The Role of Functional Prototyping within the KADS methodology.

A thesis presented in partial fulfilment of the requirements for the degree of Master of Science in Computer Science at Massey University.

Damian Pacitto
January, 1995.

Abstract

Knowledge-based systems have until recent times lacked a clear and complete methodology for their construction. KADS was the result of the early 1980's project (ESPRIT-I P1098) which had the aim of developing a comprehensive, commercially viable methodology for knowledge-based system construction. KADS has subsequently proved to be one of the more popular approaches, focusing on the modelling approach to knowledge based system development.

One area of the KADS methodology that has not been examined to any great depth is that of model validation. Model validation is the process of ensuring that a derived model is an accurate representation of the domain from which it has been derived from. The two approaches which have been suggested for this purpose within the KADS framework are that of protocol analysis and functional prototyping.

This project seeks to apply the second of these choices, that of functional prototyping, to the model of expertise created by da Silva (1994) for model validation purposes. The problem domain is that of farm management, under an joint program of research between the Computer Science, Information Systems and Agricultural Management departments of Massey University. The project took the model of expertise and created a knowledge representation model in compliance with the selected object-oriented paradigm. After this the creation of a functional prototype in a Microsoft Windows based PC environment took place, using Kappa-PC as the application development tool.

The validation took place through a demonstration session to a number of domain experts. Conclusions drawn from the experience gained through the creation and use of the prototype are presented, outlining the reasons why functional prototyping was deemed to be an appropriate method for model validation.

Acknowledgements

Thanks to my supervisors Liz Kemp and Lis Todd for their patience, ideas, faith and enthusiasm. Their support made the year a lot easier to bear.

Thanks to Dave, Shamus, Paul, Mike, Sarah and Fiona for keeping my sanity for me over the last few years in the research lab. Thanks to all the others who have drifted in and out to add some colour to the time.

Table of Contents

List of Figures

List of Tables

Chapter 1: Introduction

Historically, prototyping has been associated with the experimental approach adopted by many AI researchers, known as 'rapid prototyping'. With improvements in technology, it has become easier for rapid prototypes to implement new ideas rather than evaluate these ideas beforehand. The KADS project (Wielinga et al, 1986) was envisaged to propose a methodology for knowledge base development. KADS represents an alternative to the ad-hoc style. However, KADS also recognises the potential for the successful use of prototyping during the analysis and design phases. Since functional prototyping provides a simulator of the problem solving aspects of the artefact, it can be used for model validation. The other method proposed within the KADS methodology for this purpose is protocol analysis (Wielinga et al, 1991).

Objectives

The principle goal of this project is to explore the role of functional prototyping within the KADS methodology. The first objective of the study is to construct a suitable knowledge representation model of the already existing model of expertise (collection of domain primitives). This model of expertise was generated from work by da Silva (1994). In contrast with the model of expertise, a paradigm for the knowledge representation model must be selected. This is still environment independent, but provides a format from which the knowledge contained in the model of expertise can be expressed in an application.

The second objective is to examine the role of functional prototyping in model validation. Model validation is the process of ensuring there is an accurate representation of the domain for problem solving purposes. During this project, a functional prototype based on a model of expertise will be constructed for model validation purposes, the validation being performed by domain experts. From the knowledge representation model the prototype can be constructed. The domain this approach will be applied to is that of dairy farming.

Research Methodology

Nunamaker (1991, p. 91) states that any "research methodology consists of a combination of the process, methods and tools that are used in conducting research in a research domain". The research domain itself is the subject matter under study in the project. The interaction between the research domain and the research methodology is shown in Figure 1.

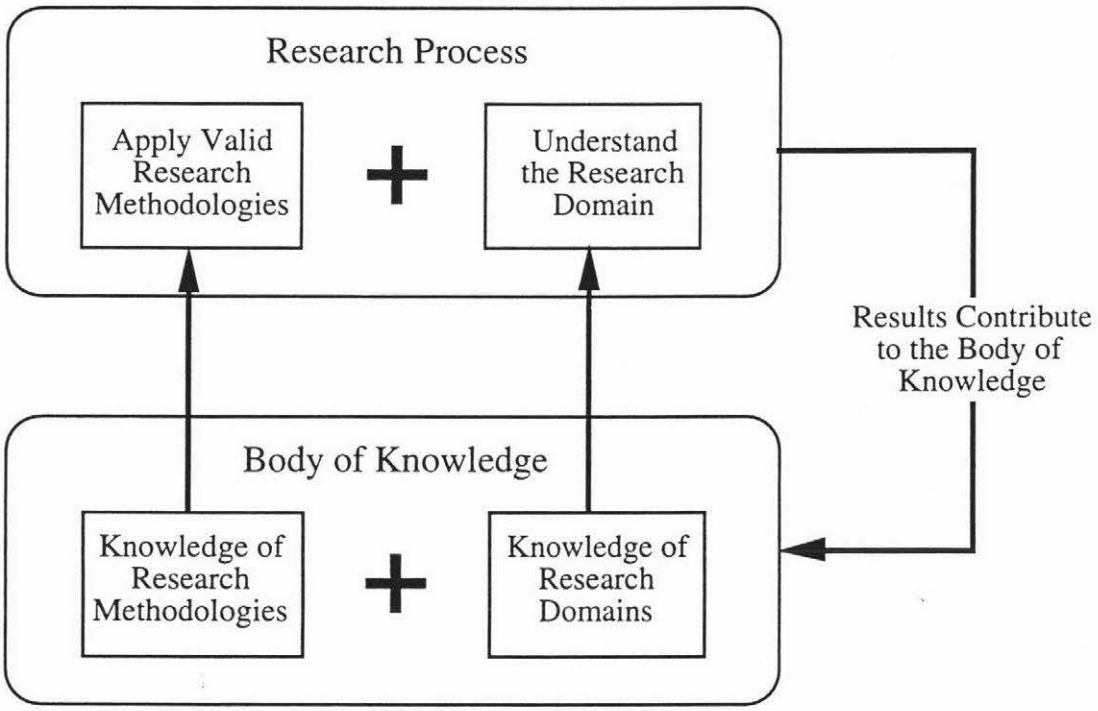


Figure 1- Research Domain and Research Methodology Interaction
(from Nunamaker, 1991)

The body of knowledge contains both the research domains and the research methodologies. Using a correct methodology on an suitable appropriate domain results in constructive research, which is fed back into the body of knowledge. This 'feeding back' process allows for technological advancement to be made.

The research domain for this project is that of the KADS methodology, specifically applied to the area of functional prototyping for model validation.

Research can be defined as a systematic, intensive study directed towards fuller scientific knowledge in the area studied (Blake, 1978). This definition can be applied to the goal of research in computer science. Nunamaker (1991) proposes that research in computer related fields follows a generic pattern of

- Observing the **problem**
- Constructing a **hypothesis**
- Confirming the hypothesis through **analysis**
- Using the results of analysis as an **argument**

From this viewpoint, problems existing in the research domain are encountered by observation. A hypothesis is formed and an attempt is made to confirm this through an analysis. Nunamaker (1991) offers a number of different possibilities for the analysis, including formal proofs, developed systems and opinion surveys.

The KADS methodology indicates the potential role of functional prototyping for model validation (Wielinga et al, 1991). However, there is a distinct lack of details of the application of this approach in published research. The effectiveness of functional prototyping is, therefore, unclear. This project attempts to remove some of the mystery shrouding the usage of functional prototyping.

In order to attempt to resolve this problem, a hypothesis has been proposed for examination: that a functional prototype can be used as an effective method for model validation of the model of expertise.

This hypothesis will be examined through the use of an analysis technique as defined by Nunamaker. For this project, analysis will be undertaken through the construction of a functional prototype and the evaluation of the completed system.

Nunamaker proposes that the information generated from the analysis phase of the research methodology can be used as an argument with reference to the hypothesis. The hypothesis of the use of functional prototyping may be supported or otherwise.

It can be seen from these steps that building a system of itself does not constitute a research project. However, Nunamaker's view permits system development to be an acceptable component of the proof required for a hypothesis, where proof is taken to be any convincing argument in support of a worthwhile hypothesis. Therefore, system development can be thought of as "proof-by-demonstration".

Application of methodology to functional prototyping

From the completed intermediate model, in this case the KADS model of expertise, a functional prototype can be created. The next step is for a knowledge representation model to be constructed. This is a description of the objects and operations that are present in the system following the object-oriented paradigm. As the knowledge representation model does not contain all the information needed for implementation of a system, external factors such as hardware and software features will be examined before the functional prototype is implemented. The functional prototype requires the problem solving knowledge contained within the task layer of the model of expertise, so there is also a link from the model of expertise to the functional prototype model. This methodology is pictorially represented in Figure 2.

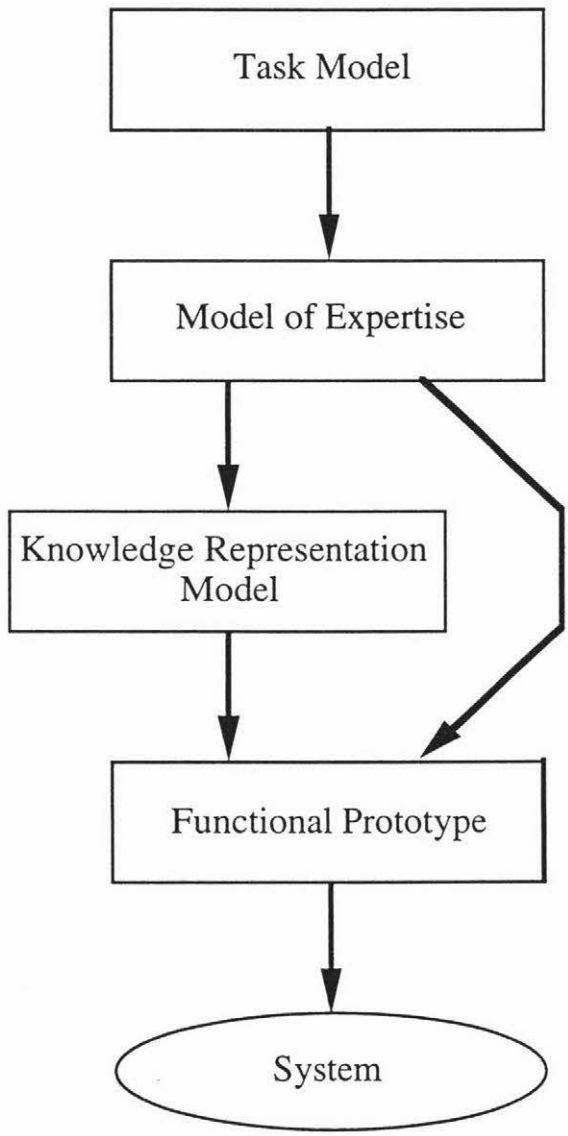


Figure 2 - Intermediate model development phases

Thesis Structure

A literature review was undertaken. This is presented in chapter two, including background as to why the research question has been proposed. The generation of the knowledge representation model is described through the course of chapter three. This chapter also provides an introduction to the problem domain, showing where the research is to be focused. The design decisions required to be made are explained, and a solution paradigm chosen. Implementation details are presented in the fourth chapter, building on the knowledge representation model. Evaluation of the constructed functional prototype occurs in chapter five, summarising the approaches used for testing the prototype. The test results are analysed, and conclusions drawn from the findings presented in chapter six.

Chapter 2: Literature Review

2.1 Introduction

Knowledge-based systems are built to represent and reason with knowledge of some specialist subject with a view to solving problems or giving advice. A brief overview of knowledge-based systems is given in the first section of this chapter. The KADS methodology has been suggested as a complete methodology for building knowledge-based systems. An examination of the major underlying principles of this approach are looked at. The model of expertise in particular is analysed, for a major part of this project is concerned with taking an existing model of expertise and constructing a functional prototype. With this goal in mind, a review of the importance of selecting a suitable knowledge representation form is also examined.

The many different forms of prototyping are also looked at. There has often been a stigma attached to prototyping within the Artificial Intelligence community, and some form of consensus on both the different forms of prototyping and the ways in which they can be examined are sought. Functional prototyping is an aspect of the KADS methodology that is neglected as a tool for model validation. An explanation of the role functional prototyping can take is also looked at.

2.2 Knowledge-Based Systems

Knowledge-based systems differ from conventional software in that they apply rules of thumb to a symbolic representation of knowledge instead of applying more algorithmic or statistical methods (Jackson, 1990). Knowledge-based system technology comes from the research field of Artificial Intelligence, the branch of computer science concerned with designing intelligent computer systems (Barr and Feigenbaum, 1981). Barr and Feigenbaum continue to define an intelligence system as being a system that exhibits the characteristics associated with intelligence in human behaviour, such as understanding language, learning, reasoning and so on. Wielinga et al (1991) propose that a knowledge-based system is not a container simply filled with knowledge extracted from an expert, but an operational model that exhibits some desired behaviour, observed or specified in terms of real-world phenomena. So, constructing a knowledge-based system can be seen as building a computational model of desired behaviour.

Jackson(1990) considers knowledge-based systems to be comprised of two modules, a knowledge base and an inference engine. The knowledge base contains the domain information. The inference engine is the part of the system containing the problem solving knowledge. Akkermans et al (1993) agree that there are two types of knowledge, although they call them application domain knowledge and control knowledge. The application domain knowledge is equivalent to Jackson's knowledge-base, consisting of concepts, relations and facts that are needed to reason about a certain domain application. Control knowledge is associated with the inference engine, describing how reasoning within the application domain is done in terms of reasoning operations on the domain knowledge and in terms of control structures and decompositions of the task as a whole.

2.2.1 Knowledge Representation

Knowledge Representation is concerned with the way in which information might be stored within the human brain, and the way in which knowledge can be formally described for the purpose of symbolic computation (Jackson, 1990). Symbolic computation is defined to be "Non numeric computations in which symbols and symbol structures can be construed as standing for various concepts and relations between them" (Jackson, 1990, pp. 8-9).

The first knowledge representation approach was that of rules. Newell and Simon (1972) generated a form of knowledge representation known as production rules. When applied to the computer domain, this led to a knowledge-base being filled with productions (or rules) in the form **IF** this condition appears, **THEN** do this action.

Rules were considered to be natural as this was the way knowledge acquisition took place, where each rule corresponded to a chunk of knowledge meaningful to the domain expert (Bramer, 1982). At this time Bramer considered rules to be the only structure for knowledge representation.

Semantic networks were originally used by Quillian for expressing knowledge contained within English sentences, by portraying information through a set of nodes connected by a set of labelled directed arcs (Quillian, 1968). These arcs represent the relationships amongst nodes. When applied to the Artificial Intelligence field, nodes can represent objects, concepts or events. Relations can be either is_a relationships, in a hierarchical taxonomy, or has_a relationships, between objects that are made up of a set of components (Duda et al, 1978).

Frames as a knowledge form were derived from semantic nets, but are different in that they are a data structure for representing stereotyped situations (Minsky, 1975). A frame consists of a series of slots, each of which represents a standard property of the element represented by the frame (Biondo, 1990).

The organising principle of knowledge representation in object-oriented systems is that of packaging both data and procedures into structures (objects) related by some form of inheritance mechanism. An object is an encapsulation of state (data values) and behaviour (operations) (Budd, 1991). Subclasses can inherit procedures as well as data from superclasses and such objects can communicate with each other via a special message passing protocol.

Objects have recently been used as an efficient knowledge representation mechanisms in a number of expert system development tools (Mehandjiska and Page, 1993). Objects have a number of features in common with frames:

- Both have an identifier or name.
- The notion of classes and instances applies to both objects and frames.
- Frames have slots with slot names, and these can take a value. Objects have attribute names and values.
- Objects and frame structures permit multiple inheritance.

Differences include:

- Frames posses daemons which are activated each time a slot is accessed, updated or deleted. Objects do not posses a similar form.
- Objects use messages for activating a method.
- A frame permits the attachment of If/Then rule states to slots.
- Pattern matching rules can easily be used without complication in frames.
- Objects can easily incorporate complex constraints, which can be difficult with frames.

Sommerville (1989) says that the object-oriented approach to software design was derived from work on information hiding, abstract data types and object-oriented programming languages. It is important to realise that despite object-oriented languages simplifying the implementation of an object-oriented design, the principle of designing a system as a set of interacting objects is distinct from implementing that system.

Coad and Yourdon (1991) state the advantages of an object-oriented approach to be:

- The emphasis placed on the understanding of problem domains.
- Analysis and specification is undertaken using the methods of organisation which pervade people's thinking.
- Increased internal consistency through attributes and services being considered together.
- Inheritance allowing commonalty to be capitalised upon
- The ability to handle volatile systems, providing stability over changing requirements.
- Reuse of analysis results, for future work.

2.3 KADS Methodology

The area of knowledge-based systems is one that is undergoing an increase in the amount of research undertaken within it. During the early nineteen eighties, there was no universally accepted methodology for the design of knowledge-based systems. Most knowledge-based systems were implemented using rapid prototyping, focusing on implementation aspects and resulting in design decisions being made as the project progressed. An ESPRIT project began what is now known as the KADS project in 1983 (Hayward, 1986). When KADS was conceived, there was little interest in methodological issues in the Artificial Intelligence community, the pervading paradigm for building software being rapid prototyping using special purpose software and hardware (Wielinga et al, 1991). This methodology seeks to provide a methodical approach to system design, emphasising analysis before implementation. Although many other methodologies have been proposed (Karbach et al, 1990), KADS remains as the dominant knowledge-based system design methodology at present (Wetter, 1992). Fundamentally, KADS takes the view that knowledge-based system development is essentially a modelling activity (Hickman et al, 1989). Although originally standing for knowledge acquisition and documentation structuring (Hayward, 1986), KADS is an acronym that has long since lost its original meaning (Breuker, 1993).

2.3.1 Introduction to KADS

There have been a number of different methodologies proposed for development of knowledge-based systems. These include the generic task approach (Bylander and Chandrasekaran, 1987), the KEEP model (Neubert and Studer, 1991), the method-to-task approach (Musen et al, 1988), the role-limiting-method approach (Eshelman, 1988) and the KADS methodology (Breuker et al 1987), (Breuker and Wielinga, 1989).

KADS assumes that knowledge and expertise can be analysed before design and implementation starts. This knowledge is subsequently distinguished through the different roles the knowledge plays in reasoning into domain, inference, task and strategic knowledge. These types of knowledge are organised into layers which have only limited interaction.

The KADS methodology notes that the historical problem with the rapid prototyping approach is that it is inadequate for the systematic development of large knowledge-based systems (Schreiber et al, 1987). This is because the rapid prototyping approaches do not strictly separate the analysis from the design and implementation phases, and developing a knowledge-based system is essentially a modelling activity. KADS involves model driven analysis of knowledge as early as possible, which has the effect of making the process of knowledge-base construction more efficient and proficient.

2.3.2 KADS Models

KADS system development is based on the premise that a number of models, each exhibiting different characteristics and functions, are built to represent the system to be constructed in a structured, logical progression. Each model reflects, through the abstraction of detail, selected characteristics of the empirical system in the real world that it stands for (DeMarco, 1982). Within KADS, knowledge acquisition is considered as the phase containing the processes of collecting, analysing, interpreting and modelling the data (da Silva, 1994). Knowledge-based system development starts with knowledge elicitation, which is an activity in the knowledge acquisition phase. Knowledge elicitation is concerned with obtaining knowledge from an expert through various elicitation techniques and reporting it in a form which is understandable by both the knowledge engineer and expert. This includes the construction of a task model and model of expertise. This review seeks to focus on the later models produced through the methodology, with a particular emphasis on the role of functional prototyping as a means for model validation.

2.3.2.1 Model of Expertise

The model of expertise is the collection of expertise primitives that have been combined from four layers of knowledge structures (Wielinga et al, 1991). These four knowledge categories (domain, inference, task and strategic) can be viewed as four levels related through each successive level interpreting the lower level description. Figure 3 (Wielinga et al, 1991) depicts this relationship.

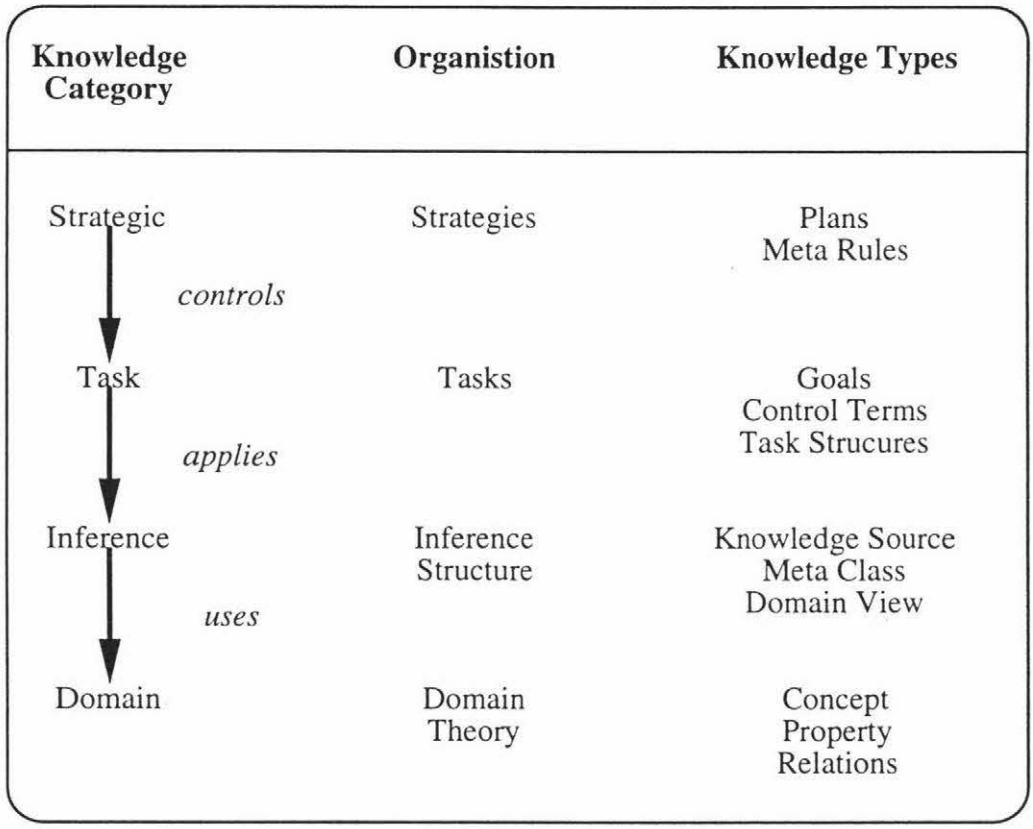


Figure 3 - Synopsis of KADS four-layer model
(from Wielinga et al, 1991)

Akkermans et al (1993) state that the model of expertise is a central model in the construction of a knowledge-based system. The model of expertise models the problem solving behaviour of an agent in terms of knowledge that is being applied in carrying out a certain task.

Wielinga et al (1991) agree that building a model of expertise is a central activity in the process of knowledge-based system construction. This is what distinguishes knowledge-based system development from conventional system development. Its goal is to specify the problem solving expertise required to perform the problem solving tasks assigned to the system.

Wielinga et al further conclude that the model of expertise is a knowledge level model. This is because the model of expertise specifies the desired problem solving behaviour for a target knowledge-based system through an extensive categorisation of the knowledge required to generate this behaviour. The model thus fulfils the role of a functional specification of the problem solving part of the artefact, focusing on what the target system should and can do.

2.3.2.2 Task Model

A task model specifies how the function of the system is achieved through a number of tasks that the system will perform (Wielinga et al, 1991). The content of the task model is determined from three aspects, task decomposition, task distribution and task environment. Task decomposition is the process of identifying tasks within the system requiring functionality, then decomposing (or breaking down) these tasks into subtasks. The composite top level task is often called the "real-life task" as it often represents the actual tasks that an expert performs in the application domain.

The task distribution is the assignment of task to agents. These agents may either be internal (for example, the knowledge-based system itself) or external (for example the user or some other system). It must be determined which subtasks to assign to the system and which to the user. The task environment is the constraints that the nature of the task domain enforces, for example whether a system has sensors for observations, or whether the system depends on the user to enter the data.

2.3.3 Model Validation and the KADS Methodology

Within the KADS methodology there has been an apparent reluctance to deal with issues relating to model validation and verification. A broader search of knowledge based system literature is required to arrive at a suitable working definition of these two concepts. Clearly model validation and verification are important parts of expert system development. As the importance of its role in the knowledge acquisition process is realised, new terminology, projects and methodologies have arisen (Hoppe, 1990).

O'Keefe et al (1987) examined the different forms of testing under the topic 'validation' and characterises the difference between validation and verification as verification meaning to build a system right, while validation meaning to build the right system.

Benbasat and Dhaliwal (1989) provide a more formal definition, proposing the following:

- Verification is the proof of certain logical properties in the development life cycle of a system.
- Validation is the determination of a homomorphism between a system and its representation.

Nazareth (1989), with regard to rule based models, considers verification to be construed as the demonstration of logical correctness of the rule set, wherein checks are performed for superfluous, incorrect or missing rules which would eventually impair system performance. Nazareth says that verification can be viewed as the processes of demonstrating that a knowledge-base is consistent (though verification of the contents of the knowledge-base) and ensuring consistency between the constructed system and its specifications. In doing this Nazareth blurs the difference between verification and validation, for the second process is considered to be part of the validation task by most authors. Difficulties in demonstrating the correct performance of an expert system (verification) include

- ill defined or missing specifications.
- inapplicability of conventional verification and validation techniques.
- lack of agreement on evaluation methodologies.

Nazareth also argues that domain knowledge should be excluded from the verification processes. The author justifies this strange ideal through looking to provide an approach for formulating generalizable principles.

Verification and validation are quite often difficult to perform because most expert systems do not have a fixed number of final states (Brule and Blount, 1989). Brule and Blount state that the solution to this is a compromise between formal methods, common sense and pragmatics. If algorithms are generated by the experts, then these can be formally examined. Any rules generated through rule induction can have their set of examples examined for completeness by an expert. Experts can also generate a suite of test cases in addition to those generated in the knowledge acquisition phase.

Within the realms of this project, the following definitions were applied.

Model Verification: The process of ensuring a derived model accurately depicts the model from which it has been constructed.

Model Validation: The process of ensuring that a model is an accurate representation of the domain from which it has been derived.

Within the KADS framework, Protocol analysis is proposed as both a method for model differentiation and for model validation (as shown in Figure 4, adapted from Wielinga et al, 1991).

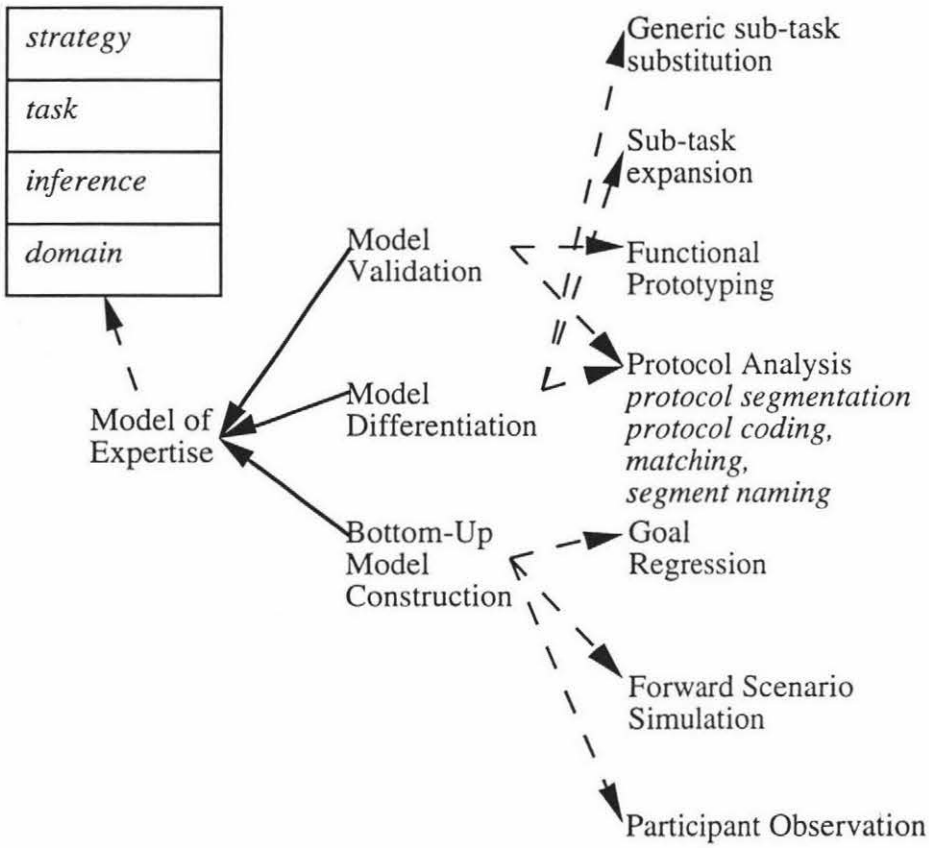


Figure 4- The Role of Protocol Analysis and Functional Prototyping (Adapted from Wielinga et al, 1991).

Protocol analysis is an aid to understanding the human problem and decision solving process. It involves a verbally elicited script containing a description of the thoughts and actions of an expert with regard to performing a certain task. The aim in interpreting a protocol is to reconstruct the expert's model for problem solving (Jackson, 1990).

Protocol analysis consists of protocol segmentation, protocol coding, matching and segment naming (Ericson and Simon, 1984). The model validation process is often strongly guided by protocol analysis, whose output is in the form of documentation.

For encoding purposes, the protocol is split into segments, each segment corresponding to a statement. A statement may have one or more segments associated with it. This connection is known as a matching. Segment naming involves the allocation of a descriptive name (or names) to a segment. This name is a brief reference to the information contained in the segment.

Each segment is encoded from the information contained within the segment itself. If the segment is fragmentary, context may need to be consulted to remove ambiguity. Coders who understand the domain can code segments with a higher probability of reliability.

Protocol analysis can be used for model validation. The accuracy of a model can be evaluated by tests of reliability. This is verified by agreement between different people encoding independently of each other (Ericson and Simon, 1984). This is not enough on its own, however. During encoding a number of considerations must be taken into account to reduce potential errors.

The encoding process must be as objective as possible. Ambiguous statements may be encoded towards the encoder's own, preferred interpretation. The context of any statement is important for this.

Another problem is the independence of encoding judgements. Problems arise here because human coders remember their previous encodings, and other information about earlier segments. This can be over come through having units defined sufficiently large so that all information required for encoding is contained in a single segment. Another possibility is having segments encoded in a random order.

2.4 Prototyping

The lack of literature on the field of prototyping within the KADS methodology lead to a search of general knowledge-based system literature on the topic. The literature by authors associated with the KADS project is presented first, then the opinions of the wider knowledge-based system community.

2.4.1 Role of Prototyping within KADS

The term 'prototyping' is often used as if it has a single meaning, which is far from the truth. Hayward (1986, p. 196) is amongst the first to comment on the process of rapid prototyping. Hayward somewhat cynically defines the process of rapid prototyping as being "it doesn't matter much where you start, try something, see if it works and keep modifying it until you get something good enough for your purpose". Hayward says that the experimental aspects of rapid prototyping lead to a "wholly inadequate answer" to the question of knowledge-base design. Hayward does, however, believe that there is potentially a place for prototyping in expert system development, just not as the complete process for knowledge-based system development.

Wielinga et al (1986) put forward a few of the benefits of rapid prototyping. There is a focus provided on the elicitation and interpretation processes carried out by the knowledge engineer. Another advantage is that the construction of a prototype may well motivate the expert, as the feasibility is displayed. However, there are drawbacks as well. The most important problem is the fact that the solution is constructed without an encompassing analysis of the problem. This is often accompanied by frequent and drastic backtracking. compulsive rapid prototyping is an underestimation of the knowledge acquisition problem. There is also a very large gap between the verbal data on expertise and the implementation constructs used to build the artefact, the knowledge-based system. When larger parts of the knowledge of the expert have to be taken into account this may become quite troublesome. The number of variations within the problem solving process grows, for which generic solutions have to be found within the existing framework, and these may not fit at all. This contrasts with one of the assumptions underneath prototyping, that is that it facilitates at a very early stage the uncovering of the basic structure of the expertise.

Hickman et al (1989) categorise prototypes as being either evolutionary or throwaway. Evolutionary prototyping (also known as rapid prototyping) abandons the methodological approach of separate analysis, design and implementation phases in favour of the immediate delivery of a rapidly constructed partial system to the client. This system dynamically changes as further requirements from the client are implemented by the development team.

Hickman et al note the difficulties of this approach include:

- the development of prototypes is an unplanned, ad hoc iterative affair.
- each iteration involves re-specification, re-design, re-implementation and re-evaluation activities.
- analysis is implementation dependent and driven by the prototype.
- early adoption of a shell means the problem must suit the characteristics of the shell.
- knowledge acquisition is driven by the shell, resulting in data from experts being interpreted in a pre-defined way which may lead to a biased system.
- major design decision must be made at an early stage.

Throwaway prototyping involves creating prototypes with a limited life span for a purpose, usually that of evaluation. Exploratory throwaway prototypes are most useful during the early stages of development, for feasibility and requirements analysis. Experimental throwaway prototypes are used when the developers have some rough design ideas. They allow for the testing of feasibility and adequacy of these design ideas.

2.4.2 General View of Prototyping

Hart (1990) defines the different types of prototyping to be:

- Throw-it-away prototyping, involving building a prototype at an early stage with the intention of learning from it and then discarding it and building the actual system.
- Incremental prototyping, being the staged delivery of a system. The single design of the system is broken down into a number of smaller projects, and at various stages in the development the prototype is presented for acceptance and testing of a further part of the system.
- Evolutionary prototyping, the continual modification of a working version of the system. Changes are incorporated as and when they are needed, meaning that clearly the system must be designed for the possibility of change in the future.

Hart also suggests there is a choice to rapid prototyping as a means of system development, offering the alternative of the KADS methodology. Hart perceives KADS as separating the activities of problem analysis from selection of solution methods. The methodology identifies three stages- orientation, problem identification and problem analysis- and provides for many elicitation techniques, types of data and the stages at which these can be used.

Hart and Berry (1992) suggest a generic definition encompassing forms of prototyping, saying that an individual prototype is a system which has some of the essential elements of the final system, but not all.

Sommerville (1989) recommends that any prototype constructed should be considered as a throwaway system and should not be used as a basis for further system development.

Roberts (1988) proposes another form of prototyping, sufficiently different from the previously mentioned three to be in a category of its own. This is structured prototyping, where a key concept is the use of some form of intermediate representation. The structured prototype involves the building of a paper model, which can then be tested (in accordance with the KADS methodology) through protocol analysis before proceeding to a machine based implementation.

Roberts wrote about experience with the process of constructing an intermediate representation model. This was found to have a number of advantages. This allowed for the most appropriate knowledge structure (rules, frames, objects) to be selected. This in turn allowed for the intermediate representation model to be environment independent.

In theory, it has always been standard practice to try and get a complete system specification before starting to design and implement a computer based system. However, experience proved this to be an inefficient way of proceeding. According to Hart (1990), this was because at the start of the project, people's ideas of what is required and how it will appear in practice are fuzzy and imprecise. Reasons for this include written specification being difficult to understand, particularly if written in computer jargon. A dynamic system is hard to understand from a static description.

2.4.3 Prototyping Life Cycle

Duffin (1988) proposes a prototyping development life cycle, consisting of five stages as shown in Figure 5.

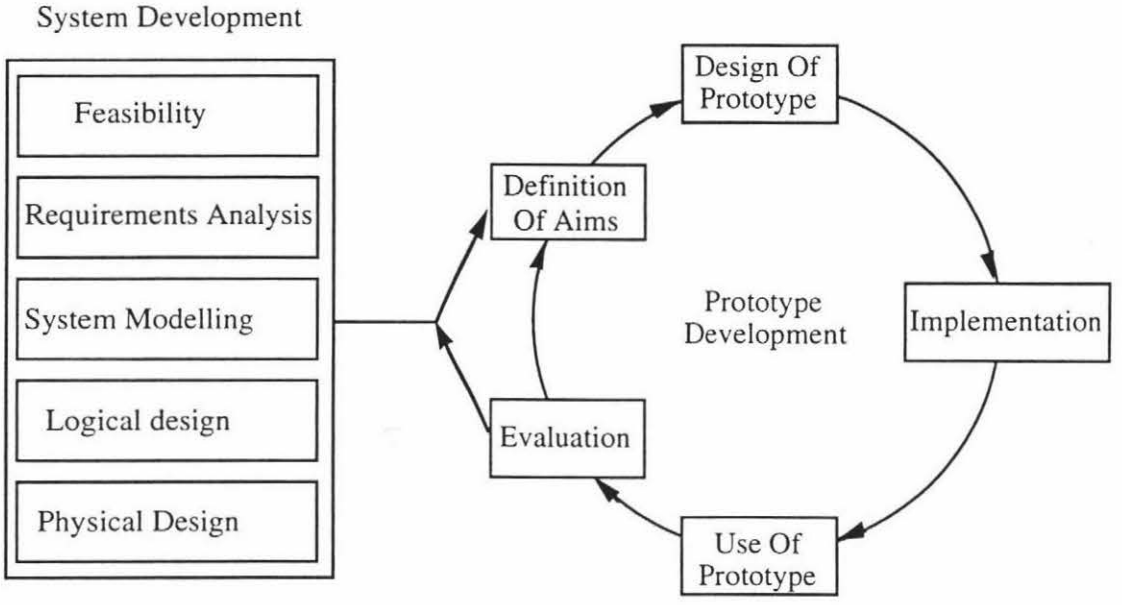


Figure 5 - prototyping development life cycle
(Duffin, 1988)

It is important for there to be an explicitly defined domain, and success criteria, before it is designed, implemented and used. The evaluation process involves a careful transformation of the gained experiences back into the system development life cycle. This evaluation may lead to a new prototyping cycle, which results in the aims being redefined.

