DECENTRALISING THE CODIFICATION OF RULES IN A DECISION SUPPORT EXPERT KNOWLEDGE BASE

by

Erika de Kock

presented in partial fulfilment of the requirements for the degree

Master of Scientiae
(Computer Science)

in the

Faculty of Engineering, Built Environment and Information Technology

University of Pretoria

Promoter: Professor J.M. Bishop

March 2003

Abstract

Title:	Decentralising the codification of rules in a decision support expert knowledge base
Author:	Erika de Kock
Promoter:	Professor J.M. Bishop
Department:	Computer Science
University:	University of Pretoria
Degree:	Master of Scientiae (Computer Science)

The paradigm of Decision Support Systems (DSS) is to support decision-making, while an Expert System's (ES) major objective is to provide expert advice in specialised situations. Knowledge-Based DSS (KB-DSS), also called Intelligent Decision Support Systems (IDSS), integrate traditional DSS with the advances of ES. A KB-DSS' knowledge base usually contains knowledge expressed by an expert and captured by a knowledge engineer. The indirect transfer between the domain expert and the knowledge base through a knowledge engineer may lead to a long and inefficient knowledge acquisition process.

This thesis compares 11 DSS packages in search of a (KB-) DSS generator where domain experts can specify and maintain a Specific Decision Support System (SDSS) to assist users in making decisions. The proposed (KB-) DSS-generator is tested with a university and study-program prototype. Since course and study plan programs change intermittently, the (KB-) DSS' knowledge base enables domain experts to set and maintain their course and study plan rules without the assistance of a knowledge engineer. Criteria are set to govern the (KB-) DSS generator search process. Example knowledge base rules are inspected to determine if domain experts will be able to maintain a set of production rules used in a student registration advice system. By developing a prototype and inspecting knowledge base rules, it was found that domain experts would be able to maintain their knowledge in the decentralised knowledge base, on condition that the objects and attributes used in the rule base were first specified by a builder/programmer.

Keywords:

Decision Support, Decision Support Systems, Expert Systems, Knowledge-Based Decision Support Systems, Intelligent Decision Support Systems, Knowledge Management, Knowledge Representation, Decision Support Generator, Rule-Based Systems, Knowledge base

Acknowledgements

I thank the following companies, who kindly sent evaluation versions of their software packages:

- Arlington.
- Distributed Computing Sys
- Expert Choice Inc.
- ILOG
- Infoharvest Inc
- Intellix
- Stasoft
- Vanguard
- Rule Machines Corp, and
- The Haley Enterprise

I extend my appreciation to my promoter, Prof Judy Bishop, for her patience and encouragement, and to my friends: Dr Susan van Zweel, Mr James Thackrah and Mrs Alex Koutsouvelis who edited my use of language.

I acknowledge the financial assistance of the National Research Foundation under Grant No 2053401.

Contents

List of figures

Page

List of tables

List of acronyms

4GL	Fourth Generation Languages
AHP	Analytical Hierarchy Process
AI	Artificial Intelligence
API	Application Programming Interface
BAL	Business Action Language
BOM	Business Object Model
BRS	Business Rule Studio
CBIS	Computer Based Information Systems
CBR	Case Based Reasoning
CDSS	Customer Decision Support Systems
COM	Component Object Model
CSF	Critical Success Factors
DB	Databases
DBMS	Database Management System
DGMS	Dialogue Generation Management Software
DLL	Dynamic Link Library
DSS	Decision Support Systems
DTD	Document Type Definition
EC	Electronic Commerce
EDI	Electronic Data Interchange
EIS	Executive Information Systems
ES	Expert Systems
GDSS	Group Decision Support Systems
GSS	Group Systems Support
GUI	Graphical User Interface
IA	Intelligent Agents
IDDS	Intelligent Decision Support Systems
IRD	Information Requirement Definition
IRS	Information Reporting Systems
IS	Information Systems
ISP	Internet Service Provider
ISS	Intelligent Support Systems
IT	Information Technology
JESS	Java Expert System Shell
JVM	Java Virtual Machine
K-T method	Kepner-Tregoe method/process
KB	Knowledge Base
KB-DSS	Knowledge-based Decision Support Systems

LAN	Local Area Network
LHS	Left-hand-side
MBMS	Model Base Management Software
MINTS	Management Intelligent Systems
MIS	Management Information Systems
MSS	Management Support Systems
NL	Natural Language
NLP	Natural Language Processing
PIM	Personal Information Systems
OAV	Object Attribute Values
OCX	ActiveX control
ODBC	Open Database Connectivity
OIS	Office Information Systems
OLAP	Online analytical Processing
OLE	Object Linking and Embedding
OOSM	Object-Oriented Structured Modelling
OOSML	Object-Oriented Structured Modelling Language
PRL	Production Rule Language
R&D	Research-and-Development
RDBMS	Relational Database Management Systems
SDLC	Systems Development Life Cycle
SDSS	Specific Decision Support System
SGML	Standard Generalised Mark-up Language
SIS	Storage Information Systems
SM	Structured Modelling
SQL	Structured Query Language
TD	Approval of Dean
TDH	Approval of Head of Department
TPS	Transaction Processing Systems
RAD	Rapid Application Development
RBR	Rule-Based Reasoning
RMI	Remote Method Invocation
RWE	Real world Example
UIMS	User Interface Management System
URL	Uniform Resource Link
VRS	Visual Rule Studio
WA	Work Area
Web	World Wide Web
WM	Working memory
XML	Extensible Mark-up Language

Chapter 1 – Introduction

The goal of knowledge acquisition is to construct an accurate and efficient formal representation of an expert's knowledge (Turban 1995). The knowledge engineer usually elicits the knowledge from the expert, conceptualises it and represents it in the knowledge base via a variety of interactive techniques. In general, expressing knowledge and transferring the knowledge to either a machine or a person is difficult. The expert is often unaware of the detailed process or rules he uses to arrive at a conclusion. The indirect transfer of knowledge between the domain expert and the knowledge base via a knowledge engineer may lead to a long and inefficient knowledge acquisition process. Transferring this knowledge to a machine requires knowledge in a detailed and organised manner.

1.1 Background and the state of existing knowledge

The actuality of the Web has turned the focus of modern Decision Support Systems (DSS) to opportunities that the World Wide Web (Web) offers DSS. The Web extends the capabilities of DSS to a large number of geographically dispersed users at a relatively low cost (Shim et al 2002). The Web's impact ensures a more efficient decision making process that is more widely used. The latest advances in DSS include tools such as data warehouses, on-line analytical processing (OLAP), data mining, Web-based DSS, collaborative support systems, virtual teams, knowledge management, optimisation-based DSS and active decision support (Shim et al 2002).

Several expert system development software packages such as EXSYS, Level5 and VP Expert simplified the syntax of production rules to make them easier to create and understand. Ideally, a natural language processor used to represent knowledge in a specific representation structure would support the domain expert best (Turban 1995).

Expert System knowledge can be transferred over the Web not only to human users but also to other computerised systems including DSS, robotics and databases. Expert Systems (ES) can even be constructed using the Internet. Intranet-based GroupWare can facilitate the process of collaborating builders, experts and knowledge engineers. Knowledge maintenance can also facilitate the use of the Web, which is useful to users (Turban et al 2001). The Web can support the spread of multimedia-based ES. These systems, also called *Intelimedia systems,* support the integration of extensive multimedia applications and ES.

