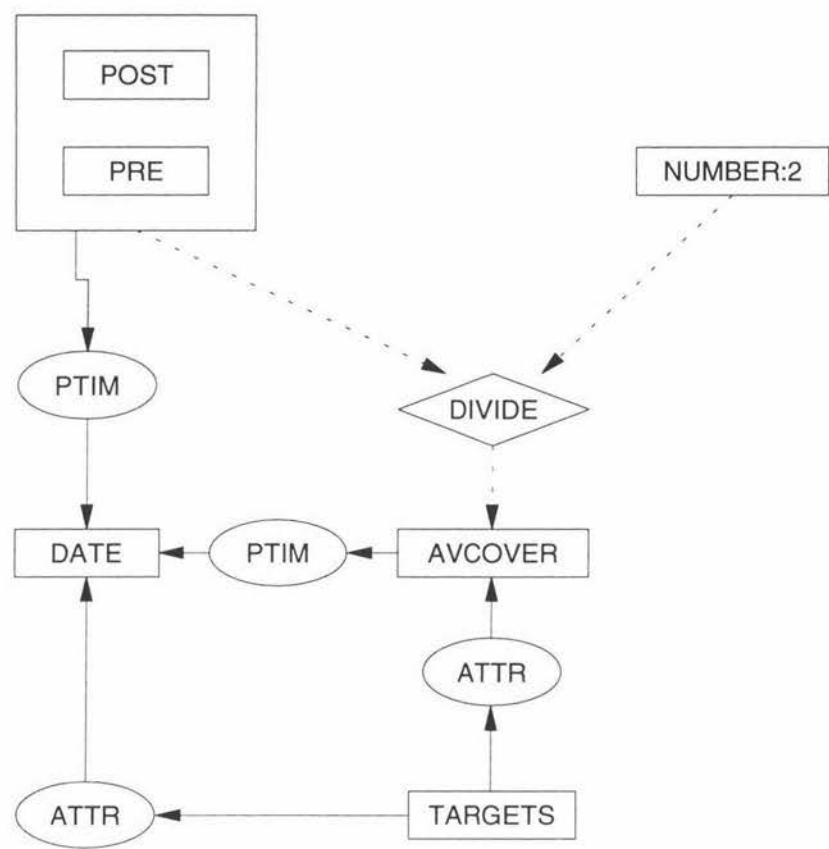
type SUPPLEMENT(x) **is**



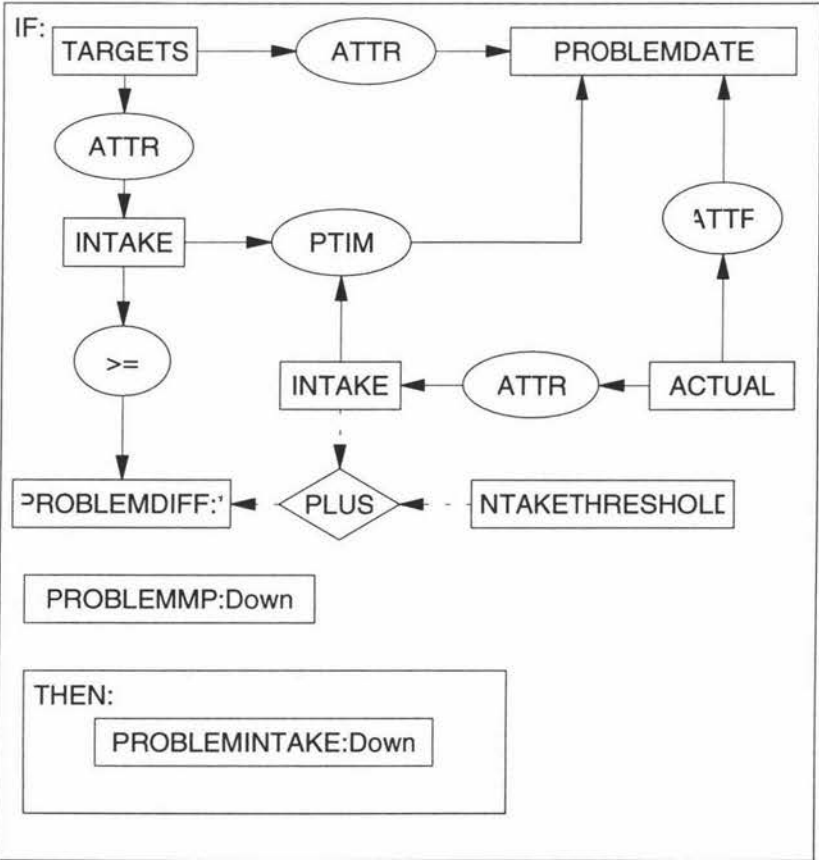
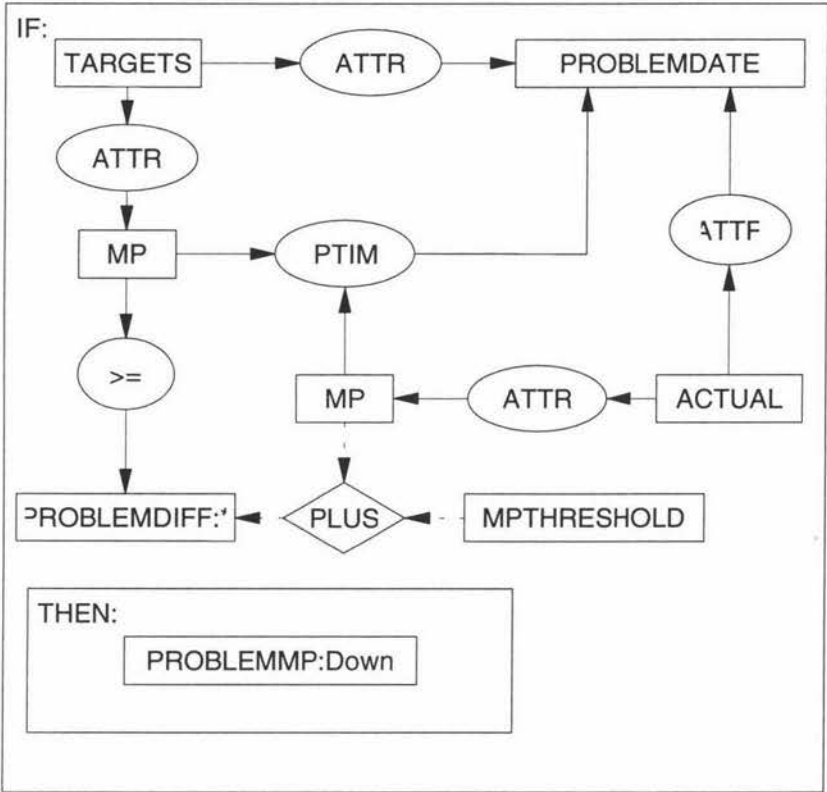
B1.2 ACTOR GRAPHS

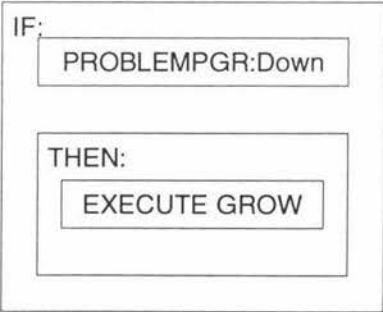
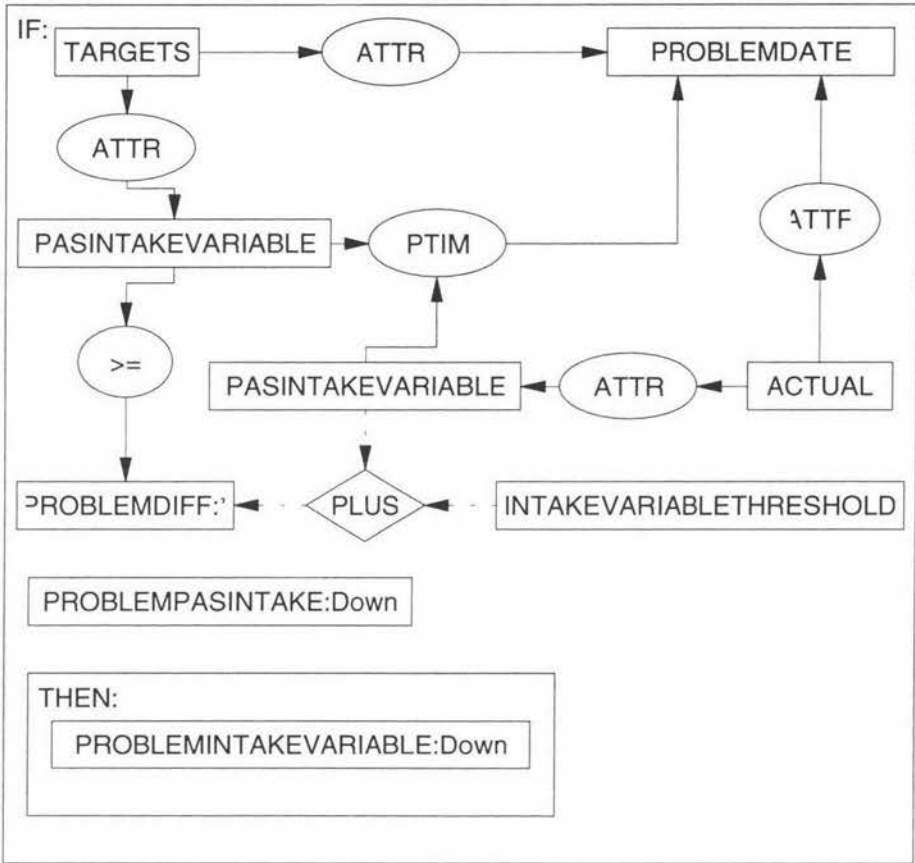