The timing of commencement of the prototype is a critical issue in the project life cycle. If the prototype is constructed without enough analysis having taken place, then the process of building, modifying and scrapping a prototype can be repeated again and again without any convergence towards a final, acceptable system (Hart and Berry , 1992). They add to this warning, saying that prototyping does not preclude the production of structured and maintainable systems, but using an undisciplined approach to prototyping almost certainly will.

A decision must also be made as to when to finish the prototyping model. Payne and McArthur (1990) recognise the difficulties by saying that it can be difficult to call an end to fine tuning. A decision must be made that the prototype contains all the necessary information. There must also be an evaluation of the prototype to see whether all the original goals have been met and if further work should be undertaken. Closure refers to the formal acknowledgment that the system is ready for an evaluation.

Functional prototyping is an approach in the KADS methodology to model validation for knowledge-based systems. There has been little research focused into this specific area of the KADS approach, possibly because KADS was in part deigned to offer an alternative to rapid prototyping. However, functional prototyping stands in its own right well away from rapid prototyping, and can be seen as a simulator of the problem solving aspects of the artefact (Wielinga et al, 1991). In the fore-mentioned document, the area of functional prototyping is hardly mentioned, other than to say that there is a tool currently being constructed to support this type of prototyping.

Sommerville (1989) says that the realism of requirements can be demonstrated by constructing a system simulator, which is analogous to using a functional prototype to verify part of a model of expertise. However, Sommerville notes problems with a system simulator as including:

- For complex systems, it can be as expensive and as time consuming to develop the system simulator as it is to develop the system itself.
- It may be difficult to change the simulator so that changes in the requirements may be impossible to assess by simulation.

Sommerville also suggests that in a software engineering domain, prototyping can be used as a means of requirements validation. The four steps for validating requirements are:

- Needs of the user should be shown to be valid.
- Requirements should be shown to be consistent.
- Requirements should be shown to be complete.
- Requirements should be shown to be realistic.

Reviews requiring both developers and users should be conducted with the development team walking the client through the requirements, explaining the impact of these requirements. It is then up to the developers and users to resolve any problems and contradictions.

Benbasat and Dhaliwal (1989) state that evaluation is the measurement of certain properties (for example quality, utility and usability) of a system. Brule and Blount (1989) add to this definition by saying that evaluation begins with a clear definition of the project, accompanied by a functional design.

Criteria for evaluating the prototype comes from three levels according to Payne and McArthur (1990):

- Development team checks for implementation stability
- Experts and end users check validity of data.
- Management review completed prototype to evaluate whether the project goals have been met.

Acceptance of the prototype occurs after the original problem definition and specifications have been reviewed by the group and agreed upon as being met. (Brule and Blount, 1990).

2.5 Conclusion

Knowledge-based system development has been one of the areas of Artificial Intelligence research to show significant advancement since initially being recognised as a separate area for research. Knowledge-based systems attempt to use rules of thumb rather than the more structured algorithmic approach employed in many other computer realms. Because of the emphasis placed on the form of knowledge representation selected, there has been much work on the evolution of symbolic knowledge representation. The latest form of knowledge representation is derived from the object-oriented paradigm, the knowledge representation form being that of objects.

There has been a void in knowledge-based system development, that of a comprehensive methodology to encompass all aspects of system development. The KADS methodology has evolved through time, and now has an emphasis on the analysis stages of system development. To achieve this, a number of models are generated. Each model reflects selected characteristics of the problem domain, and combine to form the model of expertise. This model of expertise has the goal of providing the problem solving expertise required to perform the problem solving tasks in the target system.

Model validation is an important part of any knowledge-based system development. KADS offers two approaches for this. Protocol analysis involves the construction of 'paper' models to express knowledge. Functional prototyping uses the construction of a prototype to validate knowledge. Functional prototyping has not been a common choice within the KADS framework, possibly because it involves prototype implementation, which is not in line with the KADS ideology. However, prototyping comes in many forms. Rapid prototyping, historically commonly used in AI research, is widely condemned as being too unstructured. Throwaway (or functional) prototyping involves the creation of prototypes with a limited life span, and can be used for feasibility and requirements analysis, as well as model validation.

Chapter 3: Construction of the Knowledge Representation Model

Introduction

A requirement for model development is that of an understanding of the domain. This chapter begins by outlining relevant areas of the farm management problem. For a more detailed explanation, see da Silva, 1994. For this domain, da Silva performed the knowledge acquisition phase of the KADS methodology. This involved analysis of a number of transcripts from successful, local farmers. The result of this phase was in the form of a model of expertise.

A decision had to be made with regard to choosing the most appropriate knowledge representation form being chosen; this is the beginning of the analysis and design phase of Nunamaker's methodology. The choice, and the rationale behind it, is explained in the second section of the chapter. A comprehensive example of the chosen methodology and processes involved in the transition from the KADS model of expertise to an object-oriented knowledge representation is worked through.

3.1 Domain Problem

The dairy industry plays an increasingly important role in the New Zealand economy. A knowledge based system has been proposed to help farmers make decisions during the most important time of the year, the period including the summer and autumn seasons.

The decisions involved in farm management are categorised into two types. The first type are short term plans, which, as the name suggests, are frequently made, often looking ahead seven to ten days only. Long term plans are made less frequently, often being major seasonal decisions such as when to start preparing for winter. Research has been undertaken into the first steps of the knowledge acquisition process with regard to this problem domain using the AIM methodology (da Silva, 1994). A number of successful farmers have been interviewed in the Manawatu region. From this research a task model has been developed, representing the overall activities performed by farmers in order to deal with problems and plan future actions. A model of expertise has also been constructed, with the goal of specifying the problem solving expertise required to perform the problem tasks.

Effective farm management practices are crucial for a successful dairy farm. The year on a dairy farm can be split into three major periods:

- Winter
- Spring
- Summer - Autumn

The Summer- Autumn period is when a number of crucial decisions are made. The management options available are presented in a simplified management hierarchy, as shown in Figure 6 (Gray et al, 1993).

1. Cull low producing cows.	**Least cost / Most flexible**
2. Dry off thin heifers (< 3.5 condition score).	
3. Speed up the rotation (no less than 25-30 days).	
4. Feed silage, surplus to spring/ winter requirements.	
5. Feed summer forage crop.	
6. Cull cows (normal culls).	
7. Once a day milking.	
8. Dry off the herd.	**High cost / Least flexible**

Figure 6 - Simplified hierarchy of management options.

The most critical of these is that of the drying off date, the point at which farmers stop milking the cows. If the cows are dried off too late, then their body condition will reduce, which may prove difficult to replace before calving. This has the flow on effect of resulting in low production during early lactation (milking) in the following season, as well as potentially reducing reproduction performance. Also feed that should be conserved for the winter season shortage is required to maintain a satisfactory production level.

However, drying off the cows too early results in a reduction of income for the farmer in the current season. The goal is thus to leave the drying off date until as late as possible without jeopardising the next season's production. It is the importance placed on this final decision that emphasises the need for monitoring both production and the herd's condition score. Records are kept of the factors associated with the herds

New Zealand farming systems are usually pastoral based. This pasture is a relatively cheap source of livestock feed, but is subject to seasonal fluctuation. During the spring and winter periods, there is commonly a pasture surplus. However, New Zealand summers are typically dry, with herd demand exceeding pasture growth for four to six weeks during February and March. To ensure there will be sufficient feed supply for this period, feed budgets are used to minimise the impact of any deviation (Gray et al, 1993).

Feed budgeting is for a large part of the year carried out quite informally. However, as the summer autumn period progresses, more formal techniques of often employed, to increase the chance of optimisation of resources.

3.2 Applying the Object-Oriented Approach to Knowledge Representation

The Knowledge Representation model takes the KADS model of expertise and creates a model of the system in accordance with one of the available knowledge representation paradigms.

For this project, an object-oriented model of the system was created. Sommerville (1989) identifies advantages of using the object-oriented approach:

- Objects are independent entities, which may readily be changed because all state and representation information is held within the object itself.
- All shared data areas are eliminated as communication between objects is via message passing.

Schreiber and Wielinga (1992, p. 2) note that despite the differences between KADS and object-oriented approaches, "there are quite a number of similarities between the approaches." For example, the KADS model of expertise when combined with the task model provides the equivalent of the output of the analysis phase of the object-oriented approach. Schreiber and Wielinga, while focusing on the object modelling technique approach proposed by Rumbaugh et al (1991), produced a table showing the relations between various software engineering techniques, including the object-oriented approach, and the KADS methodology. Table 1 is adapted from their table.

Object-Oriented	KADS
Analysis	Task Model and Conceptual Model
Part of Analysis Model	Task Model and Model of Co-Operation Model of Expertise
Design Model	Design Model

Table 1 - Comparison of KADS and Object-Oriented Analysis
Adapted from Schreiber and Wielinga, 1993

3.2.1 Object-Oriented Principles

The object-oriented approach adopted is the Coad & Yourdon methodology (1991). Object-oriented analysis is based on the following principles for managing complexity.

- Abstraction
- Encapsulation
- Inheritance
- Communication with messages

Abstraction can take two forms, that of procedural abstraction or data abstraction. Procedural abstraction is concerned with the level at which something is viewed. For example, a novice car driver may view the car engine as one, whole unit, whereas a mechanic perceives an engine as being comprised of pistons, valves, distributor etc. Procedural abstraction is commonly associated with "function / sub-function" abstraction, where a high level task (for example "Fix Car") is decomposed into a number of smaller tasks ("Fix engine", "Fix electrical system" etc.).

Data abstraction is the principle of defining a data type in terms of the operations that apply to objects of the type, with the constraint that the values of such objects can be modified and observed only through the use of the operations (Coad and Yourdon, 1991). For example, to find the colour of a car is not done by saying "What is the value of the car attribute colour". Instead the operation "Return Car Colour" (or similar name) is invoked, which will simply return the car colour.

Encapsulation, or information hiding, is an important concept within the object-oriented approach. Each module is defined such that it reveals as little as possible about its internal workings. This is the so-called "black box" principle, whereby a user does not require (or need) to know how the output is reached, just that it is generated. For example, a researcher may feed five numbers into a function and their standard deviation is returned. The researcher need not know the formula, but is just concerned with the value returned.

Inheritance is another underlying principle of the object-oriented approach. It is a mechanism for expressing similarity amongst classes, simplifying definition of classes which are alike ones previously defined. It portrays generalisation and specialisation, making common attributes and services explicit within a class hierarchy or lattice (Coad and Yourdon, 1991). Inheritance thus allows for the expression of commonalty, where common attributes and services need only to be specified once. A specialisation (instance) of a generalisation (class) may have additional services or attributes added. For example, a generalisation "Vehicle" may have a specialisation "Truck". The "Truck" may have added attributes such as "Number of axles", and added services such as "Calculate maximum weight per axle", which might not be relevant or appropriate for "Vehicle" in general.

Communication with messages is the way in which requests for some action to be performed are made. The result of this request may or may not return a value. Messages are the way in which operations (or methods) are invoked, when one module requests from another module some action or data store value.

3.2.2 Object-Oriented Constructs

The eight object-oriented constructs, which are used during object-oriented analysis and design, are

- Class & Object
- General - Specific Structure
- Whole - Part Structure
- Attribute
- Service
- Instance Connection
- Method Connection
- Subject

Class & Object

A class is a description of one or more objects with a uniform set of attributes and services.
An object is an abstraction of something in a problem domain. It is an encapsulation of attribute values and their exclusive services. Each object (instance of a class) *inherits* the characteristics of the class.

Class-&-Object is a term meaning "A class and the objects in that class", and is symbolised as in Figure 7.

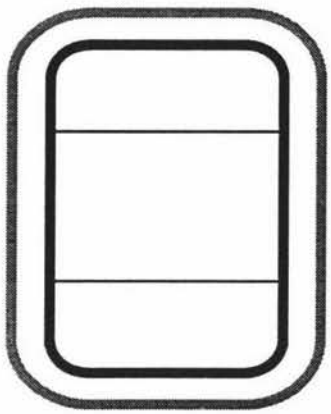


Figure 7 - Class & Object Structure

General - Specific Structure

A Generalisation - Specialisation structure is an approach to distinguishing between classes as a method of organisation. The example in Figure 8 is that of Vehicle being a generalisation, and Truck being a specialisation. This can be thought of as an 'Is_A' relationship, where (for example) a truck Is_A vehicle.

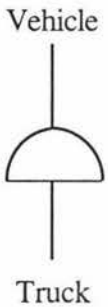


Figure 8 - Is_A hierarchy

Whole - Part Structure

A Whole - Part structure is an approach to identifying components which are grouped together to make a class as a method of organisation. An example of Vehicle being the whole, and Engine being a part is shown in Figure 9. This can be thought of as an 'Has_A' relationship, where (for example) a vehicle Has_A engine.

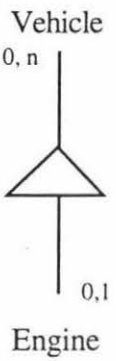


Figure 9 - Has_A hierarchy

Each end is marked with a constraint or range, indicating the number of parts that a whole may have (and vice versa) at any given moment in time. From the above example, a vehicle may have no engine (if it is a glider) and at most *n* engines. An engine is part of at most one vehicle, or possibly no vehicle.

Attribute

An attribute is some data (state information) for which each object in a class has its own value. The attributes are denoted as shown in Figure 10.

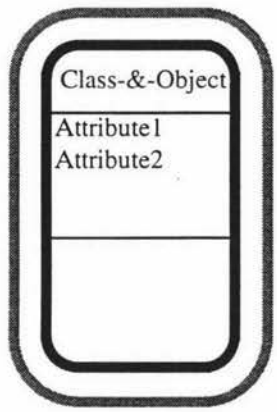


Figure 10 - Class & Object with Attributes

For example, all engines have an attribute denoting their size in cubic centimetres, but the value of this attribute varies for different instances.

Service

A service is a specific behaviour that an Object is responsible for exhibiting. A service is how the internal values of an object can be retrieved, and is depicted in Figure 11.

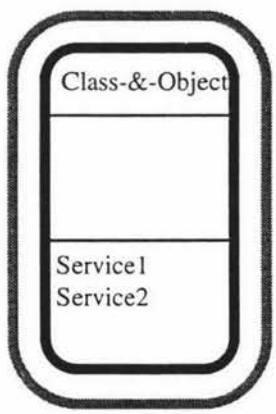


Figure 11 - Class & Object with Services

For example, to determine the size of an engine (which has the attribute size), the attribute *size* cannot be directly accessed. Instead a service must be requested (named such as Return_Engine_Size), which will return the engine:size value. This request of a service is called a message.

Instance Connection

Instance connections form a model of problem domain mappings that one object needs with other objects in order to fulfil its responsibilities. The connection is denoted by a solid black line.

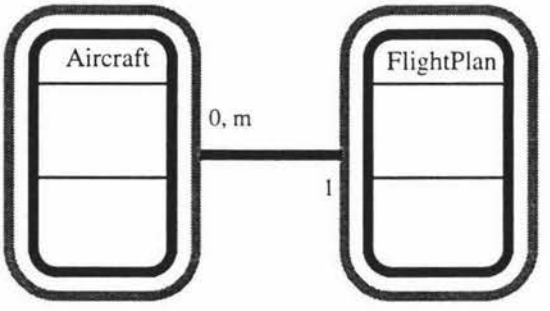


Figure 12 - Instance Connection

In the example in Figure 12, an aircraft may have either zero or many flight plans, while a flight plan must be assigned to one aircraft only.

Message Connection

A message connection models the processing dependency of an object, indicating the need for services in order to fulfil its responsibilities. The arrow points in the direction that the message is to be sent.

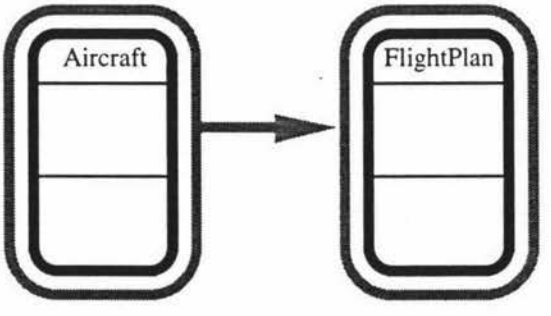


Figure 13 - Message Connection

For example, in Figure 13, one of the instances of the class aircraft may have a service which requires to know where the aircraft is flying to. A message could be sent from the instance to the flight plan to retrieve the destination.

Subject

A subject is a mechanism for guiding a reader (be they an analyst, problem domain expert, manager or client) through a large complex model. Subjects are also helpful for organising work packages on larger projects, based upon initial object-oriented analysis investigations. Subjects are used because of the commonly large number of objects within an object-oriented model. Denoted by thick grey border, subjects can be collapsed to abstract more detail from the system. In Figure 14, the box on the left denotes a collapsed subject.

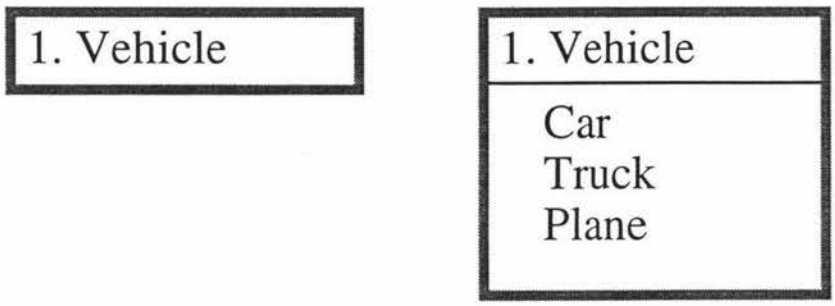


Figure 14 - Examples of Subjects

3.3 Creation of an Object-Oriented Task Model

The task model created by da Silva (1994) and shown in Figure 15 led to an object-oriented approach being taken to the overall task of Manage_the_Farm. This overall goal was identified as consisting of four parts:

- Planning Management decisions, required to optimise the current season's production without placing at risk the next seasons production.
- Monitoring Events, to ensure what events are happening are consistent with the goals to be achieved
- Diagnose Problems, to identify the cause of any observed problems.
- Prediction of Events, to identify future states on the farm in order to foresee potential problems.

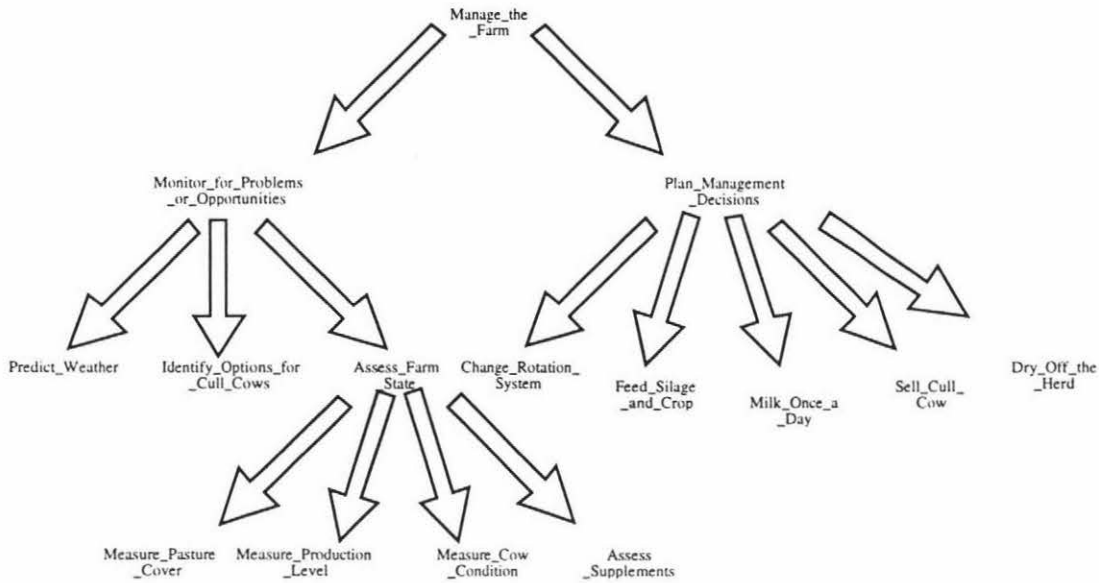


Figure 15 - Task Model of the Domain (da Silva, 1994)

The object-oriented model constructed from this decomposition is shown in Figure 16.

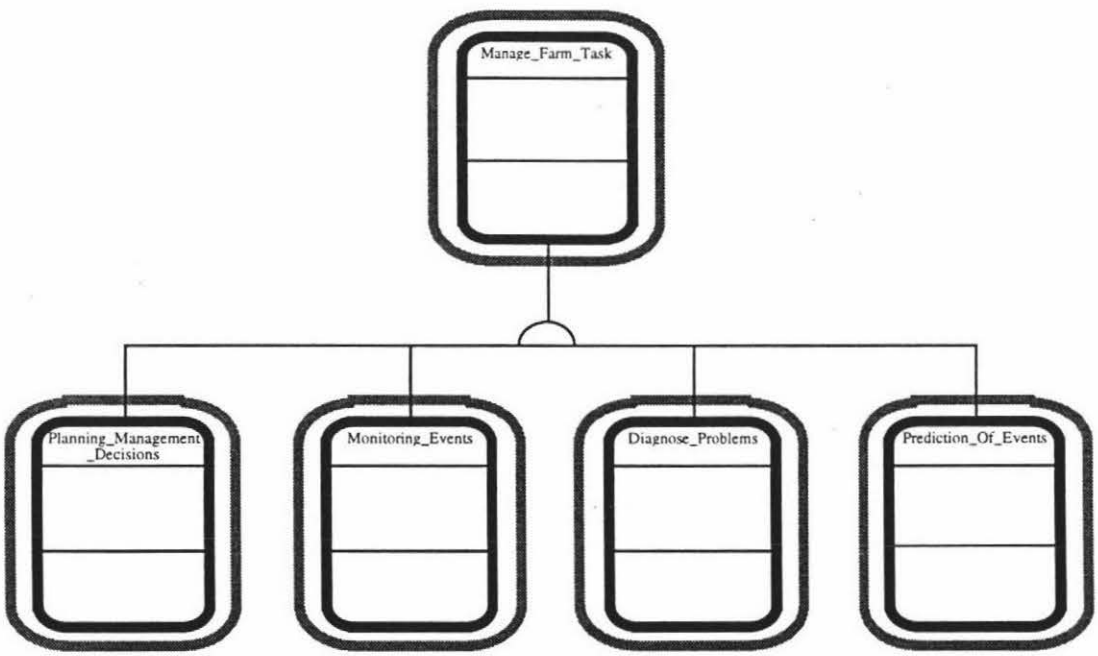


Figure 16 - Object-Oriented view of Manage_The_Farm task

These four divisions were again decomposed. Figure 17 is an example of this, showing how Monitoring Events can be decomposed itself into:

- Pasture Cover Reading
- Pregnancy Test Monitoring
- Production Level
- Body Condition
- Crop Monitoring

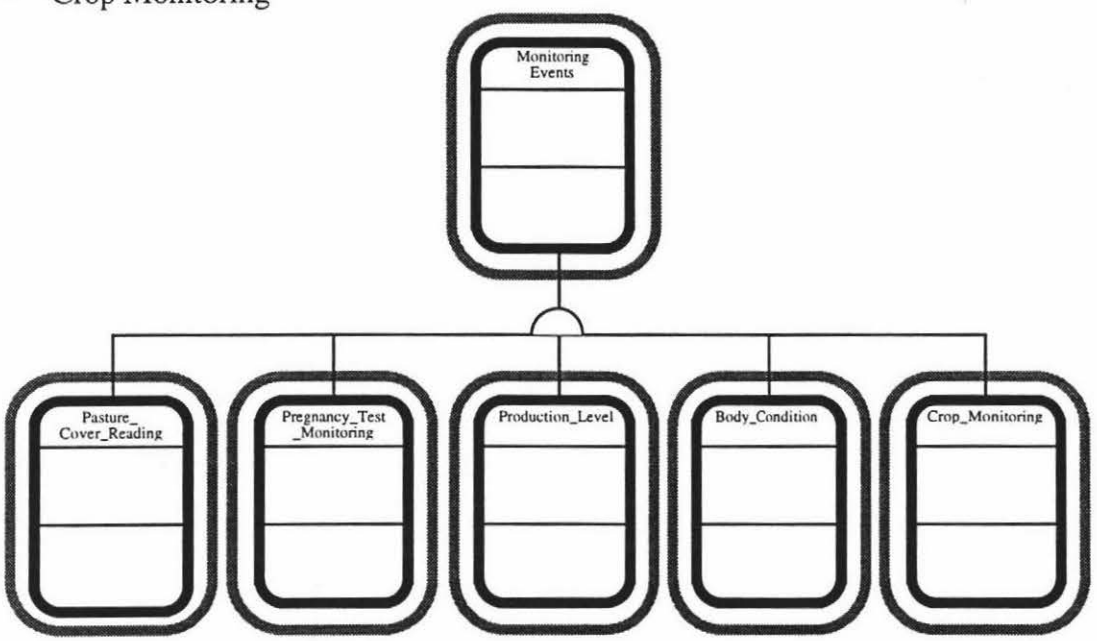


Figure 17 - Decomposition of Monitor Events Task

So with the hierarchy constructed as in Figure 18, the services could now be added to the objects.

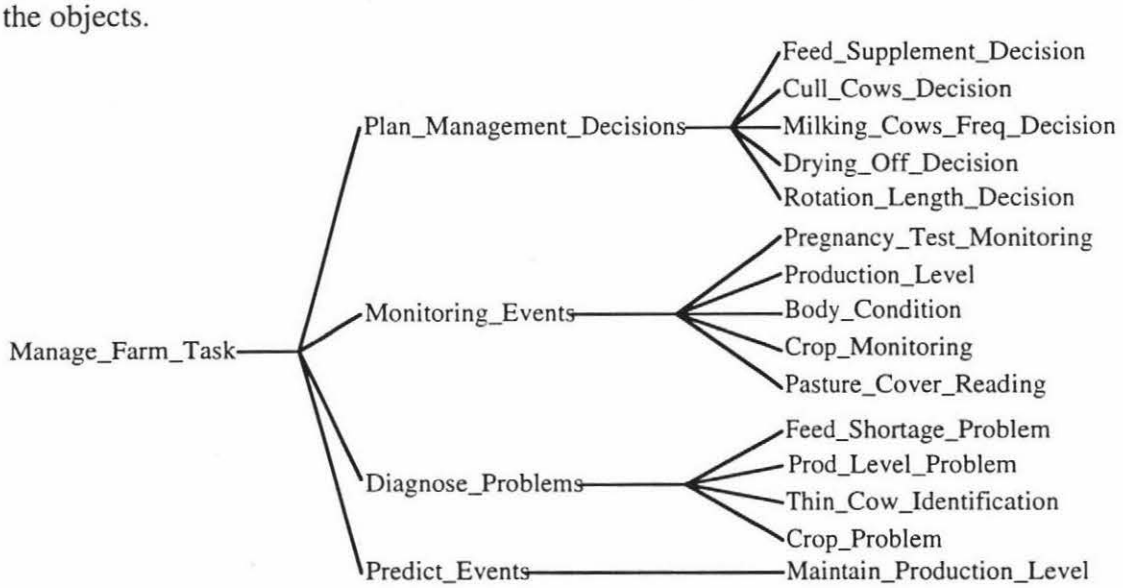


Figure 18 - Task Hierarchy

The transition from inference layer to task layer were carried out by da Silva (1994). The conventions used in describing these transitions are:

- The arrows in the task structure describe the relation between input and output knowledge roles which are represented by concepts of the domain.
- The names of the primitive inference actions are italicised in the task structure.
- In case a transfer task is necessary for the achievement of the subtask being described, its name will be italicised.

'Monitoring Events' subtasks were used to ensure that the results being obtained are in accordance with the goals to be achieved. The components of the task structure are

select (Observable --> Variable Values)
select (Parameters --> Parameter Values)
measure (Variable Values --> Results)
compare (Results and Parameter Values --> Differences)

Selecting variable values involves the decision as to which observables on the farm are relevant to the area being monitored. For example, the average pasture cover is relevant for monitoring the farms pasture cover, whereas the herd production level is not.

Once relevant observables have been determined, parameter values are selected. These are ideal values with which the real situation on the farm is compared. For dairy farmers, historical Figures are quite often a source of parameter figures. For example, last years level of milkfat production can give a good estimate of where production should be at present, especially with the knowledge of what subsequently happened.

Measuring involved the instantiation of the variable values by performing some form of quantitative measurement, for example performing a pregnancy test returns a percentage of cows successfully impregnated.

Once these measurements have been made, a comparison is made between the actual figures and the ideal figures, and the difference recorded. No decisions are made regarding this figure; the amount is simply stored for future reference.

The task 'Diagnose Problems' is composed of five individual activities:

select (Observable --> Variable Values)
select (Parameters --> Parameter Values)
measure (Variable Values --> Results)
compare (Results and Parameter Values --> Differences)
classify (Differences --> Problems)

The first four activities determine whether, from the calculated difference, there is a problem present. If there is found to be a problem, from the difference the problems is identified and given a name, for example 'Feed Shortage'. This occurs in the classify activity.

The 'Prediction of Events' task also consists of four activities:

select (Observable --> Variable Values)
select (Parameters --> Parameter Values)
measure (Variable Values --> Results)
predict (Results and Parameter Values --> Events)

This process is similar to the monitoring activities, but with a view to the future. Actions are suggested for the short term to prevent foreseen future problems. For example, if with current crop growth there will be excess feed, making hay or silage may be suggested to reduce wastage.

Finally, 'Planning Management Decisions' are made with a view to optimising production in the current season without jeopardising the next season's production. The task structure is as follows:

construct (Problems, Strategies, Historical Data and Events --> Management Decisions)

The construct activity deals with the conclusions reached in the 'Diagnose Problems' and 'Prediction of Events' tasks. The conclusion reached in this activity are major decisions on the farm, for example the timing of when to dry off the herd.

3.4 Moving from the Model of Expertise to the Knowledge Representation model

Coad and Yourdon's (1991) object-oriented methodology provides a comprehensive outline for the object-oriented analysis of the problem domain. When moving from the model of expertise to the object-oriented model, some elements map naturally (and easily) into the other, for example hierarchical part_of networks. Other models require more thought, and are not as straight forward to convert.

The model of expertise used as the basis consisted of three levels, these being the domain, inference and task. The domain layer generated the concepts, attributes and relationships at a high level for the object-oriented model. By high level it is meant that it does not contain the problem solving strategies. A number of concepts were identified as potential objects, while attributes from the domain layer became attributes within the object-oriented model. Relationships express the interaction amongst concepts within the problem domain. These were mapped to the object-oriented model, creating the various links necessary between the objects and classes to create a complete model.

The inference layer of the model of expertise provides the structure of the management domain, as in Figure 19. An inference specified at the inference layer is assumed to be primitive in the sense that it is fully defined through its name and input/output specification and a reference to domain knowledge it uses (Wielinga et al, 1991).

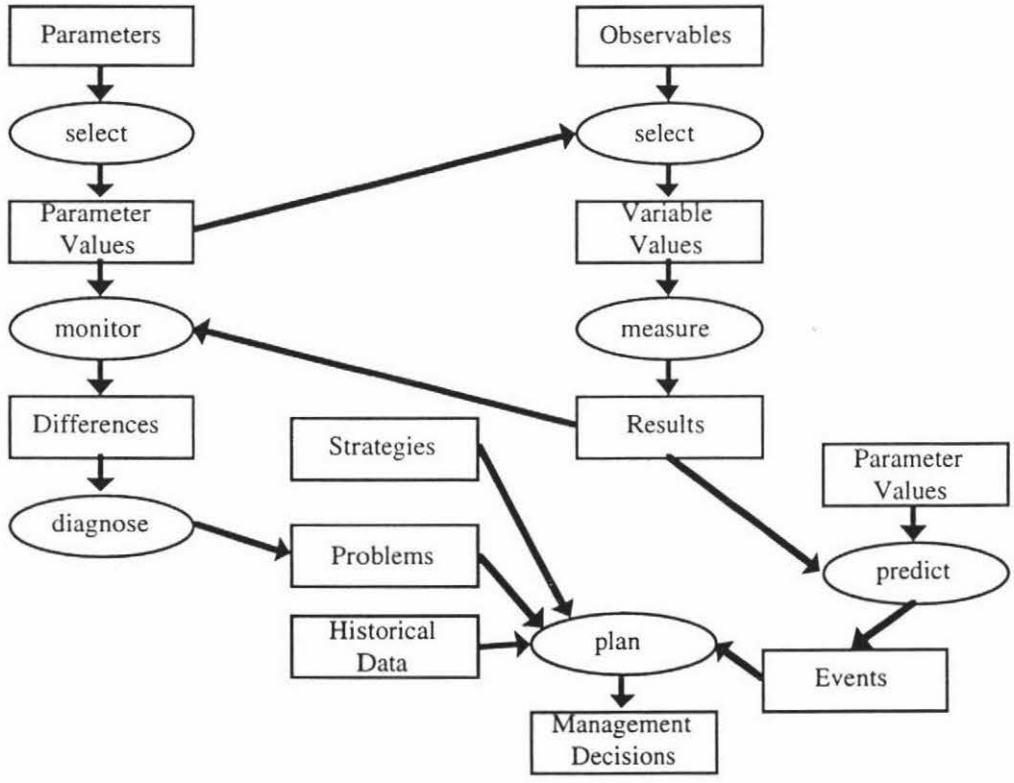
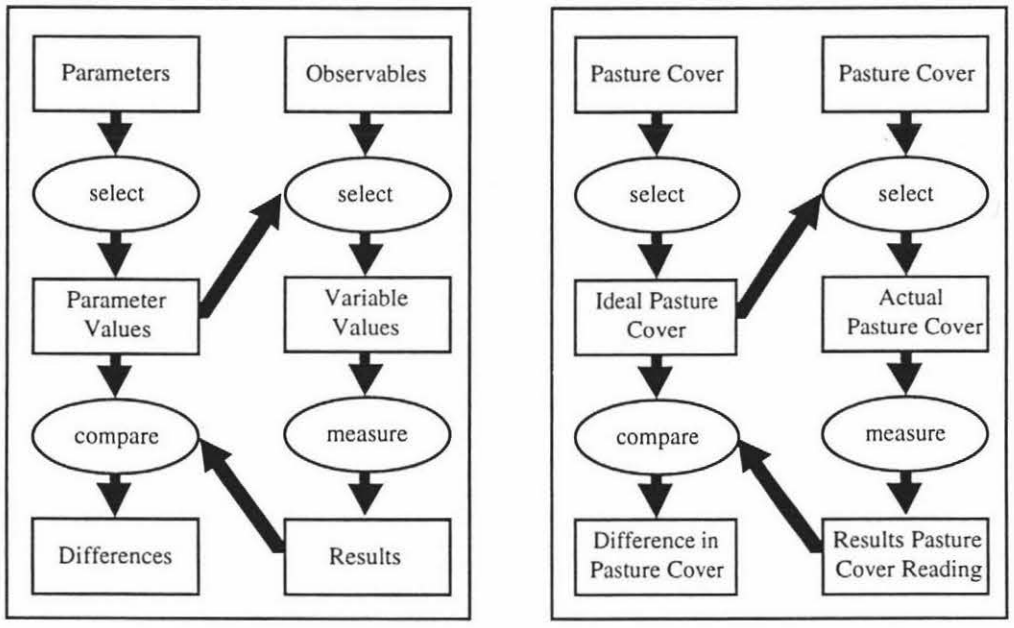


Figure 19- Inference Structure of Summer-Autumn management period (da Silva, 1994)

The structure represents the management of a dairy farm during the summer-autumn period. Four activities are involved in this management process: monitoring, diagnosis, planning and prediction. This decomposition is shown in Figure 18. To monitor the farm, the farmer would select an element such as cow condition. This element would be measured and the result compared to previous results to check if the current results were expected. A noticed difference trigger the diagnose process to identify the cause of the difference. To help with selected the optimal management decisions, prediction of events on the farm is used. The decisions taken by the farmers would be planned in accordance with the problems identified, the strategies designed to solve them and the prediction of events (da Silva, 1994).