According to Shim et al (2002), ES technology is now being replaced by Intelligent Systems (IS) to fulfil two key functions:

- Screening, sifting and filtering of a growing overflow of data, information and knowledge, and
- The support of effective and productive use of Executive Information Systems (EIS)

1.2 Motivation

Because of the ambiguity of natural language, it is difficult to represent knowledge that is free from any structure. Recent research in this area has been towards evaluating text for keywords used in a domain and identifying relations between these keywords to parse a meaningful framework of valid concepts about the system when specifying the system's requirements (Gervasi and Nuseibeh 2002).

Shim et al (2002) recommends that DSS researchers and developers:

- Identify areas where tools are needed to transform uncertain and incomplete data into useful knowledge alongside qualitative insights
- Be more prescriptive about effective decision making by using intelligent systems and methods
- Exploit advancing software tools to improve the productivity of working and decision making time, and
- Assist and guide DSS practitioners in improving their core knowledge of effective decision support

They state that this process will be enhanced by continued development in Web-enabled tools, wireless protocols and group support systems, which will expand the interactivity and pervasiveness of decision support technologies. This study aims to investigate the possibility that a domain expert can maintain a frequently updated knowledge base resulting in a productive and effective support of a web-based intelligent decision support system.

1.3 Problem definition

This study investigates the possibility of a web-based intelligent DSS model where domain experts could maintain a decentralised knowledge base in an easy and domain-expert-friendly way without the assistance of a knowledge engineer.

An intelligent DSS has an expert system component that emulates a human expert's skill in a specific problem domain. The skill of the human expert is codified in the knowledge base component of the ES. When presented with input, the inference engine component of the expert system uses this knowledge base to derive a solution. The decision support system then presents a knowledge worker or decision-maker with all the skills (the ES component with explanation facility) to make a well-informed decision. This study focuses on a set of similar problem domains in which the knowledge contained in knowledge bases is changed periodically. Codifying the knowledge presented in a knowledge base usually involves a complex procedure where the expert and codifier share the responsibility of the codified result.

It would be valuable if the expert himself could define and maintain the knowledge in the frequently updated knowledge base. The definition and maintenance of this knowledge base should be easy and domain-expert-friendly. In Figure 1-1, Turban (1995) presents a conceptual model of DSS. In Figure 1-2 a modified version of his conceptual DSS model is presented to emphasise the focus of this study.

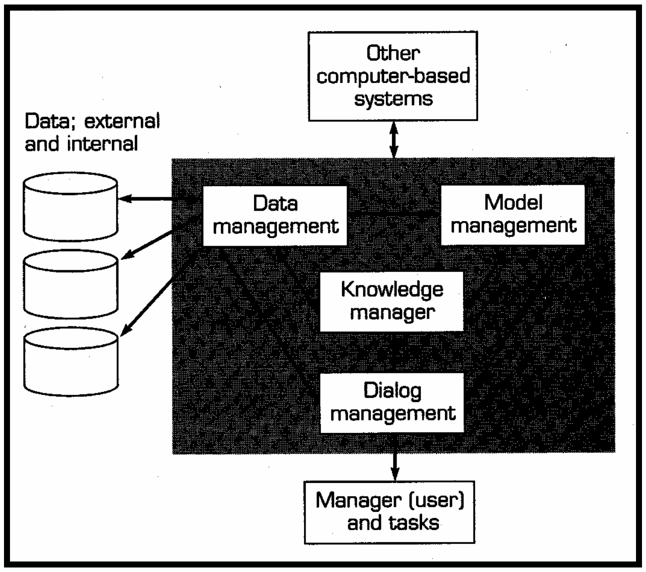


Figure 1-1 A conceptual model of Decision Support Systems (Turban 1995)

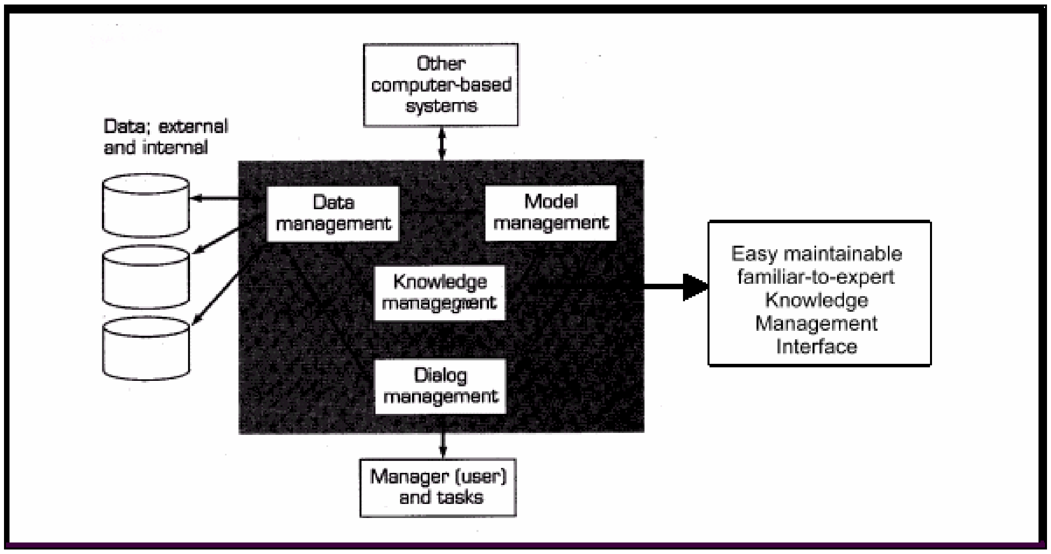

Other computer-based systems
Data; external and internal
Data management
Model management
Knowledge management
Dialog management
Manager (user) and tasks
Easy maintainable familiar-to-expert Knowledge Management Interface

Figure 1-2 Turban's (1995) DSS model modified to include the focus of this study

In order to allow the expert to maintain his knowledge, a special kind of interface allowing the expert to express himself is necessary. The maintained knowledge serves as input to the Intelligent Decision Support System (IDSS). This study aims to investigate the possibility of such a system (See Figure 1-2).

1.4 Research hypothesis and objectives

It is possible to provide an interface to a knowledge base where course-skilled domain experts can develop and maintain their knowledge that serves as an input to the expert or intelligent component of an Intelligent Decision Support System.

Objectives of the research:

1. This study will describe an Intelligent Decision Support System (IDDS) model that includes the following features:
   - A knowledge base interface
     - That allows course-skilled experts to specify and maintain their knowledge
   - A decentralised knowledge base
     - Used by an Intelligent Decision Support Web-application
   - A specific DSS (model)
     - That provides the decision-maker with the knowledge from the knowledge base to reach a well-informed decision
   - A web-based DSS user interface that provides the user with
     - Alternatives to explore, and
     - Explanation of suggestions made to the user or knowledge worker; both contributing to the effectiveness of the decision support system
   - A data component
     - That provides information to the intelligent component and user interface, and

2. Maintenance of the knowledge in the knowledge base should not cause a new release of the IDDS application and thus ensure productivity

1.5 Structure of the thesis

The primary purpose of this thesis is to describe a web-based knowledge based decision support system (KB-DSS) model that involves a knowledge base interface that provides domain experts with a tool to maintain the knowledge used by the IDDS.