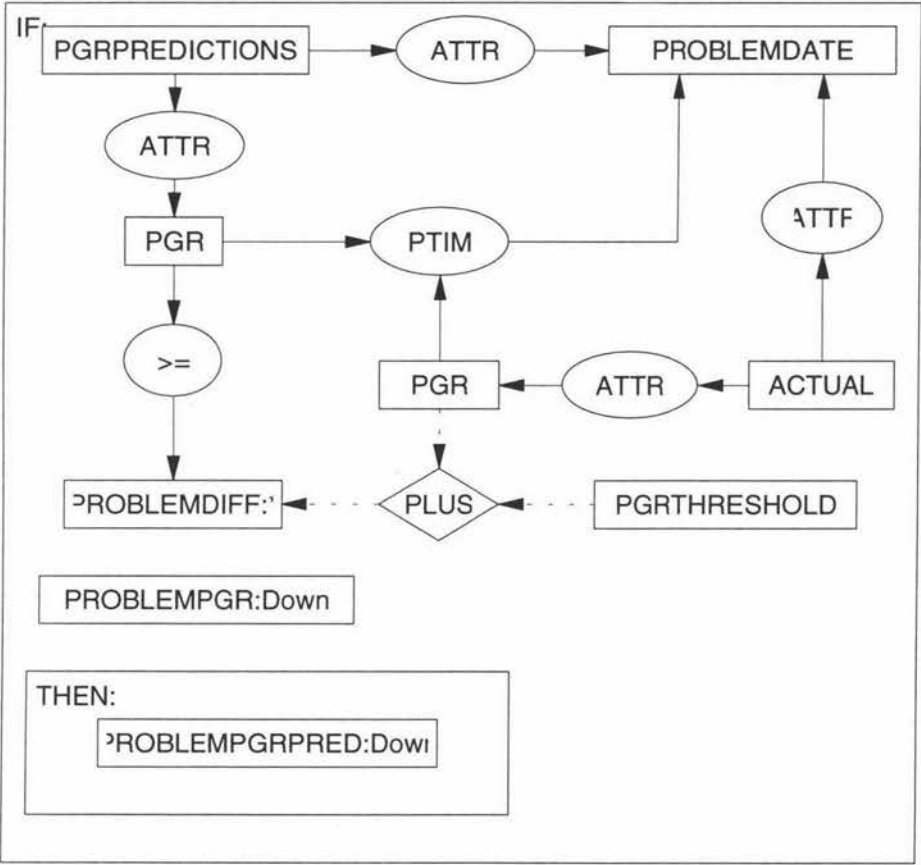




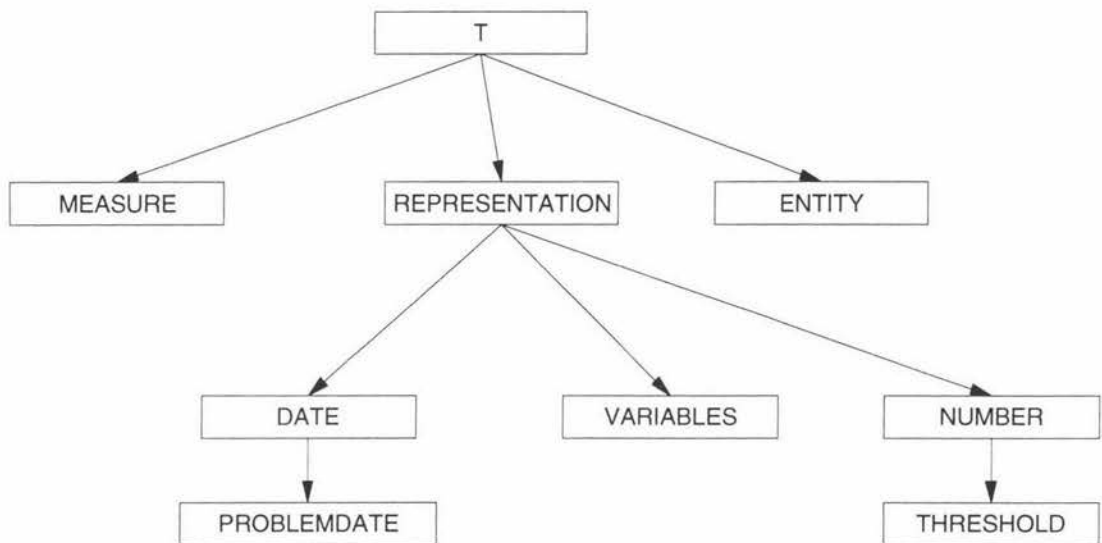
B1.3 IF-THEN CONCEPTUAL GRAPHS

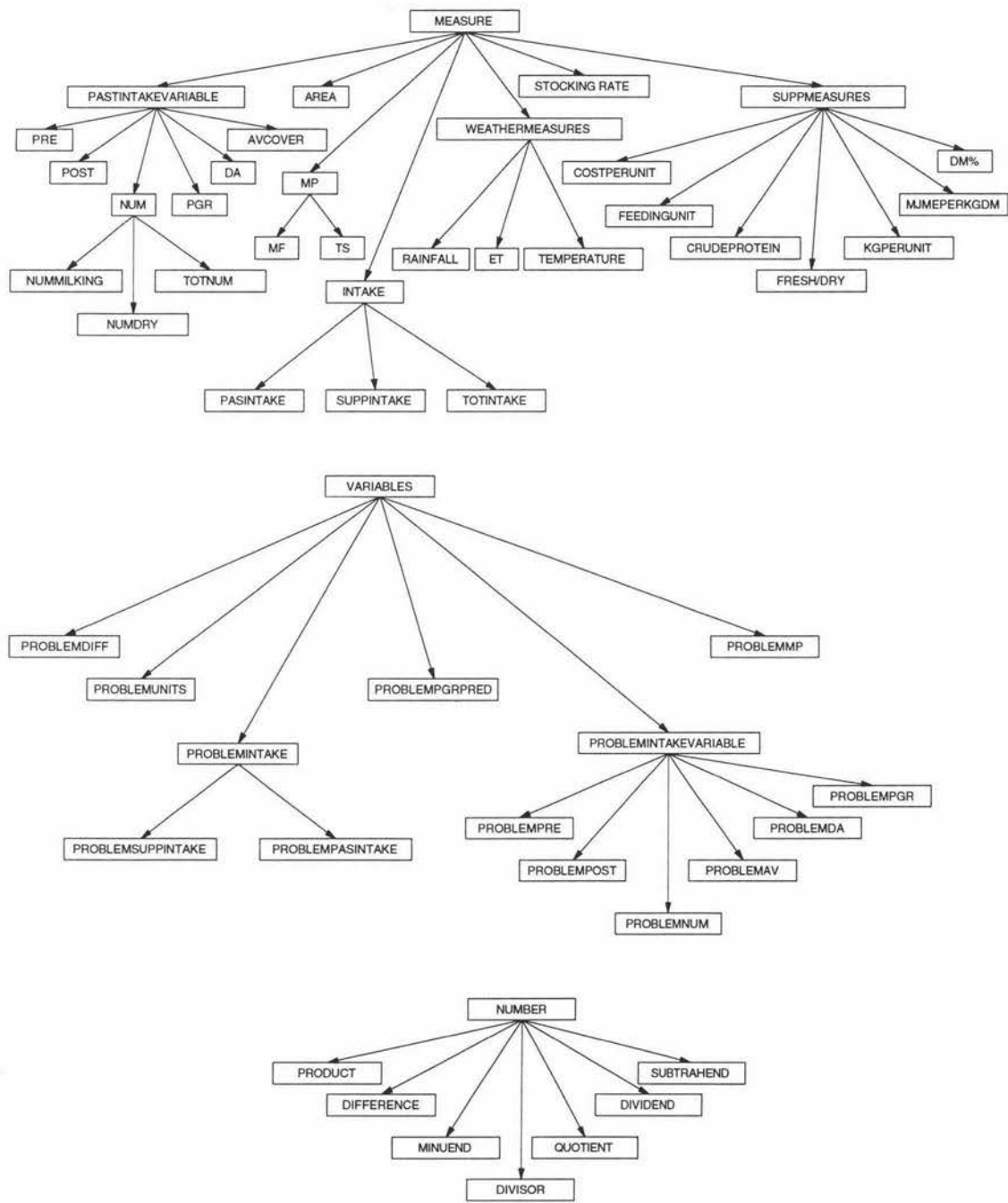


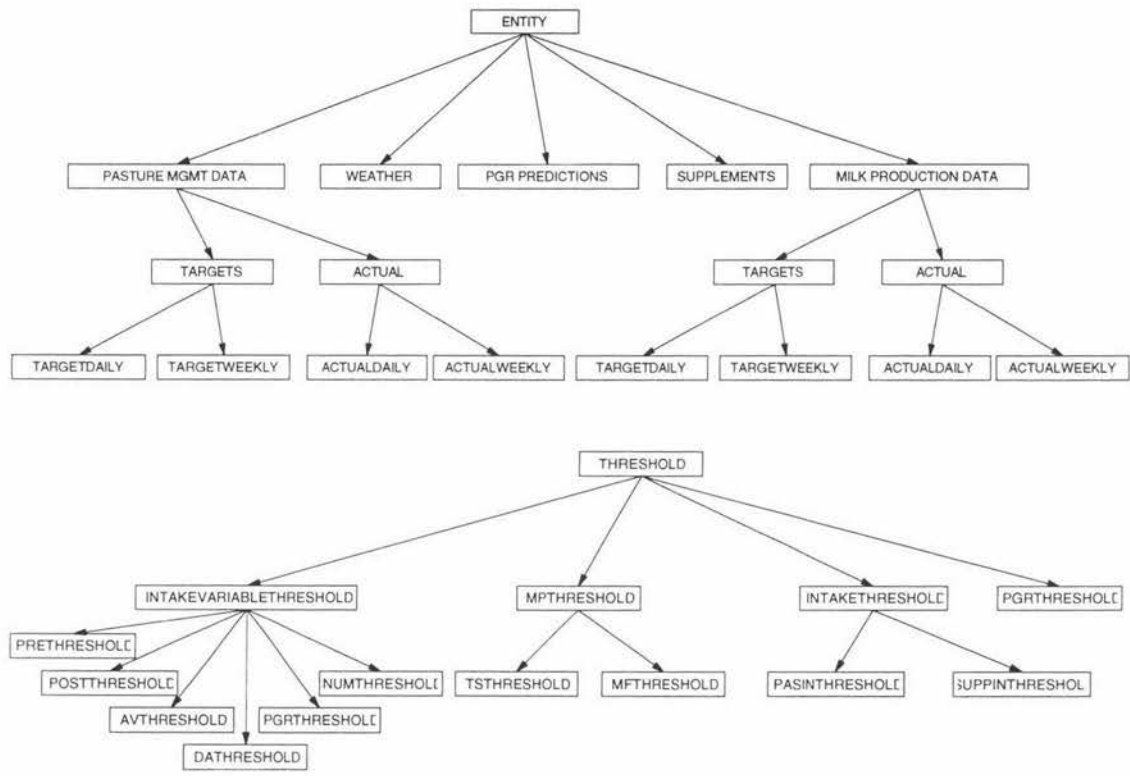




B1.4 TYPE LATTICE







B1.5 CONCEPTUAL CATALOGUE OF CONCEPTS

AREA < MEASURE. Area is a measure of the area of the farm being grazed by the milking herd in hectares.

AVCOVER < PASTINTAKEVARIABLE. Average pasture cover is a measure of the amount of pasture across the farm, measured in KgDM/ha.

DA < PASTINTAKEVARIABLE. Daily allocation is the area of the farm the cows are allowed to graze in one day.

DATE < REPRESENTATION. Date is a representation of time.

ENTITY < T. Entities include physical objects as well as abstractions (Sowa 1984).

INTAKE < MEASURE. Intake is a measure of the feed eaten by an animal, measured in KgDM/cow/day.

MEASURE < T. Measure has no supertypes other than T (Sowa 1984).

MF < MP. Milk fat is a measure of milk production, measured in kg/cow/day

MP < MEASURE. Milk production level is a measure of milk produced by cows.

NUMBER < REPRESENTATION. A number is a representation (Sowa 1991).

PASTINTAKEVARIABLE < MEASURE. Pasture intake variables are measures of pasture intake.

PASTUREMGMTDATA < ENTITY. Pasture management data is an entity which represents information related to the current state of the farm and herd.