The inference layer contributed the high level of the knowledge representation model. It provided an abstract representation, and is not concerned with detail.

The monitoring part of da Silva's inference structure is shown on the left side of Figure 20. An example of the form it took when instantiated in the inference layer is shown on the right side of Figure 20.



Monitoring Part of Inference Structure (Inference Layer) (Da Silva, 1994)

Instantiated Model of Pasture Cover Monitoring (Inference Layer)

Figure 20 - Inference structure and instantiation.

The task layer is concerned with expressing knowledge about how inferences can be combined to achieve a certain goal. The instantiation of the 'Manage the Farm' template to the inference layer formed the basis of the task layer. The 'Manage the Farm' template is shown in Figure 21.

task **Manage_the_farm**
goal: Maximise current milkfat production without jeopardising next season's yield.
task structure
select(Observable ---> Variable Values)
select(Parameters ---> Parameter Values)
measure(Variable Values ---> Results)
monitor(Results and Parameter Values ---> Differences)
diagnose(Differences ---> Problems)
plan(Problems and Strategies ---> Management Decisions)
predict(Results and Parameter Values ---> Events)

Figure 21 - Generic task template of summer-autumn management period

The Task layer takes the generic 'Manage the Farm' template and decomposes this into low level problem solving tasks. The first step is the creation from the generic task template of the generic tasks. There were four generic tasks created through decomposition (given in Appendix 1). Figure 22 depicts one of these, the task 'Monitoring Events', which consists of assessing the events on the farm.

task **Monitoring Events**
goal: Make sure the results being obtained are in accordance to the goals to be achieved.
task structure
select (Observable --> Variable Values)
select (Parameters --> Parameter Values)
measure (Variable Values --> Results)
compare (Results and Parameter Values --> Differences)

Figure 22 - Generic 'Monitoring Events' Task
(da Silva, 1994)

Figure 22 and the other generic tasks in the task layer were instantiated into subtasks and goals associated with them. Figure 23 shows the instantiated generic task 'Measure Pasture Cover'. This was derived from the generic 'Monitoring Events' task in Figure 22.

subtask: Measure Pasture Cover

goals: - Make sure there will not be a feed deficit in the next few days.

- Verify that the cows are being well fed at the moment.

task structure:

select(Pasture Cover --> Average Pasture Cover)

select(Pasture Cover System --> Ideal Pasture Cover)

Walk the farm

Select a paddock as a sample

measure(Pasture Cover --> Results of Pasture Cover Measurement)

If the month is January, February or March

Then

Calculate Pasture Cover informally once a fortnight;

Else

Measure Pasture Cover using the Plate Meter once a week

End If

compare(Ideal Pasture Cover with Results of Pasture Cover Measurement

--> Differences in the Pasture Cover)

Figure 23 - Instantiated Monitoring Task - Measure Pasture Cover
(da Silva, 1994)

From Figure 23, a part of the Monitoring Events hierarchy was identified. An instance was created named Pasture_Cover_Reading. This has the task of performing a monitoring the pasture cover. To achieve this, a service named *Measure_Pasture_Cover* was created. The attributes pertaining to this instance were identified from Figure XXX as being *Average_Pasture_Cover*, *Ideal_Pasture_Cover* and *Pasture_Cover_Measurement*. This is depicted in Figure 24.

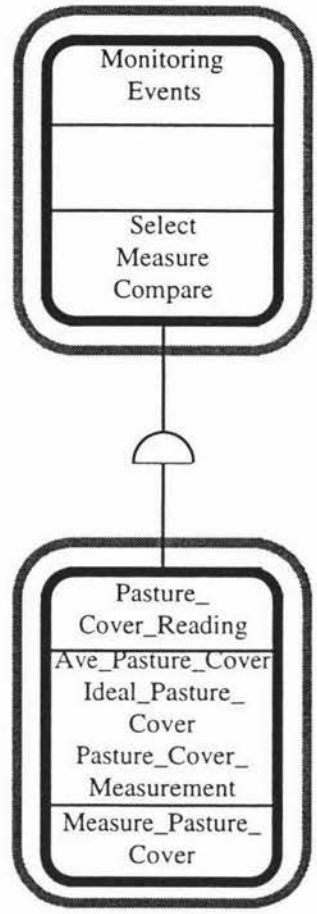


Figure 24 - Relationship between *Monitoring_Events* and *Pasture_Cover_Reading* (See Appendix 2)

3.4.1 Moving from the Model of Expertise to an Object-Oriented Model

The process of constructing an object-oriented model that was followed consisted of five activities. These were

Finding Classes and Objects (Highest Level of Abstraction)
Identifying Structures
Identifying Subjects
Defining Attributes
Defining Services (Lowest Level of Abstraction)

These activities are not unconditionally sequential steps. Coad and Yourdon (1991) order the activities this way as a result of their experiences which show this is the most common overall approach. Each activity has an associated layer of the object-oriented model.

- Class and Object Layer
- Structure Layer
- Subject Layer
- Attribute Layer
- Service Layer

Each layer contributes to the final model. Coad and Yourdon compare this to a collection of transparencies, where each additional layer presents more and more detail.

The following is an example of part of the transformation from the model of expertise into an object-oriented form. The initial model is a network of primitives (da Silva, 1994) as shown in Figure 25 which was converted to an object-oriented notation.

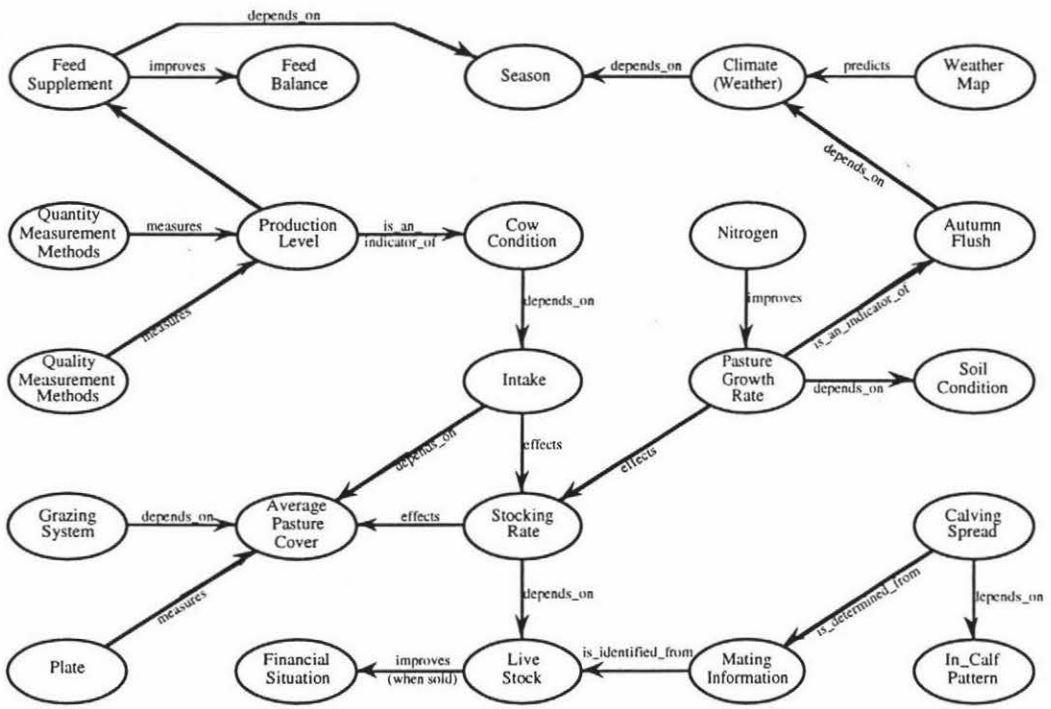


Figure 25 - Network of Primitives (da Silva, 1994)

Class and Object Layer

Classes and Objects were identified from the task decomposition work by da Silva (1994). Tasks were categorised as being either part of an implicitly expressed object, or part of another object not obviously identified. In accordance with Coad and Yourdon's (1991) guidelines, potential objects with more than one attribute were accepted, while potential objects with one attribute were regarded as being part of another object.

As an example, Table 2 shows the classes and objects generated from the analysis of subtasks of the generic task 'Monitoring Events'.

Subtask	Class and Object
Measure Production Level	Pasture Cover
Measure Production Level	Production Level
Measure Body Condition	Body Condition
Monitor Crop	Feed Supply
Do the Pregnancy Test	Mating Information

Table 2 - Class and Object generation

Further classes and objects were identified from the causal dependencies within the network of primitives. The following were selected as classes and objects as they are required by the system:

Grazing System
Climate
Herd
Season

The other components of causal dependencies present in the network of primitives were not included. These were:

Intake
Stocking Rate
Autumn Flush

The first two were acknowledged as attributes of the herd, to be included in the attributes layer. 'Autumn Flush' is the name for the increase in pasture growth rate straight after the seasonal rainfall from March onwards.

A decision was required to be made as to how to represent the livestock on the farm, both as a whole (herd) and on an individual basis. The Livestock network was constructed as shown in Figure 26.

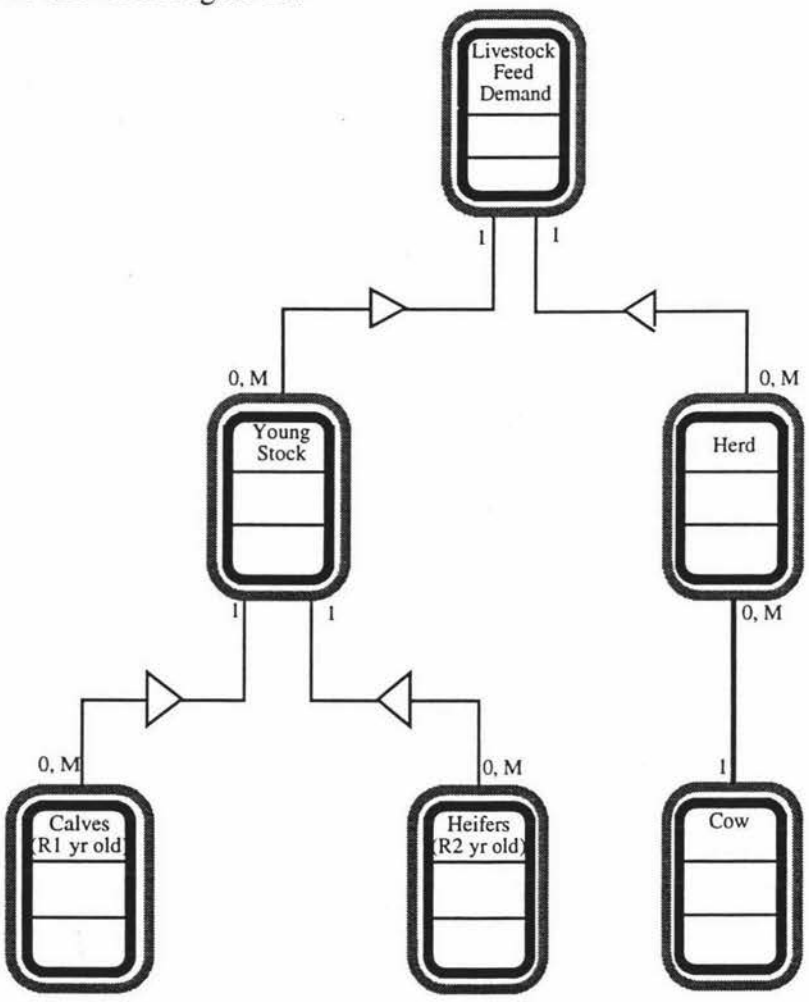


Figure 26 - Livestock Network

Young Stock was separated from the *Herd* class as cows during their first two years required considerably different attributes to mature cows.

A new class-and-object was created, *Cow*. This was to represent the relationship between the herd as a whole, and its components, the individual cows. This resulted in the following class and objects being chosen:

- Cow Condition
- Production_Level
- Mating_Information
- Climate
- Grazing_System
- Feed_Supply
- Herd
- Cow
- Season
- Pasture_Growth_Rate
- Pasture_Cover

Structure Layer

In an object-oriented context, structure is defined as an expression of the problem domain complexity pertinent to the system's responsibilities (Coad and Yourdon, 1991). The term "Structure" is used as an overall term, describing both generalisation-specialisation structure and whole-part structure. As an example of the construction of a structure, the feed supply network from the domain layer network of primitives (da Silva, 1994) took the form as shown in Figure 27.

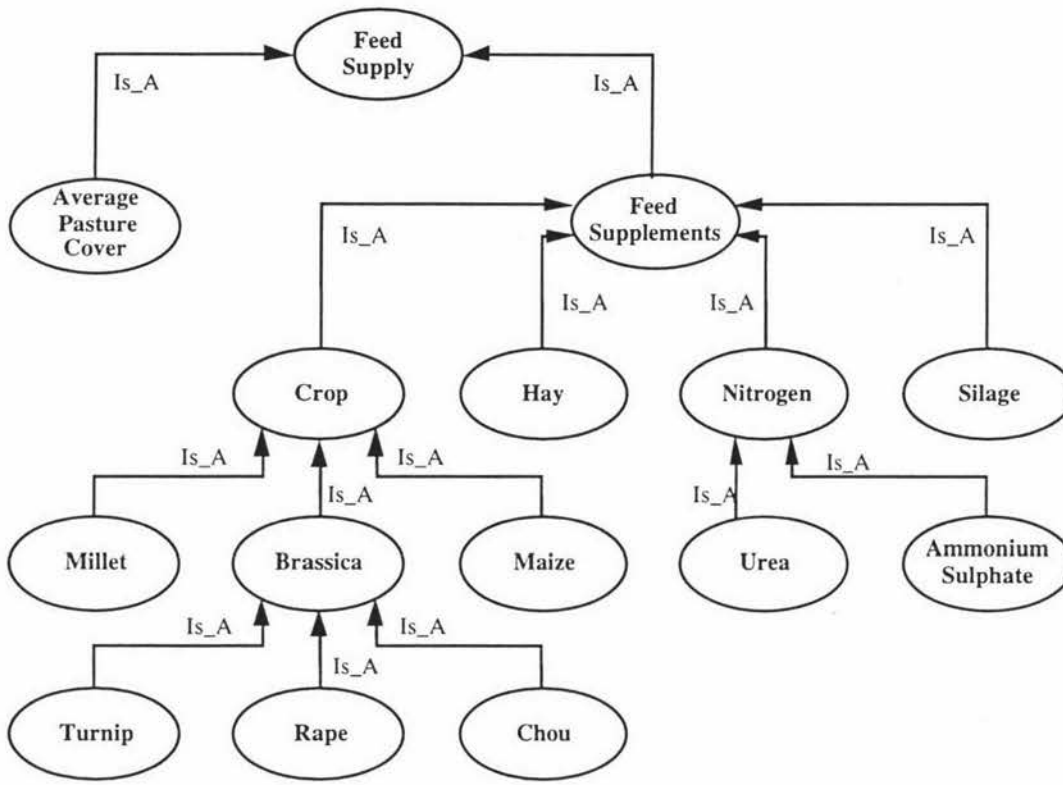


Figure 27 - Example of domain layer network of primitives
(da Silva, 1994)

Applying object-oriented analysis to this area resulting in Figure 28.

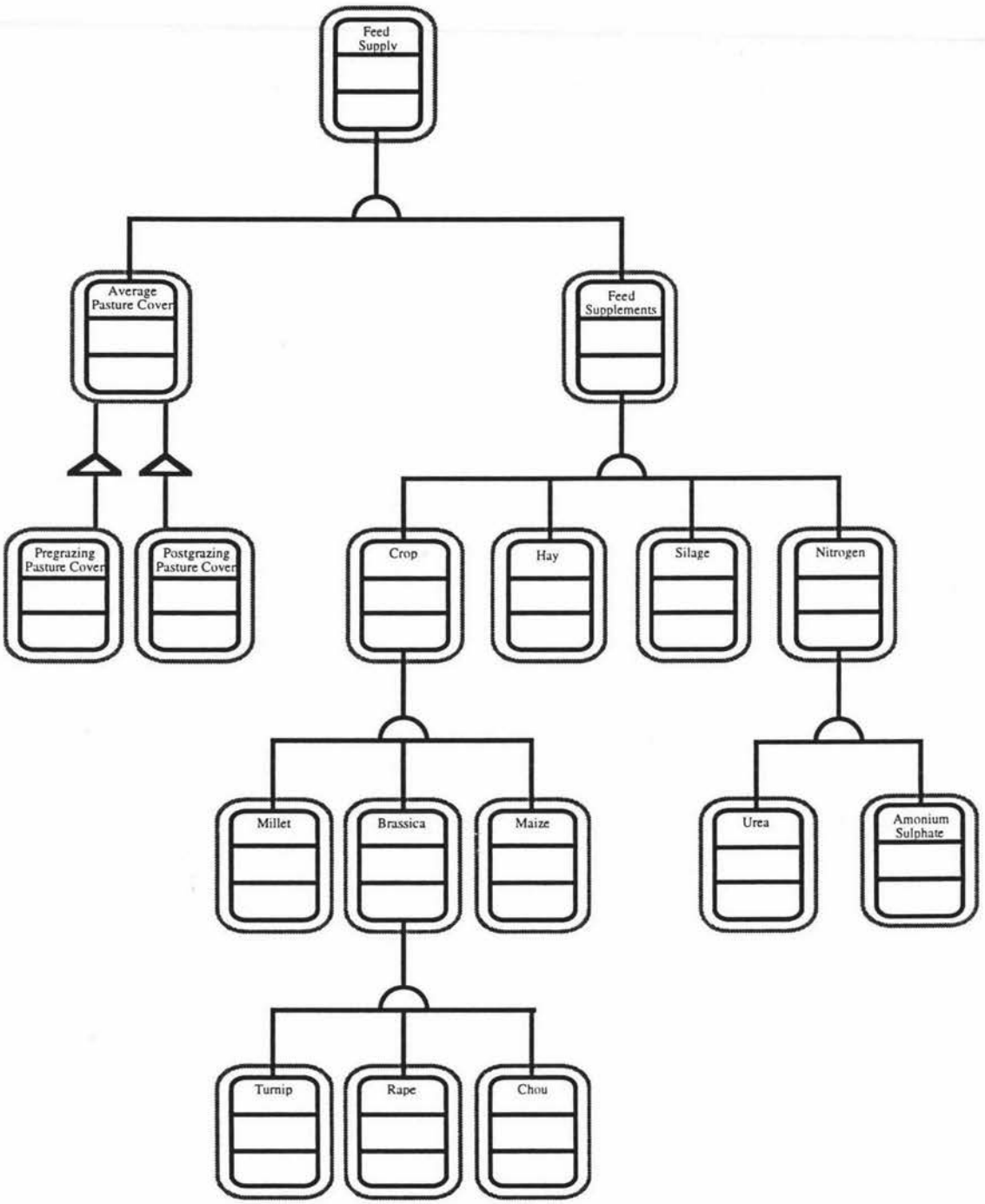


Figure 28 - Object-Oriented Feed Supply Network

The structure layer for the network of primitives introduces one relationship, that Pasture_Cover is part of Feed_Supply, shown in Figure 29.

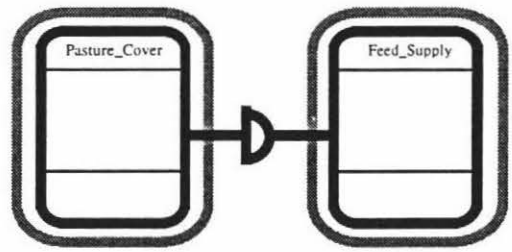


Figure 29 - Feed Supply - Pasture Cover Part_Of Relationship

Subject Layer

Subjects fundamentally show a very much simplified version of the system. From the top level of Classes and Objects, three main subject areas were determined. The division consisted of Land, Weather and Stock as shown in Figure 30.

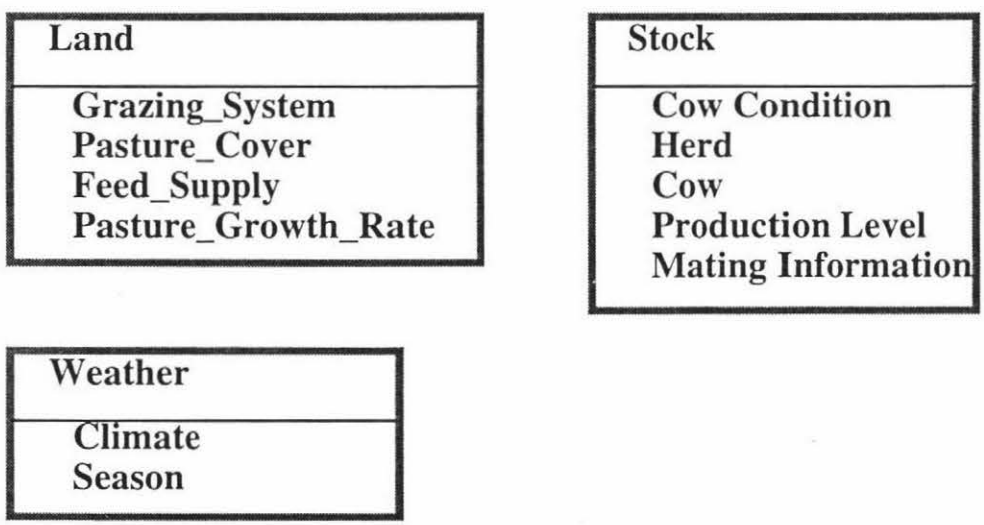


Figure 30 - Subject Layer

Attribute Layer

The attributes were generated from using both the data dictionary, and from examination of the task structures. Local attributes, or attributes required for intermediate working, were not included.

Figure 31 is an excerpt from the Task model generated by da Silva (1994) describing the subtask of the 'Monitoring Events' task *Measure Cow Condition.*

subtask: Measure Cow Condition (Condition Score)

goals: - Confirm that the cows are being well fed.

- Identify thin cows to be dried off.

task structure:

select(Condition Score ---> Condition Score of the cows)

select(Condition Score System---> Ideal Condition Score)

For herd

measure(Condition Score of the cows ---> Results of Cow Condition)

compare(Results of Cow Condition with Ideal Condition Score ---> Differences in Condition Score)

Figure 31 - Measure Cow Condition subtask
(da Silva, 1994)

From Figure 31, two attributes were generated for the class Cow Condition. They were *Ideal Condition Score* and *Current Condition Score*, shown in Figure 32.

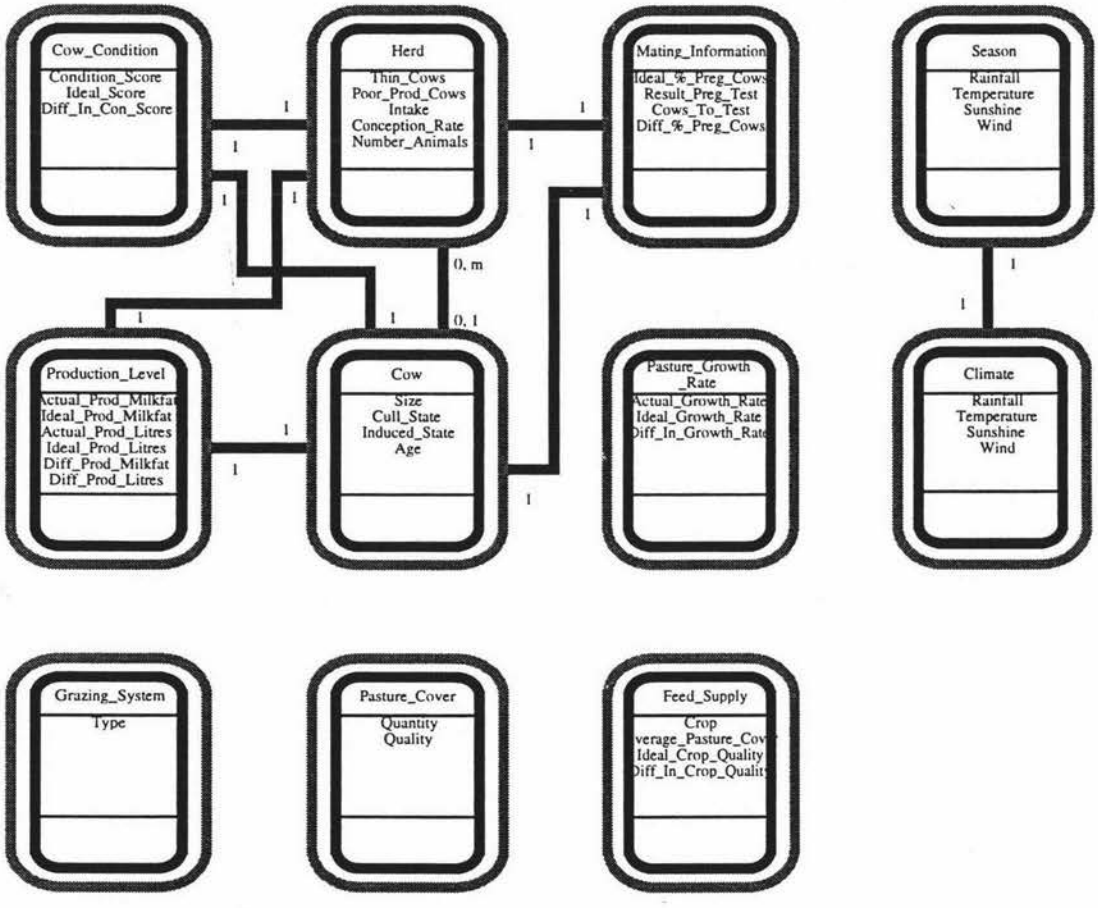


Figure 32 - Attribute Layer

Service Layer

In the service layer, services to which both classes and objects respond (or messages) are defined.

Figure 33 is an excerpt from the Task model describing the subtask of the Plan Management Decisions task *Identify Cull Cows.*

subtask: Identify Cull Cows

goals: - Maintain cow intake by selling marked cows.

- Improve average pasture cover.

task structure:

If *Identify Feed Shortage* is true **and**

Predict empty cows is true

Then

construct(Problem: Empty Cows; Strategy for empty Cows ---> Sell empty cows)

End If

If empty cows have already sold **and**

Identify Feed Shortage is true **and**

Identify Poor Producing Cows is true

Then

construct(Problem: Poor Producing Cows; Strategy for Poor Producing Cows ---> Sell Poor Producing Cows)

End If

If empty and poor producing cows have already sold **and**

Identify Feed Shortage is true **and**

Identify thin Cows is true

Then

construct(Problem: Thin Cows; Strategy for Thin Cows ---> Sell Thin Cows)

End If.

Figure 33 - Identify Cull Cows subtask
(da Silva, 1994)

From Figure 33, a number of services were determined. Identify feed shortage requires determining whether or not a feed shortage has been found. So, the service *Identified Feed Shortage* is added to the Feed Supply class. Similarly services are created for identified poor producing cows and identified thin cows. Within the herd class these return whether or not the herd's poor producing or thin cows have been identified. This is shown in Figure 34.

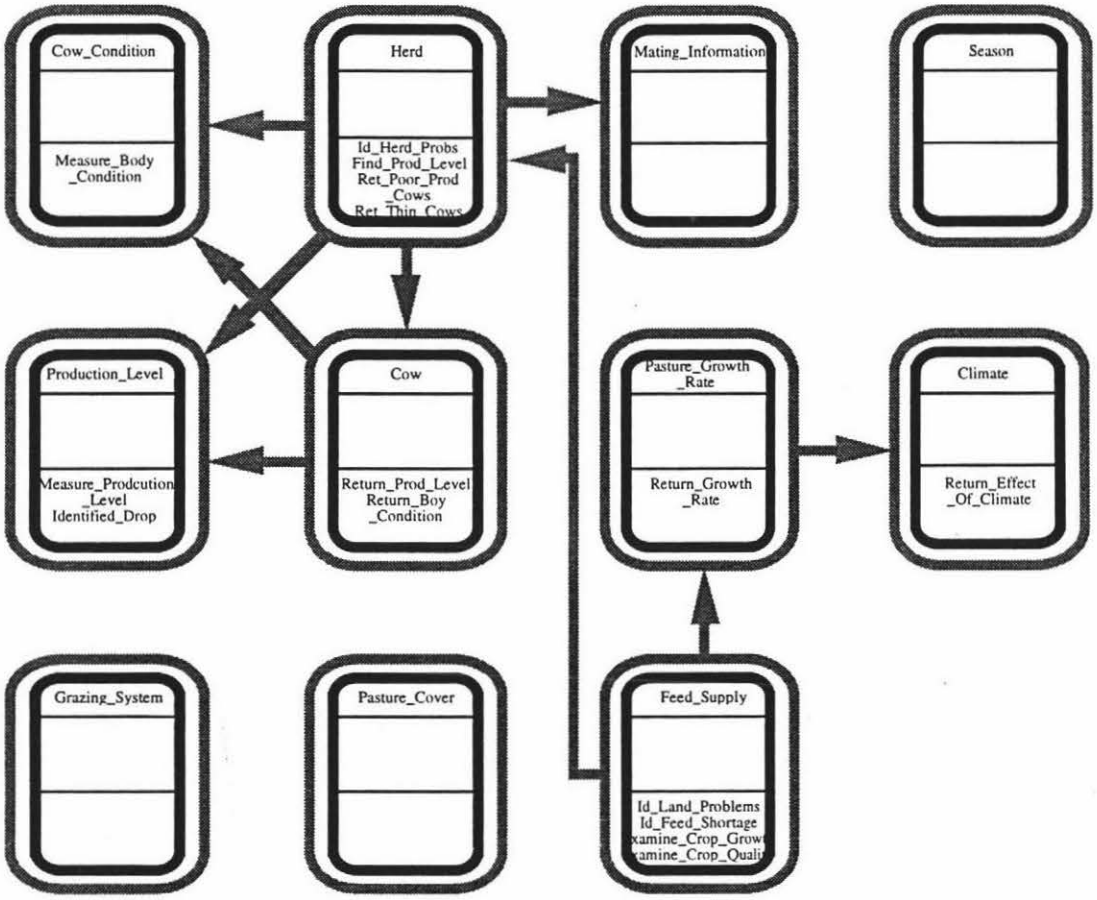


Figure 34 - Service Layer

When these layers are combined into one final model, the object-oriented model shown in Figure 35 is produced.

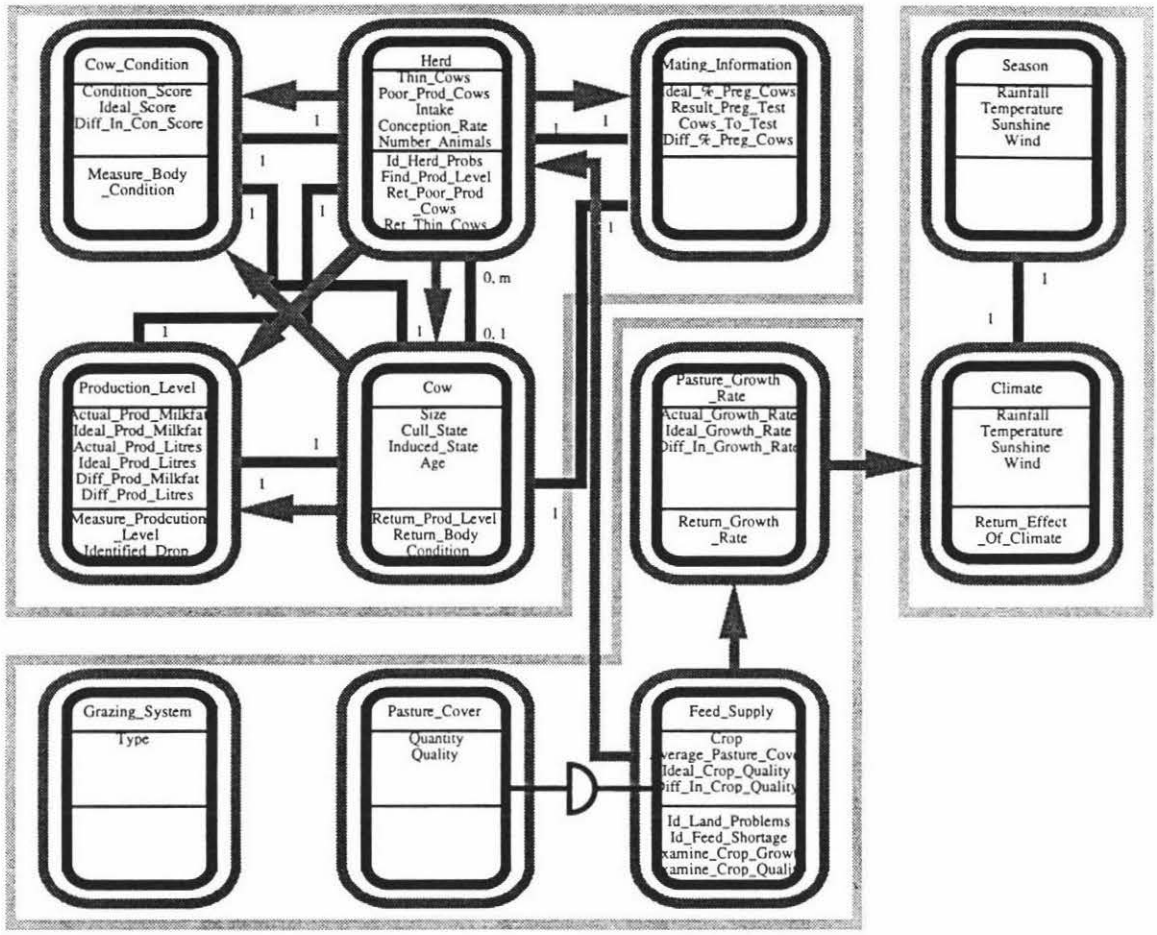


Figure 35 - Object Oriented Model of Network of Primitives

This analysis was carried out for all domain elements necessary for the functional prototype to be constructed. The remainder are shown in Appendix 2.

3.5 Conclusion

A problem domain with a genuine need for a knowledge based system has been established. The domain is that of farm management. Two styles of management have historically been shown to be required, that of short term or reactive management, and also long term planning.

Following Coad and Yourdon's (1991) approach, the object-oriented paradigm provided the knowledge representation form. Objects were shown to have a number of characteristics making them desirable, including abstraction, encapsulation, inheritance and communication with messages. Additional support of the choice of objects came from the mapping from the given model of expertise to knowledge representation model occurring quite naturally and logically. This was due to the similarities between the KADS methodology and the object-oriented approach.

The domain layer object-oriented analysis of the model of expertise produced an object-oriented task model and an object oriented view of the network of primitives. Within the domain layer was found all the potential services and attributes for the object-oriented model. The inference layer of the model of expertise is a high level abstraction of the problem domain. It does not contain the problem solving knowledge required for a full system; but instead contains the structure of the implemented system. The inference layer provides many of the classes and objects required for the knowledge representation model. The task model is concerned with the low level aspects of the domain, and provides in addition to further classes and objects the attributes and services for the object-oriented model. The contribution of each layer of the model of expertise is shown in Table 3.

KADS Methodology	**Knowledge Representation Model**
Domain Layer	Identifies potential Services and Attributes.
Inference Layer	Selects Classes and Objects at high level. Identifies Class and Object Hierarchy.
Task Layer	Selects further Classes and Objects. Selects Services and Attributes.

Table 3 - Contribution of Model of Expertise Layers to Knowledge Representation Model

The object-oriented analysis of the model of expertise, the output being in the form of the knowledge representation model, is now ready to be applied to the next phase of model development.

Chapter 4: Construction of Functional Prototype

Introduction

With the knowledge representation completed, the knowledge contained within this model (and correspondingly the model of expertise) requires validation and verification. Functional prototyping has been included within the KADS methodology without ever being examined in any depth for its model validation potential. The procedure selected to achieve model validation is through the construction of a functional prototype. A self-contained part of the problem domain which can be modelled completely without requiring knowledge which has not been validated is identified. This chosen area of the problem domain is expressed within the knowledge representation model. The reasons for the choice of functional prototyping are given in the first section of this chapter. The methodology used for the functional prototype development is outlined in the second part of this chapter.

The knowledge representation model is used as the basis of the implementation of the functional prototype, the third phase of Nunamaker's methodology. A brief description of the software used for the prototype construction is presented at the start of the third section. The construction of the functional prototype took place over a period of months. A number of design decisions were made at an early stage to determine which areas of the problem domain were to be prototyped. An explanation of these decisions and the rationale behind them is given at the completion of this section.