Chapter one describes the problem definition and hypothesis of the thesis. Chapter two presents a summarised literature study of Intelligent Decision Support Systems also known as Knowledge-Based Systems. Chapter three describes the construction process of Decision Support Systems and Expert Systems and the synergy they produce as Intelligent Decision Support Systems. Chapter four presents an Intelligent System case study describing and prototyping an easy-to-use decentralised knowledge base interface, as part of a web-based KB-DSS model.

Because of the decision support and expert capabilities of the intelligent system, an in-depth investigation is launched into both Decision Support Systems and Expert Systems in chapters five and six. Chapter seven presents evaluation models for DSS, ES and KB-DSS and describes how the student registration advice system prototyped in chapter four can be evaluated. Chapter eight concludes with final remarks and possible future research suggestions.

A final note: The acronyms used in the thesis are summarised as a List of acronyms on p. ix, and can represent the singular or the plural form.

Chapter 2 - Knowledge-based Decision Support Systems

2.1 Knowledge-based Decision Support Systems, Decision Support Systems and Expert Systems

It is impossible to make good decisions without information. Harbridge House in Boston, MA (Turban et al 2001) conducted a survey to determine the importance of certain management practices and the performance level of management in these practices within a company. They included 6500 managers of more than 100 companies in the survey. "Making clear-cut decisions when needed" was ranked as the most important management practices. Ranked second in managerial importance was "getting to the heart of the problems rather than dealing with less important issues". Most of the remaining eight management practices were related directly or indirectly to decision-making. In the survey, only 10 percent of the managers thought that management performed "very well" at any given practice. The participants accredited this to the difficult decision-making environment and the fact that the trial-and-error method is expensive and ineffective. Using models when making decisions would reduce costs and make the trial-and-error process more effective (Mallach 1994).

Processing information manually while making decisions is becoming increasingly difficult because of the following trends (Turban et al 2001):

- Advances in communication, accessibility to global markets, the use of the Internet and electronic commerce (EC) increase the number of alternatives involved in a decision
- Many decisions need to be made within a time constraint: To process the needed information efficiently and effectively is not possible if done manually
- Information technology (IT) and the sophisticated analysis that it provides becomes a factor in making a good decision, and
- Rapid access to remote information such as consulting experts or providing a group decision meeting is often necessary

On the contrary, conventional computer programs are completely inflexible in responding to unexpected situations and lack the judgement humans possess, to make decisions in situations deficient in information. Modelling is an important feature of Decision Support Systems (DSS) (Turban 1995) (See Figure 2-3, p13). DSS provides the framework that allows decision-makers to view alternatives and make good decisions.

When experiencing a problem in a specific problem domain, a decision-maker normally consults a specific domain expert to assist him in his decision-making process. An expert is a person who has specific knowledge and experience in a problem area, who has acquired his expertise usually over several years. Expertise is the extensive, task-specific knowledge acquired from training, reading about and experience in a specific domain (Turban et al 2001). The less structured the problem domain, the more specialised and expensive the advice of the expert becomes. When solving complex problems, expertise enables experts to make better and faster decisions than non-experts.

Intelligent Systems describe the various applications of Artificial Intelligence (AI) (Turban et al 2001). Applications of Intelligent Systems include Expert Systems (ES), natural language processing (NLP), speech understanding, robotics and sensory systems, fuzzy logic, neural computing, computer vision and scene recognition, and intelligent computer-aided instruction. Expert Systems (ES) is one of the sub-disciplines of AI that is used and applied more than any other AI technology (Turban et al 2001). An intelligent decision support system (IDSS) or knowledge-based decision support system (KB-DSS) includes an Expert System (ES) as one of the main components (Klein & Methlie 1995). This component supplies knowledge of special interest using artificial intelligence (AI) (Turban et al 2001) to the decision support system user. The ES component provides us with:

- A system, which can simulate reasoning, and
- A system that can explain its reasoning and conclusions

ES is therefore ideal to assist a decision-maker in an area where expertise is required (Turban 1995). An ES' knowledge is stored in electronic format and called upon whenever the need for information arises. The basic idea behind ES is the transferring of knowledge from an expert to the computer to the user or knowledge worker or decision-maker. Like a human expert, the ES advises non-experts and explains the logic behinds its conclusion (Turban et al 2001).

2.2 Decision Support Systems (DSS) and Knowledge-based Decision Support Systems (KB-DSS)

Decision support is a context-free expression. It means different things to different people. There is no universally accepted definition of DSS (Turban 1993). Decision Support Systems exist to help people make decisions. DSS do not make decisions by themselves (Mallach 1994) but attempt to automate several tasks of the decision-making process of which modelling is the core (Turban et al 2001). At the heart of DSS lie decisions and decision-making. To comprehend DSS a person needs to understand the process of making decisions. In this paragraph, a summary of the most important DSS and decision concepts is presented. The terms used are explained in Chapter 5 (p74).

2.2.1 Decisions

Simon (1960, 1977) first recognised two polar types of decisions and named them programmed and non-programmed decisions. Finlay (1994), Mallach (1994) and Turban (1993, 1995 and 2001) referred to it by using alternative terms: structured, semi-structured and unstructured (See Paragraph 5.1.2: p74).

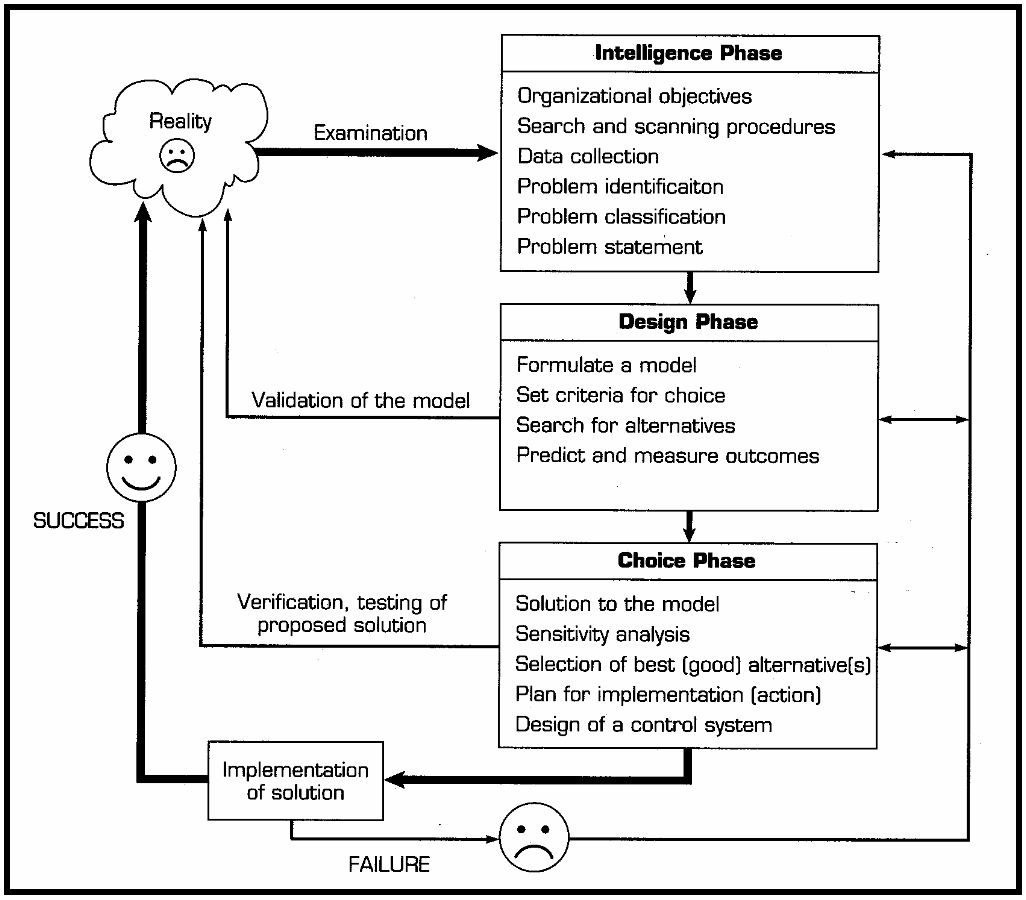


Figure 2-1 The Decision-making/Modelling Process (Turban 1995; Turban et al 2001)