PGR < PASTINTAKEVARIABLE. Pasture growth rate is a measure of the rate of growth of pasture.

PRE < PASTINTAKEVARIABLE. Pre is a measure of pre grazing pasture cover, measured in KgDM/ha.

POST < PASTINTAKEVARIABLE. Post is a measure of post grazing pasture cover, measured in KgDM/ha.

RAINFALL < WEATHERMEASURES. Rainfall is a measure of the amount of rain.

REPRESENTATION < T. A representation is a subtype of the universal type.

STOCKINGRATE < MEASURE. Stocking rate is a measure of the number of cows grazed per hectare.

SUPPINTAKE < INTAKE. Supplement intake is the measure of supplement fed to cows, measured in KgDM/cow/day.

SUPPMEASURES < MEASURES. Supplement measures are a measure of supplement attributes.

THRESHOLD < NUMBER. A threshold is a number which represents tolerance levels of an expert between actual and target levels.

TIME < T.

VARIABLES < REPRESENTATION. A variable is a representation of intermediate values used in heuristic evaluation.

WEATHERMEASURES < MEASURE. Weather measures are indicators of the weather situation

\perp < all other types.

T < all other types.

B1.6 CONCEPTUAL CATALOGUE OF RELATIONS

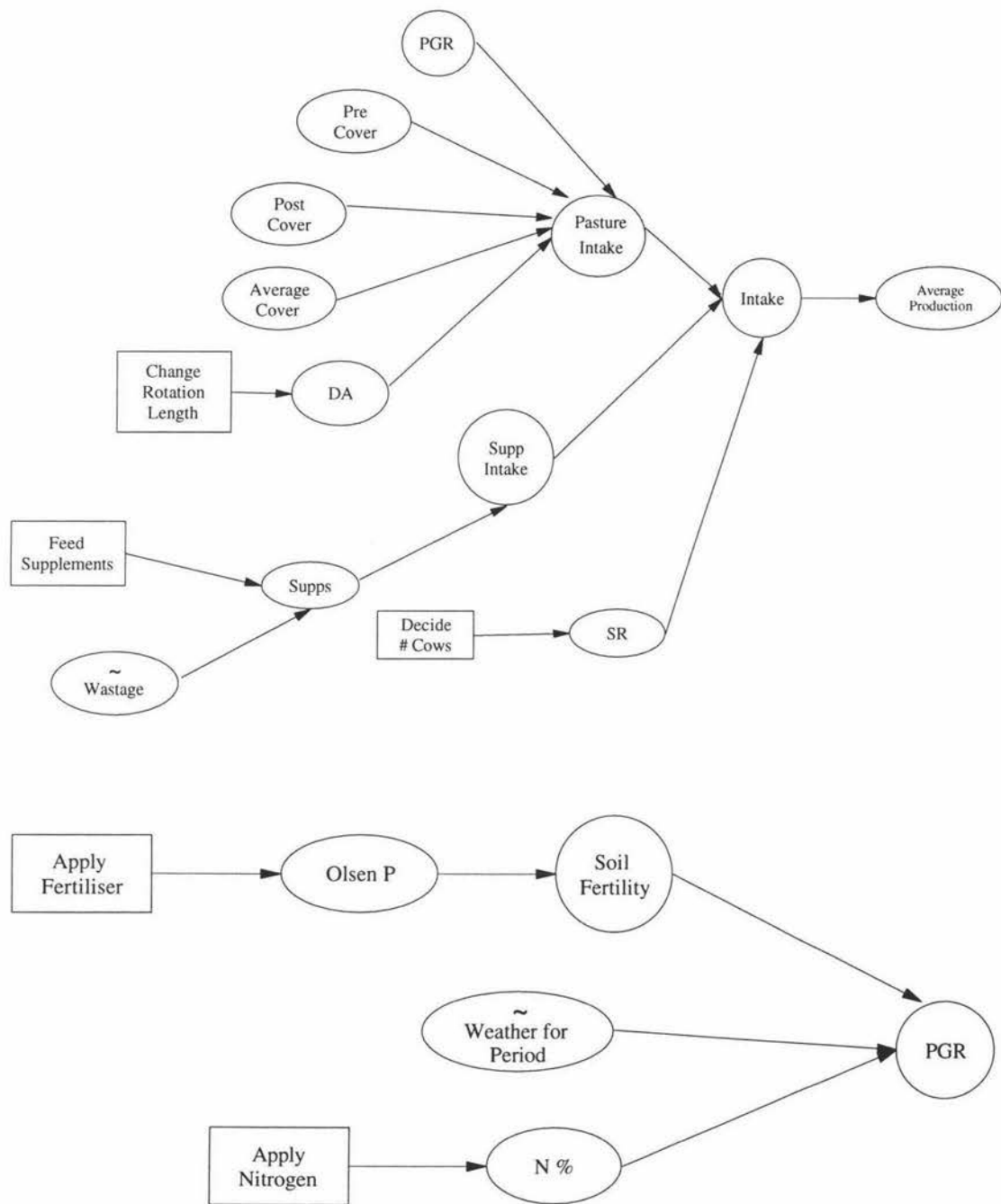
attribute. (ATTR) links [ENTITY:*x] to [ENTITY:*y], where *x has an attribute *y (Sowa 1984).