4.1 Functional Prototyping vs Protocol Analysis

Within the KADS methodology, the model of expertise can be validated by either protocol analysis or functional prototype development. Protocol analysis, performed by segmentation, coding, matching and naming (see section 2.3.3), provides a method for validation by comparison of encoded models constructed by knowledge engineers working independently. This is best suited to the knowledge engineers working in parallel, that is at the same time on their models. Functional prototyping was chosen as the validation technique for this project, making use of the existing model created by da Silva (1994). Functional prototyping takes the work of a knowledge engineer up to the model of expertise phase, and creates a prototype containing the knowledge. This knowledge can then be verified and validated against the model of expertise. The difference between the protocol analysis and functional prototyping methodologies is demonstrated in Figure 36.

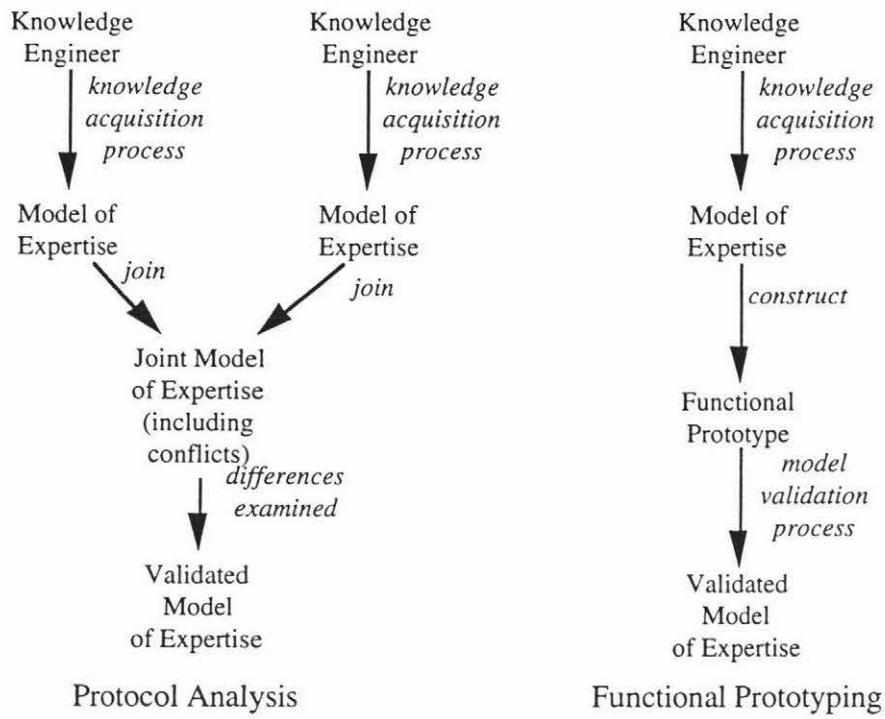


Figure 36 - Techniques of Validation

4.2 Applying the Prototyping Lifecycle

Duffin's model of the prototyping life cycle (presented in Figure 5) can now be applied. The corresponding aspects specific to this project and their generic model equivalents are listed in Table 4.

Definition of Aims	To implement a functional prototype for validation of the model of expertise knowledge
Design of Prototype	Knowledge representation model
Implementation	Construct of the functional prototype
Use Of Prototype	Domain expert's examination of the system
Evaluation	Critical analysis of the constructed prototype

Table 4 - Stages of the prototyping development life cycle

Clearly the first two parts of the life cycle have been fulfilled. This chapter contains the result of the middle stage of this iteration around the life cycle.

The functional prototype development process was modelled on a variation of Sommerville's model of evolutionary development shown in Figure 37 (Sommerville, 1989).

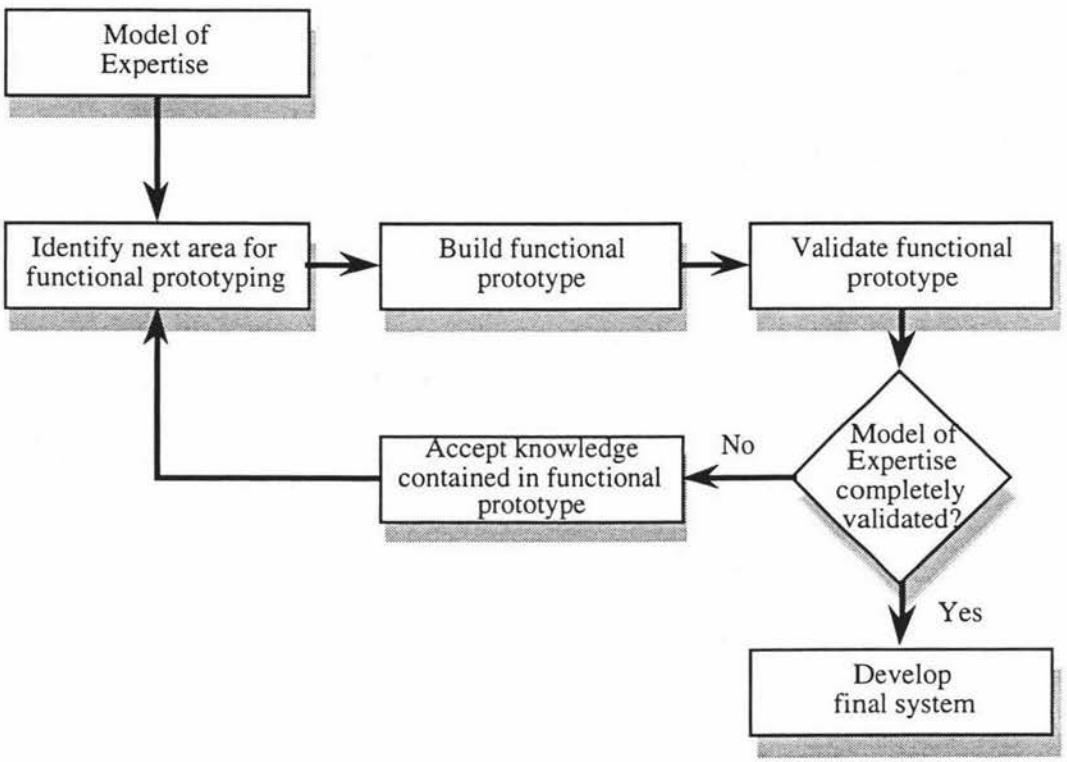


Figure 37 - Model of expertise validation through use of functional prototyping (Adapted from Evolutionary development, Sommerville 1989)

From the given model of expertise and subsequently constructed knowledge representation model, areas were identified which could stand alone for functional prototyping purposes. The prototype was then constructed and tested. Problems identified at this point could be classified as either generated by the model of expertise (and therefore the knowledge representation model) or the prototype developer.

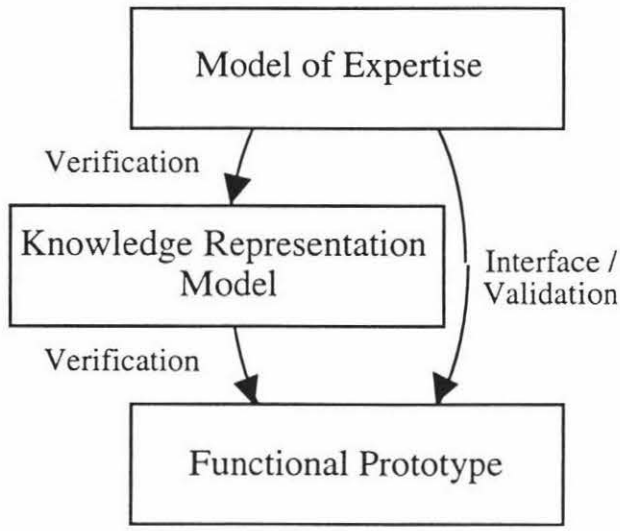


Figure 38 - Examination of transitions between models

Problems involving the model of expertise require refinements to be made to it; problems of the second type are caused by either the implementor misunderstanding the knowledge contained within the model of expertise or knowledge representation model, or by the implementor not expressing the knowledge in a form easily understood by the domain expert. The three phases of error detection are shown in Figure 38- the verification process, and validation process and the interface design analysis. Misunderstanding the model of expertise or knowledge representation model results in an error being detected through the verification phase. This can be overcome by the further research and understanding of the previous model. Failure to express the knowledge in a form understandable by the domain expert is an interface design problem. This requires further work into the human- computer interface issues applicable to this domain. A problem within the model of expertise is picked up in the validation process, when it is realised the problem domain is not accurately represented by the functional prototype.

If the prototype is accepted as being valid, the area of knowledge from the model of expertise that has been simulated is accepted as correct. Subsequently developed areas of the model of expertise may be examined, building on the knowledge previously accepted as being correct, or developed in parallel.

4.3 Application Functionality Overview

The main areas considered within the bounds of the developed prototype were that of the production level and feed budgeting. These were chosen as both are important fields for Dairy Board Consulting Officers, the end users of the system, who require the ability to both compare and analyse various farmers' performances as well as to examine a farmer's performance in the current season against either their historical figures or their projected results. The application was designed with this in mind.

4.3.1 Software Package Overview

Kappa-PC (version 2.2) is an object-oriented windows based applications development system, which allows for the construction of applications in a graphical environment.

Elements within a domain are referred to as objects. In Kappa, objects are either classes or instances within the class; anything within the system is called an object. This is a different meaning to objects within the Coad and Yourdon object-oriented methodology (Section 3.2.2). In Figure 39, the difference is shown. The Coad and Yourdon notation on the left shows the class-&-object 'Vehicle'. In the Coad and Yourdon approach, instance is a synonym for object. The figure depicts Vehicle as a class which can have instances. (It is possible for a class to have no instances; this is called an abstract class). 'Car' and 'Plane' are instances (or objects) of the class 'Vehicle'. Within the Kappa-PC program, anything that is created within the system is referred to as an object. A class is separate from an instance, as with the Coad and Yourdon approach, but an object can be either a class or an instance. Other differences in terminology include:

- Attribute (Coad and Yourdon) = Slot (Kappa-PC)
- Service (Coad and Yourdon) = Method (Kappa-PC)

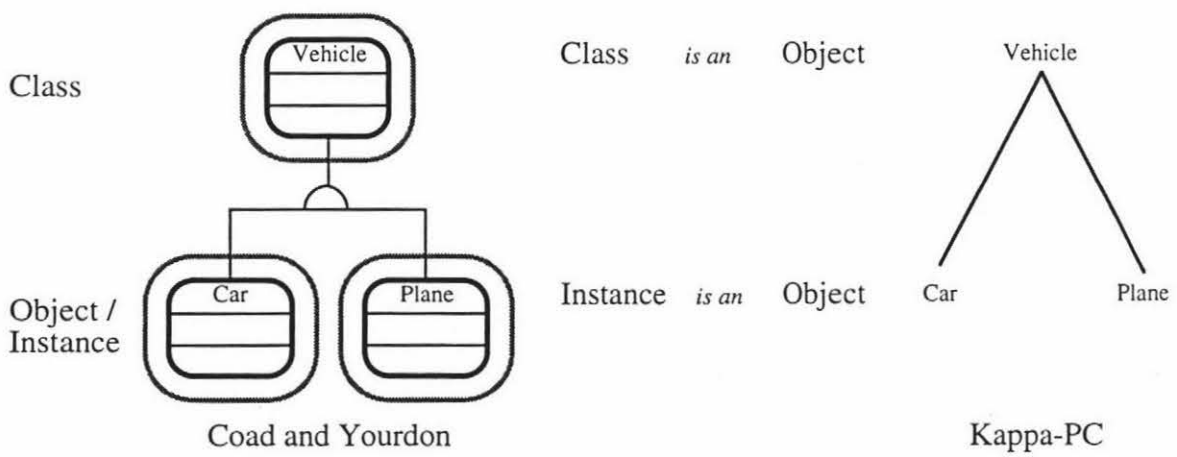


Figure 39 - Coad and Yourdon Notation and Kappa-PC Notation

The object-oriented programming style allows for the construction of methods which specify the operations the object can perform. Within Kappa-PC, attributes of objects exhibit additional features associated with frames in the form of monitors. These monitors, which are equivalent to daemons in a frame based system, can be triggered by various actions:

- If_Needed.
- When_Accessed.
- Before_Change.
- After_Change.

These can invoke methods attached to the instance.

Elements of the graphical user interface (GUI) may have functions attached to them, for example a button may trigger a rule or another call function. Similarly, object slots within the GUI can have automatically triggered methods, such as updating a value when a combination box alters its choice.

Quattro-Pro for Windows was selected to provide the worksheets from which data was obtained. As farmers are used to recording their results on a worksheet, it was felt that it would be preferable to create links directly from the worksheets through the use of dynamic data exchange (dde's, also known as hotlinks). However, the version of Kappa-PC used does not support hotlinks from other applications. The next version will, allowing the ability to create hotlinks. Kappa could act as a sender, but not as a receiver. This meant figures could be altered within Kappa's memory, but changes could not be made to the worksheet.

The only explanation facilities supported within Kappa-PC are those applied to rules. There is no automatic transcript generation facility available. This meant that a simple transcript writing function was written to record inferences, options selected and other user actions. The transcript file could be annotated by the user, cleared or viewed. Viewing was possible through the use of Microsoft Word for Windows. An example of the transcript generation is shown in Appendix 3.

4.3.2 The Knowledge Representation Model and Implementation Issues

Moving from the knowledge representation model to the functional prototype involved selection of areas of the model of expertise to prototype. The examples looked at were that of examination of the production level and feed budgeting. These were key elements identified in the knowledge representation model of the network of primitives. Feed budgeting is derived from analysis of the feed supply. Production level and feed supply were self contained within their subject areas, as shown in Figure 40 and Figure 41.

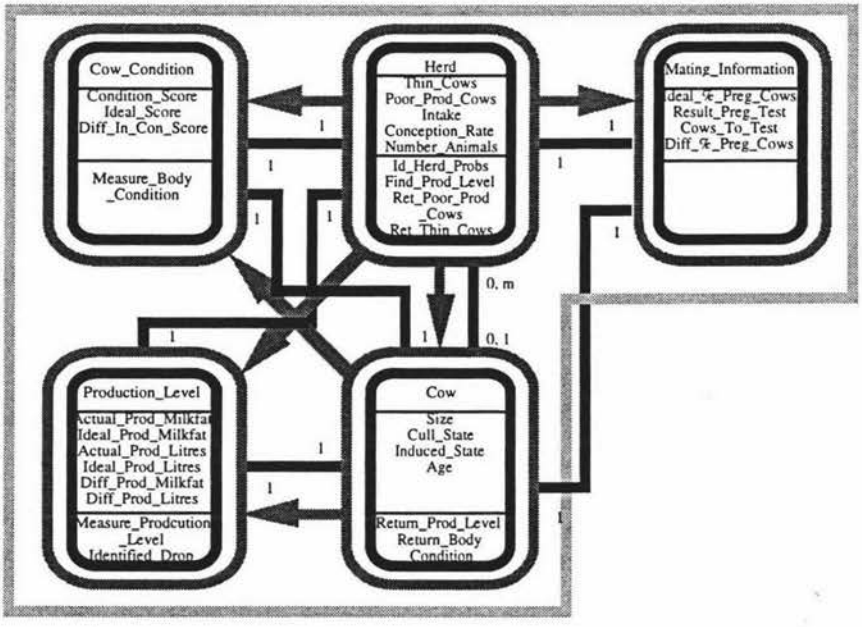


Figure 40 - Stock subject featuring the Production Level class.

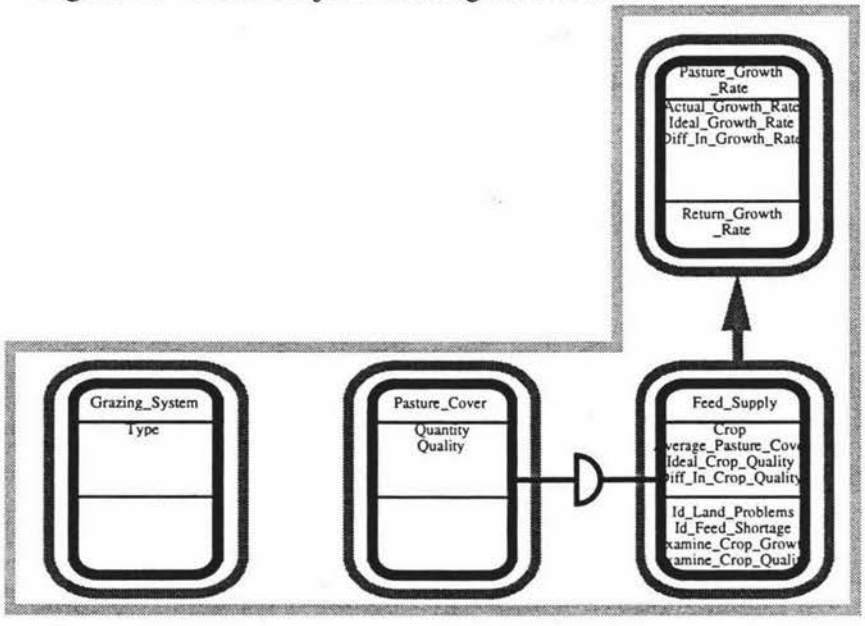


Figure 41 - Land subject featuring the Feed Supply class.

The knowledge representation model supplies the top level aspects of the object hierarchy within the functional prototype. This includes the services and attributes. The code implementing the services was written when constructing the functional prototype after examining the task layer of the model of expertise.

4.3.2.1 Object-Oriented Feed Budgeting

The four components of the generic task decomposition- monitoring, prediction, planning, and diagnosis- each had a role to play within the feed budgeting domain. A comprehensive example of the transition from knowledge representation model to functional prototype is given for the 'diagnosis' area. The knowledge representation model has produced the framework for the functional prototype: now the internal workings are added.

The 'Monitoring Events' component of feed budgeting involved examination of pasture cover. As the point of feed budgeting is to ensure there is sufficient feed for the herd, the potential problem being examined is that of a feed shortage. In accordance with the object-oriented approach, for example, a method was created for feed shortage identification, and named identify_feed_shortage. A parameter for the lowest acceptable pasture cover level when taking into account the effect of available supplements was established and comparison made between the figure in reality and the ideal value. If there was an unacceptable deficit identified, then an indication of a potential feed shortage problem exists.

'Prediction of Events' could be used on the current figures to identify a future problem, for example a potential fall in production. This could be determined by an examination of the herd's dry matter intake over a certain area.

The 'Diagnosis' task is invoked if a problem is detected either at present or foreseen in the future. The diagnosis task determines what the cause of the problem is. The result of this analysis is the input for the planning management decisions activity. An example of the implementation of a task is moving from the task model of 'Identify Feed Shortage' to the *Diagnose_Problem* instance *Feed_Shortage_Problem*. The relevant part of the subtask within the model of expertise is given in Figure 42. (There is also a contingency for the period being examined in July, which is not within the bounds of this project).

subtask: Identify Feed Shortage (Between April and August)

goals: - Maintain cows being well fed

- Make sure that there will be enough feed for the cows at calving time and early lactation

task structure:

Measure Pasture Cover

If Average Pasture Cover is below 1400 kg of Dry Matter /hectare and Supplements are not enough to cover the deficit

Then

classify(Difference in Pasture Cover --> Feed Shortage)

End If.

Figure 42 - Identify Feed Shortage Task layer Decomposition
(da Silva, 1994)

From this, Figure 43 was generated during the construction of the knowledge representation model.

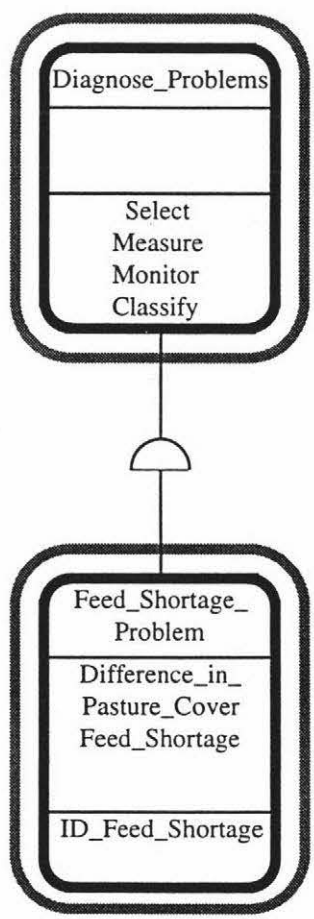


Figure 43 - Diagnose_Problems:Feed_Shortage_Problem relationship

The service *ID_Feed_Shortage* (shown in Figure 44) is structured as follows. The *Pasture_Cover_Reading: Average_Pasture_Cover* attribute is instantiated when a message is sent to the service Pasture_Cover_Reading. From this, it was diagnosed whether there is feed shortage problem or not. This result is stored in the Boolean slot *Feed_Shortage*.

```
SendMessage(Pasture_Cover_Reading,Measure_Pasture_Cover);
If ( ( Pasture_Cover_Reading: Average_Pasture_Cover) < 1400 )
        Then    {
                Trans_And_File( Analysis_Text, "A feed shortage has been identified,
                        as the average pasture" );
                Trans_And_File( Analysis_Text, "level, in addition to the
                        supplements, is still not sufficient.\n" );
                Feed_Shortage_Problem:Feed_Shortage = TRUE;
                }
        Else
                Feed_Shortage_Problem:Feed_Shortage = FALSE;
```

Figure 44 - Code for Service *ID_Feed_Shortage*

If a problem concerning the feed supply has been observed, then a 'Management Planning Decision' must be made to determine which option is to be selected. Each choice impacts on the farm in a different manner. A message is sent to each component of the Planning Management task, triggering a method to examine both the applicability of the option and the priority the user has assigned it. Once an acceptable combination has been identified, the selected option is returned together with the reasoning behind its choice.

The guide used for the numeric aspects of the program was the 'Dairy Farm Feed Budget' worksheet prepared by the Massey University Agricultural and Horticultural Systems Management Department. The template took the form as shown in Figure 45.

DAIRY FARM FEED BUDGET FOR THE PERIOD ENDING
FARMER ADDRESS
Massey University Agricultural & Horticultural Systems Management
For a monthly feed budget insert the number of the starting month in B10.
For a weekly or 10 day feed budget enter 25 or 37 in B10 respectively.

EFFECTIVE AREA:
INITIAL COVER:
STARTING PERIOD:

PERIOD:

FEED DEMAND AND FEED SUPPLY
Growth rate/Day:
Intake/Day:
Difference/Day:
Supplements (enter total)
Hay:
Silage:
Nitrogen:

FINAL COVER:

ANIMAL INTAKE:
Milking Cows
Intake/Head/Day
Dry Cows
Intake/Head/Day
Heifers 12-24 mth
Intake/Head/Day
Heifer calves
Intake/Head/Day
Bull calves
Intake/Head/Day
Bulls
Intake/Head/Day
Other
Intake/Head/Day

TOTAL DEMAND/DAY

DAYS/PERIOD:

Figure 45 - Feed Budget Template
(Dairy Farm Feed Management Budget, Massey University)

Examination of one farmer's performance requires three worksheets to be initially selected:

- the farmer's original plan.
- the farmer's record of what has actually happened.
- the farmer's original plan modified to include the figures that have been recorded since the original plan was derived.

When adding a farmer for comparison purposes, objects were created with unique identifiers composed from a combination of the farmer's name and the number of the period. These were created for implementation reasons, being required by the Kappa-PC program, and called implementation objects. For example, 'Smith2' would represent farmer Smith's second recorded period. An example of part of such a hierarchy is displayed in Figure 46, where two farmers have been selected, each having figures for four periods. 'Doris' and 'Joe' are sub-classes of Hist (for History), and Doris1, Doris2 etcetera are instance of the class with instantiated attributes.

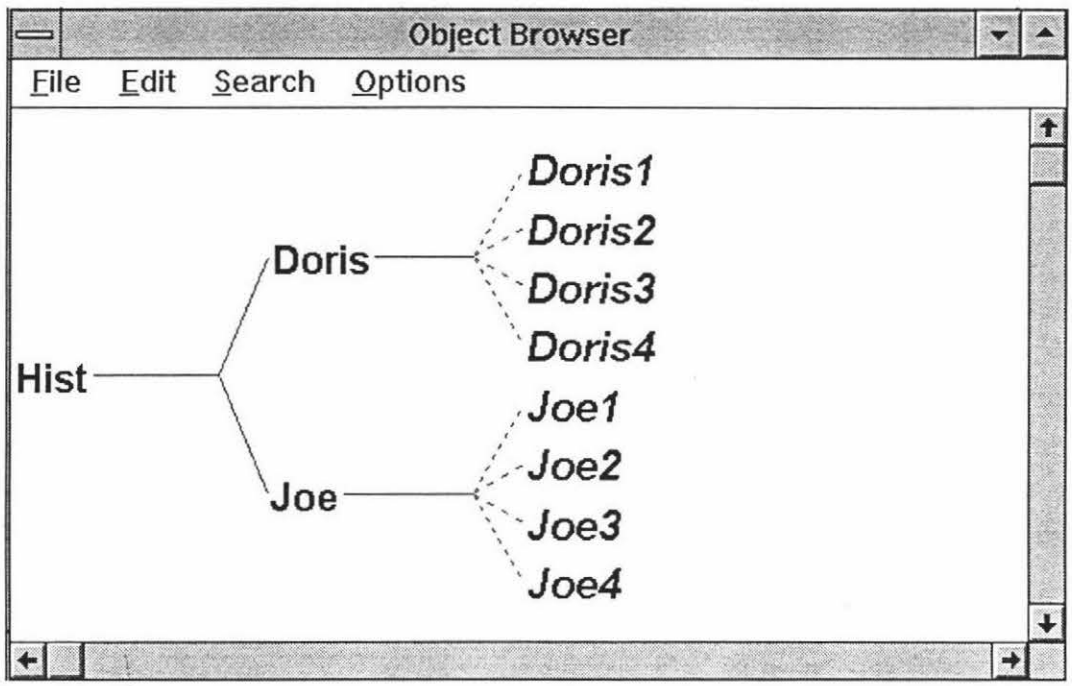


Figure 46 - Hierarchy containing two farmer's figures for comparison

By examination of the worksheet's formula in Figure 45, and after consulting the knowledge representation model, the following fields were identified as being worthy of inclusion for prototyping purposes:

- Number of Animals
- Consumption (kg DM/cow/day)
- Length of Round (days)
- Supplements (kg DM)
- Growth Rate (kg DM/cow/day)
- Grazing (ha/day)
- Initial Cover (kg DM)
- Final Cover (kg DM)

Some of these figures could simply be read from the worksheet and only modified by the user directly (for example grazing) whereas others, after being read from the worksheet, required a dynamic link to other fields (for example final cover is a combination of nearly all of the other fields).

If examination was taking place on just one farmer, an additional suffix of either R, OP or M was added to denote reality, predicted or modified worksheet figures as shown in the bottom part of Figure 47.

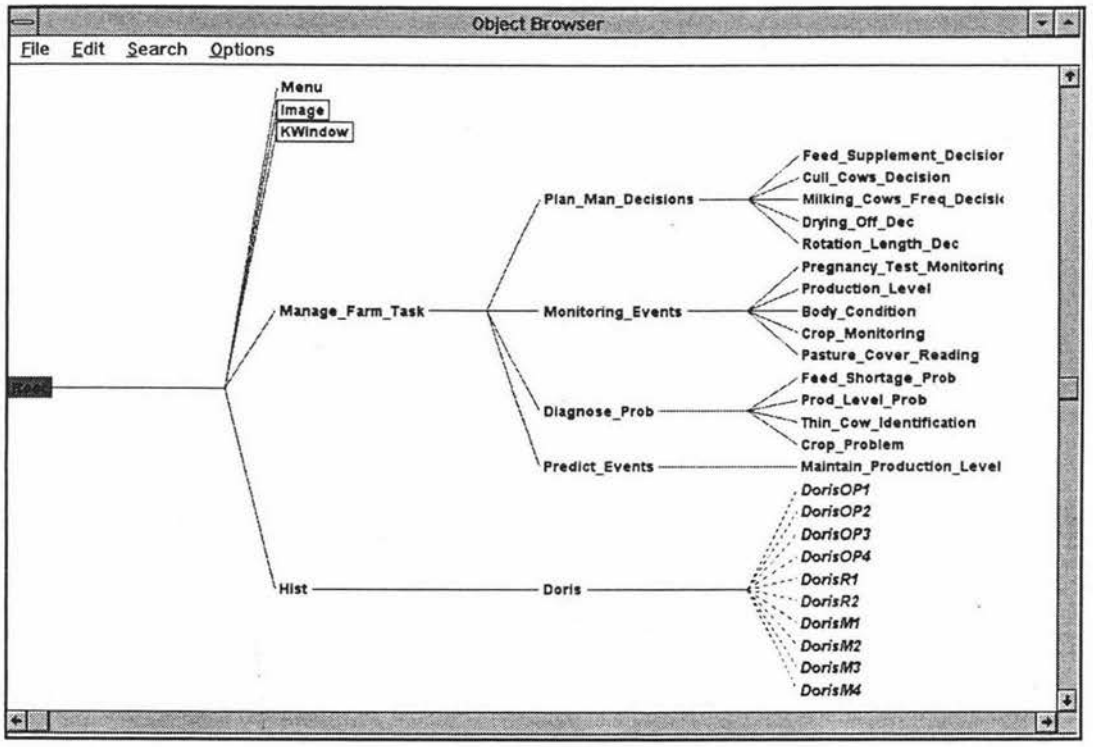


Figure 47 - Hierarchy after selecting one farmer's worksheets

Each of these objects has the attributes necessary for analysis as defined previously (consumption, length of round etc.) and is instantiated to the worksheets values as shown in Figure 48.

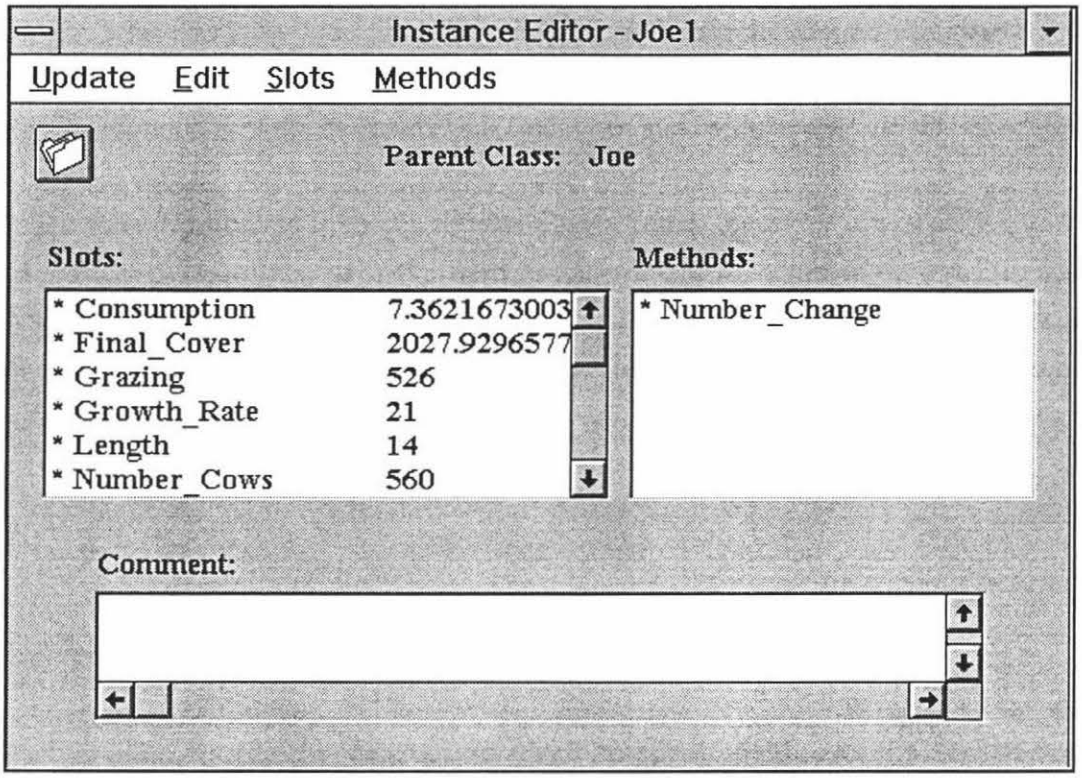


Figure 48 - Instance with instantiated values

When analysis takes place, the values are transferred into the appropriate part of the decision hierarchy from the object representing the selected period. For example, *Length of Round* is used by *Rotation Length decision*, and *Initial and Final Cover* used by *Pasture Cover Reading* (shown in Figure 49).

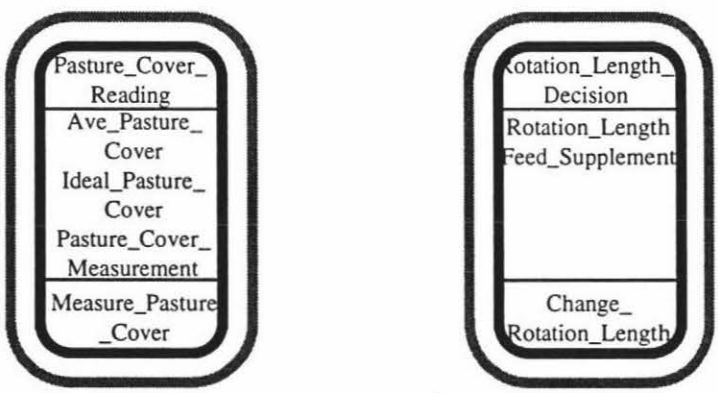


Figure 49 - Objects instantiated by data from object representing current period

4.3.2.2 Object-Oriented Production Level Analysis

Monitoring of the production level figures involves examination of both the herd's production level and the herd's overall condition score. The herd's production level for both milkfat and litres (per cow) is calculated, and these figures are compared with an ideal (or desirable) amount, taken from historical data. The condition score reflects the physical condition of the herd, which is calculated and compared to known parameters.

Diagnosis of problems involves determining whether there are irregularities with the milkfat production rate, or if the herd has a problem reflected in its condition score. Production level problems arise if either the production level falls below a specified amount, or if the production falls at an unacceptable rate over a period of time. Acceptable condition scores vary with time of year, but a comparison can be made to see whether the score is too low for the most recent period.

Planning and management decisions follow a similar pattern to that for feed budgeting decisions. The most appropriate choice needs to be made through a combination of the users priorities and general preferences.

Deriving the figures for production level examination follows the format for the worksheet prepared by I.M. Brooks of the department of Animal Science, Massey University as shown in Figure 50.

COWYIELD:BY I.M.BROOKES. DEPT. of ANIMAL SCIENCE, MASSEY UNIVERSITY			
NAME		CALVING MONTH: (1st Half =1, 2nd Half -2):	
BREED LACTATION LENGTH BREEDING INDEX		CON SCORE LWT POST CALVING	
ENTER DATA THEN PUSH F9		DRY OFF NEXT CALVING NEXT CALVING	
NOTE: MILKFAT = SOLIDS / 1.71		LWT(K) CS SOLIDS DM INTAKE	
DM INTAKE MONTH DAYS		TOTAL (Kg)	

Figure 50 - Cowyield worksheet template
(Brookes, Dept of Animal Science, Massey University)

Objects representing each row of figures are created for analysis purposes. These objects have the following attributes:

- Condition Score
- Days
- Dry Matter Intake
- Liveweight
- MD (A measure of metabolism related to dry matter intake)
- Milkfat
- Solids

These objects are used as a dynamic store of figures required for analysis. They are created when the appropriate worksheet is selected, and destroyed when the analysis screen is exited.

As an example of the transition from model of expertise to functional prototype, Figure 51 gives the Measure Production Level subtask (da Silva, 1994).

subtask: Measure Production Level

goals: - Confirm that the animals are being well fed.

- Make sure the production goals are being achieved.

task structure:

Repeat every day for herd

select(Production Level --> Production/cow/day)

select(Production System --> Ideal Production)

Measure total of litres

Measure kg of milkfat

Divide total of litres by numbers of cows

Divide total of milkfat by number of cows

measure(Milkfat/cow/day --> Results of milkfat/cow/day)

measure(Litres/cow/day --> Results of litres/cow/day)

compare(Ideal Production with Results of milkfat/cow/day --> Difference in Production Level)

compare(Ideal Production with Results of litres/cow/day --> Differences in Production Level)

End.

Figure 51 - Task layer subtask Measure Production Level.

In the object-oriented knowledge representation model, Figure 52 was incorporated into the *Production_Level* class-&-object. The service was the first component of the class to be identified, and called *Measure_Production_Level.* After this, the attributes were identified: *Actual_Production_Milkfat, Ideal_Production_Milkfat* etcetera. This is shown in Figure 52.