Simon (1960, 1977, and referenced by Turban et al 2001) propose a four-phase model:

1. Intelligence
2. Design
3. Choice, and
4. Implementation

Finlay (1994) refers to the implementation phase as the review phase. Turban (1995) and Finlay (1994) slightly extended and modified the models (See Figure 2-1 and Table 2-1: p9).

In the case study (See Paragraph 4.4: p53), various factors come into play when a student decides his courses for the coming year or semester. These factors include

- Course availability during the semester for registration
- A suitable timetable period match
- Fairly structured prerequisites for the subject
- Other factors such as the student's preferences and talents, and
- Demands of the industry with respect to the knowledge and skills required

Some of these factors are not easily quantifiable and so the decision would be classified as semi-structured.

Table 2-1 Phases and Stages according to Finlay's (1994) model for problem tackling

Phases	Stages
Structuring	Problem detection Problem definition
Understanding	Detailed systems design Exploring courses of action Decision taking
Action	Implementation of change Review

♦ Human decision-making processes

In order to automate assisting humans in decisions, one should keep in mind that people are not entirely rational (Turban 1993). The way that people react to problems, the way they perceive problems, their values and beliefs may cause people to make decisions differently (Turban 1993). Different psychological personality types exist. These personality types play an important role in the decision making process. When presented with all the facts, the ways that people approach decisions are influenced by preference as determined by their personality type. Knowing the personality type of the decision-maker will help design appropriate tools to support that person (Mallach 1994). Huitt (1992), as referenced in Mallach (1994) summarises the preferred decision-making techniques of the eight personality types in Table 5-1 (p80). To be useful, a decision support system should include some of a decision-maker's preferred decision-making techniques.

▪ The Kepner-Tregoe process

When being presented by a multi-attributed problem, it is difficult to choose the best alternative because not all attributes can be optimised simultaneously. Eliminating some of the inferior alternatives by ranking the alternatives in order of importance can simplify the decision-making. Systematic decision-making ensures that all aspects of the decision-making receive consideration. Before using computers, a process aiming to improve the human decision-making was the Kepner-Tregoe process (K-T method). The following stages (Mallach 1994) were proposed as part of the decision-making process:

- State the decision purpose
- Establish objectives
- Classify the objectives by their importance
- Generate alternatives
- Evaluate the alternatives against their objectives
- Make a tentative choice
- Assess its potential adverse consequences, and
- Make a final choice

♦ Decision-making and models

Decision-making often involves the exploration of situations that do not yet exist. Analysing such situations requires a model or abstraction of reality rather than reality itself. A model is a simplified representation or abstraction of reality (Mallach 1994; Turban et al 2001). Models are used to portray the important aspects of reality while eliminating other aspects, which cause difficulties in a particular situation. Models make the structure of the problem explicit. Examining a simple model may show general principles of how the system in question behaves and may lead to a deeper understanding of the problem. This behaviour might be hidden behind the mass of details resulting from a more complex model. A simple model can be included in a more complex model and it may be a good starting point even if the objective is to a build a more complex model of the system. Building a simple model may clarify and illustrate the process of modelling the type of system stated as the objective (Mallach 1994).

Turban et al (2001) and Mallach (1994) provide extensive lists of benefits gained when presenting a problem by using a model. These benefits are listed in Paragraph 5.2.1 (p81). Turban et al (2001) further classifies models as being **iconic**, **analogue**, **mathematical** and **mental.** Mallach (1994) classifies models into **graphical**, **narrative, physical** or **symbolic** models (See Paragraph 5.2.2: p82).

2.2.2 Using information technology (IT) to support decision-making

The alternatives amongst which a decision must be made can range from a few to a few thousand. The decision-maker needs to narrow the possibilities down to a reasonable number. Decision support, such as a selective information retrieval system can help with this task. Computers can evaluate alternatives, especially where the alternatives can be put into numerical terms. Even when this is not the case, the computer can assist the decision-maker in presenting the alternatives in a form that facilitates the decision (Mallach 1994). If decision-making involves a group of people, group DSS ensures that the members of the group are trying to decide the same thing. Decision support tools such as electronic mail can help people from different locations to communicate about a joint decision statement.

Many activities are involved with decision-making. To support these activities, systems must be able to include one or more of the following (Finlay 1994):

- Help to detect existing or incipient problems
- Help to model a problem situation in order to clarify it
- Provide the tools so that options can be considered, and
- Help with implementation of change and its review

An important key to the success of Information Technology (IT) is the ability to provide users with the right information at the right time (Turban et al 2001). Decision Support covers all areas of Information Systems (IS) including programming languages, database, networking and systems analysis. DSS brings all these technologies together for a specific type of application. Every information system has decision support aspects.

Four information technologies have been successfully used to support managers or decision-makers (Turban et al 2001):

- DSS that primarily supports analytical and quantitative types of decisions
- Executive information systems (EIS), which supply information and have lately been expanded to include analysis and communication tools to decision-makers
- Group Decision Support Systems (GDSS) or Group Support Systems (GSS) for group decision support, and
- Intelligent support systems that provides expert knowledge via an expert component to optimise a decision

Collectively the above-mentioned systems are known as Management Support Systems (MSS) and can be used independently or combined, each providing a different capability. Computerised aids that can assist a decision-maker are shown in Figure 2-2 (p12). One example of a set of decision aid tools available for managers to be more organised is the Personal Information Manager (PIM). These tools can play an extremely important role in decision support (Manheim 1989, in Turban et al 2001) by providing facilities through which the user can increase his effectiveness.

2.2.3 The initial Decision Support System concept

Making good informed decisions is one of the tasks managers engage in (Turban et al 2001). Anthony (1965) categorised managerial activities into strategic planning, management control and operational control. This is referred to as the scope of decision-making (Mallach 1994; See Paragraph 5.1.3: p74). The type (structured to less structured) and scope of a decision are related. In general, operational decisions tend to be more structured and strategic decisions less structured.

Earlier definitions of DSS classified DSS as systems that support managerial decision-makers in semi-structured decision situations, extending their capabilities but not replacing them. Judgement was required in the decision-making. Pure algorithms could not replace these decisions. Early definitions implied that the system should be computerised, on-line and have some graphic output capabilities (Turban 1993; Shim et al 2002). Turban (1995) summarises the concepts underlying DSS definitions as given by various authors in Table 2-2 (p12). Gorry and Scott-Morton (1971) integrated Simon's (1960, 1977) decision types with Anthony's (1965) levels of management activities to form the original DSS concept or framework (Shim et al 2002).