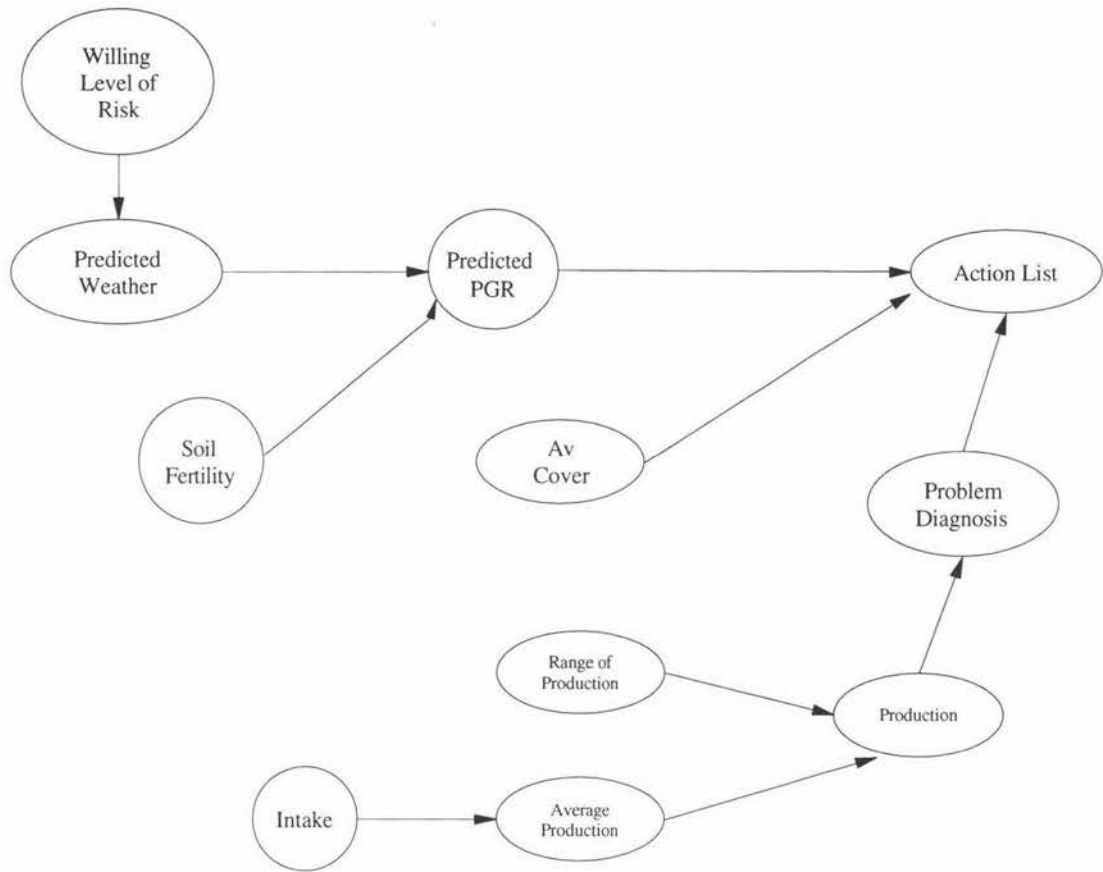
greater than or equal to. (\geq) links an [ATTRIBUTE] to a [NUMBER].

influences. (INFL) links an [ATTRIBUTE] to an [ENTITY] where the attribute influences the entity.

point in time. (PTIM) links [T] to a [TIME] at which it occurs (Sowa 1984).

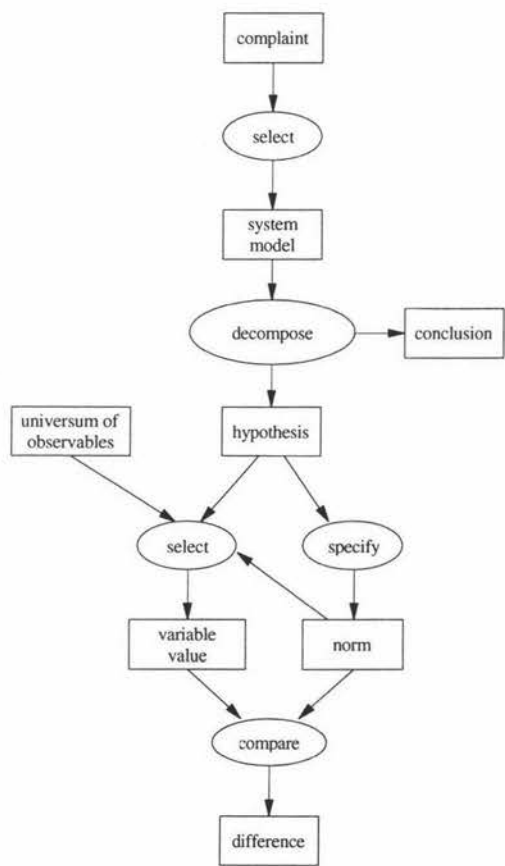
B1.7 INFLUENCE DIAGRAMS





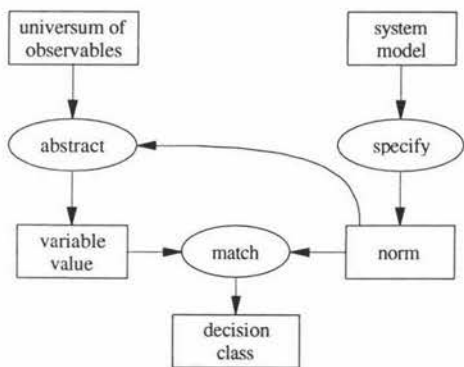
B2 INFERENCE LAYER

B2.1 DIAGNOSIS INFERENCE STRUCTURE FOR MULTIPLE FAULT DIAGNOSIS (BASED ON (HICKMAN, 1989)).



- 1. select system model on the basis of the complaint;
 - 2. decompose system model into hypothesis;
 - 3. for **each** attribute of **each** hypothesis:
 - a. select a variable and find its value;
 - b. specify a norm from the hypothesis comparable to the variable;
 - c. compare the norm to the variable value;
- if the difference exceeds tolerance the current hypothesis becomes the new system model and recurse (from2), else disregard the current hypothesis and go on to the next until all hypothesis have been tested.

B2.2 ASSESSMENT INFERENCE STRUCTURE (HICKMAN, 1989)



- 1. Specify a norm from the system model;
- 2. abstract from universum of observables taking the norm into account, to give the variable value in question
- 3. match the norm to the abstract case description and produce decision class

B3 TASK LAYER

B3.1 DIAGNOSIS

See Figure 5.8

To diagnose problems affecting intake do

select intake relationships and intake targets

decompose into pasture intake and supplement intake

specify norm pasture intake (target) = 12 kgDM/cow/day

select actual pasture intake = 10 kgDM/cow/day

compare target pasture intake and actual pasture intake to give difference

as difference exceeds threshold

and pasture intake is not atomic

decompose pasture intake into pre, post, av, da, PGR⁴

specify norm pre (target) = 2500 kgDM/ha

select actual pre = 2400 kgDM/ha

compare target pre and actual pre to give difference

as difference exceeds threshold

and pre is atomic add pre to problem list

specify norm post (target) = 1900 kgDM/ha

select actual post = 1800 kgDM/ha

compare target post and actual post to give difference

as difference exceeds threshold

and post is atomic add post to problem list

specify norm av (target) = 2200 kgDM/ha

select actual av = 2050 kgDM/ha

compare target av and actual av to give difference

as difference exceeds threshold

and av is atomic add av to problem list

specify norm da (target) = 1 ha

select actual da = 1 ha

compare target da and actual da to give difference

difference does not exceed threshold

specify norm PGR (target) = 6 kgDM/ha/day

select actual PGR = 5 kgDM/ha/day

⁴pre = pre grazing pasture cover
 post = post grazing pasture cover
 av = average pasture cover
 da = daily allocation
 PGR = Pasture growth rate

compare target PGR and actual PGR to give difference
as difference exceeds threshold
and PGR is atomic add PGR to problem list

specify norm supplement intake (target) = 3 kgDM/cow/day
select actual supplement intake = 3 kgDM/cow/day
compare target supplement intake and actual supplement intake to give difference
difference within threshold