Figure 52 - Production Level Class-&-object

The contents of the service *Measure_Production_Level* were identified from the task layer of the model of expertise. This was constructed on paper in pseudocode, then converted into Kappa-PC scripting language. Figure 53 contains an excerpt of code from the other service for Production_Level, *Identified_Drop*. This examines production for the last two periods, and establishes whether there is a problem with the production level. In the code, the reference to Figures refers to an implementation specific object created for the purpose of storing the data currently under evaluation.

```
Figures:TheText = "During the last two periods milkfat production ";
If ( ( Figures:Last_Milkfat - Figures:Second_Last_Milkfat ) == 0 )
        Then ( Figures:TheText = Figures:TheText # "remained constant at "
                # FormatValue( "%2.6f\n", Figures:Last_Milkfat ) # "\n" )
        Else    {
                If ( ( Figures:Last_Milkfat - Figures:Second_Last_Milkfat )> 0 )
                        Then ( Figures:TheText = Figures:TheText # "increased " )
                        Else Figures:TheText = Figures:TheText # "decreased ";
                Figures:TheText = Figures:TheText # "from " # FormatValue( %2.6f",
                Figures:Second_Last_Milkfat ) # " to ";
                Figures:TheText = Figures:TheText # FormatValue( "%2.6f",
                Figures:Last_Milkfat ) # "\n";
                };
Trans_And_File ( Analysis, FormatValue( Figures:TheText));
ResetValue( Figures, TheText );
If ( ( Figures:Last_Milkfat - Figures:Second_Last_Milkfat )> 0 )
        Then Figures:TheText = Figures:TheText # "The Milkfat production is
                increasing, which is good.\n\n";
If ( ( Figures:Last_Milkfat - Figures:Second_Last_Milkfat ) < 0 )
        Then    {
                Figures:TheText = Figures:TheText # "This decrease was at a
                                rate of ";
                If Slot?( Figures, Delta_Milkfat )
                        Then DeleteSlot( Figures, Delta_Milkfat );
                If Slot?( Figures, Delta_Days )
                        Then DeleteSlot( Figures, Delta_Days );
                If Slot?( Figures, Decrease_Rate )
                        Then DeleteSlot( Figures, Decrease_Rate );
                MakeSlot( Figures, Delta_Milkfat, Figures:Last_Milkfat-
                        Figures:Second_Last_Milkfat );
                MakeSlot( Figures, Delta_Days, Figures:Last_Days -
                        Figures:Second_Last_Days );
                MakeSlot( Figures, Decrease_Rate, Abs(Figures:Delta_Milkfat /
                        Figures:Delta_Days ) );
                Figures:TheText = Figures:TheText # FormatValue( "%1.6f",
                        Figures:Decrease_Rate ) # " kg MF/cow/day.\n";
                If ( Figures:Decrease_Rate > 0.01 )
                        Then ( Figures:TheText = Figures:TheText # "This is a cause
                                for concern.\n" )
                        Else Figures:TheText = Figures:TheText # "This rate is
                                acceptable.\n";
                };
Trans_And_File( Analysis, FormatValue( Figures:TheText ) );
```

Figure 53 - Sample code from *Identified_Drop* service

4.3.3 Interface Description and Program Walkthrough

The analysis text generated for the user to view is the same as that written to the transcript file. It was felt that this was an effective way of conveying the reasoning behind decisions to the domain expert to examine for validation purposes. In the case of feed budgeting, this allowed for a form of 'what if' exploration to take place, to see the effects of changing figures at various points in time. For examination of the production level, the analysis took place automatically on the worksheet selected by the user.

There are three main screens from which users could interact with a view to analysis and prediction of the season's figures. These were Feed Budgeting, Milkfat Production and Farm Management Preferences

The feed budgeting analysis takes two different forms within the constructed prototype. After consultation with people involved in the training of DBCOs, it was decided there should be a mechanism for providing both comparison of individual farmers from within a similar geographical region, and the ability to compare a farmers original plans over a period of time with what has happened in reality.

Both of these were suggested as best implemented with a dynamic nature, rather than purely static, allowing for the examination of alternative possibilities through the alteration of figures. This is termed "What if" analysis.

A graphical representation of individual fields was also included. As only one set of figures could be displayed at a time, the graph allowed for trends over time to be identified, and comparisons made.

When comparing the performance of various farmers, there were three modifiable options for the display of figures and the graph. These were available periods, available farmers and graph subjects. These options were all modifiable through the use of combination boxes. Periods were given meaningful names from the worksheet read in, such as 'Early Jan' and 'Late Feb'. The 'Available Farmers' combination box provided a list of the available (if any) farmers. Functions were written so farmers could simply be added or removed from the comparison simply by selecting the appropriate farmers name or worksheet.

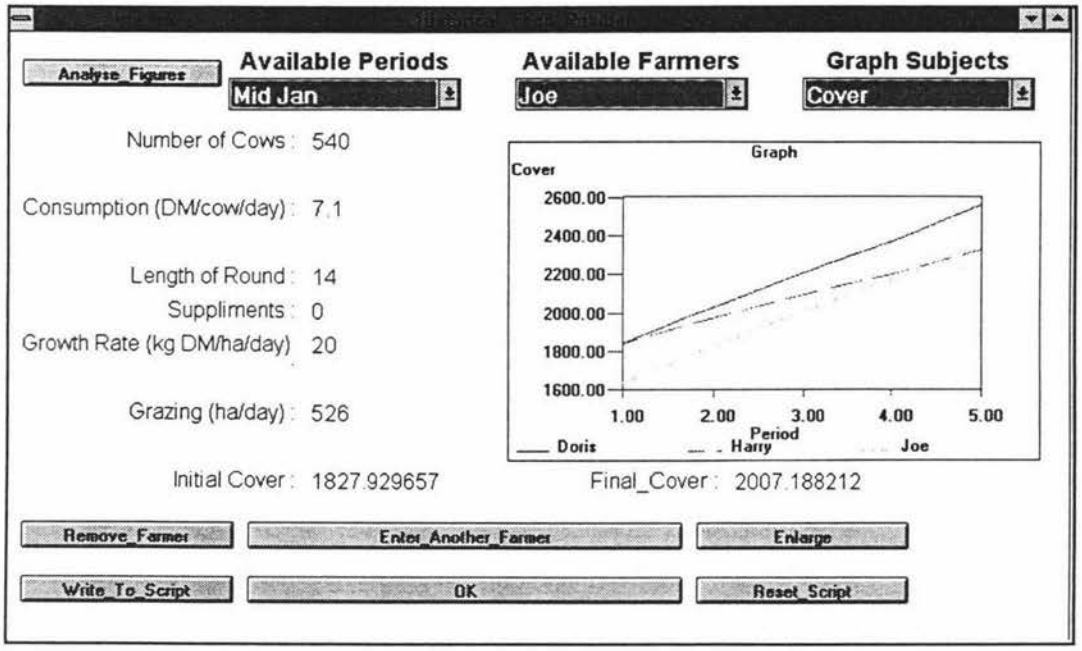


Figure 54 - Comparison of Farmers

A maximum of six farmers can be displayed at once. The graph can be enlarged to allow closer inspection of the figures. Any of the seven different fields can be selected to view. (Initial Cover and Final Cover are incorporated into the Cover option). Figure 54 contains three farmers, and is displaying the cover values over time.

Any of the figures in the fields displayed can be changed, with the exception of 'Final Cover' as this is purely derived from the other figures. Any changes made are used immediately to recalculate both figures concerned with the current period and subsequent periods, and these changes are shown on the graph (if the subject of the graph is altered).

For examination of one farmer, three worksheets are specified, and a similar screen to compare farmers performance is displayed. The main difference is that instead of there being a choice as to which farmer to view, there is a choice of which plan to view (that is original, reality or modified original). This screen is shown in Figure 55.

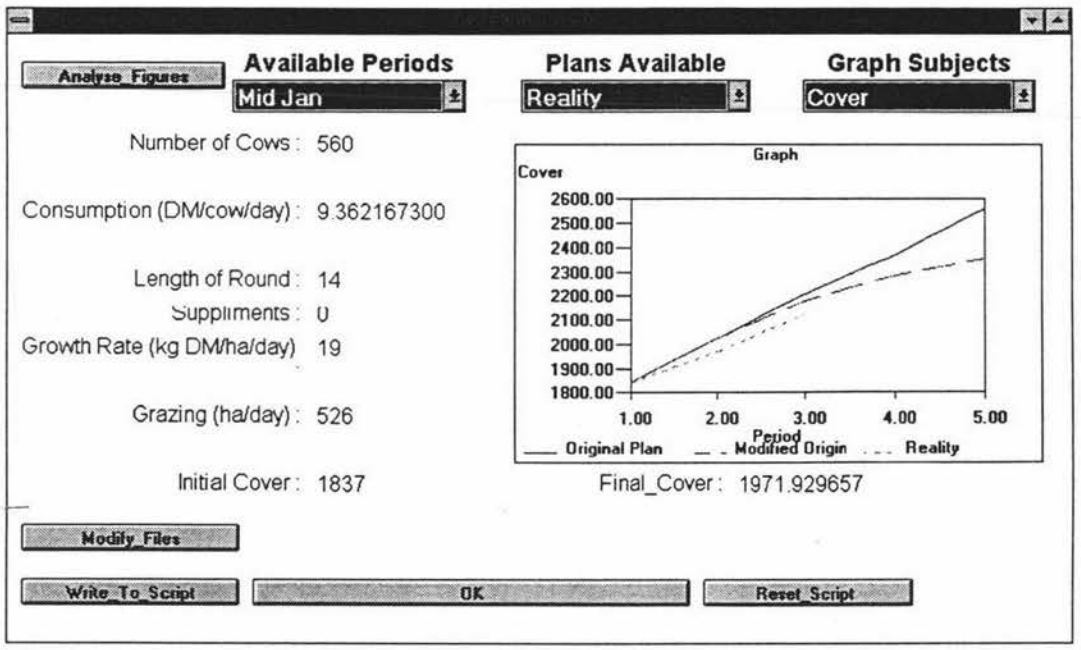


Figure 55 - Examination of one farmer

Analysis results are displayed in a window which is created as shown in Figure 56.

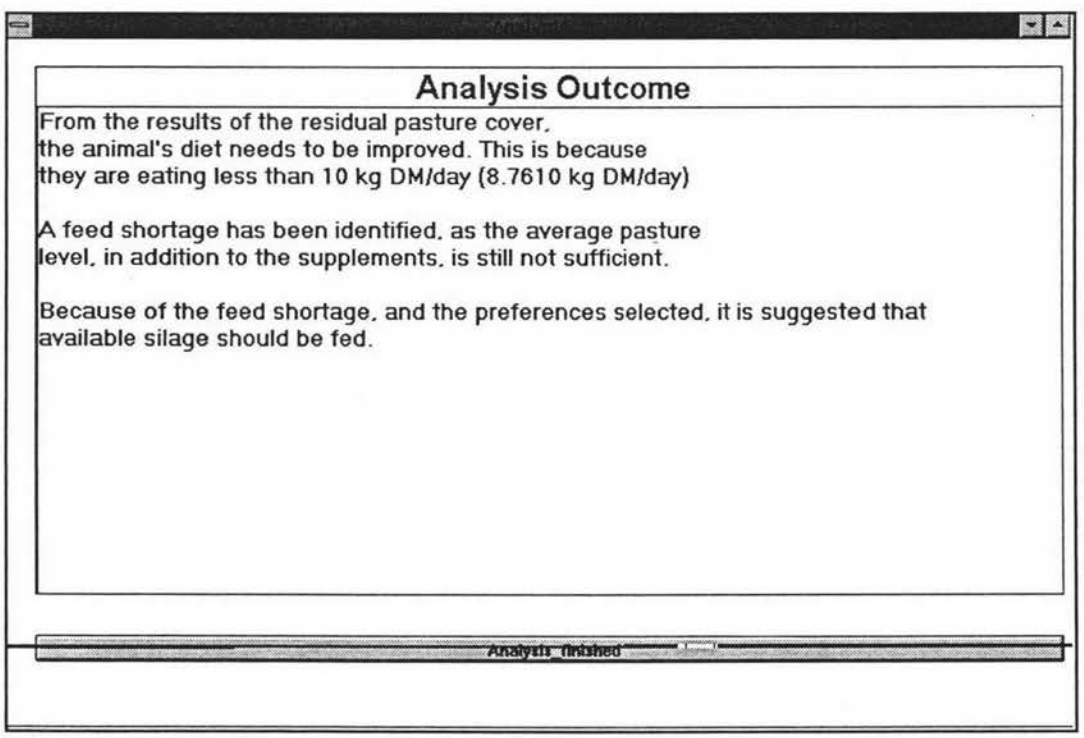


Figure 56 - Analysis of farmer's data

Milkfat production requires the worksheet to be selected, and analysis is automatically performed on this data. The analysis is displayed in the scrollable transcript window, shown in Figure 57. The top portion of the window is filled with a graph of one of the sets of data from the worksheet. The figures that can be viewed are:

- Live weight.
- Condition Score.
- Solids.
- Milkfat Production.
- Dry Matter Intake.

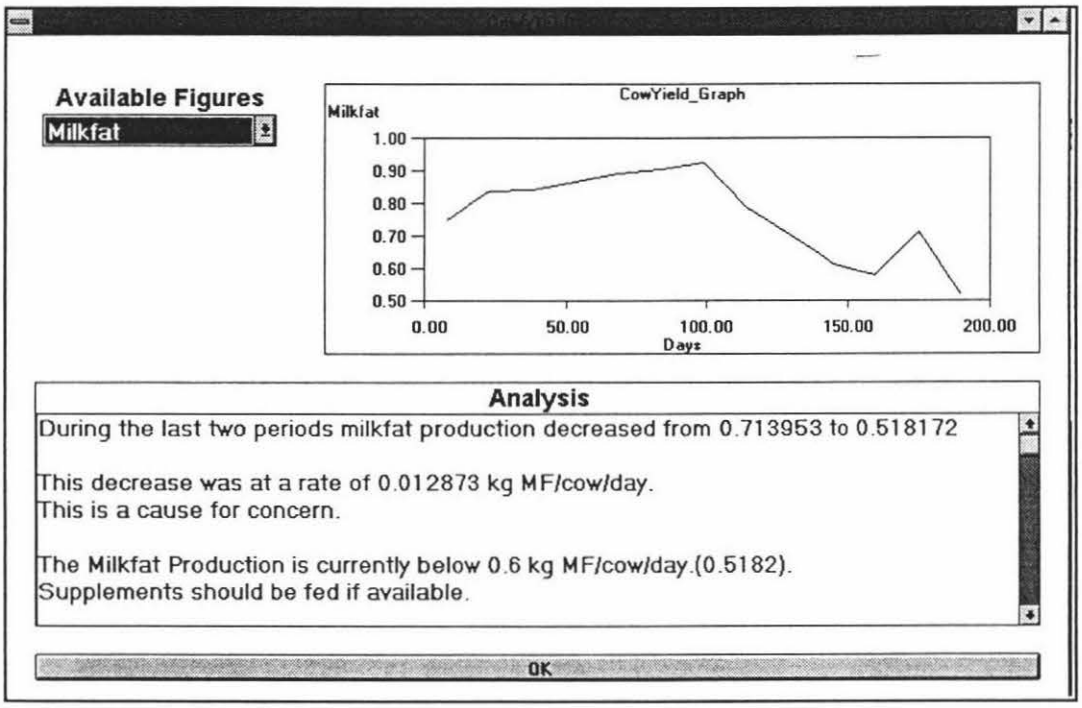


Figure 57 - Farmer's production level screen

It is also possible for the user to alter the preference of the various management options proposed by Gray et al (1993). The options were given in Figure 6. Depending on the individual conditions on a farm, the easiest option may not be the best, or even possible. For example, there is no point in suggesting that summer forage should be fed if there is none available. These options are reflected when analysis takes place on either the feed budgeting situation or the production level data. This window is shown in Figure 58.

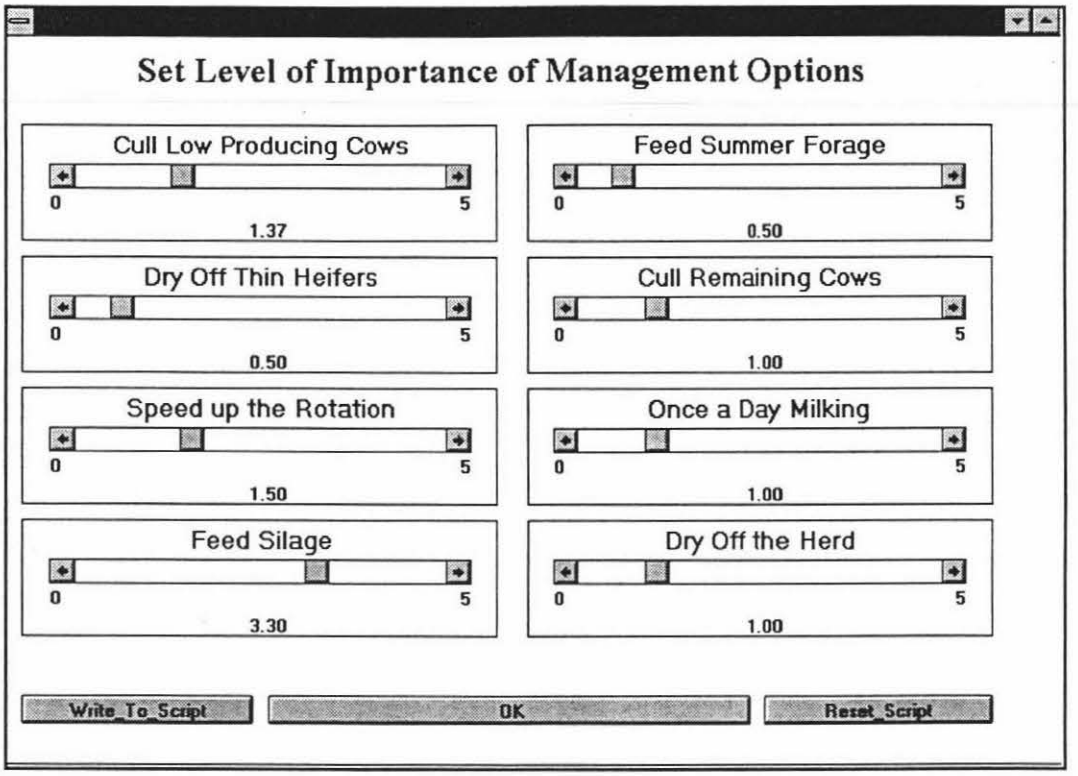


Figure 58 - Management option modification

All screens contain the button 'Write to Script', which allows for annotation of the transcript of the current session. The transcript is automatically written to when any changes are made to figures by the user, or when analysis is requested.

Interface design was carried out on a quite ad-hoc basis. From initially generated ideas, a number of screen mock ups were made. These were only modified minimally during the course of prototype implementation.

4.4 Conclusion

The analysis phase of Nunamaker's methodology has continued with the construction of the functional prototype. The high level structures for the functional prototype were identified in the knowledge representation model. These were the monitoring, prediction, planning and diagnosis tasks. These structures, shown in Figure 18, are the first implemented classes and objects within the functional prototype, from which all further implementation proceeded. From here the low level actions were derived from the task layer.

The creation of objects not contained within the knowledge representation model but necessary in the prototype system was required. These objects were termed implementation objects, and were distinct from objects identified within the knowledge representation model. The implementation-specific objects were instantiated with figures from selected worksheets for analysis and display purposes. If the system had been built using an application other than Kappa-PC, the structure of these objects may well have been different.

The prototype was built with the aim of allowing the examination of the knowledge contained in two main areas of the model of expertise. The success or otherwise of the functional prototype as tool for model validation is whether the prototype accurately reflects both correct and incorrect aspects of the model of expertise. This is to be examined in the following chapter.

Chapter 5: Evaluation of Prototype

Introduction

In accordance with Nunamaker's research strategy, the hypothesis proposed for this project, that of a functional prototype being used as an effective method for model validation, is now to be examined. This is to be done through the use of the implemented prototype as an argument, either for or against the hypothesis.

The prototype requires examination and evaluation from two perspectives. The first of these is from the prototype developer's point of view, and secondly the knowledge within the prototype is examined by the domain experts. The results of this are given in the final section of the chapter.

5.1 Prototype Acceptance Overview

Sommerville's (1989) steps for checking a prototype provide a base for the developing a prototype acceptance procedure. The developer is responsible for ensuring that when comparing the functional prototype to the problem domain, the prototype should be shown to be consistent, and complete.

The domain expert examines the consistency and completeness of the model, in addition to ensuring that the prototype is of practical use, and that the prototype is realistic. By the prototype being of practical use, it is meant that the areas modelled are useful and relevant for future system development. This involves checking that areas in the problem domain that are not important do not have too much significance placed upon them. The realistic element deals with the data being processed by the prototype. Extreme data which is possible is acceptable, but data which will never occur should not be used. The prototype is not built as a commercial quality piece of software, so robustness should not be an issue. The conclusions generated should be compared with real world situations, to make sure that they are correct.

These testing steps and the responsibility for each are shown in Figure 59.

Needs of the user shown to be valid		
Prototype should be shown to be consistent	Prototype	Domain
Prototype should be shown to be complete	Developer	Expert
Prototype should be shown to be realistic		

Figure 59 - Testing Components and Responsibilities

Due to the lack of an established methodology for the use of a functional prototype, a methodology had to be constructed. The methodology for acceptance of the functional prototype is given in Figure 60.

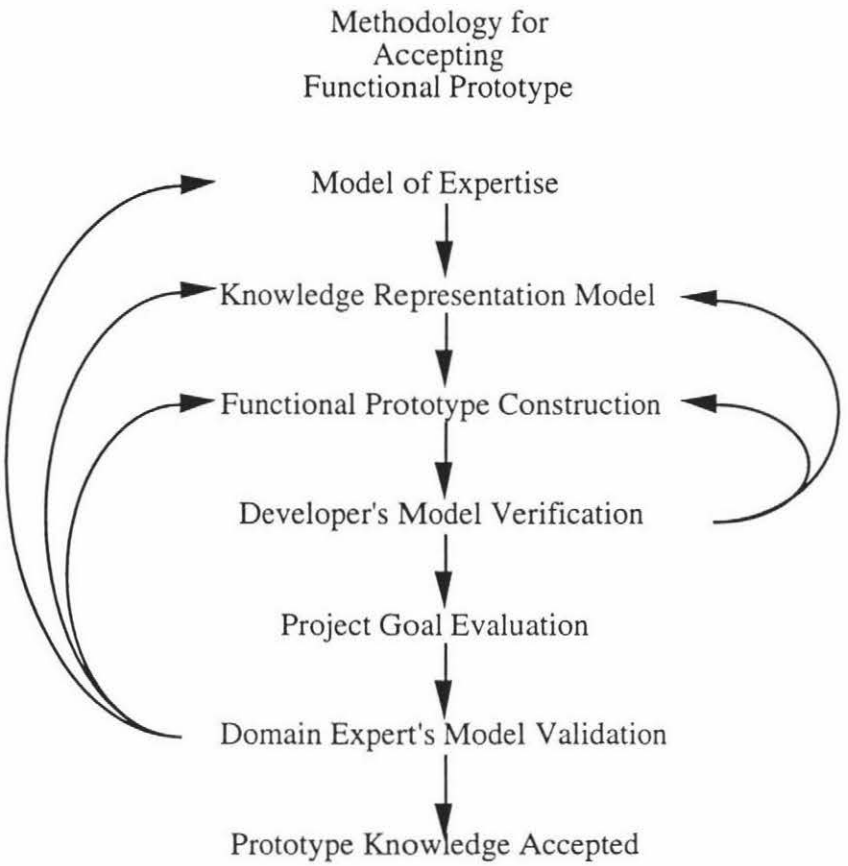


Figure 60 - Prototype Acceptance

There are two testing phases involved in the methodology. The first is from the prototype developer's point of view, ensuring that the prototype is an accurate representation of their understanding of the model of expertise and knowledge representation model. Secondly the knowledge within the prototype is examined by the domain experts. Their validation fulfils the role of validation of the model of expertise. The difference in these two phases can be summarised as being the first has the aim of verifying the *prototype developer's* understanding of the model of expertise, while the second phase validates *knowledge about the problem domain* itself.

The developer's verification is concerned with ensuring an accurate mapping exists between each stage of the development lifecycle. This involves examining the model of expertise, the knowledge representation model and the constructed functional prototype.

The domain expert's validation gives an indication of how well the model of expertise was constructed. If the verification by the prototype developer can be performed smoothly, then it can be concluded that the model of expertise was well constructed from the point of view of conveying knowledge in a form which can be subsequently built upon. If the domain expert's model validation is successful, then the model of expertise expresses the problem domain well. An accurate model of expertise is vital, as subsequently constructed models after the model of expertise increase the possibility of an error being introduced. Additionally, mistakes made in earlier models are magnified through succeeding generations.

5.2 Developer's Model Verification

The initial step in evaluating the prototype from the developer's point of view was the construction of a definition of what the prototype is to perform. Although minor modifications were made throughout the course of the project, these original aims provided guidelines to work to. It became clear that features could be continually added to the prototype, but extra features not consistent with the defined aims from Duffin's (1988) life cycle were not implemented.

Payne and McArthur's (1990) criteria for evaluating prototypes involves three levels. The first of these, checking for implementation stability, was considered to be only slightly relevant to the project. This was due to the project having an emphasis placed on the knowledge contained within the prototype, rather than requiring a system capable of handling every conceivable eventuality. Checks were made to ensure the system was stable for demonstration purposes, for example button working as expected, but exhaustive testing was not undertaken.

For Payne and McArthur, from a business oriented view, another layer was that of the management reviewing the completed prototype to ensure the project goals have been met (Payne and McArthur, 1990). This corresponds to ensuring the constructed prototype can be used for model validation purposes. It also requires that, from the knowledge engineer's perspective, the knowledge representation model (and hence the model of expertise) is accurately portrayed.

The model validation process is the third of Payne and McArthur's criterion. Model verification took place before validation. Through the use of test data, derived figures and the inferences drawn were examined. The verification was completed when the prototype performed in accordance with the developer's understanding and expectations of the problem domain. Model verification involved ensuring that the knowledge contained within the model created corresponded to the knowledge contained within the model of expertise. This involved comparing the model of expertise with the knowledge representation model, and subsequently comparing the functional prototype's knowledge with both the model of expertise and the knowledge representation model. Verification could be thought of as ensuring that the constructed functional prototype, which was accepted as being a completed system, was indeed an accurate representation of the model of expertise.

Figure 61 contains an example of the verification procedure. Verification involved tracing the same problem domain element through the three models, as shown in Figure 38. The figures contained within Figure 61 are explained within the previous chapter (Section 4.3.2.2).

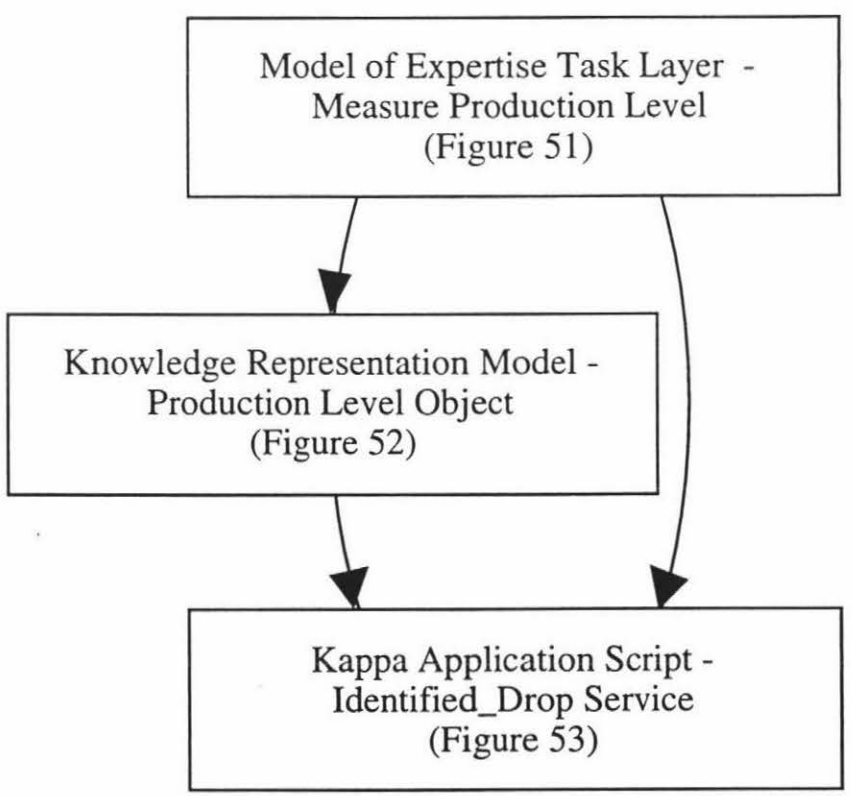


Figure 61 - Verification Paths

From Figure 61, comparisons were made between:

- The Knowledge of Representation Model and the Model of Expertise
- The Functional Prototype and the Knowledge Representation Model
- The Functional Prototype and Model of Expertise

This was also the order in which the comparisons were made. The third of these is the most important. Once this comparison has realised a match, the functional prototype is considered verified.

The interface also needs to be accepted. The criteria for finishing was that all data and information required was displayed on the screen in a clear and understandable manner, and the cost of further development outweighs the benefits. The interface design was identified was being an area of importance. Since the prototype was to be used for expressing the knowledge contained within the system, it needed to be designed so someone not familiar with the prototype could easily understand what was being displayed.

5.3 Domain Expert's Model Validation

The developer is required to have completed the verification of the functional prototype before the domain expert performs model validation. This verification is the first stage in Sommerville's validation methodology (Figure 59). Once this stage is completed, the second stage, that of functional prototype validation, may proceed.

The validation process was undertaken by constructing a functional prototype and the demonstrating of it to several problem domain experts. The domain experts, who were people responsible for the training of Dairy Board Consulting Officers, provided a series of figures in advance which were used for demonstration purposes. These figures were those of successful (or desirable) farmers results. A few sample worksheets were also created with results being less than ideal to generate a larger degree of problem analysis and decision making. These worksheets followed the format of Figure 45 and Figure 50.

A 'hands-off' presentation for the domain experts was agreed upon to be to lessen emphasis on the interface. To this end, the developer guided the demonstration, responding to queries and questions. This was to allow the domain experts to concentrate on the problem domain aspects of the software rather than trying to figure out application issues, for example how to perform a comparison of farmer's records. The interface design is important for conveying the information contained within the knowledge base accurately to the domain experts. Any comments generated about the interface should be noted for future reference when subsequent models are built.

The demonstration followed a similar pattern to the example in Appendix 3. There was a great deal of discussion, with new ideas and viewpoints being raised for both the domain expert and the system developer. The feedback generated was positive. The application was recognised as being a tool for both mathematical evaluation, and analysis of the generated figures. This analysis exhibited clearly the knowledge-based characteristics of the system. The mathematical evaluation aspect was recognised as corresponding to the 'Monitoring Events' and 'Prediction of Events' sub-tasks of the 'Manage the Farm' task.

The implemented functional prorotype was validated successfully. The domain experts considered the system to be consistent, complete, realistic and relevant to the problem domain in accordance with Figure 59. Should there have been any inconsistencies or other problems identified, the stage at which the error first appears in the model would be required to be located. The implementor's translation from model of expertise to knowledge representation model to functional prototype is susceptible to errors. If the error first appears in either of the last two stages of translation, then it is classified as a design fault. This requires the developer to understand with respect to the problem domain why the error occurred. The alternative is that the model of expertise has had a flaw identified.

The functionality of the prototype may appear accurate, but the underlying structure is not necessarily correct. For this reason the explanation facilities within the prototype must be explicit enough to ensure the knowledge, and the reasoning behind decisions made, is semantically correct. This is especially relevant for large scale systems, where not all reasoning and conclusions reached are immediately visible. Any faulty reasoning may not be picked up until later in the system development process, when major re-structuring could be required.

Functional prototypes must be kept to a relatively small scale for the problem domain. This enables specific areas to be targeted for model validation. Presenting too much information at once may result in minor aspects being missed.

5.4 Conclusion

The functional prototype has been constructed and is required to be tested from two viewpoints. The first of these is the implementor's perspective, deriving a prototype which accurately maps the model of expertise through to the prototype. Once this process was completed, the evaluation by domain experts took place, comparing their knowledge to that displayed through the functional prototype. The results of this analysis can now be used as an argument in support of the hypothesis, that a functional prototype can be used as an effective method for model validation of the model of expertise.

As the functional prototype is designed for the purpose of model validation, it was felt that the implementor was best suited to demonstrate the system. Test data was generated well in advance of the demonstration and passed onto the demonstrator. This allowed the demonstration to proceed efficiently without trivial application specific problems (for example, in the prototype the farmers name must not contain a space or full stop character).

Another advantage of having the developer present was that queries and questions arising were able to be considered immediately. Often these queries proved to be related to the interface not expressing the knowledge or data precisely enough. Despite the lack of emphasis on the interface, clearly this is an important issue to be considered. The idea of having a knowledge engineer design the underlying structure of the program while working in conjunction with a human-computer interface expert would enhance the benefit of the functional prototype.

A side effect of using functional prototyping for model validation is the positive effect on domain experts, who are not necessarily computing experts, seeing a working simple system. This can engender enthusiasm in the project. Having a prototype at this stage also generates ideas for final system development. At this stage, although constructed for model validation purposes, the functional prototype doubles as an initial final prototype model (throw away style).

The success of model validation through the functional prototype creation was helped by the quality of the existing model of expertise. Model validation would have to be performed on the complete model of expertise before any unequivocal acceptance of it .

What has been shown is the viability of using functional prototyping within the KADS framework to perform model validation. This development, in accordance with Nunamaker's terminology, can be thought of as "proof-by-demonstration".

Chapter 6: Summary and Conclusions

Functional prototyping has been shown to be an effective, viable approach to model validation within the KADS methodology.

A model of expertise had been created. Construction of a functional prototype was achieved through the development of the knowledge representation model. From the model of expertise, it was required that there be a suitable knowledge representation form selected for the prototype. The object-oriented paradigm was selected for system develoment because of their advanced features in comparison with other knowledge representation forms. These features include inheritance and encapsulation.

From the knowledge representation model, areas were selected to be prototyped for model validation purposes. Implementation took place using Kappa-PC, a PC based application development environment. The implemented prototype was examined by a number of domain experts during a demonstration session. The session generated many favourable comments on the knowledge contained within the system and the way in which it was presented.

Nunaker's four stage research methodology was followed. The problem was defined as the literature indicates that functional prototyping could be used as a means of model validation, but this has not been proven. Therefore the hypothesis was that a functional prototype can be used as an effective method for model validation of the model of expertise. Subsequently, the analysis was undertaken through the construction of the functional prototype. Finally, the argument that functional prototyping can be used as a viable option for model validation was shown to be true through proof by demonstration.

6.1 Findings

Model validation through the construction of a functional prototype has been shown to be a feasible, but under-utilised, part of the KADS methodology.

The use of functional prototyping does provide additional advantages over other techniques for model validation. For example, the object-oriented model of the system, although constructed for the prototype development process, is clearly reusable for future system development. Protocol analysis, the main alternative to functional prototyping within the KADS literature, is specific for the model validation phase. Analysis generated cannot be used to assist future development.

The knowledge representation model proved to be a necessary intermediate model between the KADS model of expertise (on paper) and the implemented functional prototype. The knowledge representation model resulted in knowledge being expressed in a format compatible with the object-oriented paradigm. This allowed for implementation to proceed in an object-oriented development environment. The advantage of the knowledge representation model was that it replaced the giant leap required from model of expertise to functional prototype with two smaller, simpler steps.

The similarities between object-oriented approaches and the KADS methodologies have been noted before (Schreiber and Wielinga, 1993) and these came out during the knowledge representation model construction. A number of structures within the domain layer of the model of expertise mapped directly into the object-oriented paradigm. The model of expertise, with domain, inference and task layers, identified the classes, objects, services and attributes for the object oriented model. This is of great benefit to developers not familiar with the problem domain presented though a model of expertise, as correct selection of classes and objects is one of the most difficult phases of object-oriented approaches.

The quality of the model of expertise can be gauged in two ways. The first aspect of this is the ability of the model of expertise to express knowledge about the problem domain in a way understandable to the prototype developer. This person does not necessarily have any background in the problem domain, so the success of the model of expertise in providing the basic elements of the problem domain and building upon this to incorporate complex constraints is vital. Secondly, the accuracy of the problem domain knowledge is examined through the validation process. Fewer errors being detected suggests the model of expertise was well constructed. A successful validation does not imply the entire knowledge base is accurate; should mistakes be discovered, then examination of previously constructed models is required as depicted in Figure 60.

The construction of a working prototype of an area of the domain does engender enthusiasm in the project. The ability for domain experts to see progress, particularly in a long term project, is motivational for all parties concerned. The functional prototype produced a great deal of feedback, with a lot of new ideas and views generated.