The concepts in Table 2-2 (p12) are converse and collectively they ignore the central issue in DSS namely the support and improvement of decision-making. Turban (1995) states it is far more beneficial to rather deal with the characteristics and capabilities of a DSS (See Figure 2-3: p13). He formulates his working definition by defining a range of basic DSS to an ideal DSS as

> "At minimum we can say: A DSS is an interactive, flexible and adaptable CBIS (Computer Based Information System), specially developed for supporting the solution of a particular management problem for improved decision-making. It utilises data, it

provides a user-friendly interface and it allows for the decision-maker's own insights. The most sophisticated DSS definition will add to this: DSS also utilises models (either standard and/or custom-made), it is built by an iterative process (frequently by end-users), it supports all the phases of the decision making, and it includes a knowledge base".

Table 2-2 Concepts underlying DSS definitions (Turban 1995)

Source	DSS defined in Terms of
Gorry and Scott-Morton	Problem type, system function (support)
Little	System function, interface characteristics
Alter	Usage pattern, system objectives
Moore and Chang	Usage pattern, system capabilities
Bonczek et al.	System Components
Keen	Development process

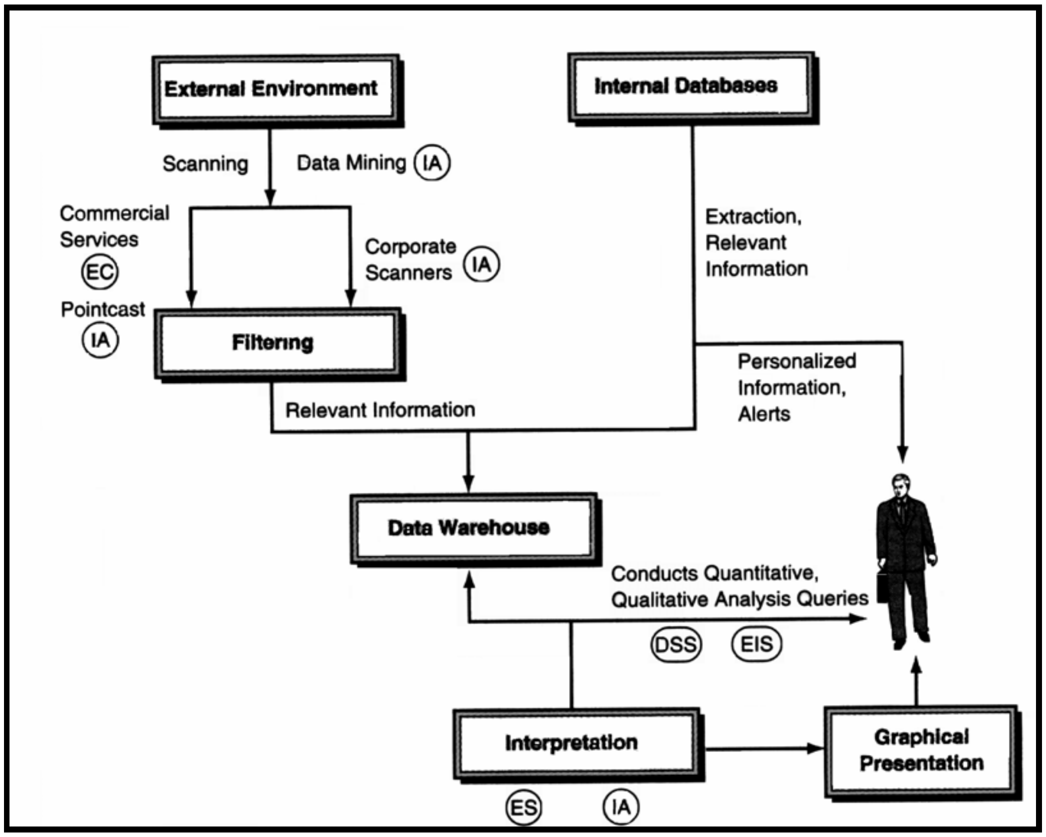


Figure 2-2 Computerised support (MSS) for decision-making. IA - Intelligent Agents, ES= Expert Systems, EC = Electronic Commerce, DSS = Decision Support Systems, EIS = Executive Information Systems (Turban et al 2001)

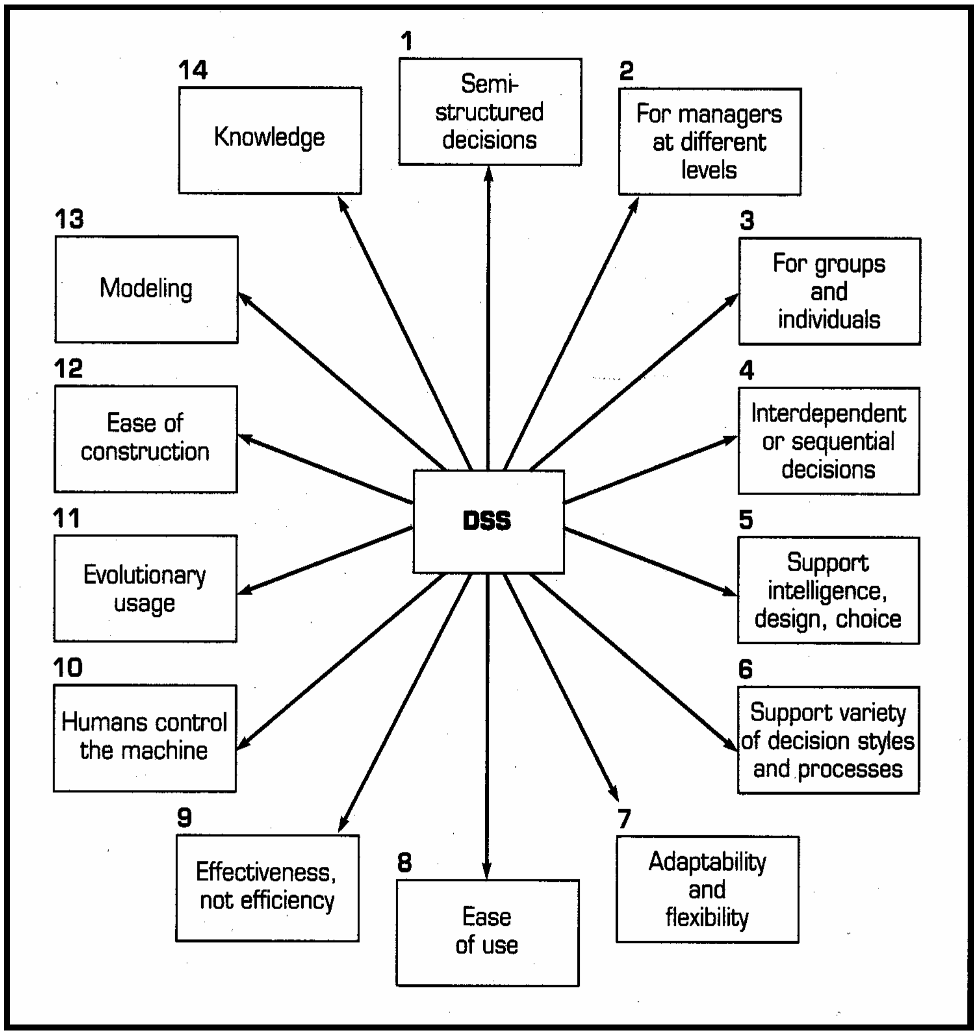
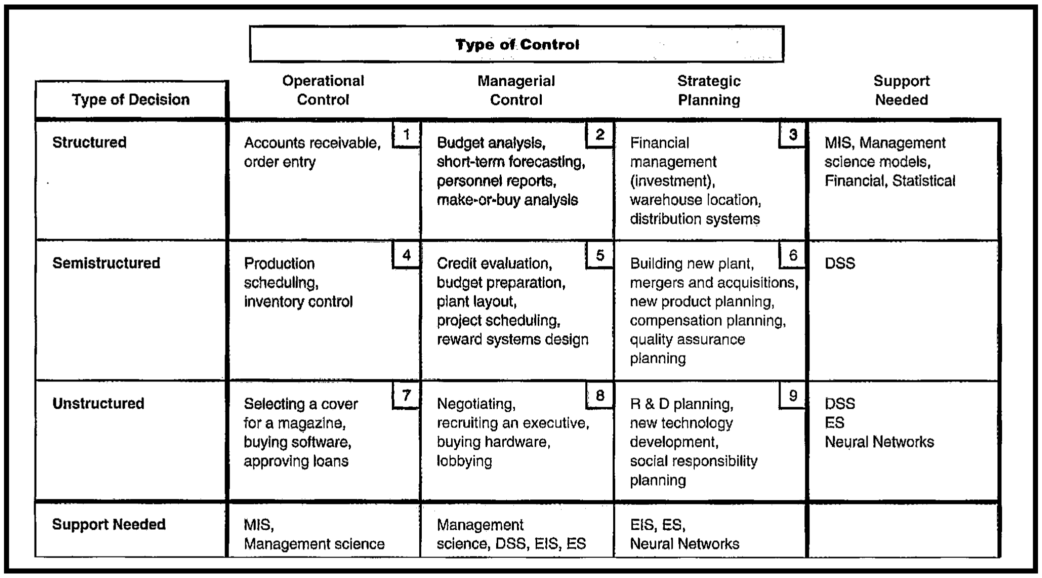