B3.2 ASSESSMENT

See Figure 5.9

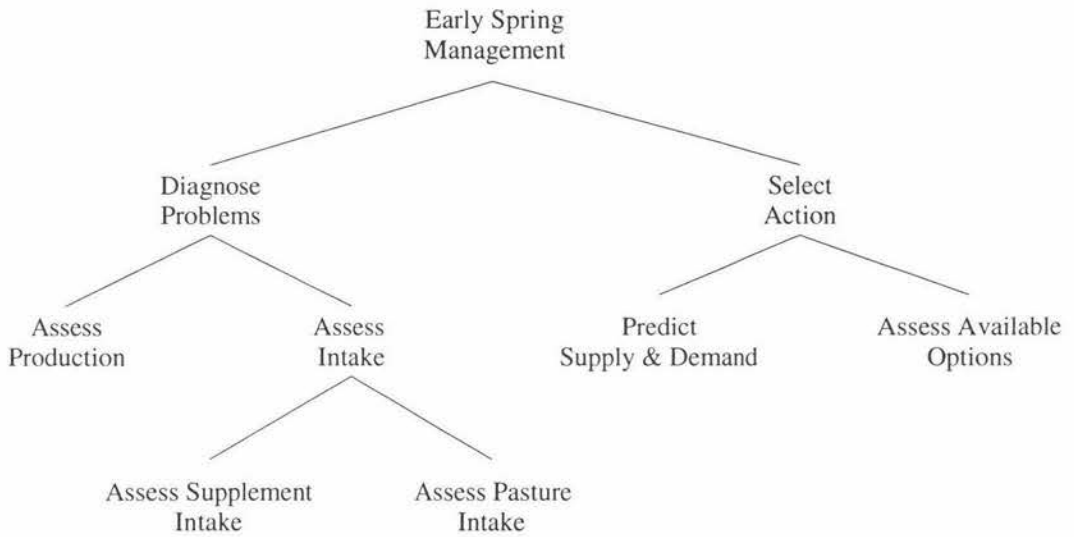
Repeat for each available option in the system model

specify tolerance level for Olsen P from system model = 30
abstract actual Olsen P levels = 24
match actual levels (24) less than tolerance (30) therefore decision = add Olsen P
to feasible options

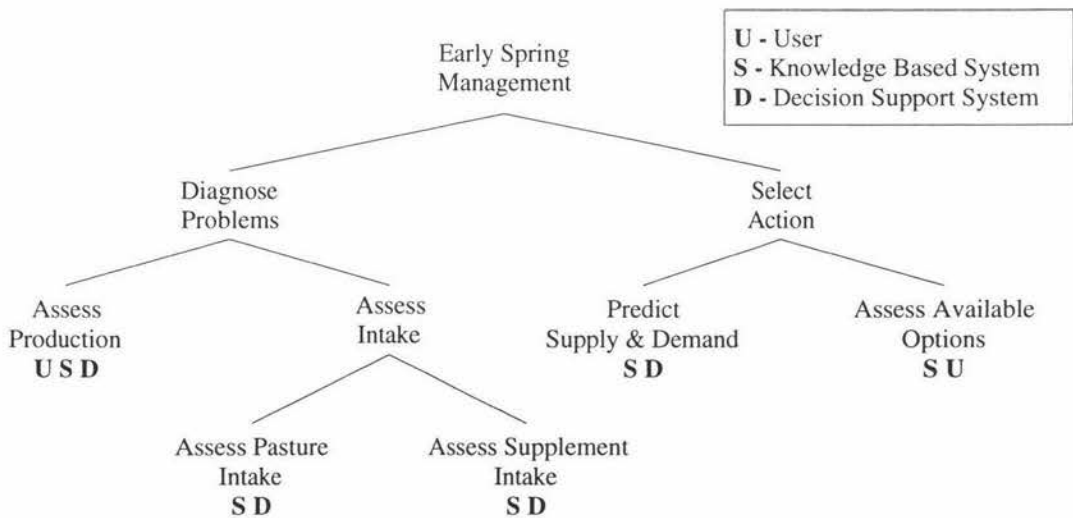
specify tolerance level for Nitrogen from system model = Application within last
two weeks
abstract actual Nitrogen = No recent application
match actual levels (No application) within tolerance (No application) therefore
decision = add apply Nitrogen to feasible options

specify tolerance level for supplementary feed from system model = Supplements
less than 30% intake
abstract actual supplement feed levels = 40% intake
match actual levels (40%) above tolerance (30%) therefore decision = feeding
supplements is not an option

B4 TASK MODEL



B5 MODEL OF COOPERATION



B5.1 SPECIFICATION OF FUNCTIONALITY

Assess Production

Agents:

User

FarmTracker

Intelligent Component

Functionality:

Intelligent Component reads production data from ASCII file produced by *FarmTracker*

Intelligent Component accepts unit and week for evaluation of milk production from user in order to diagnose problems with milk production

Assess Pasture Intake

Agents:

Intelligent Component

Farmtracker

Functionality:

Intelligent Component reads intake variables data from ASCII file produced by FarmTracker to diagnose problems with pasture intake

Assess Supplement Intake

Agents:

Intelligent Component

Farmtracker

Functionality:

Intelligent Component reads intake variables data from ASCII file produced by FarmTracker to diagnose problems with pasture intake

Predict Supply & Demand

Agents:

Intelligent Component

Farmtracker

Functionality:

Intelligent Component uses simplified feed budget to predict pasture cover using data from ASCII file produced by FarmTracker

Assess Available Options

Agents:

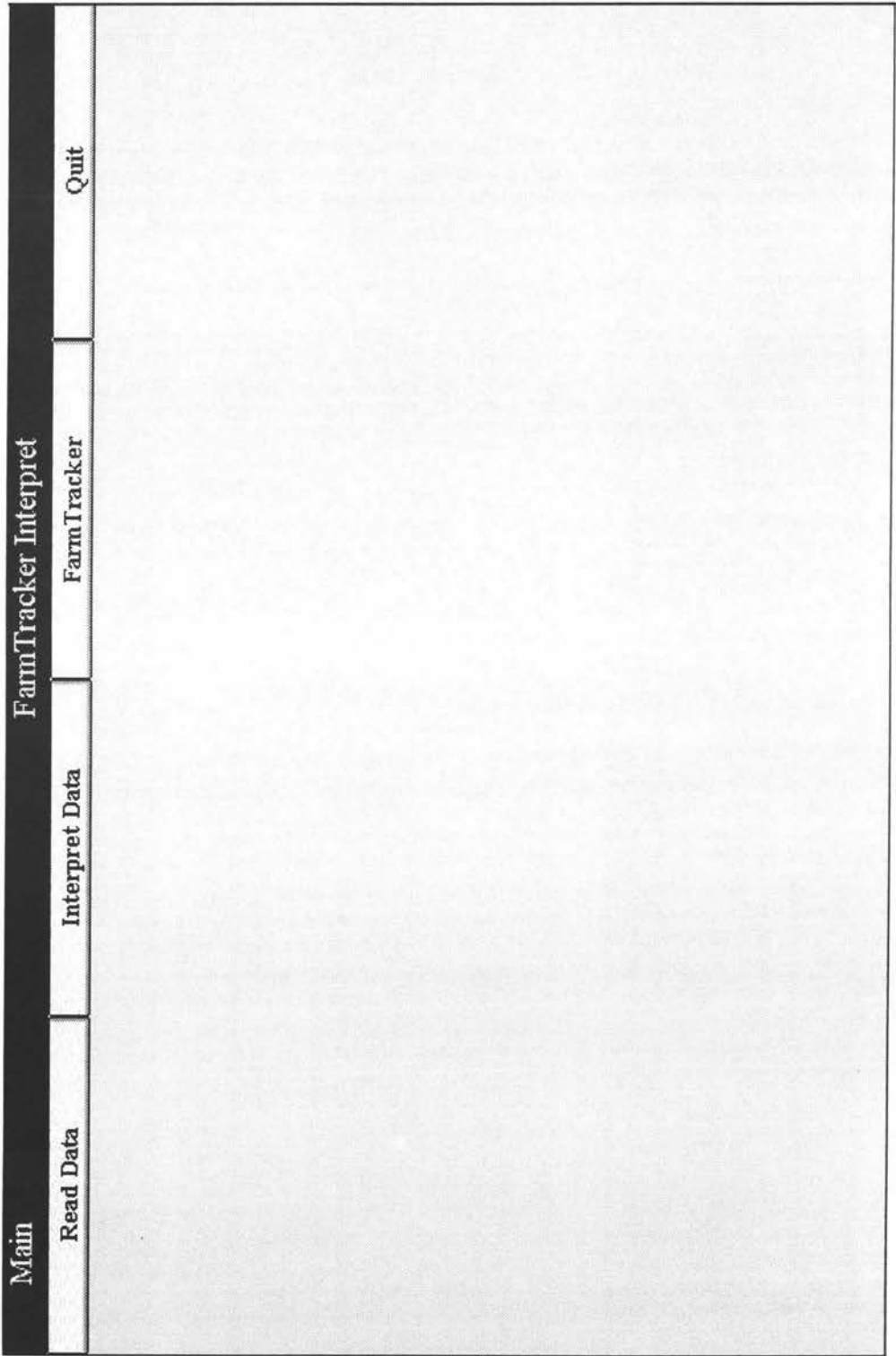
Intelligent Component

User

Functionality:

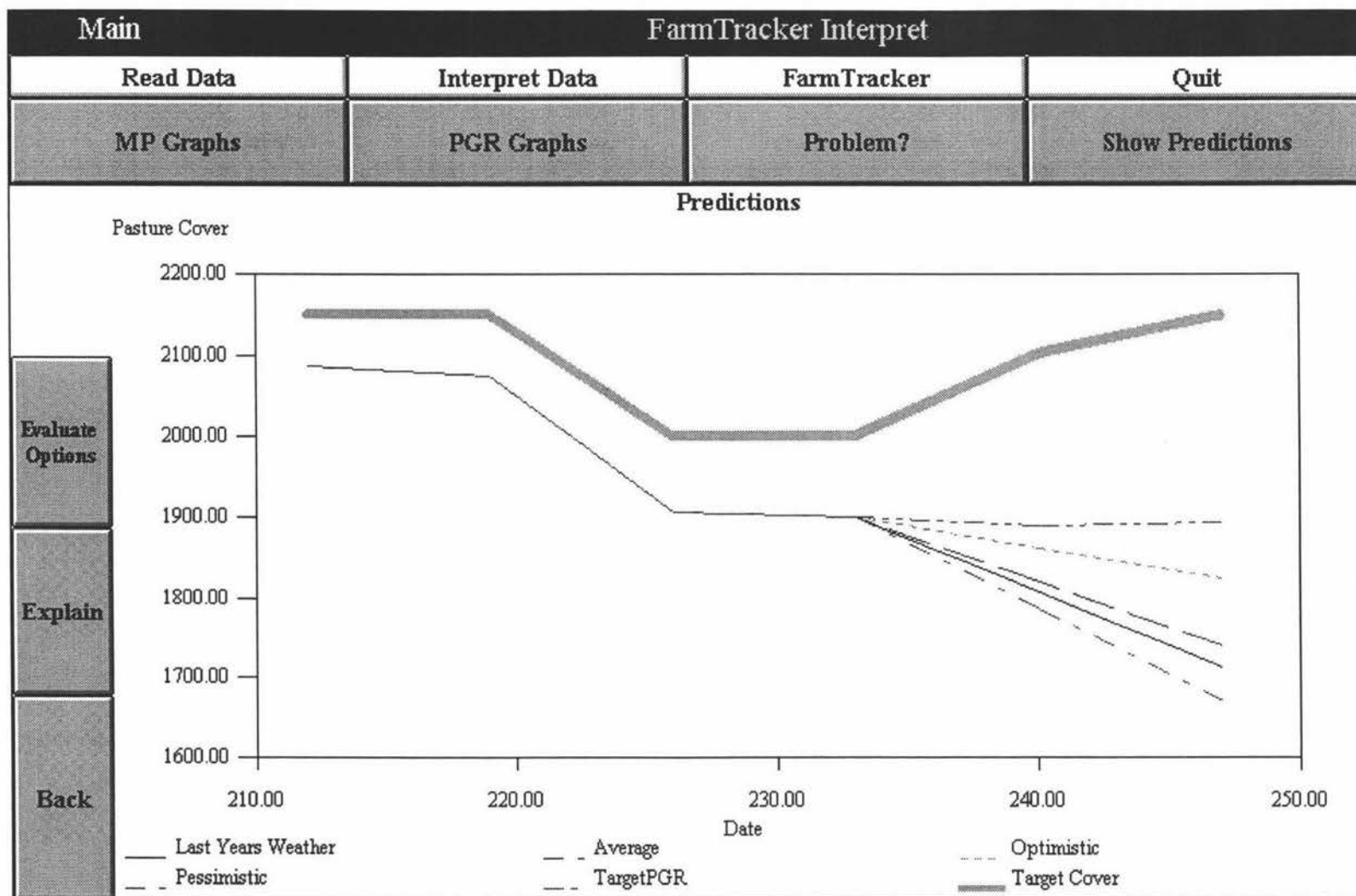
Intelligent Component assesses feasible courses of action given the state of the farm and presents them to the user

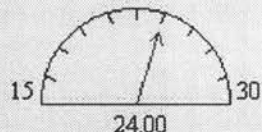
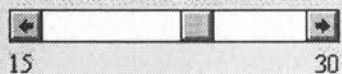
C1 SCREEN SHOTS



Introductory Screen

Screen Shot 2 - Show Various Predictions Screen



Main		FarmTracker Interpret	
Read Data		Interpret Data	
MP Graphs		PGR Graphs	
Problem?		Show Predictions	
The following windows contain suitable options, each may be adjusted and a prediction using these values can be calculated.			
Reset Values	Choose Weather averagew	Cow Numbers 50 102.00 120	Amount of N 30 kgN/ha
			Response Rate 5:1
Run Pred	 <p>24.00 Actual Farm Olsen P</p>		Available Supplements Pasture_Silage
Back	<p>Choose New OlsenP level</p>  <p>24.00</p>		<p>Apply at a rate of</p> <p>4 kgDM/cow/day</p>

C2 RULES

```

/*****
**** RULE: TSDown
*****/
MakeRule( TSDown, [],
  ( GetNthElem( Global:actualfileweekly:PastOnlyTS,
    GetElemPos( Global:targetfileweekly:Date, Problem:Date ) )
    < ( GetNthElem( Global:targetfileweekly:TS,
      GetElemPos( Global:targetfileweekly:Date, Problem:Date ) )
        - 0.05 ) ) And ( Problem:Units #= TotalSolids ),
  {
    PostMessage( "Total milk solids is down on target" );
    Problem:MP = Down;
  } );

/*****
**** RULE: MFDown
*****/
MakeRule( MFDown, [],
  ( GetNthElem( Global:actualfileweekly:PastOnlyMF,
    GetElemPos( Global:actualfileweekly:Date, Problem:Date ) )
    < ( GetNthElem( Global:targetfileweekly:MF,
      GetElemPos( Global:targetfileweekly:Date, Problem:Date ) )
        - 0.025 ) ) And ( Problem:Units #= MilkFat ),
  {
    PostMessage( "Total milk fat is down on target" );
    Problem:MP = Down;
  } );

/*****
**** RULE: SuppIntakeGood
*****/
MakeRule( SuppIntakeGood, [],
  ( GetNthElem( Global:actualfileweekly:SuppIntake,
    GetElemPos( Global:targetfileweekly:Date, Problem:Date ) )
    >= ( GetNthElem( Global:targetfileweekly:SuppIntake,
      GetElemPos( Global:targetfileweekly:Date, Problem:Date ) )
        - 1 ) ) And ( Problem:MP #= Down ),
  {
    Problem:SuppIntake = Good;
  } );

/*****
**** RULE: PreDown
*****/
MakeRule( PreDown, [],
  ( GetNthElem( Global:actualfileweekly:Pre, GetElemPos( Global:targetfileweekly:Date,
    Problem:Date ) )
    < ( GetNthElem( Global:targetfileweekly:Pre,
      GetElemPos( Global:targetfileweekly:Date, Problem:Date ) )
        - 50 ) ) And Problem:PasIntake,
  {
    PostMessage( "Pre grazing cover is down on target" );
    Problem:Pre = Down;
  } );

```

```

/*****
**** RULE: PostDown
*****/
MakeRule( PostDown, [],
  ( GetNthElem( Global:actualfileweekly:Post, GetElemPos( Global:targetfileweekly:Date,
    Problem:Date ) )
    < ( GetNthElem( Global:targetfileweekly:Post,
      GetElemPos( Global:targetfileweekly:Date, Problem:Date ) )
        - 50 ) ) And Problem:PasIntake,
  {
    PostMessage( "Post grazing cover is down on target" );
    Problem:Post = Down;
  } );

/*****
**** RULE: DADown
*****/
MakeRule( DADown, [],
  ( GetNthElem( Global:actualfileweekly:DA, GetElemPos( Global:targetfileweekly:Date,
    Problem:Date ) )
    < GetNthElem( Global:targetfileweekly:DA,
      GetElemPos( Global:targetfileweekly:Date, Problem:Date ) ) )
    And Problem:PasIntake,
  {
    Problem:DA = Down;
  } );

/*****
**** RULE: PGRDown
*****/
MakeRule( PGRDown, [],
  ( GetNthElem( Global:actualfileweekly:PGR, GetElemPos( Global:targetfileweekly:Date,
    Problem:Date ) )
    < ( GetNthElem( Global:targetfileweekly:PGR,
      GetElemPos( Global:targetfileweekly:Date, Problem:Date ) )
        - 2 ) ) And Problem:PasIntake,
  {
    PostMessage( "PGR's are less than target" );
    Problem:PGR = Down;
  } );

/*****
**** RULE: InvPastWastage
*****/
MakeRule( InvPastWastage, [],
  Problem:TotIntake #= Good,
  InvestigateWastage( ) );
SetRulePriority( InvPastWastage, 10 );

/*****
**** RULE: InvDA
*****/
MakeRule( InvDA, [],
  Problem:DA #= Down,
  PostMessage( "DA is down on target" ) );

/*****
**** RULE: MPGood
*****/