Interface development is an important part of any system, but in functional prototyping, where the system's aim is to accurately convey the knowledge for validation purposes, this is vital. Although not an explicit part of the functional prototyping model validation process, the interface should be considered an important part of the developed system. On a large scale project, where many people are involved in the development, an interface design specialist should be considered. Although designed for a throw-away prototype, the effectiveness of the interface with a view to final system interface requirements should be kept in mind. However, this may depend on how much of the interface is expressly for model validation purposes.

The most obvious limitation of the functional prototyping approach is the 'black box' feeling of some outputs. This was not very prevalent within this project, but for more complex parts of the problem domain, this needs to be avoided for model validation purposes. Ideally, the explanation facilities in the completed system should range from minimal (for example for experts wanting confirmation of their conclusions) through to full transcripts covering all decisions made and the rational behind these decisions (for model validation purposes).

6.2 Future Work

The obvious extension to this work is the completion of the model validation action on the model of expertise for this domain. The areas of the problem domain covered within this project could be used as a base for future development. From this point, the knowledge contained within the model of expertise has been validated, and work on a complete system begun. It should be remembered that the functional prototype was designed for model validation purposes. Aspects of the prototype that were considered desirable should be taken into account, but not considered an absolute necessity.

Validation of the model of expertise through another means, such as protocol analysis, could be undertaken and the two techniques compared. The possible integration of protocol analysis with functional prototyping for model validation purposes is another area with potential for further study. Some problem domain areas will naturally lend themselves to one or other of these approaches. The generation of selection criteria would be of use to future knowledge based system developers.

The success of the object-oriented paradigm and its relation to KADS opened some avenues for future research. A formal system of mapping KADS models into an object-oriented knowledge representation form could be looked at. There is work being carried out by the KADS consortium into designing a formal language for KADS models of expertise (Schreiber et al, 1992) called $(ML)^2$. The authors acknowledge there is a substantial gap between the model of expertise and the construction of an implemented system, and propose $(ML)^2$ as a stepping stone. The transition from $(ML)^2$ to an object-oriented model could be examined. The formalism of $(ML)^2$ provides the potential for automated moving from $(ML)^2$ to an object-oriented knowledge representation model.

Bibliography

Akkermans, H., Wielinga, B. and Schreiber, G. (1993). Steps in Constructing Problem Solving Methods. in Aussenac, N., Boy, G., Gaines, B., Linster, M., Ganascia, J,-G. and Kodratoff, Y. (eds.),*Knowledge Acquisition for Knowledge-Based Systems: EKAW-93*. Berlin: Springer-Verlag.

Barr, A. and Feigenbaum, E. (1981). The Handbook of Artificial intelligence. Los Altos CA: Morgan Kaufmann.

Benbasat, I. and Dhaliwal, J.S. (1989). The Validation of Knowledge Acquisition: Methodology and Techniques. in *Proceedings of the EKAW-89.*

Biondo, S. (1990). Fundamentals of Expert Systems Technology. Norwood: Ablex Publishing Corporation.

Bischofberger, W. (1992). Prototyping-oriented Software Development: Concepts and Tools. Berlin: Springer-Verlag.

Blake, S. (1978). Managing for Responsive Research and Development. San Francisco: WH. Freeman and Company.

Bødker, S. and Grønbaek, K. (1991). Co-operative Prototyping: Users and Designers in Mutual Activity. International Journal of Man-Machine Studies, 34 453-478.

Boncek, R., Holsapple, C., and Whinston, A. (1982). Prototyping for Decision Support Systems. in Ginzberg, M.J., Reitman, W. and Stohr, E. (eds.), *Decision Support Systems: Proceedings of the NYU Symposium on Decision Support Systems.* Amsterdam: North-Holland Pub. Co, New York.

Bramer, M. (1982). A survey and critical review of expert system research. in Michie, D. (ed.), *Introductory Readings in Expert Systems*, 3 - 32. New York: Gordon and Breach.

Breuker, J. (1993). Modelling Artificial Legal Reasoning. in Boy, G., Aussenac, N., Gaines, B. and Boose, J. (eds.), *Proceedings of the European Workshop on Knowledge Acquisition - EKAW '93,* 66- 78. Berlin: Springer- Verlag.

Breuker, J. and Wielinga, B. (1989). Models of Expertise in Knowledge Acquisition. in North-Holland, Guida, Giovanni and Tasso (eds.)*Topics in Expert System Design, Methodologies and Tools* , 265 - 295. Amsterdam.

Breuker, J., Wielinga, B., van Someren, M., de Hog, R., Schreiber, G., de Greef, P., Bredeweg, B., Wielemaker, J., Billault, J-P., Davoodi, M. and Hayward, S. (1987). Model Driven Knowledge Acquisition: Interpretation Models. Esprit Project P1098 Deliverable D1. University of Amsterdam and STL, Amsterdam.

Brule, J.F. and Blount, A. (1989). Knowledge Acquisition. New York: McGraw-Hill.

Budd, T. (1991). Object-Oriented Programming. Reading, Massachusetts: Addison-Wesley.

Bylander, T. and Chandrasekaran, B. (1987). Generic Tasks for Knowledge- Based Reasoning: The Right Level of Abstraction for Knowledge Acquisition. Technical Report 87-TB-KNOWAC, Ohio State University:Columbus.

Coad, P. and Yourdon, E. (1991). Object-Oriented Analysis. Englewood Cliffs, New Jersey: Yourdon Press.

da Silva,A. (1994) The AIM Methodology: Analysis, Interpretation and Modelling, a methodology for the role of the knowledge analyst in knowledge acquisition. Massey University.

DeMarco, T. (1982). Controlling Software Projects. New York: Yourdon Press.

Duda, R., Hart, P. and Sutherland, G. (1978). Semantic Network Representation in Rule-Based Inference Systems. in Waterman, D. and Hayes-Roth, F. (eds.), *Pattern-Directed Inference*,. Academic Press: New York.

Duffin, P. (1988). GEMINI: Government Expert System Methodology Initiative. Advanced Technology Group:London.

Ericson, K.A. and Simon, H.A. (1984) Protocol Analysis: Verbal Reports as Data. Massachusetts: MIT Press.

Eshelman, L. (1988) MOLE: A Knowledge Acquisition Tool for Cover-and-Differentiate Systems. in *Automating Knowledge Acquisition for Expert Systems.* 37 - 80. Boston:Kluwer Academic Press

Gray, D., Greig, A., Lockhart, J., Malone, J. and MacMillan, M. (1993). Summer-Autumn Management from a Farmer's Perspective. Massey University.

Gray, D.I. and Parker, W.J. (1989). The Planning, Implementation and Control of Pastoral Based Systems. Massey University.

Hart, A. (1990). Knowledge Acquisition for Expert Systems. New York: McGraw-Hill

Hart, A. and Berry, D. (1992). Knowledge Acquisition: Human Issues. New York: McGraw-Hill

Hayward, S. (1986). A structured development methodology for expert systems. in *Proceedings of the international conference on Knowledge Based Systems* , 195- 203. Online: London

Hickman, F.R., Killin, J.L., Land, L., Mulhall, T., Porter, D. and Taylor, R.M. (1989). Analysis for Knowledge-Based Systems: A Practical Guide to the KADS Methodology. Chichester: Ellis Horwood.

Hoppe, T. (1990). Validation of User Intention. in Ginzberg, M.J., Wielinga, B., Boose, J., Gaines, B., Schreiber, G. and van Someren, M. (eds.), *Current Trends in Knowledge Acquisition*, 161 - 172. IOS Press, Netherlands.

Jackson, P. (1990). Introduction to Expert Systems. Reading, Massachusetts: Addison-Wesley.

Johnson, P., Zualkernan, I and Garber, S. (1987). Specification of Expertise. International Journal of Man-Machine Studies, 26 161 - 182.

Karbach, W., Linster, M. and Voß. (1990). Models of Problem-Solving: One label - One idea? in Ginzberg, M.J., Wielinga, B., Boose, J., Gaines, B., Schreiber, G. and van Someren, M. (eds.), *Current Trends in Knowledge Acquisition*, 173 - 189. IOS Press, Netherlands.

Kemp, E.A., Todd, E.G., da Silva, A. and Gray, D. (1993). Knowledge Acquisition Applied to Farmer Decision Making. in *Proceedings of the Second Singapore International Conference on Intelligent Systems SPCIS '94.*

Mehandjiska, D. and Page, D. (1993). Object-Oriented Development of Expert Systems. in *Proceedings of ANNES 1993* IEEE Computer Society Press:University of Otago

Minsky, M. (1975). A framework for representing knowledge. in Winston. (ed.), *The Psychology of Computer Vision*, 211 - 277. New York: Magraw - Hill.

Musen, M., Fagan, L., Combs, D. and Shoftliffe, E. (1988). Use of a domain model to drive an interactive knowledge editing tool. International Journal of Man-Machine Studies, 26 105-121.

Nazareth, D. (1989). Issues in the verification of knowledge in rule-based systems. International Journal of Man-Machine Studies, 30 255-271.

Neubert, S. and Studer, R. (1991). The KEEP Model, a Knowledge Engineering Process Model. University of Amsterdam, Amsterdam.

Newell, A. and Simon, H. (1972). Human Problem Solving. Englewood Cliffs NJ:Prentice - Hall.

Nunamaker, J., Chien, M. and Purdin, T (1991). Systems Development in Information Systems Research. Journal of Management Information Systems Vol. 7 Number 3, 89-106.

O'Keefe, R.M., Balci, O. and Smith, E.P (1987). Validating Expert Systems. IEEE Expert Winter.

Payne, E. and McArthur, R. (1990). Developing Expert Systems. New York: John Wiley & Sons.

Quillian, M. (1968). Semantic Memory. in Minsky, M. (ed.), *Semantic Information Processing*, 227 - 270. Cambridge MA: MIT Press.

Roberts, H. (1988). Expert Systems Clubs: Design Methods. in The Computer Journal, 33 129 - 133.

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F. and Lorensen, W. (1991). Object-Oriented Modelling and Design. New Jersey: Prentice Hall.

Schreiber, G., Bredeweg, B., Davoodi, M. and Wielinga, B. (1987). Towards a Design Methodology for KBS. Esprit Project P1098 Deliverable D8. University of Amsterdam, Amsterdam.

Schreiber, G. and Wielinga, B. (1993). Comparing KADS-I to Conventional Software Engineering. in Schreiber, G., Wielinga, B. and Breuker, J. (Eds.) KADS: A Principled Approach to Knowledge-Based System Development. Cambridge: Academic Press.

Sommerville, I. (1989). Software Engineering. Reading, Massachusetts: Addison-Wesley.

van Harmelen, F. (1992). (ML)2: A formal language for KADS models of expertise. in Schreiber, G., Wielinga, B. and Breuker, J. (Eds.) KADS: Knowledge Acquisition and Design Structuring. London: Academic Press.

Wetter, T. (1992). Preface in Wetter, K.D., Boose, J. Gaines, B.R., Linster, M. and Schmalhofer (eds.),*Current Developments in Knowledge Acquisition - EKAW' 92.*(pp V-IX). Berlin: Springer-Verlag.

Wielinga, B., Breuker, J., and van Someren, M. (1986). The KADS System Functional Description. Esprit Project P1098 No. T1.1. University of Amsterdam, Amsterdam.

Wielinga, B.J., Schreiber, A.Th and Breuker, J.A. (1991). KADS: A Modelling Approach to Knowledge Engineering. Esprit Project P5248 KADS-II, University of Amsterdam.

Appendix 1: Model of Expertise

Appendix 1 contains the relevant aspects within the model of expertise used for the functional prototype development. It is taken from da Silva (1994).

Domain Layer

The domain layer consists of the domain primitives and the network of primitives. This layer is the declarative part of the domain knowledge and so, does not contain problem-solving strategies. It consists of concepts, attributes and relationships.

Domain Primitives

The domain primitives describe the set of the concepts, attributes and relationships between concepts.

List of concepts and attributes

- Animal Health Problems
- Breed
- Calving Spread
- Cow Condition
- Cull Cows: empty cow, old cow, poor producing cow, mastitis cow.
- Dairy Cow
- Dry Matter: quantity, quality.
- Drying off
- Feed
- Feed Balance
- Feed Budget
- Feed Demand
- Grass Reproduction
- Feed Supply (supplements): silage, hay, crop, urea.
- Grazing System: set stocking, rotational system
- Intake
- Leased Block
- Mating
- Milking Area
- Paddocks
- Pasture Cover: quality, effect, seedhead
- Pasture Growth Rate
- Planning Horizons
- Pasture Scoring: plate meter, measurement method
- Postgrazing Pasture Cover (Residual)
- Pregrazing Pasture Cover
- Production Level: milkfat (kg), protein, litres
- Selenium
- Spore Count
- Reproductive Performance: in calf pattern, conception rate
- Rotation System: rotation length, paddock size
- Soil Temperature
- Somatic Cell Count

- Stocking Rate
- Weather Map
- Weather (Climate Data): rainfall, hot dry wind, dryness, shower of rain
- Wintering at home
- Young Stock: heifers, calves
- Zinc

Summer Autumn Management Period

Short Term Planning: Jan-Mar/Apr

- The main goal of this period is to maintain production levels as high as possible

Long Term Planning: Mar/Apr-May

- The main goal of this period is to make sure there will be enough feed (pasture and supplements) during calving time.

Relationships

Following are the main relationships which characterise short and long term planning.

Short Term Planning

- At this time of year dry matter is very low
- A good pasture cover leads to cows being well fed.
- Quality pasture may be evaluated by measuring milkfat and protein produced.
- Supplements are fed in order to allow pasture cover to build up.
- Rainfall in December leads to good pasture in the summer period.
- The rotation length is planned to maintain production level.
- the rotation length influences how well the cows are being fed.
- Feeding the cows is more beneficial than trying to have feed stocked up.
- Crop growth depends on the summer.
- The animals may be given silage to improve their feed balance.
- The supplements are fed in case the production falls below 0.6 kg of milkfat/cow/day.
- Silage is fed when the crop is gone.
- Supplements are fed when the cow condition drops.
- Nitrogen may be used in the autumn to improve pasture growth rate.
- The autumn flush is a time of high pasture growth rates.
- The young stock is kept on a leased block.

- The young stock is on a set stocking.
- Production level is an indicator of the condition of the cows.
- Production level may drive the decision of feeding supplements and drying off the cows.
- Cow condition is one of the main factors which influences the drying off process.
- The reproduction information is from the pregnancy test.
- Crop may lose quality if left for too long to be fed.
- The weather which follows the rain is a major factor for the pasture growth rates.
- Crop may be a problem for two year olds, because sometimes they don't eat it.
- The methods of production level evaluation are: kg of milkfat/cow/day, litres/day.
- The amount of litres measured are the main indicator for changing the grazing system.
- The cows scored 3.5 before being milked once per day.
- Pasture quality is still not an issue.
- The residual pasture cover is an indicator of the pasture cover situation.
- A pasture cover situation of 1700 DM/ha may trigger a major evaluation.
- The main reasons for culling are: empty cows, age, cell count, empty two year olds.
- Cows losing weight leads to a drop in production level.
- High stocking rates demand larger amounts of feed.
- The rotation system is extended if the grass is not growing more than requirements.
- Monitoring pasture cover gives a prediction of the cover ahead.
- The quickest way to build the cover is to leave more residual pasture cover.
- Residual pasture cover is another indicator of cow intake.

Long Term Planning

- The number of cows that will be wintered is very important when doing the feedbudget.
- An intake of 10 kg is about right (9 is acceptable, but 12 is a waste).
- The pregrazing pasture cover and postgrazing pasture cover are used to calculate cow intake.
- The pasture cover is being monitored once a week.
- Pasture quality is still not a problem.
- The pasture cover is used mainly for the feedbudget.
- A pasture cover below 2000 kg of dry matter/hectare might be risky and could lead to lower late production and cycling problems for the cows.
- The factors that can change the rotation system are: pasture cover, intake and cow condition.
- Pasture growth rates have been used to pick up the autumn flush.
- From April until calving date, growth rates and pasture cover are more important than production.
- The drying off decision is influenced by the stocking rate, in the sense that a higher stocking rate means drying off early.
- The use of nitrogen can provide a longer milking time.
- The reason why the good condition cows are dried off with the poor producing cows is the fact that the poor producing cows being dried off does not improve the pasture growth rates as they still need to be fed in order to gain weight.
- If cows are dried off early they can be at a condition of 4 and there is still time to gain condition until calving date. But if the drying off decision is delayed, the cows have to have a 4.5 condition at least because there is not much time to gain condition until calving date.
- A bad decision in terms of the balance between feed supply and cow condition is done when the cows are fat and there is nothing to feed them in the spring.
- Production is still an indicator whether the cows are being well fed.
- Pasture growth rate is the critical factor at this time of year in terms of the drying off decision.

Network of Primitives

The Network of Primitives represent the connection between the concepts and the relationships. Not all domain primitive relationships are contained, but the most important ones are.

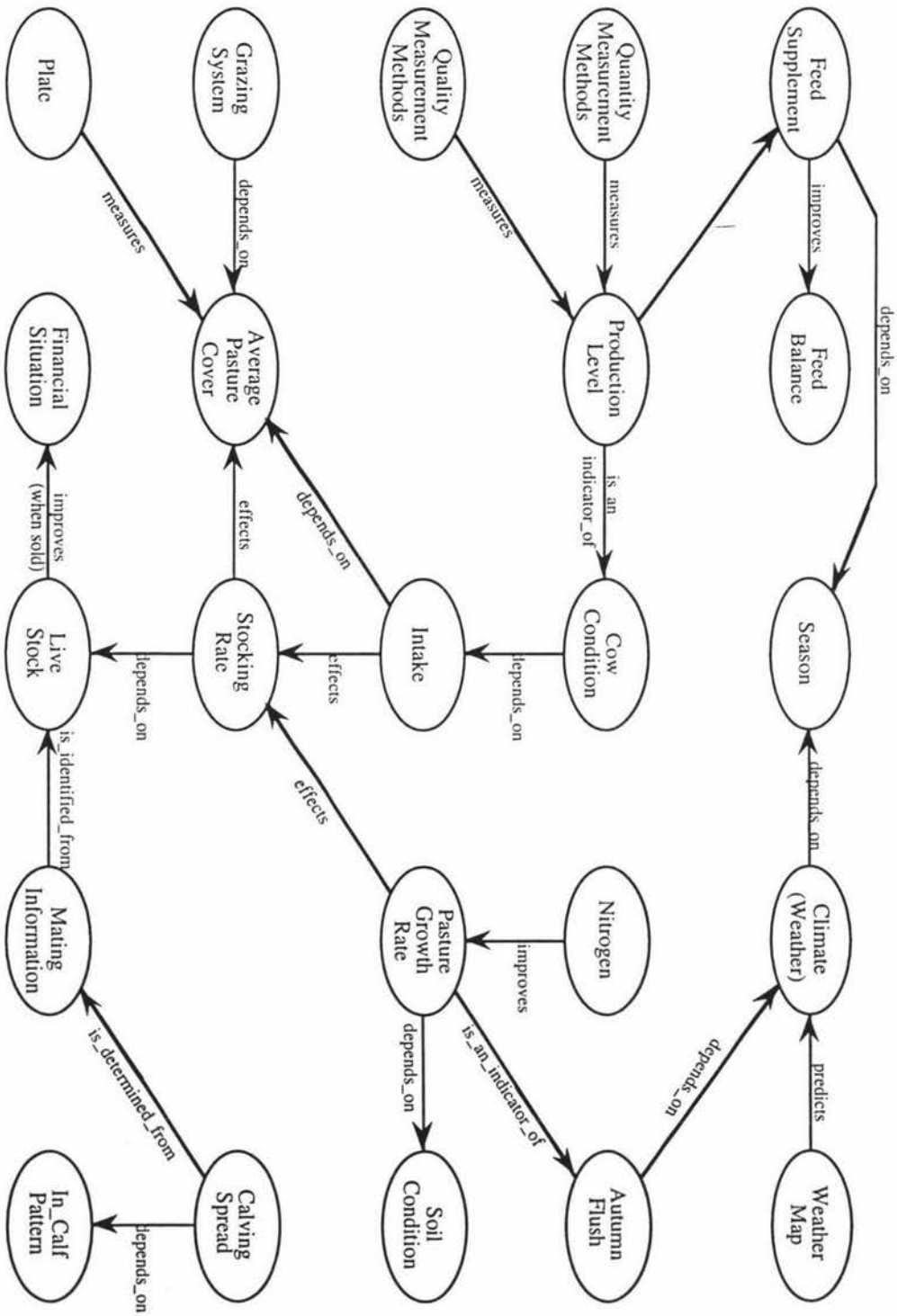
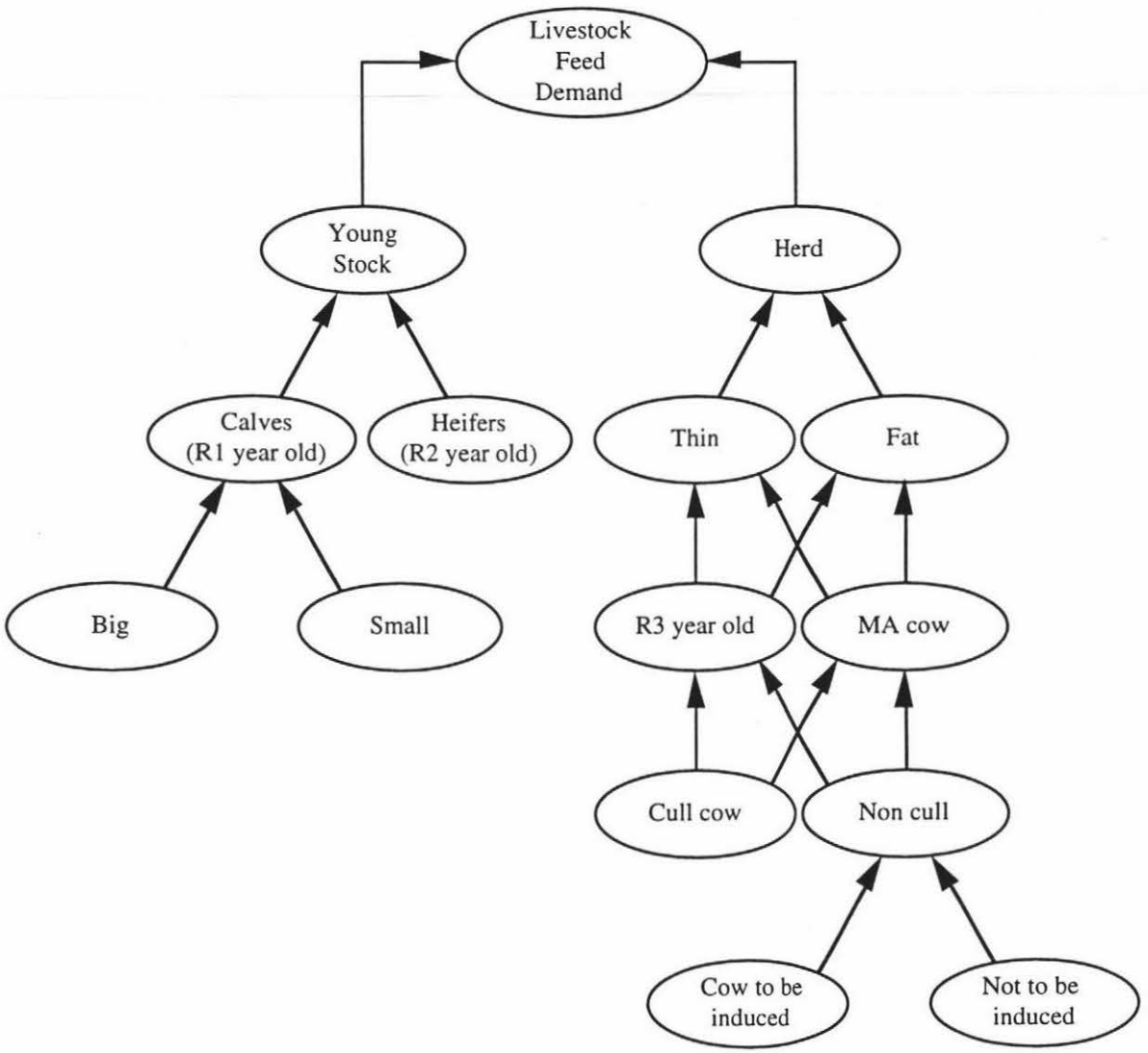


Figure 62 - High level Network



Note: Every arrow denotes an "is_part_of" relationship

Figure 63 - Livestock network - "is_part_of" relations

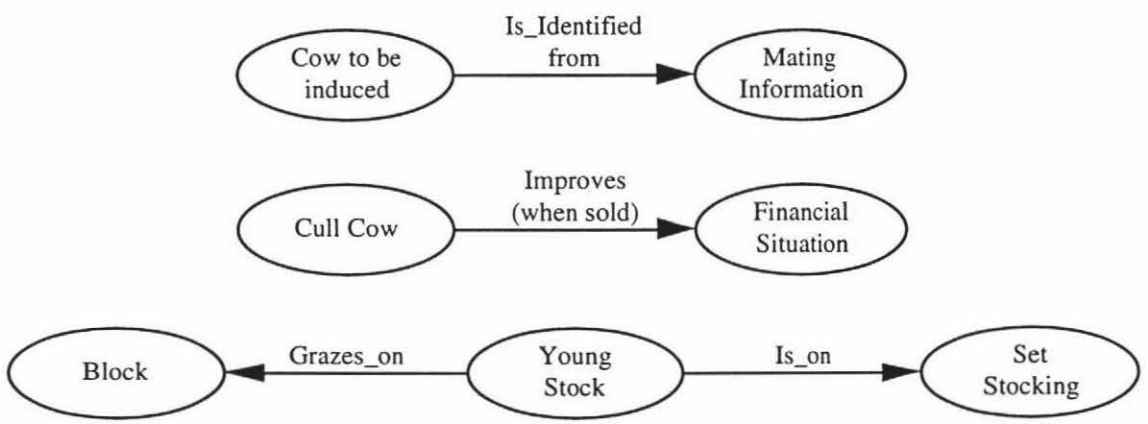


Figure 64 - Other relations involving livestock

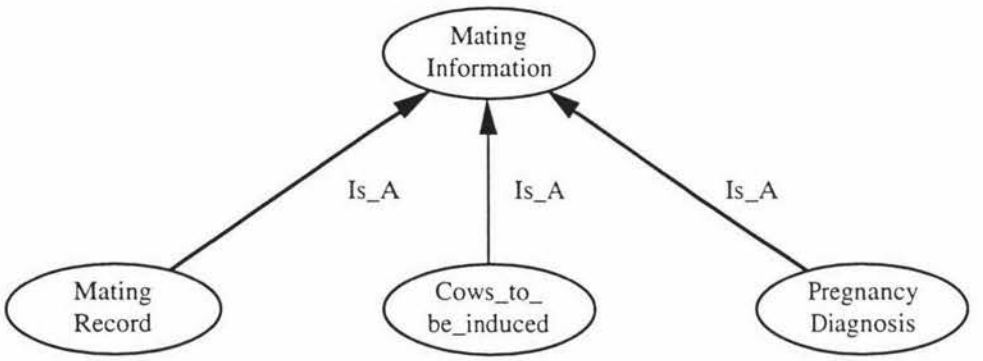


Figure 65 - Mating Information - "Is_A" relations

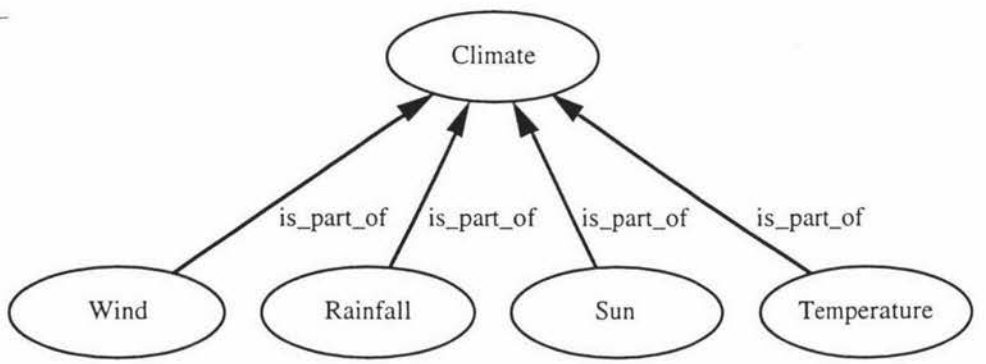


Figure 66 - Climate Network - "is_part_of" relations

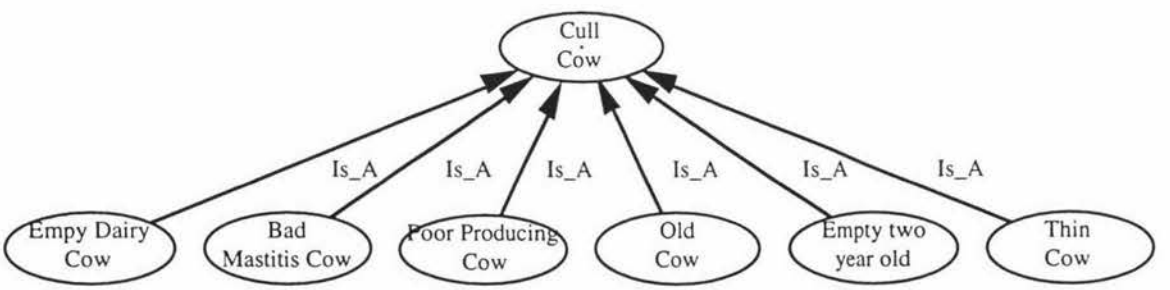


Figure 67 - Cull cow network - "Is_A" relations

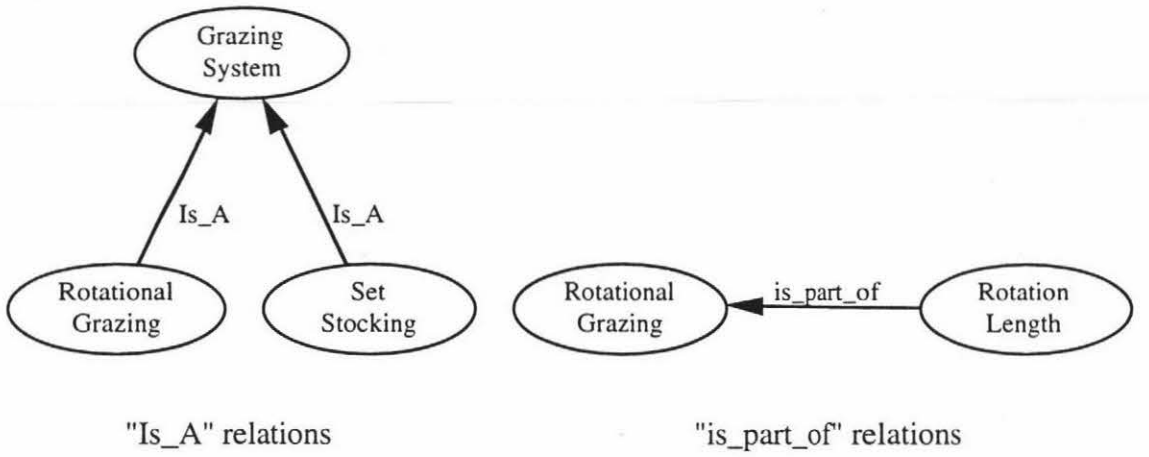


Figure 68 - Grazing Systems network

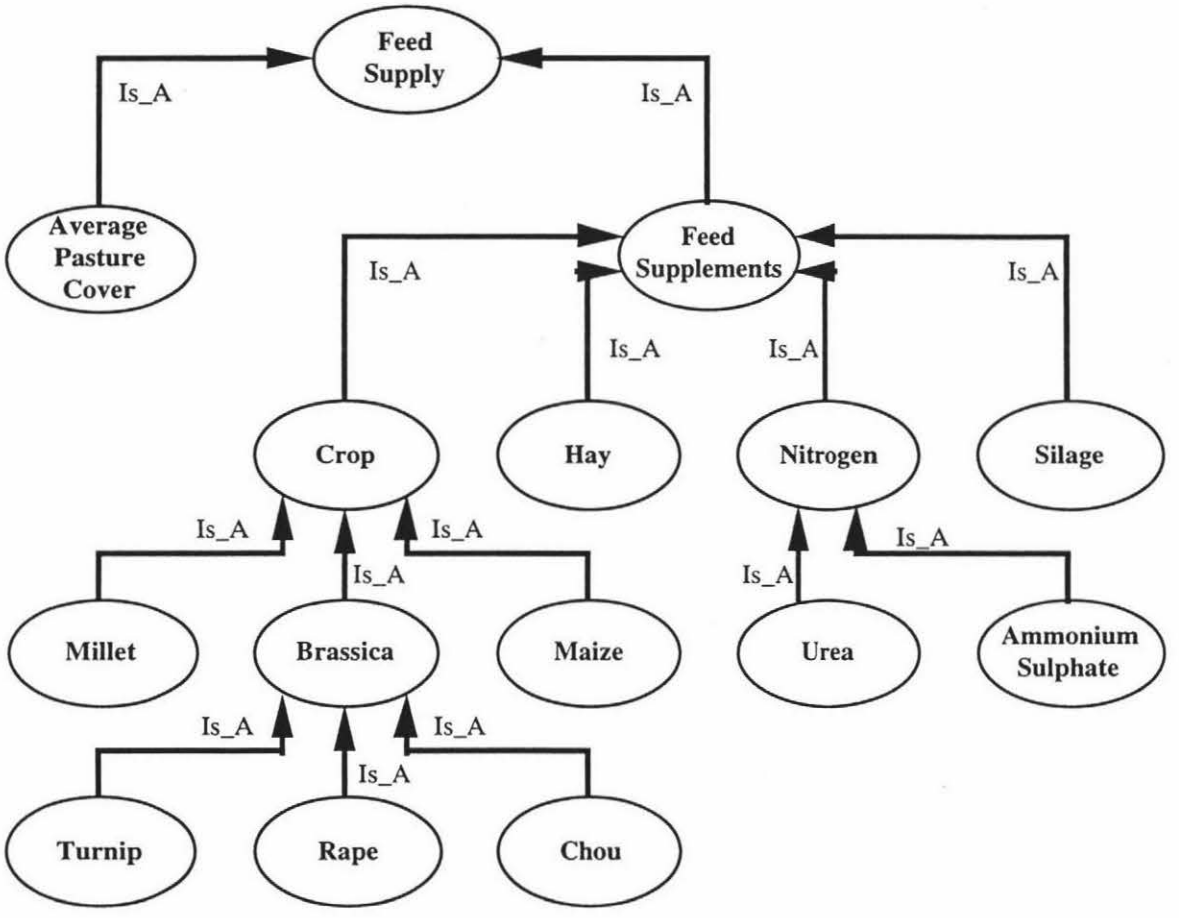


Figure 69 - Feed supply network "Is_A" relations

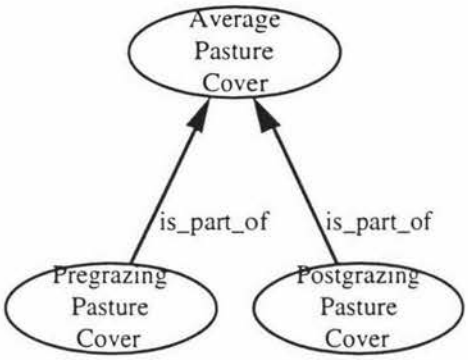


Figure 70 - Average Pasture Cover network "is_part_of" relations

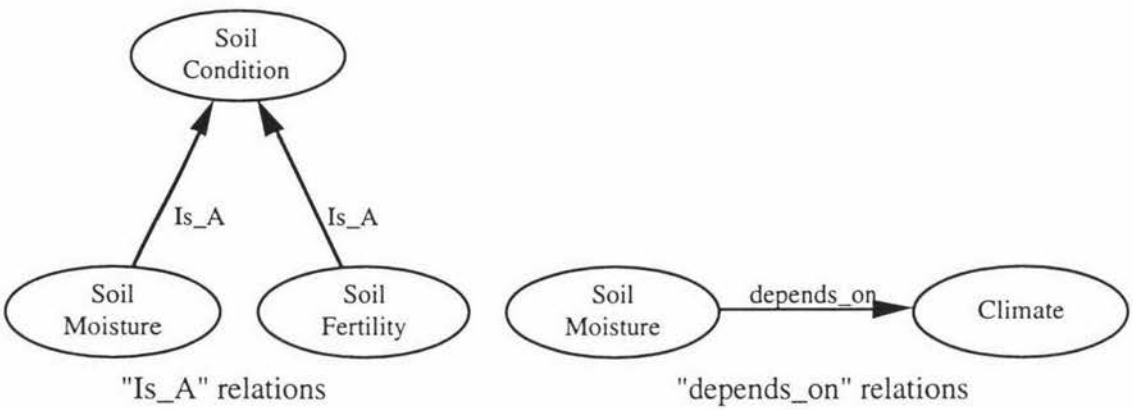


Figure 71 - Soil condition network

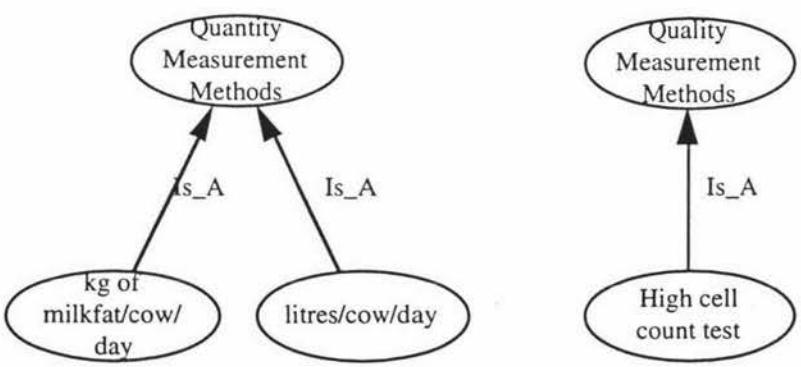


Figure 72 - Measurement methods network- "Is_A" relations

Inference Layer

The inference layer consists of the inference structure of the summer-autumn management domain and the instances of the domain which were instantiated with the primitive inference actions and knowledge roles. The final inference structure designed for the domain is shown in Figure 73.