DSS
1
Semi-structured decisions
2
For managers at different levels
3
For groups and individuals
4
Interdependent or sequential decisions
5
Support intelligence, design, choice
6
Support variety of decision styles and processes
7
Adaptability and flexibility
8
Ease of use
9
Effectiveness, not efficiency
10
Humans control the machine
11
Evolutionary usage
12
Ease of construction
13
Modeling
14
Knowledge

Figure 2-3 The characteristics and capabilities of DSS (Turban 1995)

Turban et al (2001) explores a framework (See Figure 2-4: p14) to determine the decision support needed. This framework was first proposed by Gorry and Scott-Morton (1971) and is based on the combined works of Simon (1960; 1977): the type of decision (programmed to non-programmed or well structured to ill structured), and Anthony (1965): the three broad categories of managerial activities (Strategic, management and operational control). The technologies supporting the various decision types and managerial control are indicated.

2.2.4 The customary decision process in a DSS environment

The process of a more customary-used decision-making model as used in a DSS environment is shown in Figure 2-5. The emphasis of this customary model is on the problem analysis and model development. Once the problem is recognised, it is defined in a way compatible with model creation. Models are used to analyse the various alternatives and the choice made and implemented as described by Simon (1960, 1977). These phases overlap and blend together and have frequent iterations to previous phases as the problem is understood and chosen solutions fail (Shim et al 2002).



	Type of Control			
Type of Decision	**Operational Control**	**Managerial Control**	**Strategic Planning**	**Support Needed**
Structured	1 Accounts receivable, order entry	2 Budget analysis, short-term forecasting, personnel reports, make-or-buy analysis	3 Financial management (investment), warehouse location, distribution systems	MIS, Management science models, Financial, Statistical
Semistructured	4 Production scheduling, inventory control	5 Credit evaluation, budget preparation, plant layout, project scheduling, reward systems design	6 Building new plant, mergers and acquisitions, new product planning, compensation planning, quality assurance planning	DSS
Unstructured	7 Selecting a cover for a magazine, buying software, approving loans	8 Negotiating, recruiting an executive, buying hardware, lobbying	9 R & D planning, new technology development, social responsibility planning	DSS ES Neural Networks
Support Needed	MIS, Management science	Management science, DSS, EIS, ES	EIS, ES, Neural Networks	

Figure 2-4 Decision support framework showing the technology to be used (Turban et al 2001)

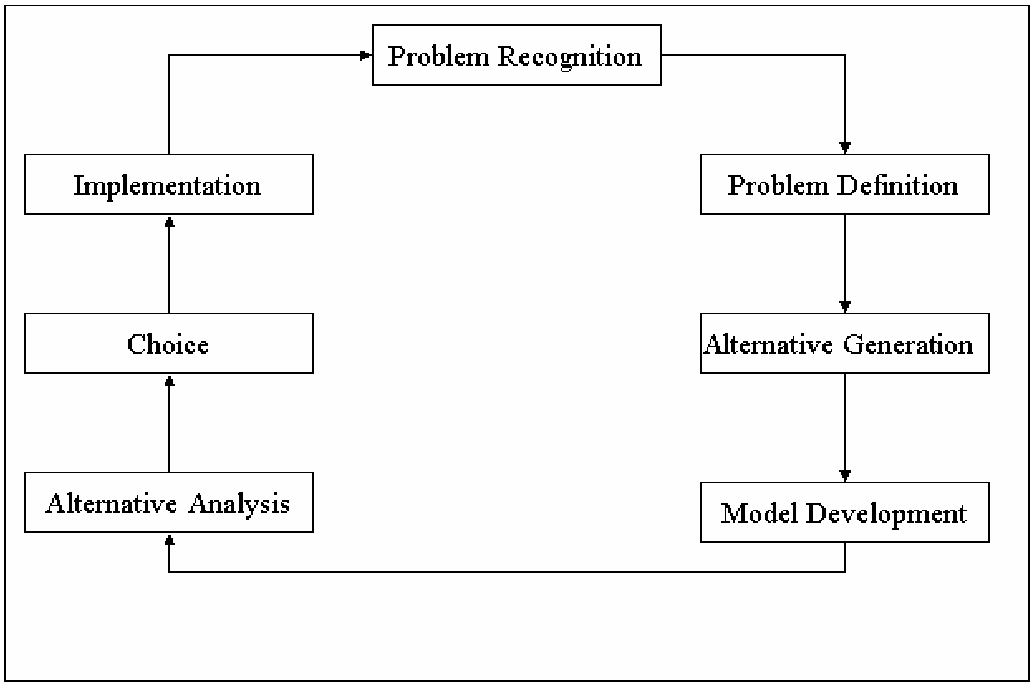


Figure 2-5 The DSS decision-making process (Shim et al 2002)

2.2.5 Classifications of Decision Support Systems

The varieties of DSS are overwhelming. It ranges from a single user spreadsheet to a multi-user database to Internet-based DSS that uses re-usable agents (Bui & Lee 1999). DSS differ in their scope, the decisions they support and the people that use the DSS (Mallach 1994). Finlay (1994) states two important reasons why DSS should be classified into types. The ways in which DSS are developed differ markedly depending on the type of DSS. His second reason states that the classification provides a framework for research. Without this classification or framework, it is impossible to communicate the results and findings about such disparate types effectively and without this communication the understanding of the factors of importance will be inhibited (Mallach 1994). A DSS, which has as major component a logical model, is likely to be developed by people with different skills rather than a DSS where the major component is a data model. The interaction of the user and the developer will be diverse.