```



```

*****/
MakeRule( MPGood, [],
  ( ( GetNthElem( Global:actualfileweekly:PastOnlyTS,
    GetElemPos( Global:targetfileweekly:Date, Problem:Date ) )
    >= ( GetNthElem( Global:targetfileweekly:TS,
      GetElemPos( Global:targetfileweekly:Date,
        Problem:Date ) ) - 0.05 ) )
    And ( Problem:Units #= TotalSolids ) ) Or ( ( GetNthElem( Global:actualfileweekly:PastOnlyMF,
      GetElemPos( Global:targetfileweekly:Date,
        Problem:Date ) )
      >=
      ( GetNthElem( Global:targetfileweekly:MF,
        GetElemPos( Global:targetfileweekly:Date,
          Problem:Date ) )
        -
        0.025 ) )
      And
      ( Problem:Units
        #=
        MilkFat ) ),
    Problem:MP = Good );

*****/
**** RULE: InvestigateRange
*****/
MakeRule( InvestigateRange, [],
  Problem:MP #= Good,
  InvestigateRange( ) );

*****/
**** RULE: ReqFertGROW
*****/
MakeRule( ReqFertGROW, [],
  Problem:PGR = Down,
  {
    ReqFertGROW( Problem );
  } );

*****/
**** RULE: DownOnPrediction
*****/
MakeRule( DownOnPrediction, [],
  ( Problem:Predict #= Done ) And ( GetNthElem( Global:actualfileweekly:PGR,
    GetElemPos( Global:targetfileweekly:Date,
      Problem:Date ) )
    < ( GetNthElem( Global:pred:PGR,
      GetElemPos( Global:pred:Date,
        Global:z ) )
      - 2 ) ),
    Problem:PGRPred = Down );

*****/
**** RULE: InvPred
*****/
MakeRule( InvPred, [],
  Problem:PGRPred #= Down,
  InvPred( ) );

```

```

/*****
**** RULE: GoodcfPrediction
*****/
MakeRule( GoodcfPrediction, [],
  ( Problem:Predict #!= Done ) And ( GetNthElem( Global:actualfileweekly:PGR,
    GetElemPos( Global:targetfileweekly:Date,
      Problem:Date ) )
    >= ( GetNthElem( Global:pred:PGR,
      GetElemPos( Global:pred:Date,
        Global:z ) )
      - 2 ) ),
  Problem:PGRPRED = Good );

/*****
**** RULE: BlameWeatherFert
*****/
MakeRule( BlameWeatherFert, [],
  Problem:PGRPRED #!= Good,
  PostMessage( "The PGR's were to be expected given the weather conditions and fertility for the
  problem period, however they were below target" ) );

/*****
**** RULE: SuppIntakeDown
*****/
MakeRule( SuppIntakeDown, [],
  ( GetNthElem( Global:actualfileweekly:SuppIntake,
    GetElemPos( Global:targetfileweekly:Date, Problem:Date ) )
    < ( GetNthElem( Global:targetfileweekly:SuppIntake, GetElemPos( Global:targetfileweekly:Date,
      Problem:Date ) )
      - 1 ) ) And ( Problem:MP #!= Down ),
  {
    Problem:SuppIntake = Down;
  } );

/*****
**** RULE: PasIntakeGood
*****/
MakeRule( PasIntakeGood, [],
  ( GetNthElem( Global:actualfileweekly:PasIntake,
    GetElemPos( Global:targetfileweekly:Date, Problem:Date ) )
    >= ( GetNthElem( Global:targetfileweekly:PasIntake, GetElemPos( Global:targetfileweekly:Date,
      Problem:Date ) )
      - 1 ) ) And ( Problem:MP #!= Down ),
  {
    Problem:PasIntake = Good;
  } );

/*****
**** RULE: InvSupp
*****/
MakeRule( InvSupp, [],
  Problem:SuppIntake #!= Down,
  InvSupp( ) );

/*****
**** RULE: PasIntakeDown
*****/
MakeRule( PasIntakeDown, [],

```

```

( GetNthElem( Global:actualfileweekly:PasIntake,
  GetElemPos( Global:targetfileweekly:Date, Problem:Date ) )
  < ( GetNthElem( Global:targetfileweekly:PasIntake, GetElemPos( Global:targetfileweekly:Date,
    Problem:Date ) )
    - 1 ) ) And ( Problem:MP #≠ Down ),
{
PostMessage( "The Pasture Intake is Down on Target" );
Problem:PasIntake = Down;
} );

/*****
**** RULE: AvDown
*****/
MakeRule( AvDown, [],
  ( GetNthElem( Global:actualfileweekly:Av, GetElemPos( Global:targetfileweekly:Date,
    Problem:Date ) )
    < ( GetNthElem( Global:targetfileweekly:Av,
      GetElemPos( Global:targetfileweekly:Date, Problem:Date ) )
        - 50 ) ) And Problem:PasIntake,
{
PostMessage( "Average cover is down on target" );
Problem:Av = Down;
} );

/*****
**** RULE: PGRDate
*****/
MakeRule( PGRDate, [],
  ( Problem:Predict #≠ Done ) And Not( Member?( Global:pred:Date,
    Problem:Date ) ),
  FindPGRDate( ) );
SetRulePriority( PGRDate, 10 );

/*****
**** RULE: OptionFert
*****/
MakeRule( OptionFert, [],
  Global:farmfile:OlsenP < 30,
  Problem:OptionFert = Y );

/*****
**** RULE: NotOptionFert
*****/
MakeRule( NotOptionFert, [],
  Global:farmfile:OlsenP >= 30,
  Problem:OptionFert = N );

/*****
**** RULE: OptionSupp
*****/
MakeRule( OptionSupp, [],
  GetNthElem( Global:actualfileweekly:SuppIntake,
    GetElemPos( Global:targetfileweekly:Date, Problem:Date ) )
    / GetNthElem( Global:actualfileweekly:PasIntake,
      GetElemPos( Global:targetfileweekly:Date, Problem:Date ) )
    <= 0.3,
  Problem:OptionSupp = Y );

```

```

/*****
*** RULE: NotOptionSupp
*****/
MakeRule( NotOptionSupp, [],
  GetNthElem( Global:actualfileweekly:SuppIntake,
    GetElemPos( Global:targetfileweekly:Date, Problem:Date ) )
  / GetNthElem( Global:actualfileweekly:PasIntake,
    GetElemPos( Global:targetfileweekly:Date, Problem:Date ) )
  > 0.3,
  Problem:OptionSupp = N );

/*****
*** RULE: NPred
*****/
MakeRule( NPred, [],
  NApp:yn #= No And Problem:ComingRight #= N,
  {
    Solution:N = "Add Nitrogen";
  } );

/*****
*** RULE: FertPred
*****/
MakeRule( FertPred, [],
  Problem:OptionFert #= Y And Problem:ComingRight #= N,
  {
    Solution:Fert = "Apply Fertiliser";
  } );

/*****
*** RULE: SuppPred
*****/
MakeRule( SuppPred, [],
  Problem:OptionSupp #= Y And Problem:ComingRight #= N,
  {
    Solution:Supps = "Feed Supplements";
  } );

/*****
*** RULE: NotSuppPred
*** Feeding Supps is not an option as already more than 30% of intake is supplements
*****/
MakeRule( NotSuppPred, [],
  Problem:OptionSupp #= N And Problem:PGR #= Down,
  {
    ResetValue( Solution:Supps );
  } );
SetRuleComment( NotSuppPred, "Feeding Supps is not an option as already more than 30% of intake is
supplements" );

/*****
*** RULE: NotNPred
*** Addition of Nitrogen is not an option because some has been applied in the last fortnight
*****/
MakeRule( NotNPred, [],
  NApp:yn #= Yes And Problem:ComingRight #= N,
  {
    ResetValue( Solution:N );
  } );

```

```
});
SetRuleComment( NotNPred, "Addition of Nitrogen is not an option because some has been applied in
the last fortnight" );
```

```
/******
**** RULE: NotComingRight
*****/

MakeRule( NotComingRight, [],
  GetNthElem( Predict:Average, GetElemPos( Predict:Date,
    Problem:Date + 14 ) )
    < GetNthElem( Predict:Target, GetElemPos( Predict:Date,
    Problem:Date + 14 ) ),
  Problem:ComingRight = N );

/******
**** RULE: ComingRight
*****/

MakeRule( ComingRight, [],
  GetNthElem( Predict:Average, GetElemPos( Predict:Date,
    Problem:Date + 14 ) )
    >= GetNthElem( Predict:Target, GetElemPos( Predict:Date,
    Problem:Date + 14 ) ),
  {
    PostMessage( "Will be on target in 2 weeks with average weather so suggest no action" );
    Problem:ComingRight = Y;
  } );
```

```
/******
**** GOAL: Solution
*****/

MakeGoal( Solution,
  {
    Solution:Fert;
    Solution:N;
    Solution:Supps;
  } );
```

C2.1 RULE GROUPS

FORWARD CHAINING RULES

```
MakeSlot( Problem:FwdRules );
SetSlotOption( Problem:FwdRules, MULTIPLE );
SetValue( Problem:FwdRules, AvDown, BlameWeatherFert, DADown, DownOnPrediction,
GoodecfPrediction, InvDA, InvestigateRange, InvPastWastage, InvPred, InvSupp, MFDown, MPGood,
PasIntakeDown, PasIntakeGood, PGRDate, PGRDown, PostDown, PreDown, ReqFertGROW,
SuppIntakeDown, SuppIntakeGood, TSDown );
```

BACKWARD CHAINING RULES

```
MakeSlot( Problem:BwdRules );
SetSlotOption( Problem:BwdRules, MULTIPLE );
SetValue( Problem:BwdRules, NotOptionFert, NotOptionSupp, OptionFert, FertPred, NPred, SuppPred,
OptionSupp, NotSuppPred, NotNPred, ComingRight, NotComingRight );
```