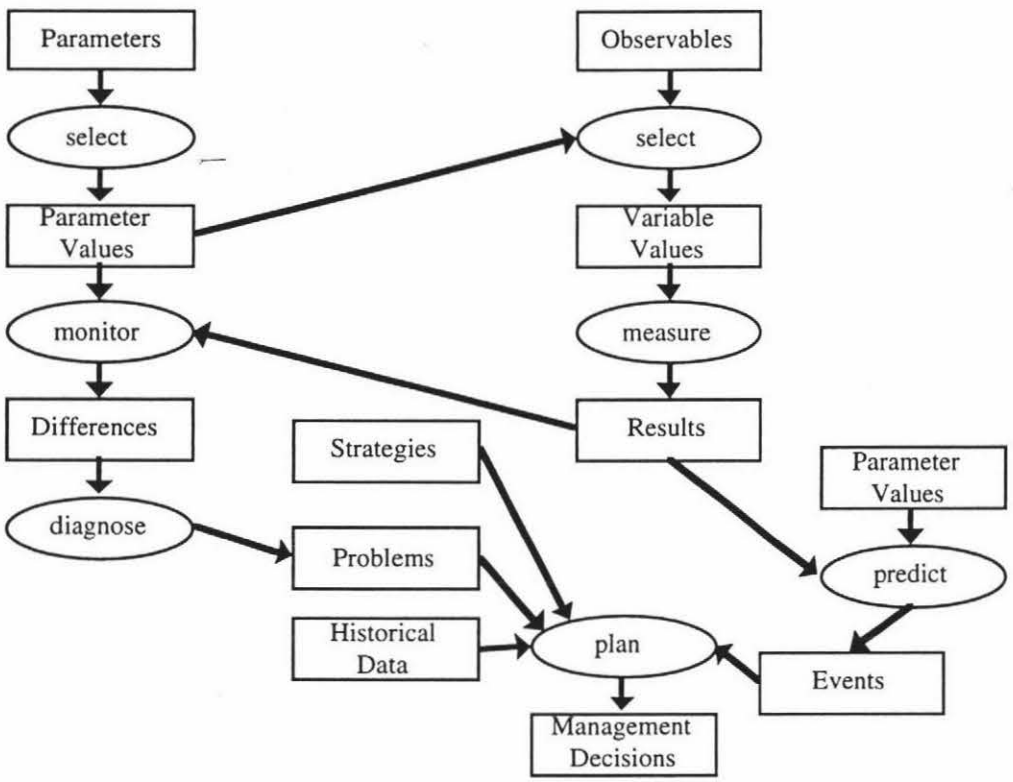


Figure 73- Final inference structure of the summer-autumn management period

Primitive inference actions and their input and output knowledge roles

select (Parameters): The inference structure starts with the selection of a number of parameters concerning the structure of the system being developed. The selection of parameters can be compared to an initial study of how an ideal system should work. This primitive inference action also instantiates the parameters to their values.
input: Overall structure of the dairy system.
output: Parameter Values.

select (Observables) are selected from the real system. This selection may be guided by the parameter variables. The observables are transformed into variables through data abstraction.
input: General data concerning the farm current situation.
output: Variable Values.

measure: Before comparing the variables obtained from the system it is necessary to evaluate them. These primitive inference actions transform the observables into results that can be compared to the ideal parameters.
input: Variable Values.
output: Results

compare: The desired (parameter) values are compared with the result of the test and measure of the actual values. This comparison may generate no difference between the desired and actual values. But, in case a difference is produced, the problem causing the difference is identified.
input: Results and Parameter Values.
output: Differences.

classify: This primitive inference action classifies the difference to determine whether it is significant or not.
input: Differences.
output: Problems.

construct: Identifies the actions that have to be taken in order to overcome the problems and maintain the right results.
input: Problems, Strategies, Historical Data and Events.
output: Management Decisions.

predict: It uses some of the results to determine what will happen next in terms of events.
input: Results and Parameter Values.
output: Events.

Concepts of the domain which were instantiated with the knowledge roles:

Knowledge Roles	Concepts of the Domain
Parameter (ideal) and Observables (actual)	- Dairy Cow Breed - Types of Supplements - Grazing System Management - Reproductive Performance - Measurement Methods - Soil Types
Parameter and Observable Values	- Rainfall - Cow Condition - Feed Demand (Dry Matter/hectare) - Feed Supply (Dry Matter/hectare) - Type of Grazing System - Animal Intake (kg of Dry Matter/day) - Pasture Cover (kg of Dry Matter/hectare) - Pasture Growth Rate - Postgrazing Pasture Cover (kg of dry matter/hectare) - Pregrazing Pasture Cover (kg of dry matter/hectare) - Production Level (kg of milkfat/cow/day) - Reproductive Performance (number of empty cows, conception rate) - Stocking Rate
Results	- Average cow condition of 4.5 - Pregrazing Pasture cover of 1800 kg of dry matter/hectare - conception rate of 75% - 10 empty cows - Stocking rate of 3 cows/hectare - Rainfall of 3 mm overnight

Problems	- Feed Shortage - Young Stock below target weight - Low in calf rate - Low Production Level - Empty Cows - Low Condition Score - Non ideal weather conditions
Management Decisions and Strategies (from Planning)	- Feed Silage - Change Rotation Length - Feed Crop - Reduce Cow Intake - Sell unwanted cows - Dry off thin cows - Dry off young cows - Milk the herd once per day - Dry off the herd
Events (from Prediction)	- Make hay and silage - Feed Crop - Identify unwanted cows

Relationships of the domain instantiated with the primitive inference actions

select (Parameters):

- Study the general picture of a dairy farm.
- From this study select the parameters with the ideal values to compare to the real situation on the farm. For the dairy farmers, these parameters can be records of the last years level of production, stocking rate, average pasture cover, drying off date etcetera.

select (Observables):

- Walk the farm (usually on a fortnightly basis) to analyse the situation and obtain the variables to be tested and measured.

measure:

- Do the pregnancy test to identify empty cows (pregnancy test).
- Condition score the cows to make sure they are at an acceptable condition.
- Test the milk for milkfat constituents (Herd Test).
- Use the plate meter to measure pasture cover.
- Use levels of milkfat and protein to measure pasture quality.
- Measure how much feed the cows are eating (cow intake) to make sure the production level is being maintained.

compare:

- There are no examples of this primitive inference action as it consists of the comparison of the results of the tests and measurements with the results of the last years. Sometimes this comparison is done based on tacit knowledge.

classify:

- Feed deficiency makes the milk production drop.
- Feed deficiency causes low in calf rate.
- Low in calf rate causes empty cows.
- Young stock below target weight jeopardises the following season.
- Condition scores below acceptable levels causes drop in production and low in calf rates.

construct:

- Rotation length is planned to maximise pasture growth.
- The feedbudget is planned according to feed demand and feed supply.
- The pasture cover management plan consists of not allowing the cover to drop too much.
- The drying off date is planned according to cow condition and pasture cover.

predict:

- The weather predicts when to make hay.
- Residual pasture cover predicts production level.
- Production levels predict cow condition.

Task Layer

The task layer consists of the decomposition of the generic task and was based on the instantiation of the "Manage_the_farm" template to the inference layer.

task **Manage_the_farm**
goal: Maximise current milkfat production without jeopardising next season's yield.
task structure
select(Observable ---> Variable Values)
select(Parameters ---> Parameter Values)
measure(Variable Values ---> Results)
monitor(Results and Parameter Values ---> Differences)
diagnose(Differences ---> Problems)
plan(Problems and Strategies ---> Management Decisions)
predict(Results and Parameter Values ---> Events)

This instantiation detected the necessity of decomposing the generic task into subtasks. Following are the four subtasks which resulted from the decomposition of the generic task 'Manage_the_farm'.

task **Planning Management Decisions**
goal: Organise a plan of actions that have to be taken in order to maintain the production level of the current system without risking next season's production.
task structure
construct (Problems, Strategies, Historical Data and Events --> Management Decisions)

The second activity consists of assessing the events on the farm:

task **Monitoring Events**
goal: Make sure the results being obtained are in accordance to the goals to be achieved.
task structure
select (Observable --> Variable Values)
select (Parameters --> Parameter Values)
measure (Variable Values --> Results)
compare (Results and Parameter Values --> Differences)

The third activity consists of identifying the cause of the problems:

task **Diagnose Problems**
goal: identify cause of problems
task structure
select (Observable --> Variable Values)
select (Parameters --> Parameter Values)
measure (Variable Values --> Results)
compare (Results and Parameter Values --> Differences)
classify (Differences --> Problems)

The fourth activity consists of the prediction of events on the farm:

task **Prediction of Events**
goal: Identify the future state of the events on the farm in order to take specific actions to avoid problems.
task structure
select (Observable --> Variable Values)
select (Parameters --> Parameter Values)
measure (Variable Values --> Results)
predict (Results and Parameter Values --> Events)

Task decomposition

This is the main part of the task layer, the specification of the generic tasks: Planning Management Decisions, Monitoring Events, Diagnosis of Problems and Prediction of Events. It consists of the decomposition of the generic tasks into the subtasks and goals associated with them. This specification is the overall framework of the tasks performed by the dairy farmers in order to achieve their goal. The tasks described are related to the inference structure of Figure 73.

For all subtasks:

- The names of the primitive inference actions are italicised in the task structure.
- The arrows in the task structure describe the relation between input and output knowledge roles which are represented by concepts of the domain.
- In case a Transfer Task is necessary for the achievement of the subtask being described, its name will be italicised.

Subtasks of the generic task: Monitoring Events

subtask: Measure Pasture Cover

goals: - Make sure there will not be a feed deficit in the next few days.
- Verify that the cows are being well fed at the moment.

task structure:

select(Pasture Cover --> Average Pasture Cover)

select(Pasture Cover System --> Ideal Pasture Cover)

Walk the farm

Select a paddock as a sample

measure(Pasture Cover --> Results of Pasture Cover Measurement)

If the month is January, February or March

Then

Calculate Pasture Cover informally once a fortnight;

Else

Measure Pasture Cover using the Plate Meter once a week

End If

compare(Ideal Pasture Cover with Results of Pasture Cover Measurement --> Differences in the Pasture Cover)

subtask: Measure Production Level

goals: - Confirm that the animals are being well fed.
- Make sure the production goals are being achieved.

task structure:

Repeat every day for herd

select(Production Level --> Production/cow/day)

select(Production System --> Ideal Production)

Measure total of litres

Measure kg of milkfat

Divide total of litres by numbers of cows

Divide total of milkfat by number of cows

measure(Milkfat/cow/day --> Results of milkfat/cow/day)

measure(Litres/cow/day --> Results of litres/cow/day)

compare(Ideal Production with Results of milkfat/cow/day --> Difference in Production Level)

compare(Ideal Production with Results of litres/cow/day --> Differences in Production Level)

End.

subtask: Measure Body Condition (Condition Score)

goals: - Confirm that the cows are being well fed.

- Identify thin cows to be dried off.

task structure:

select(Condition Score ---> Condition Score of the cows)

select(Condition Score System---> Ideal Condition Score)

For herd

measure(Condition Score of the cows ---> Results of Cow Condition)

compare(Results of Cow Condition with Ideal Condition Score ---> Differences in Condition Score)

subtask: Monitor Crop

goals: - Make sure the quality of the crop is not being compromised.

task structure:

select(Type of Crop --> Crop)

select(Supplement System --> Ideal Quality)

measure(Crop --> Results)

compare(Crop with Ideal Quality --> Differences in Quality)

subtask: Do the Pregnancy test

goals: - Identify empty cows.

task structure:

select(Cow to be tested --> Selected Cows)

select(Pregnancy Test System --> Ideal Percentage of Pregnant Cows)

measure(Selected Cows --> Results of Pregnancy Test)

compare(Results of Pregnancy Test with Ideal Percentage of Pregnant Cows --> Differences in Percentage of Pregnant Cows)

<u>Subtasks of the generic task: Diagnose Problem</u>

subtask: Identify poor producing cows

goals: - Maintain production level

task structure:

Measure production level

If the cow is producing less than 0.1 kg of milkfat/day

Then

classify(Difference in Production Level --> Poor Producing Cow)

End If.

subtask: Identify thin cows

goals: - Maintain body condition

task structure:

Measure Body Condition

If the condition score <= 3.5 and there is no time to improve body condition

Then

classify(Difference in Condition Score --> Thin Cow)

End If.

subtask: Identify Feed Shortage (Between April and August)

goals: - Maintain cows being well fed

- Make sure that there will be enough feed for the cows at calving time and early lactation

task structure:

Measure Pasture Cover

If Average Pasture Cover is below 1400 kg of Dry Matter /hectare and Supplements are not enough to cover the deficit

Then

classify(Difference in Pasture Cover --> Feed Shortage)

End If.

If Average Pasture Cover will fall below 2200 kg of Dry Matter /hectare in late July

Then

classify(Difference in Pasture Cover --> Feed Shortage)

End If.

subtask: Identify drop in production level

goals: - Maintain Production Level

task structure:

Measure Production Level

If the production has fallen below 0.6 kg of milkfat/cow/day

Then

classify(Difference in Production Level --> Drop in Production Level)

End If.

subtask: Identify quality problem of summer crop

goals: - Graze crop before its quality declines

task structure:

Measure Crop

If the crop has grown too much

Then

classify(Difference in supplement --> Quality problem with summer crop)

End If.

Subtasks of the generic task: Predict Events

subtask: Predict production level by estimating cow intake

goals: - Maintain production level by predicting how much the cows are eating

task structure:

For one paddock

Measure Pasture Cover before grazing (Pregrazing)

Measure Pasture Cover after grazing (Postgrazing)

Subtract Postgrazing from Pregrazing to get Residual Pasture Cover

Divide the result of this subtraction by the number of cows

If the cow intake is below 10 kg of Dry Matter/day

Then

predict(Results of Residual Pasture Cover --> Improve Animal's Diet)

End If.

Subtasks of the generic task: Plan Management Decisions

subtask: Change Rotation Length

goals: - Maintain cow intake by selling marked cows.

- Improve average pasture cover.

task structure:

If *Identify Feed Shortage* is true **and**
Predict empty cows is true
Then
construct(Problem: Empty Cows; Strategy for empty Cows ---> Sell empty cows)
End If
If empty cows have already sold **and**
Identify Feed Shortage is true **and**
Identify Poor Producing Cows is true
Then
construct(Problem: Poor Producing Cows; Strategy for Poor Producing Cows ---> Sell Poor Producing Cows)
End If
If empty and poor producing cows have already sold **and**
Identify Feed Shortage is true **and**
Identify thin Cows is true
Then
construct(Problem: Thin Cows; Strategy for Thin Cows ---> Sell Thin Cows)
End If.

subtask: Feed Supplement
goals: - Maintain Production Level
task structure:

If *Identify Drop in Production Level* is true
Then
If crop is available
Then
If crop quality is declining
Then
construct(Problem:Drop in Production Level; Crop Strategy --> Feed Crop
Else
construct(Problem:Drop in Production Level; Silage Strategy --> Feed Silage
End If
Else
construct(Problem:Drop in Production Level; Silage Strategy --> Feed Silage)
End If
End If.

subtask: Identify Cull Cows
goals: - Maintain cow intake by selling marked cows
- Improve Average Pasture Cover
task structure:

If *Identify feed shortage* is true **and**
Predict empty cows is true
Then
construct(Problem:Empty Cow; Strategy for Empty Cows --> Sell empty cows)
End If
If empty cows have already been sold **and**
Identify feed shortage is true **and**
Identify poor producing cows is true
Then
construct(Problem:Poor Producing Cows; Strategy for Poor Producing Cows --> Sell Poor Producing Cows)
End If

If empty and poor producing cows have been already sold **and**
Identify feed shortage is true **and**
Identify thin cows is true
Then
construct(Problem:Thin Cows; Strategy for Thin Cows; Strategy for Thin Cows --> Sell Thin cows)
End If.

subtask: Milk cows once per day
goals: - Maintain Average Pasture Cover and Cow Condition
task structure:
If *Identify Feed Shortage* is true **and**
Cull cows have already been sold **and**
Cow Condition can not be used
Then
construct(Problem:Feed Shortage and Cow Condition; Drying off Strategy --> Milk herd once a day)
End If.

subtask: Dry off the herd
goals: - Maintain Average Pasture Cover and Cow Condition
task structure:
If *Identify Feed Shortage* is true **and**
Cull cows have already been sold **and**
Cow Condition can not be used **and**
The herd is being milked once a day already
Then
construct(Problems:Feed Shortage and Cow Condition; Drying off Strategy --> Dry off the Herd).
End If.

Appendix 2: Knowledge Representation Model

This appendix gives both the object-oriented model of the domain primitives, and the object-oriented task model.

Domain Primitives

The object-oriented model of the domain primitives took place at a high level. The existing structures were converted into the object-oriented paradigm, but further decomposition, (the identification of attributes and services) did not take place unless it was necessary for the prototype which was constructed.