Alter (1980) compared different types of DSS and placed all DSS in one of seven categories. He classified the DSS by the type of operation the DSS performs. Mallach (1994) refers to these categories as different levels of DSS. The seven levels of categories (See Paragraph 5.3.3: p84) are:

- File drawer systems
- Data analysis systems
- Analysis information systems
- Accounting models
- Representational models
- Optimisation systems, and
- Suggestion systems

Finlay (1994) classifies these levels into Management Information systems (MIS) or Management Intelligent Systems (MINTS) depending whether the DSS deals with information (MIS) or with intelligence (MINTS). MIS is generally more context independent, compared to the use of MINTS that heavily relies on the context. MIS is concerned with efficiency (doing the thing right) while MINTS is concerned with effectiveness (doing the right thing) (Finlay, 1994; See Paragraph 5.3.3: p84).

2.2.6 Architecture of a DSS

A decision support system is a specific type of information system that consists of many parts (Mallach 1994; Turban 1993; Turban et al 2001):

- **Data management subsystem:** A DSS uses one or more data stores (databases, sets of files and/or data warehouses) to provide relevant information to the Decision Support System. Some of them are maintained by the DSS itself and some are external data sources. Some database primarily used and maintained by another information system with its own database management system (DBMS) and some DSS applications may have no separate DSS database. The data are entered into the DSS as needed.

- **Model management subsystem:** This subsystem is a software package that includes and manages quantitative and qualitative models. Quantitative models provide the system's analytical capabilities.
- **Dialogue subsystem** or **user interface:** The DSS communicates with the decision-maker via this subsystem. The user supplies information to the DSS and commands the DSS using this subsystem. The information supplied determines what data need to be extracted from the data sources. , and
- **Knowledge management subsystem:** This optional subsystem can support any of the other subsystems or act as an independent component. It provides knowledge for the solution of the specific problem.

The above components are considered to constitute the software portion of the DSS, the final part being the decision-maker himself. A conceptual model of DSS as represented by Turban (1995; Turban et al 2001) and is shown in Figure 1-1 (p3). The components are put together by programming them from scratch or by gluing them together from existing components, or by using comprehensive tools called DSS generators (See Paragraph 3.1.3: p32).

♦ Data management subsystem

Data for DSS may be acquired from various **internal**, **external** and **personal** sources (Turban 1995) as well as from **commercial** databases and as **collected raw data** (Turban et al 2001; See Paragraph 5.4.1: p90)

♦ Model management subsystem

A distinguished characteristic of DSS is the inclusion of a modelling capability (Turban 1995). The DSS analysis is executed on a model of reality, rather than reality itself. A model is a simplified representation or abstraction of reality (Turban 1995; Mallach 1994), which has advantages such as lower cost of experimentation, compression of time, manipulation of the model itself, lower cost of error, reinforcement of learning and enhanced training (Turban 1995). To take reality to only mean that which presently exist is limiting. Some DSS tools exist to explore situations that do not yet exist. To include such tools as models, reality should include that which could come about in future (Finlay 1994).

Turban (1995) categorises DSS models further into seven groups (See Paragraph 5.4.2: p91):

- Complete enumeration – few alternatives
- Optimisation via algorithm
- Optimisation via analytical formula
- Simulation
- Heuristics
- Other descriptive models, and
- Prescriptive models

According to Turban (1995), the model management subsystem should include the following elements:

- Model base
- Model base management system
- Modelling language
- Model directory, and
- Model execution, integration and command

The model base contains statistical, financial and other quantitative models that provide analysis capabilities. The ability to invoke, run, change, combine and inspect models is what differentiates a DSS from a computer-based information system (CBIS). Models play a significant role in DSS. Models can be of various types as discussed in Paragraph 5.4.2 (p91). Models need to be managed. Software similar to a DBMS called model base management software manages all the models in DSS. A DSS may include several models as standard or free-standing software interfacing with the DSS. Mathematically based DSS still constitutes the majority of applications.

Modelling in Management Support Systems may involve non-quantitative (qualitative) models. In many cases these models can be presented in terms of rules. Non-quantitative modelling can be done separately or in combination with quantitative modelling. In some cases it is possible to transform, qualitative measures to quantitative ones by assigning values in a certain range for example one to 10 to qualitative values (Turban 1995).

♦ Dialogue subsystem or user interface

This subsystem is the key to successful use of the DSS. Various interface modes exist which determines how information is displayed and used. Graphics are especially important for problem solving, because it helps decision-makers visualise data, relationships and summaries. The User Interface Management System (UIMS) or Dialogue Generation Management Software (DGMS) is a component of the DSS that accommodates the various information representations. It is the DGMS' responsibility to provide and interface between the user and the rest of the DSS.
The important capabilities of a DGMS, as listed by Turban (1993), should include some of the following:

- Interact several different dialogue styles
- Capture, store and analyse dialogue usage (tracking), used to improve the dialogue system
- Accommodate the user with various input devices
- Present data with a variety of formats and output devices
- Give users help capabilities, prompting, diagnostic and suggestion routines, or other flexible support
- Provide user interface with data base and model base
- Create data structures to describe outputs
- Stores input and output data

- Provide colour graphics, three-dimensional graphics and data plotting
- Include different windows to allow multiple functions to be displayed concurrently
- Provide training by examples, and
- Provide flexibility to accommodate different problems and technologies

The main task of the DGMS is to transform the input from the user into languages that can be read by the DBMS and MBMS and to translate output from the DBMS and MBMS and knowledge management subsystems into a form that can be understood by the user (Turban 1993)

♦ Knowledge management

This is an optional subsystem and can support any of the other subsystems or act as an independent component. Complex DSS systems may require an additional component that provides expertise to the decision-maker. Knowledge management software provides a structure to execute and integrate one or more Expert Systems with the operation of DSS components. DSS that includes such a component is called an intelligent DSS, a DSS/ES or a knowledge-based DSS (Turban 1995). It acts as a consultant as it advises and explains to non-experts.

A knowledge subsystem comprising of all the relevant rules governing the possible course combinations of a given degree will be an integral part of the student DSS (See Paragraph 4.4: p53). Holsapple & Joshi (2001) view the knowledge system as well as the problem processing system as key DSS components. The knowledge system holds representations of descriptive, procedural and/ or reasoning knowledge and the problem processing system solves and recognises problems in a decision making process by drawing knowledge from the knowledge representations of the knowledge system. They view decision making as a prime knowledge management application and state that “DSS is a computer-based technology that aims to get the right knowledge in the right form to the right persons at the right time so they can better make decisions and make better decisions”.

2.2.7 The future of DSS

Technological developments continually allow for DSS tools that are more effective: disk storage, interactive operating systems, enabled spreadsheets, databases and flexible modelling tools (Courtney 2001). Networks and telecommunications enabled group support and Executive Information Systems (EIS). Expert Systems theory and technology enabled knowledge-based DSS. The Internet and the Web fostered the development of globally connected organisational decision environments and may in future develop technical, organisational, personal and ethical perspective decision support. The web can be used to broaden organisational decision-making and facilitate communication among a variety of stakeholders (See Paragraph 5.7: p103).

Shim et al (2002) expresses that: “In future, mobile tools, mobile e-services and wireless Internet protocols will mark the next set of remarkable developments in Decision Support Systems (DSS), expanding the accessibility of tools to decision-makers where-ever they might be.” Mobile tools will play a major role in developing e-commerce as customers will access e-services equally through their cellular phones and their personal computers (Earle & Keen 2000).