C3 SELECTED FUNCTIONS

C3.1 DIAGNOSIS

```

/*****
**** FUNCTION: Diagnosis
*****/
MakeFunction( Diagnosis, [],
{
SetForwardChainMode (BREADTHFIRST, IGNORE);
ResetValue(Problem:Units);
ResetValue(Problem:Date);
ResetValue(Problem:PGR);
ResetValue(Problem:Av);
ResetValue(Problem:DA);
ResetValue(Problem:MP);
ResetValue(Problem:PasIntake);
ResetValue(Problem:PGRPred);
ResetValue(Problem:Post);
ResetValue(Problem:Pre);
ResetValue(Problem:Predict);
ResetValue(Problem:SuppIntake);
ResetValue(Problem:TotNum);

SetPostMessageTitle (" ");

Assert (Problem:Units);
ForwardChain(NULL, Problem:FwdRules);
RunPreds ();
} );

```

C3.2 ASSESSMENT

```

/*****
**** FUNCTION: StartBChain
*****/
MakeFunction( StartBChain, [],
{
CheckNApp ();

ResetValue(Solution, Fert);
ResetValue(Solution, Supps);
ResetValue(Solution, N);
ResetValue(Problem, OptionFert);
ResetValue(Problem, OptionSupp);
ResetValue(Problem, ComingRight);
ResetValue(NApp:ChoiceAmt);
ResetValue(NApp:ChoiceResp);
ComboPrediction:SuppAmt = 0;

ClearList(Solution:List);

BackwardChain([NOASK], Solution, Problem:BwdRules);

```

```

If Not (Null?(Solution:Fert)) Then AppendToList(Solution:List, Solution:Fert);
If Not (Null?(Solution:Supps)) Then AppendToList(Solution:List, Solution:Supps);
If Not (Null?(Solution:N)) Then AppendToList(Solution:List, Solution:N);

If (LengthList(Solution:List) > 0) Then
{
ComboPrediction:TotNum = (GetNthElem(Global:actualfileweekly:TotNum,
GetElemPos(Global:actualfileweekly:Date, Problem:Date)));
ComboPrediction:OlsenP = Global:farmfile:OlsenP;
ComboPrediction:Weather = Global:AvWeather;
ShowWindow(Session11);
ShowWindow(Session12);
ShowWindow(Session13);
ShowWindow(Session14);
HideWindow(Session5);

If Member?(Solution:List, "Add Nitrogen" ) Then ShowWindow(Session8);
If Member?(Solution:List, "Apply Fertiliser" ) Then ShowWindow(Session9);
If Member?(Solution:List, "Feed Supplements" ) Then ShowWindow(Session10);
};
} );

```

C3.3 USE OF THE GROW MODEL

```

/*****
**** FUNCTION: CreateGrowIn
*****/
MakeFunction( CreateGrowIn, [wf OlsenP name],
{
OpenWriteFile ("c:\ft\grow.txt");
WriteLine ("00," "KappaPrediction," "0," "0," "0," "Yes," "24");
Write ("01");
EnumList(wf:Rainfall, rain, (Write(" ", rain))) ;
WriteLine();
WriteLine ("02,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0");
Write ("03");
EnumList(wf:Temp, temp, (Write(" ", temp))) ;
WriteLine();
Write ("04");
EnumList(wf:ET, et, (Write(" ", et))) ;

WriteLine();
WriteLine("05,0,0,0");
WriteLine("06,1,Flat,Sunny," Global:farmfile:SoilType, " ", "14," Global:farmfile:SoilType, " ",
Global:farmfile:SoilDepth, " ", "120," Global:farmfile:WHC, " ", OlsenP);
WriteLine("07,1,Ryegrass,HF,90,Whiteclover,WC,10");
WriteLine("99");
CloseWriteFile();

} );

/*****
**** FUNCTION: ExecuteGROW
*****/
MakeFunction( ExecuteGROW, [],
{

```



```

Execute("c:\ft\grow.bat");

For i From 1 To 800 Do
{
  WaitForInput();
};
Wait(10);
} );

/*****
**** FUNCTION: ReadGrowOut
*****/
MakeFunction( ReadGrowOut, [name],
{
  OpenReadFile("c:\ft\grow.out");
  Global:x = ReadWord();
  While (Global:x != 54)
  {
    ReadLine();
    Global:x = ReadWord();
  };
  ReadWord();
  Global:pred = Pred # name;
  ReadWord();
  If Instance?(Global:pred) Then
    DeleteInstance (Global:pred);

  MakeInstance((Global:pred), GROWPrediction);

  For i From 1 To 36 Do
  {
    ReadWord();
    AppendToList (Global:pred:PGR, ReadWord());
  };
  CloseReadFile();
} );

```

C3.4 SIMULATED FEED BUDGET

```

/*****
**** FUNCTION: ComboPrediction
*****/
MakeFunction( ComboPrediction, [],
{

/*From the week being investigated, the cover is calculated weekly by using target intake and pgr's for
each week. Updated SR's and intakes due to supps are included*/

ClearList (Predict:Variable);
ClearList (Predict:Date);

Global:x = 1;
Global:y = GetNthElem(Global:actualfileweekly:Date, 1);

AppendToList(Predict:Date, Global:y);

```

```

AppendToList(Predict:Variable, (GetNthElem(Global:actualfileweekly:Av,
                                           GetElemPos(Global:targetfileweekly:Date, Global:y))));

While (Global:y < Problem:Date)
{
  Global:x += 1;
  Global:y = GetNthElem(Global:actualfileweekly:Date, Global:x);
  AppendToList(Predict:Date, Global:y);
  AppendToList(Predict:Variable, (GetNthElem(Global:actualfileweekly:Av,
                                           GetElemPos(Global:targetfileweekly:Date, Global:y))));
};

Global:x = Problem:Date;

Predict:x =GetNthElem(Global:targetfileweekly:PasIntake, GetElemPos(Global:targetfileweekly:Date,
Global:x));
Predict:z = GetNthElem(Global:actualfileweekly:Area, GetElemPos (Global:actualfileweekly:Date,
Global:x));

/*      Intake per ha = Pasture Intake per cow * number of cows / area      */

Global:y = ( Floor ( (Predict:x - ComboPrediction:SuppAmt*0.25) * (ComboPrediction:TotNum) /
Predict:z *10) ) /10;

If Global:y < 0 Then Global:y=0;

For a From 1 To 2 Do
{
  Global:z = Global:x;
  While (Not (Member?(Global:pred:Date, Global:z)))
    Global:z -= 1;

  PredictCover(GetNthElem(Predict:Variable, GetElemPos(Predict:Date, Global:x)),
              (Global:y),
              (GetNthElem(PredVar:PGR, GetElemPos(PredVar:Date, Global:z))));
  AppendToList (Predict:Variable, PredictForwardCover:FinalCover);

  Global:x += 7;
  AppendToList (Predict:Date, Global:x);
};
});

/*****
**** FUNCTION: PredictCover
*****/
MakeFunction( PredictCover, [cc ti pgr],
{
  PredictForwardCover:FinalCover = (cc - (ti*7) + (pgr*7));
});

```

C3.5 AVERAGE WEEKLY DATA TO MATCH TARGETS

```

/*****
**** FUNCTION: CalcWeeklyAv

```

```

*****/
MakeFunction( CalcWeeklyAv, [z],
{
Global:x = 1;
Global:y = 1;
ClearList (Global:actualfileweekly:Date);

While (GetNthElem (Global:targetfileweekly:Date, Global:y) < GetNthElem (Global:actualfile:Date,
LengthList(Global:actualfile:Date)))
{
While (GetNthElem (Global:actualfile:Date, Global:x) != GetNthElem (Global:targetfileweekly:Date,
Global:y))
{
Global:x += 1;
};
AppendToList (Global:actualfileweekly:Date, (GetNthElem (Global:targetfileweekly:Date,
Global:y)));
Global:y += 1;
Global:count = 0;
Global:sum = 0;
While (GetNthElem (Global:actualfile:Date, Global:x) < GetNthElem (Global:targetfileweekly:Date,
Global:y))
{
Global:sum += (GetNthElem (Global:actualfile:z, Global:x));
Global:x += 1;
Global:count += 1;
};
AppendToList (Global:actualfileweekly:z, (Global:sum/Global:count));
};

AppendToList (Global:actualfileweekly:Date, GetNthElem (Global:targetfileweekly:Date, Global:y));
Global:sum = 0;
Global:count = 0;
While (GetNthElem (Global:actualfile:z, Global:x) != GetNthElem (Global:actualfile:z,
LengthList(Global:actualfile:z)))
{
Global:sum += (GetNthElem (Global:actualfile:z, Global:x));
Global:x += 1;
Global:count += 1;
};
If (Global:count = 0) Then
{Global:count = 1;
Global:sum = ((GetNthElem (Global:actualfile:z, LengthList(Global:actualfile:z))));
};
AppendToList (Global:actualfileweekly:z, (Global:sum/Global:count));
});

```