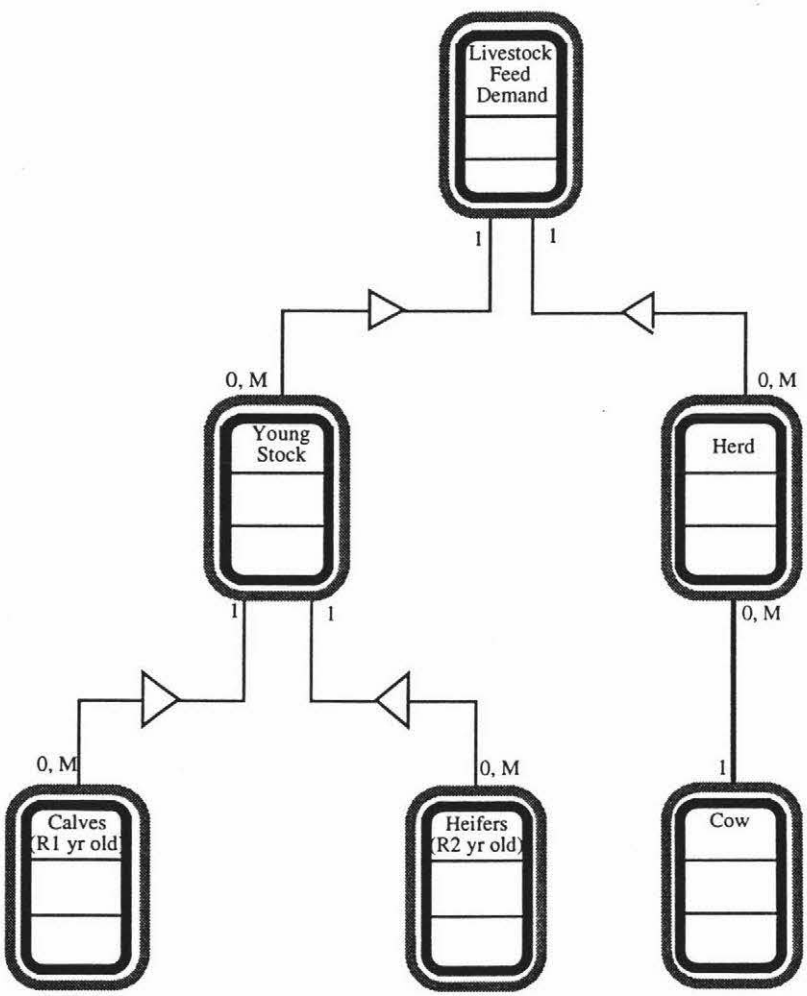


Figure 74 - Livestock Network

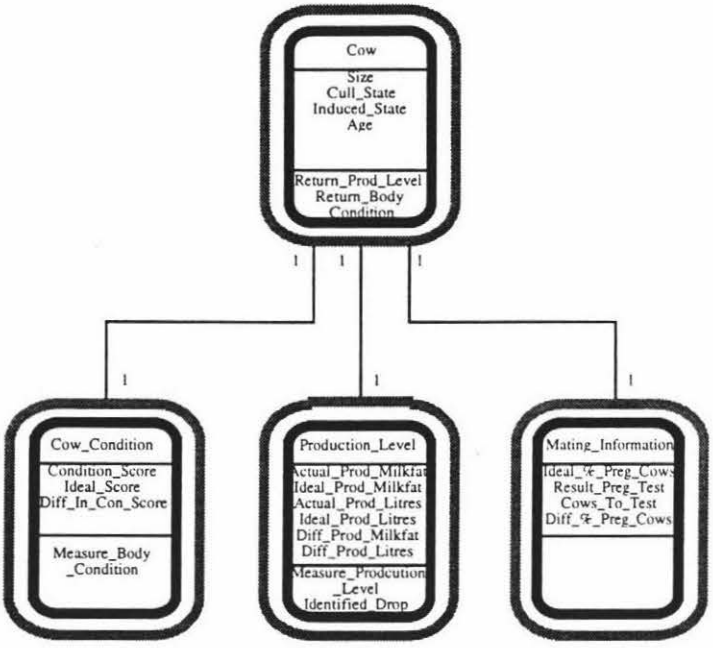


Figure 75 - Cow

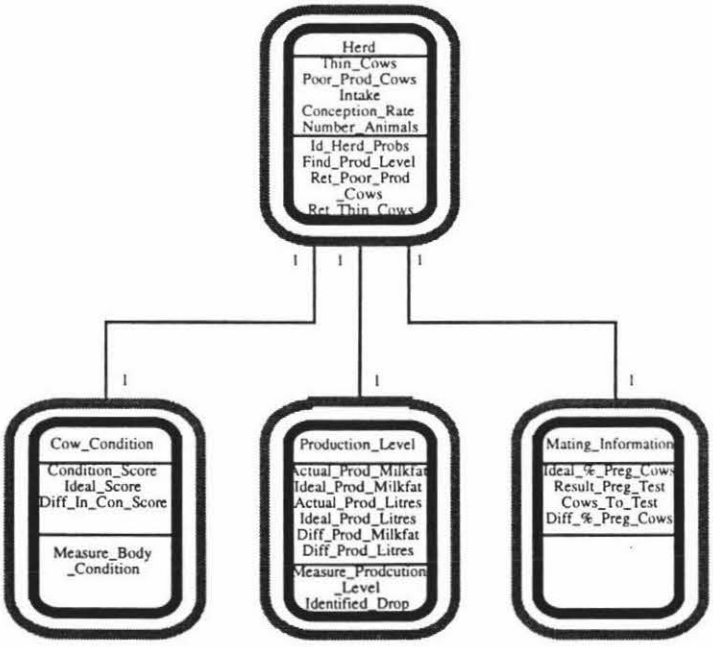


Figure 76 - Herd

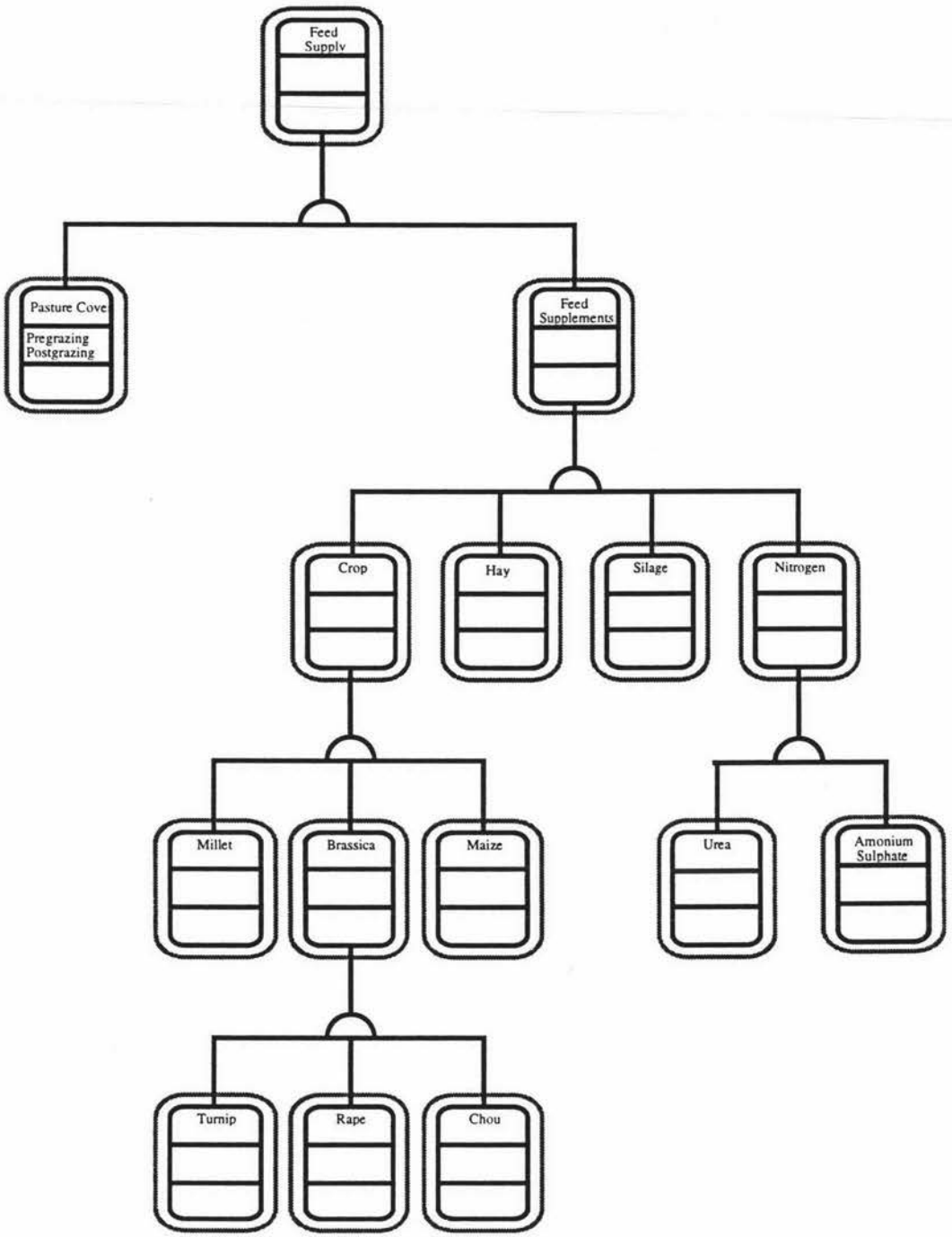


Figure 77- Feed Network

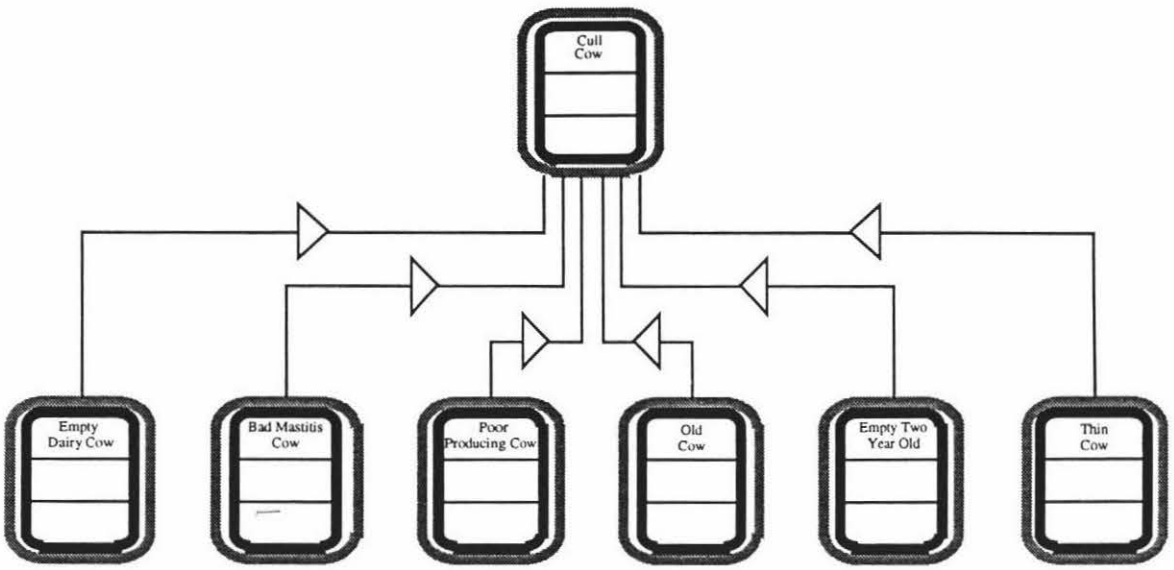


Figure 78 - Cull Cows Object-Oriented Model

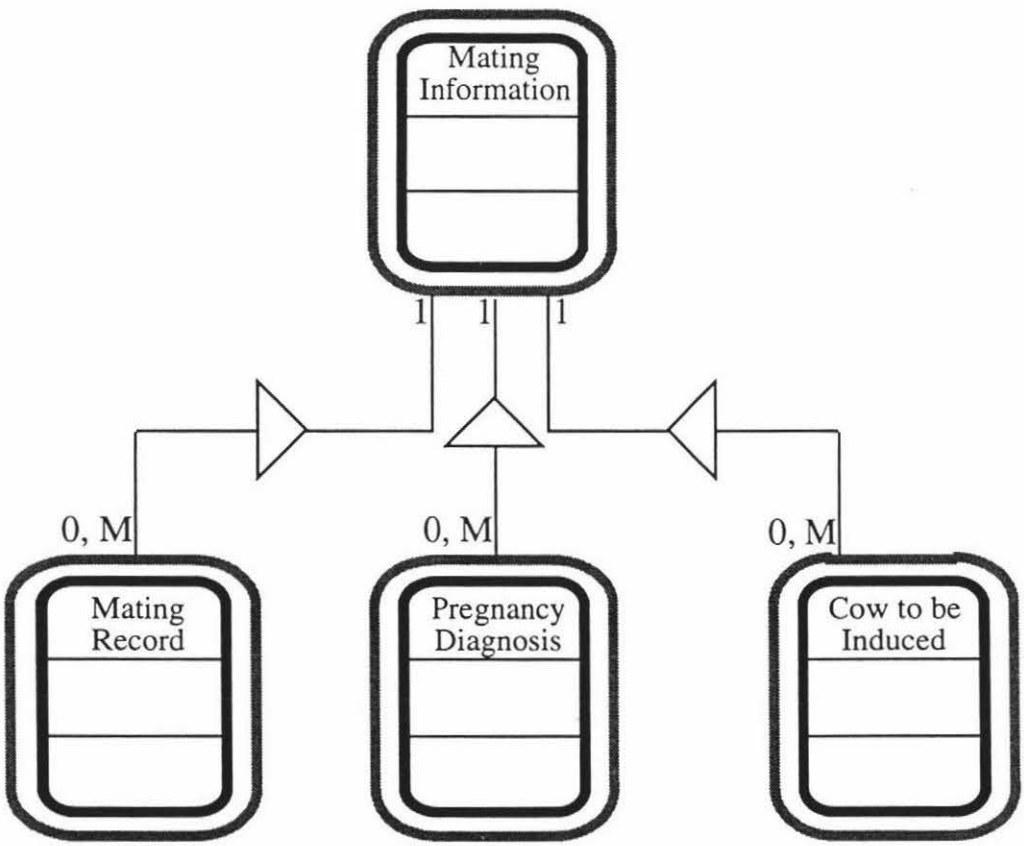


Figure 79 - Mating Information

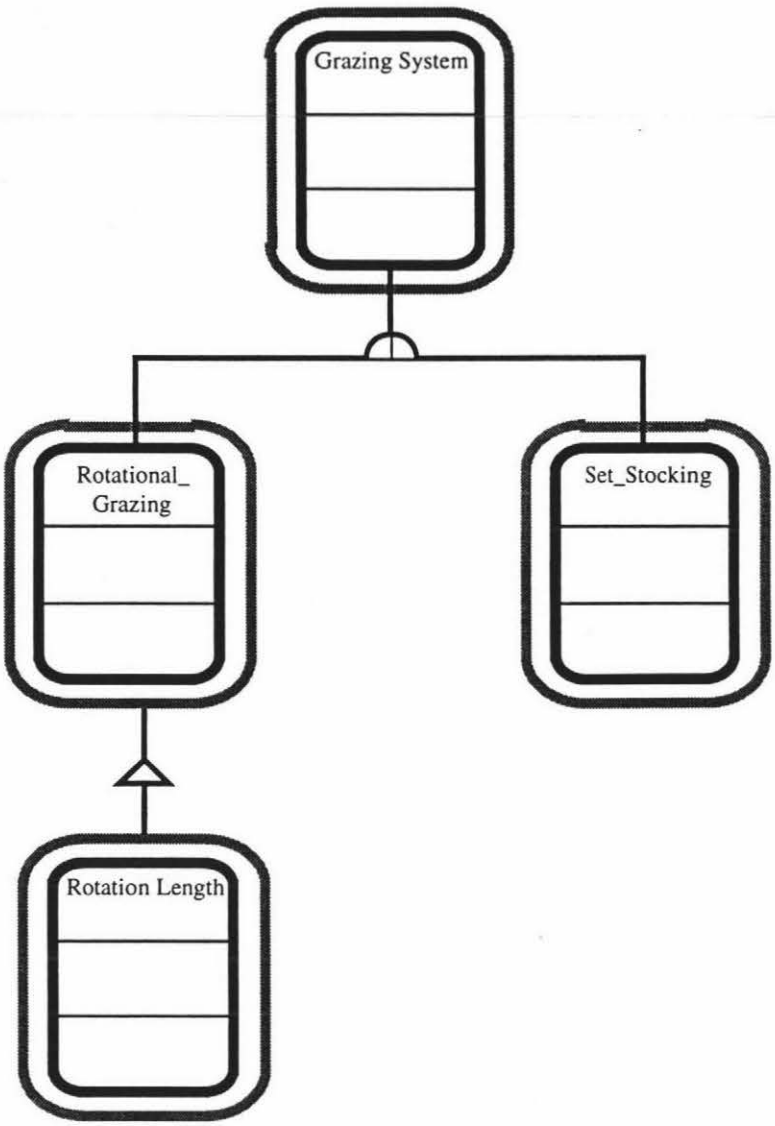


Figure 80 - Grazing System

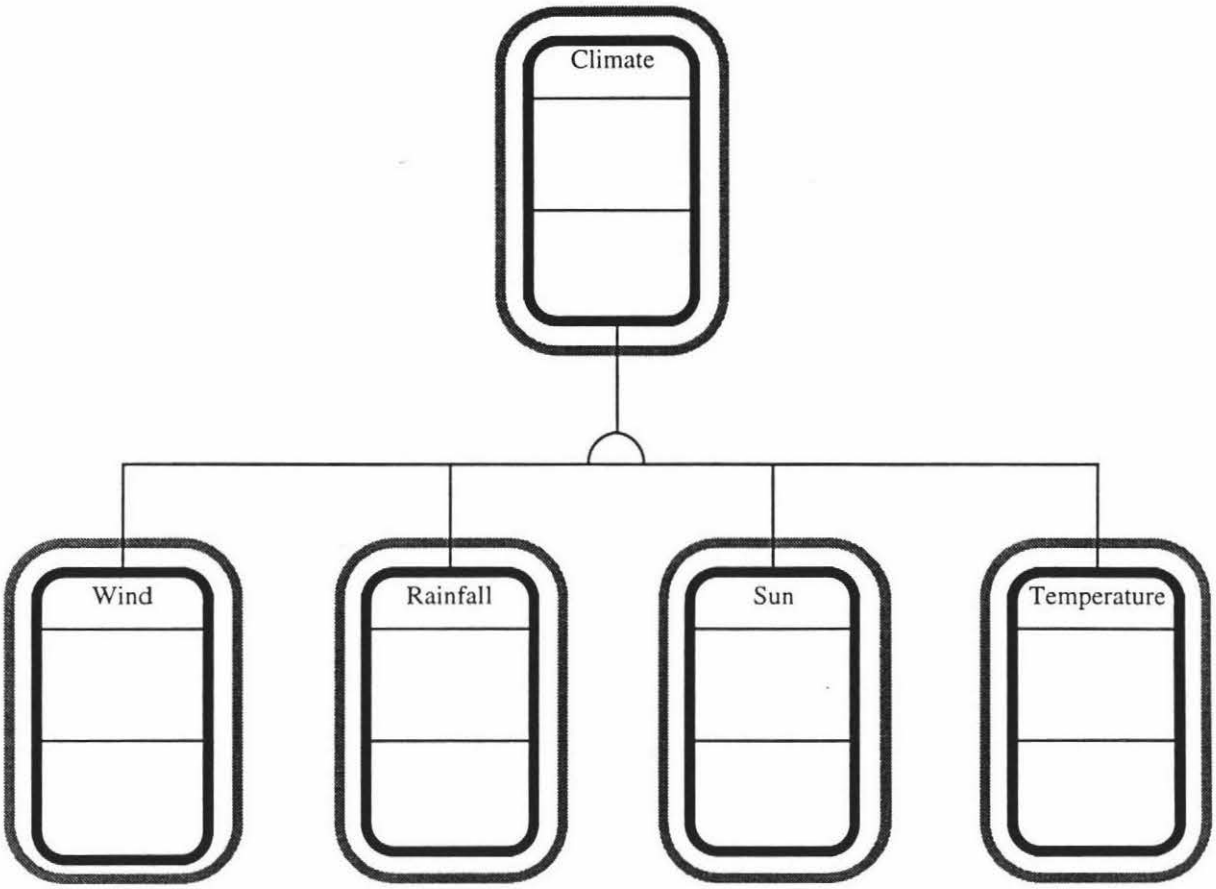


Figure 81 - Climate Network

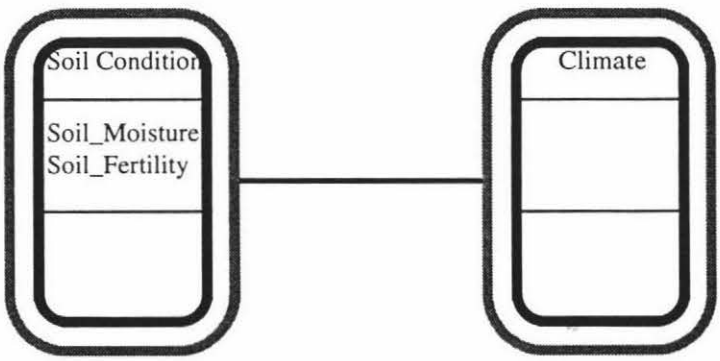


Figure 82 - Soil Condition Network

Object Oriented Task Model Decomposition

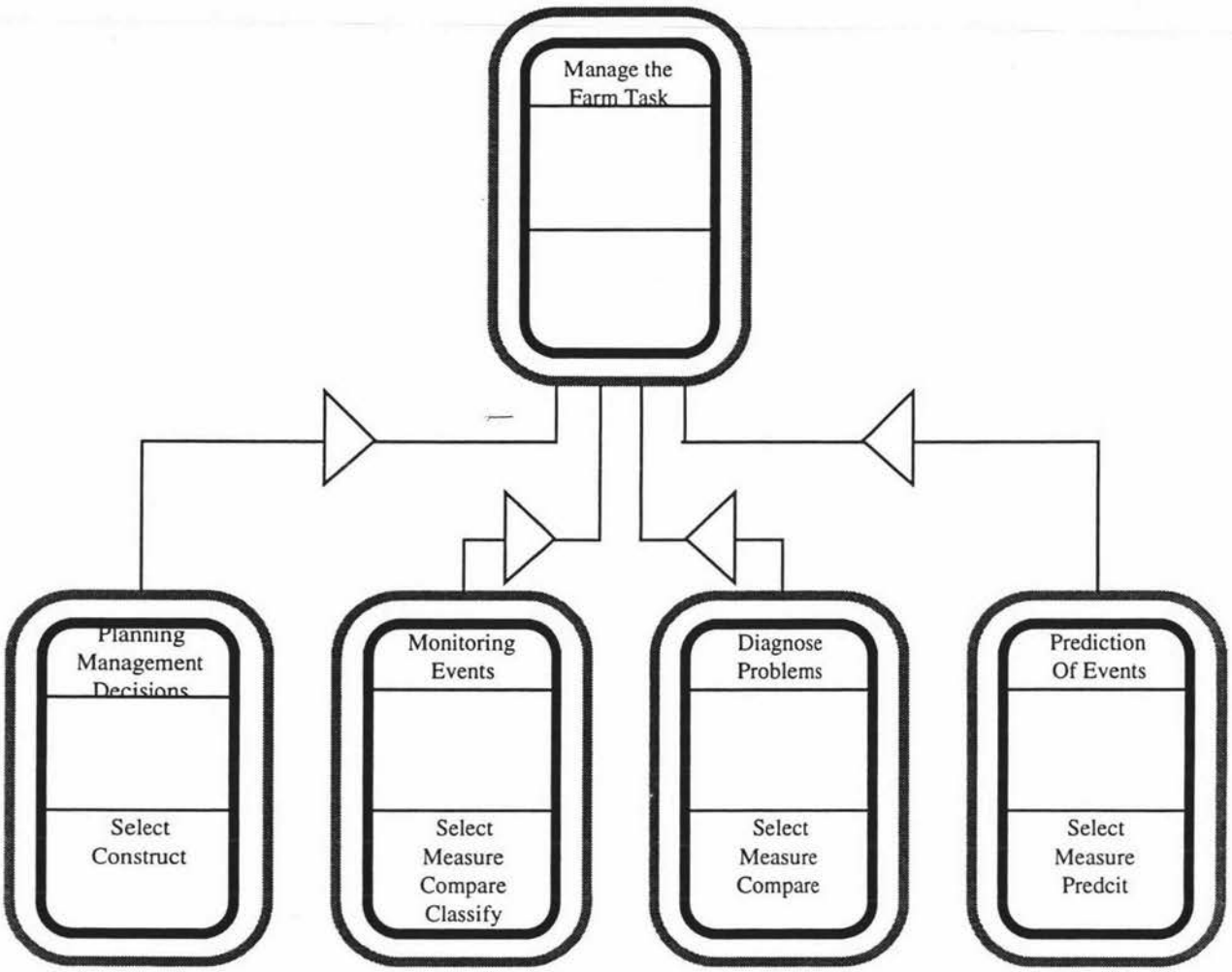


Figure 83 - High Level Task Model

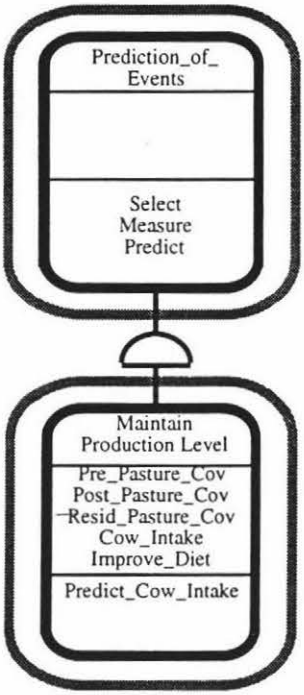


Figure 84 - Prediction of Events Decomposition

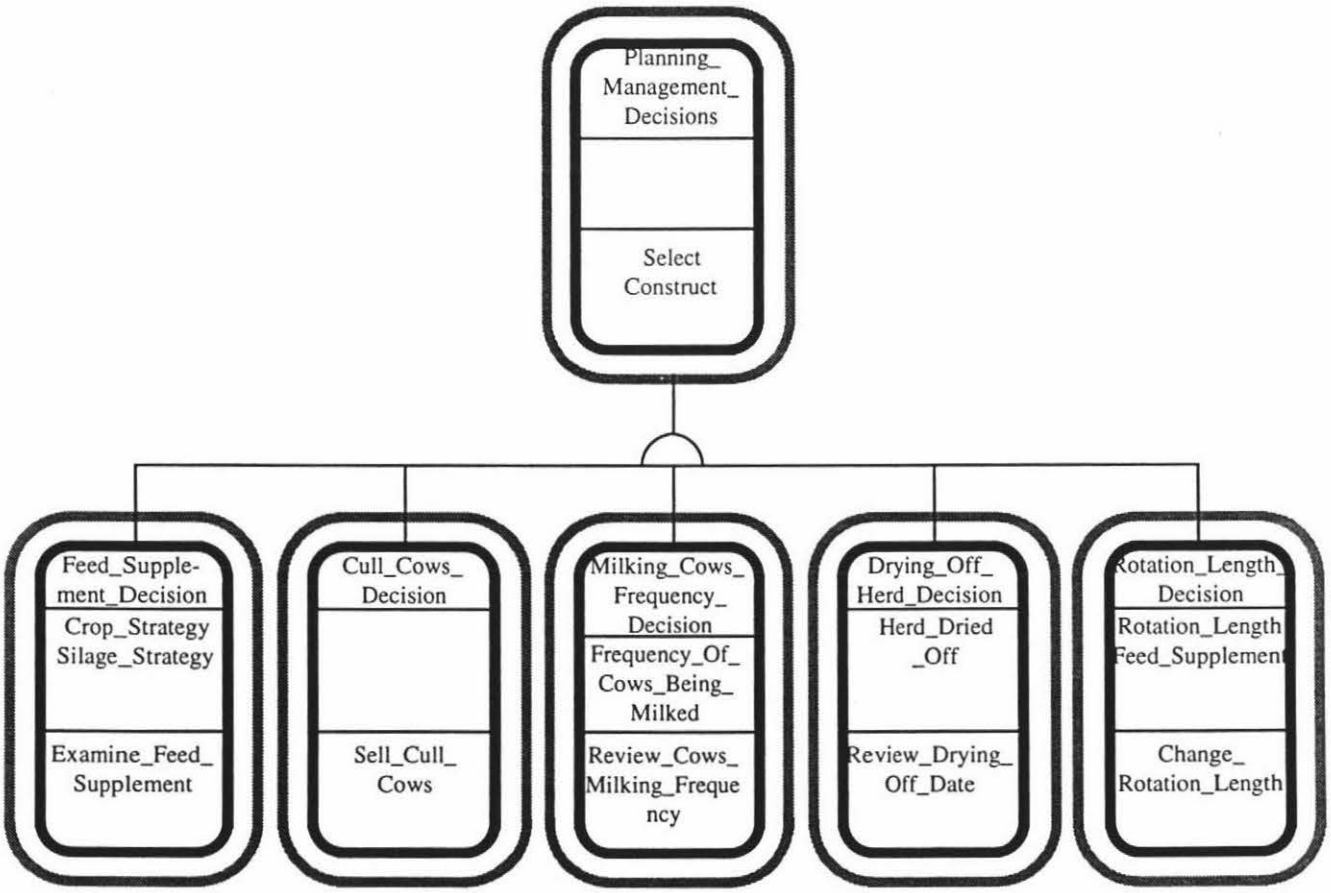


Figure 85 - Planning Management Decomposition

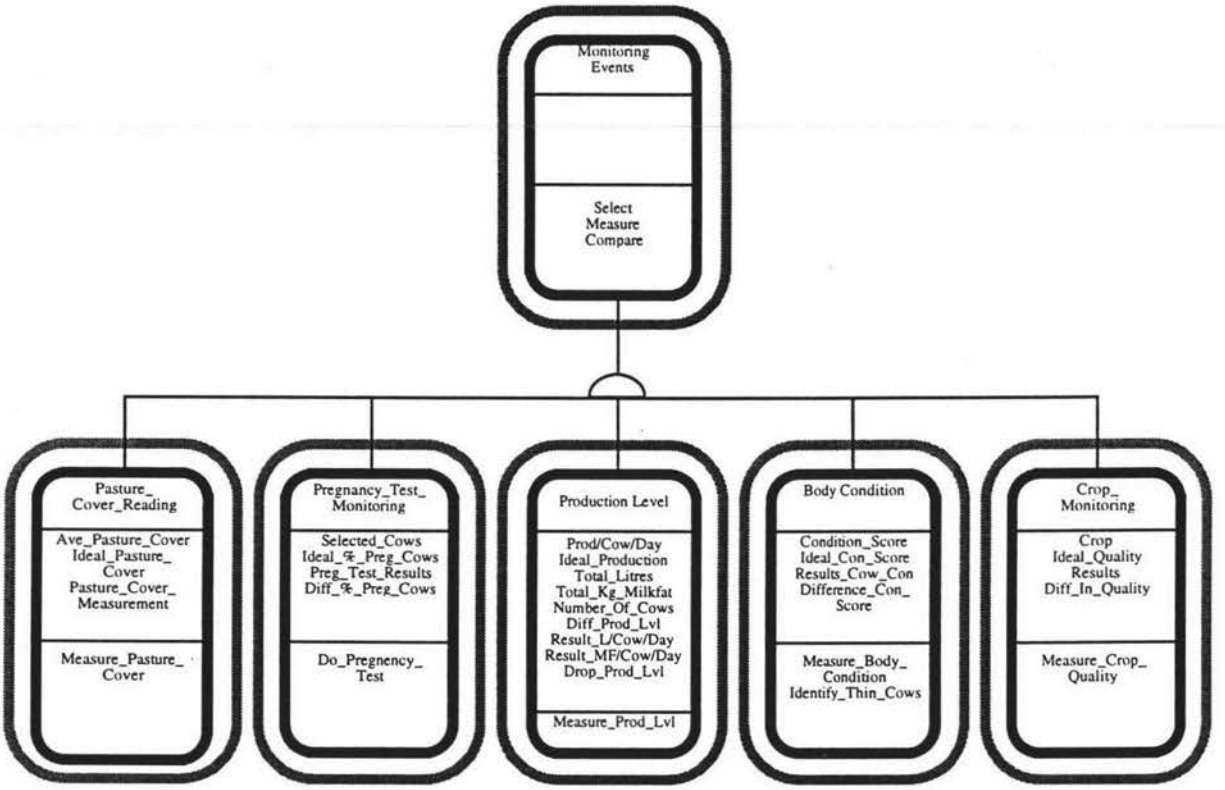


Figure 86 - Monitoring Events Decomposition

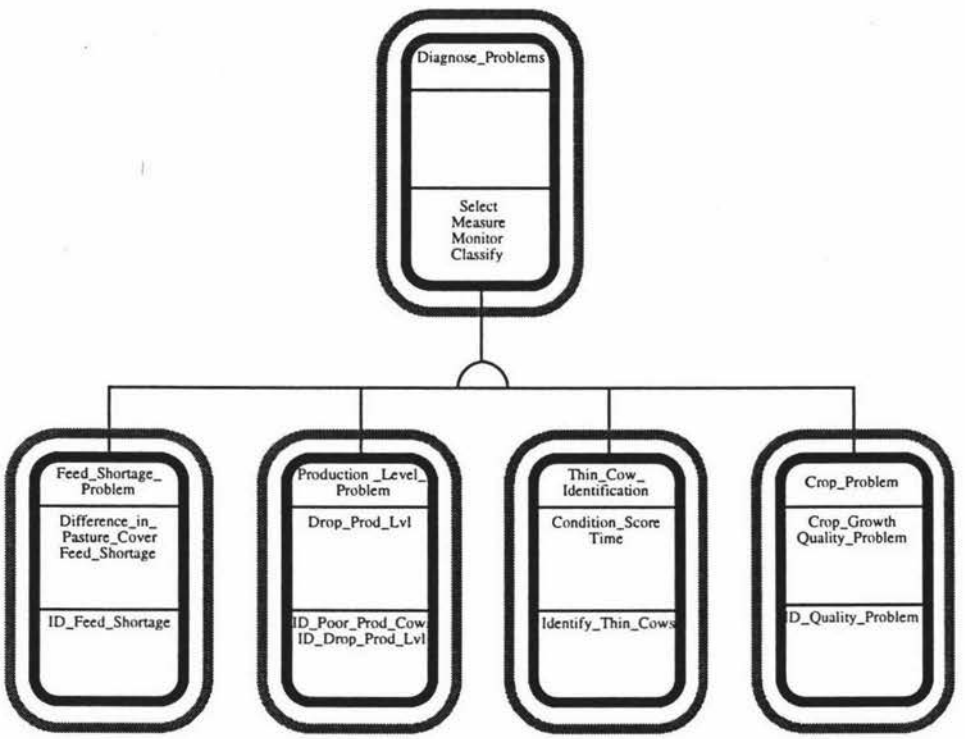


Figure 87 - Diagnose Problem Model

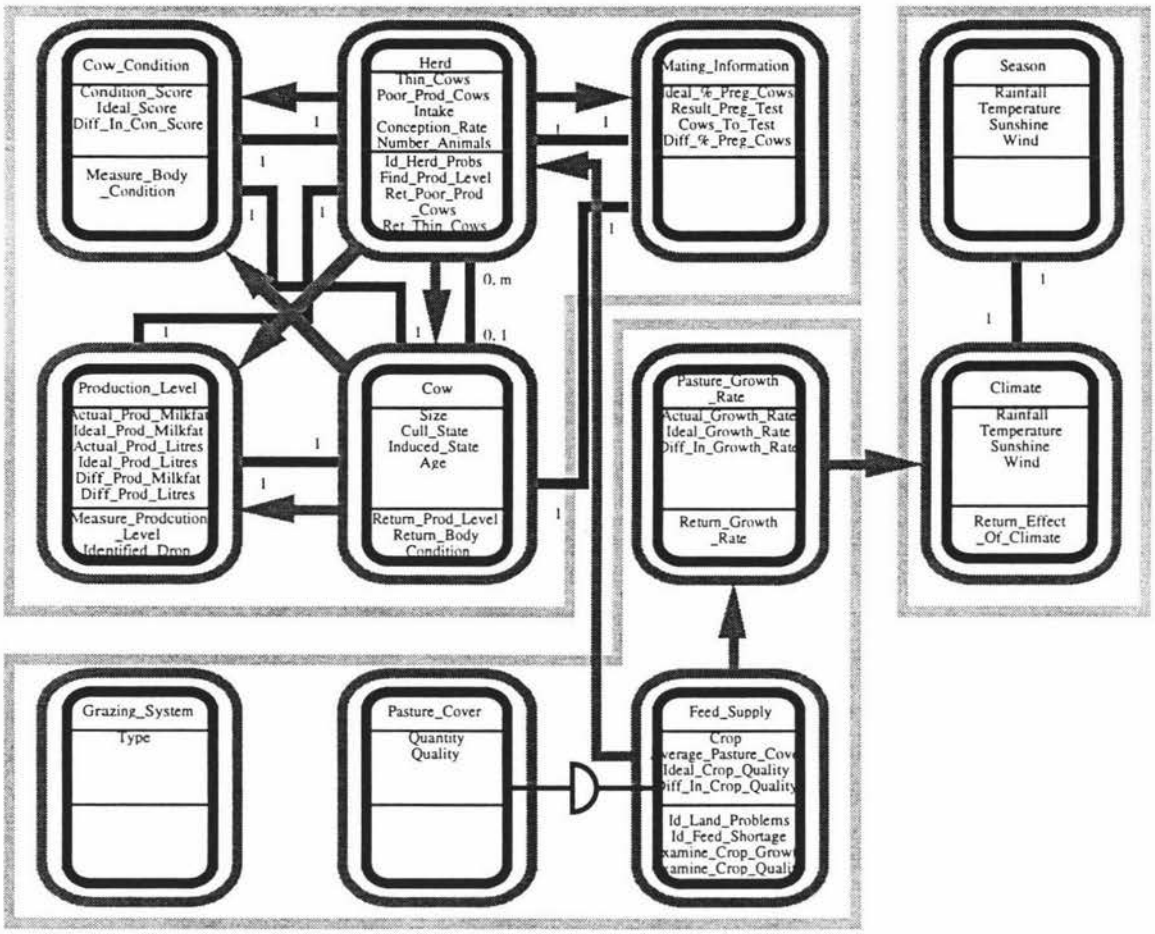


Figure 88 - Object-Oriented Model of Network of Primitives

Appendix 3: Sample Session

The example involved the "Comparison of Farmers" option. A single farmer was selected, and the figures for the first period analysed. Then management preferences were changed, to simulate a change in the farm situation, and the analysis performed. Before the third analysis was undertaken, additional supplements were added to gauge their effect.

The session is firstly depicted through the series of screens that were cycled through, and then the transcript generated is listed.

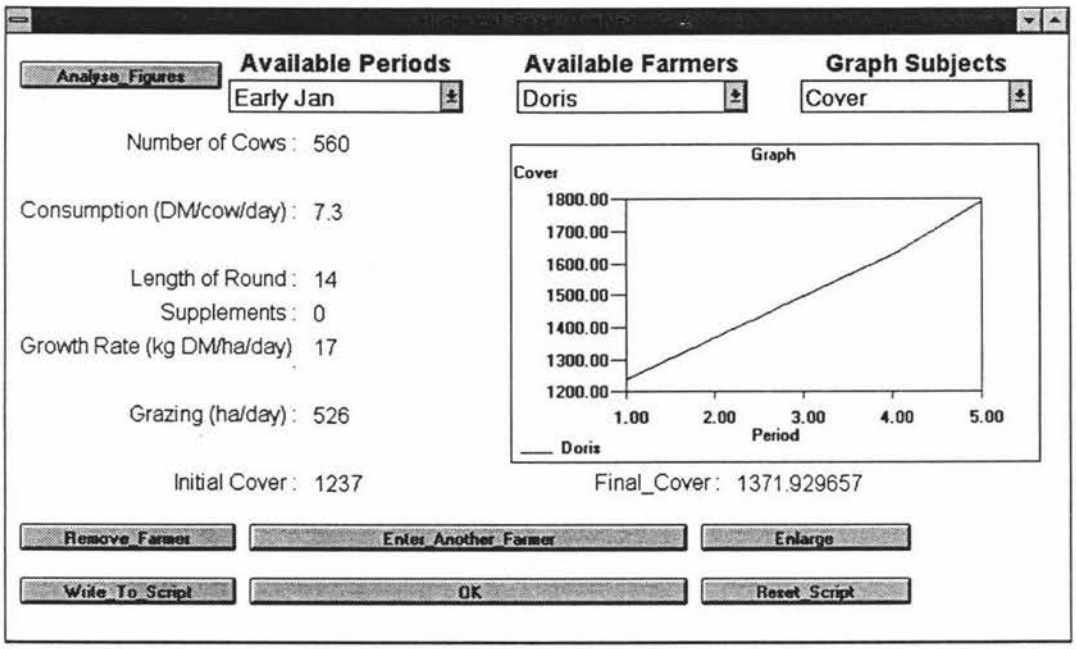


Figure 89 - Initial figures after one farmer has been added

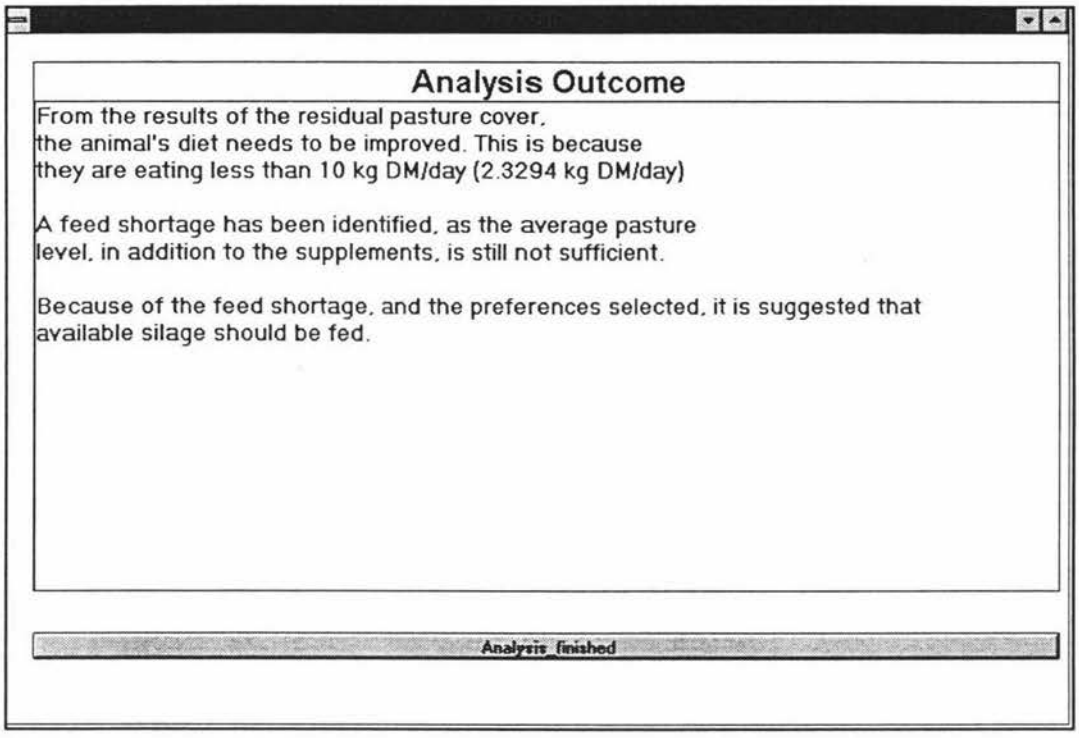


Figure 90 - The results of analysis on the first set of figures

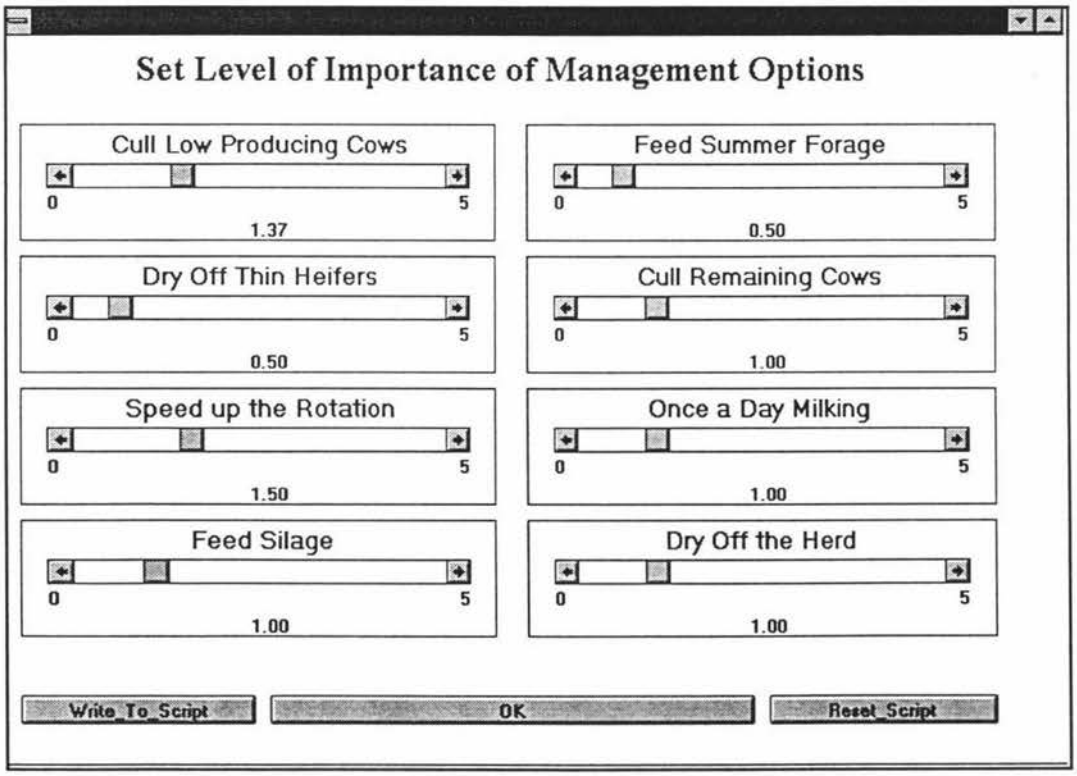


Figure 91 - Preferences after 'Feed Silage' has been altered from 4 to 1

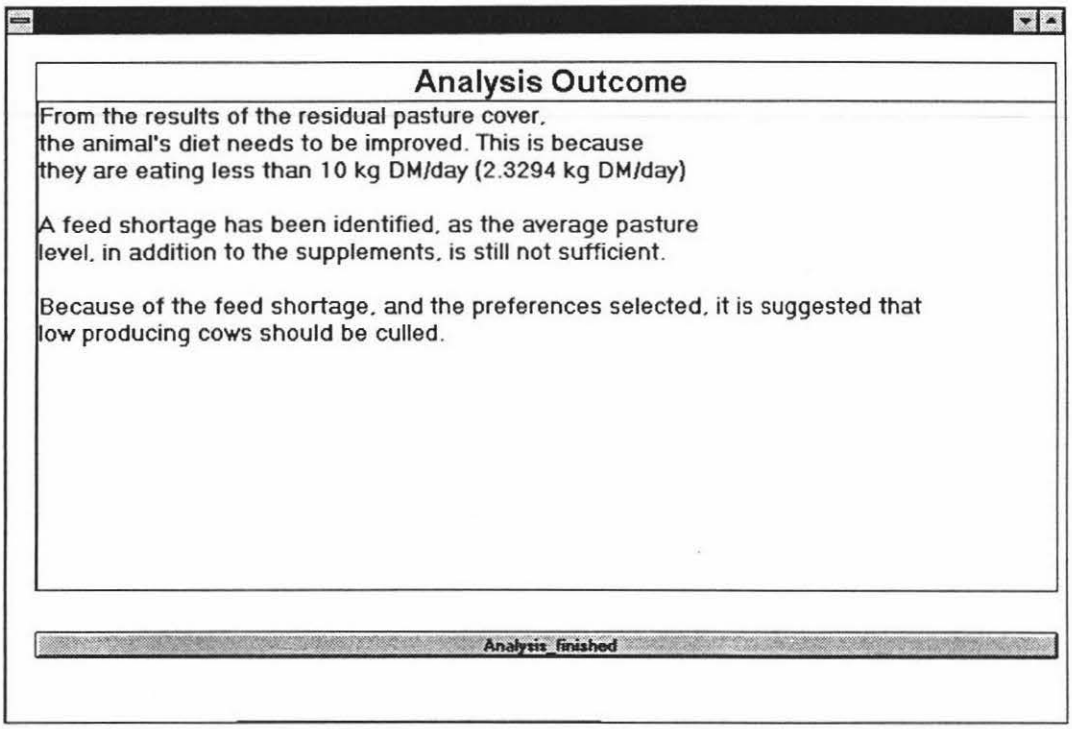


Figure 92 - The results of the second analysis on the modified set of figures

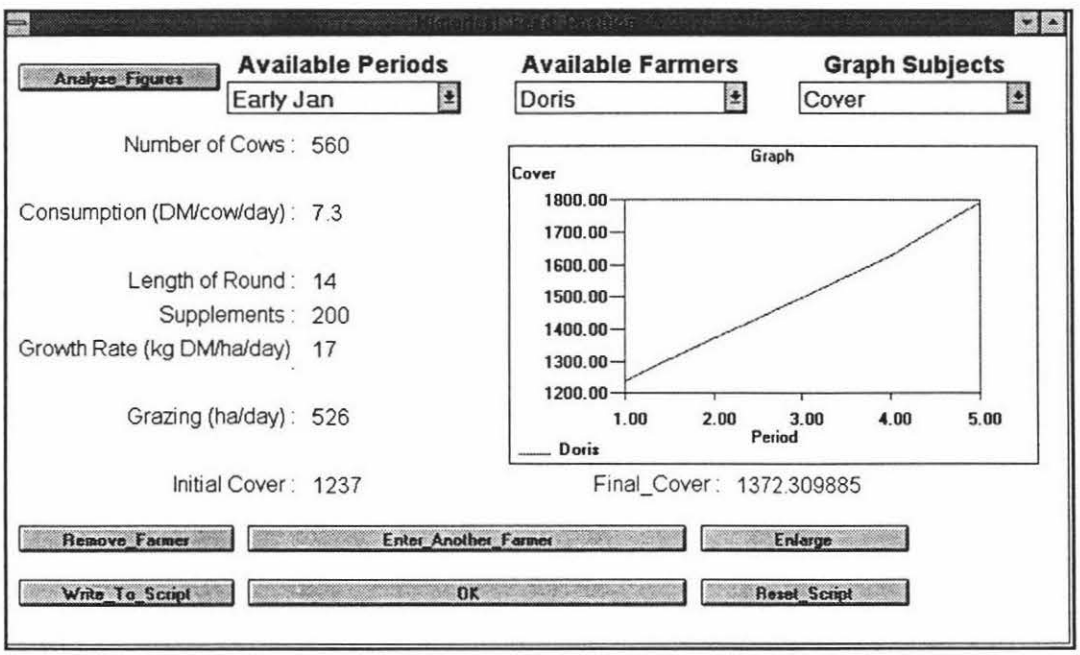


Figure 93 - Figures after 'Supplements' has been altered from 0 to 200

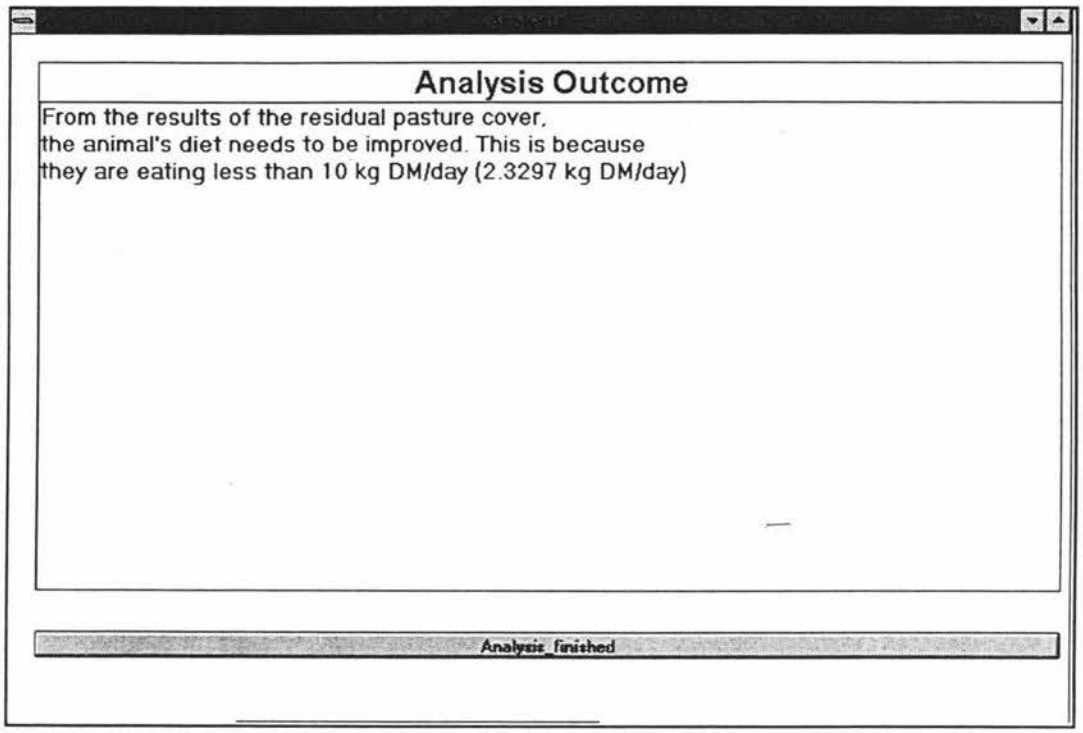


Figure 94 - The results of the third analysis on the modified set of figures

The following is the functional prototype's generated transcript from the demonstration session. Interface actions such as pressing buttons are denoted by <>. Conclusions reached are presented as normal text.

-----Transcript Begins -----

<Comparison of Farmers selected>

<New Farmer entered>

<Analysis selected>

From the results of the residual pasture cover,
the animal's diet needs to be improved. This is because
they are eating less than 10 kg DM/day (2.3294 kg DM/day)

A feed shortage has been identified, as the average pasture
level, in addition to the supplements, is still not sufficient.

Because of the feed shortage, and the preferences selected, it is suggested that
available silage should be fed.

<Analysis Finished selected>

<Alter Preferences selected>

The priority of Feed_Silage changed to 1

<Analysis selected>

From the results of the residual pasture cover,
the animal's diet needs to be improved. This is because
they are eating less than 10 kg DM/day (2.3294 kg DM/day)

A feed shortage has been identified, as the average pasture
level, in addition to the supplements, is still not sufficient.

Because of the feed shortage, and the preferences selected, it is suggested that
low producing cows should be culled.

<Analysis Finished selected>

The value of Supplements changed to 200

<Analysis selected>

From the results of the residual pasture cover,
the animal's diet needs to be improved. This is because
they are eating less than 10 kg DM/day (2.3297 kg DM/day)

-----Transcript Ends -----