Shim et al (2002) calls the database capabilities, the modelling functionality and the interface design components of the DSS, the classic tool design components of DSS. He adds tools such as data warehousing, on-line analytical processing, data mining and web-based DSS as important tool developments for future DSS. These tools together with collaborative support systems, virtual teams, knowledge management optimisation-based DSS and active decision support are important topics in the development of the DSS concept for the this millennium.

DSS will exist in future. It will be more accessible to the global decision-maker because of the popularity of the Web. Instead of using a complete DSS application or DSS generator, the user might seek the integration of DSS tools or models that ate part of different DSS using mobile technologies (Shim et al 2002).

2.2.8 Summary

As stated in Paragraph 2.2.1 (p7) the student DSS (See Paragraph 4.4: p53) is semi-structured and will include a knowledge subsystem (See Paragraph 2.2.6: p15) maintainable by course administration experts.

2.3 Expert Systems (ES)

An expert system's major objective is to provide expert advice in specialised situations. An ES application makes inferences and arrives at conclusions (Turban 1995) while a DSS provides an environment to assist a user to reach conclusions (See Paragraph 2.2: p7; Chapter 5: p74). An ES component is ideal to assist a decision-maker in an area where expertise is required (Turban 1995).

2.3.1 Definitions of Expert Systems

According to Olson and Courtney (1992) ES are computer programs within a specific domain, involving a certain amount of AI to emulate human thinking in order to arrive at the same conclusions as a human expert would. Expert Systems can deal with incomplete and uncertain data in reaching conclusions and incorporates an explanation for its reasoning process. Turban et al (2001) define ES as computer advisory programs that attempt to imitate the reasoning processes of experts in solving difficult problems. ES has the ability to perform at the level of an expert, representing domain specific knowledge, in the way an expert thinks.

The value of ES as a sub-discipline of AI can be appreciated more when compared to natural intelligence. Kaplan (1984) as referenced by Turban et al (2001) gives commercial advantages that AI has over natural intelligence (See Paragraph 6.1: p107). Advantages and disadvantages of using ES are given in Paragraph 6.1.2 (p111).

ES is appropriate for a limited number of problem domains (Raggad & Gargano 1999; Mallach 1994; Goodall 1985). ES is appropriate when systems:

- Are concerned with judgements
- Have no attempt of mathematical representation, and/or
- Have no traceable paths from inputs to conclusions

2.3.2 Architecture of an ES

The important components (See Paragraph 6.2.2: p113) of an ES (Goodall 1985; Turban et al 2001) are:

- **A dialogue structure**: Access to the ES is obtained via natural language or using a predefined syntax. The knowledge worker provides some information to the ES and receives the ES' conclusion and explanation.
- **A knowledge base**: Obtaining knowledge from experts can be a cumbersome task. Methods include manual, semi-automatic and automated acquisition. Knowledge representation is important and crucially affects the ease and speed with which the inference engine can use it. Deep and surface knowledge are identified (See Paragraphs 6.2.2: p113 and 6.4.1: p123). Knowledge can be formulated using formal theories or normative models. The forms of knowledge representation often used are:
    - Rules
    - Semantic nets
    - Frames, and
    - Cases
- **An inference engine:** The inference engine is activated when the user initiates a consultation session. Three basic techniques are identified when inferring facts or conclusions from the knowledge base:
    - Forward chaining
    - Backward chaining, and
    - Hybrid using both forward and backward chaining
- **A blackboard**: It involves an architecture that allows the independent knowledge sources to communicate. Turban et al (2001) describes the blackboard as a kind of database that an area of working memory set aside for the description of the current problem, the input and the intermediate results.
- **An explanation sub-system**: This is a by-product of the inference engine that provides a trace of the rules fired, and
- **A knowledge refining system**: Having the expert system evaluate and analyse the reasons for failure or success and learning from it will greatly enhance ES. This functionality is currently being developed in experimental ES.

The process of ES is divided into a development process and a consultation process (See Paragraph 6.2.3: p121) and is presented in Figure 2-6.

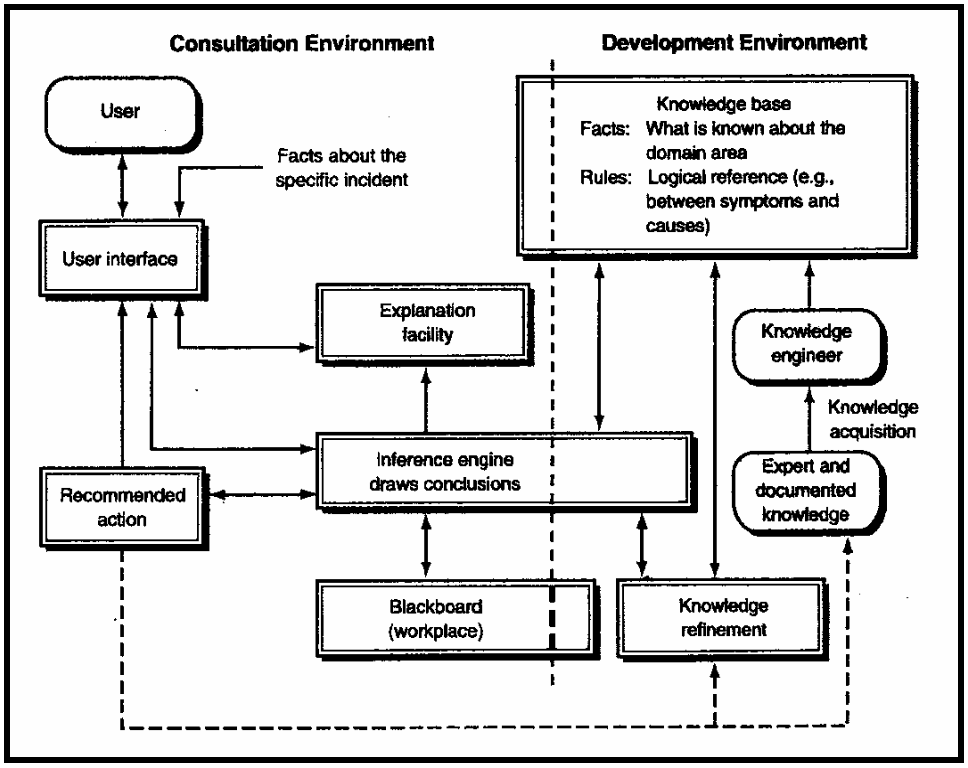

Consultation Environment
Development Environment
User
Facts about the specific incident
User interface
Explanation facility
Inference engine draws conclusions
Recommended action
Blackboard (workplace)
Knowledge refinement
Knowledge base
Facts: What is known about the domain area
Rules: Logical reference (e.g., between symptoms and causes)
Knowledge engineer
Knowledge acquisition
Expert and documented knowledge

Figure 2-6 Structure and process of an expert system (Turban et al 2001)

2.3.3 ES and ES Shells

Construction time of an ES can be reduced by using an ES shell (Beynon-Davies 1991). A shell is the explanation and inferencing mechanisms of the ES only, stripped from its knowledge (Turban 1995).

The architecture of an ES Shell includes (Beynon-Davies 1991):

- A knowledge base that serves as a repository of rules for domain-specific knowledge
- An inference engine that drives the system in making inferences using the knowledge in the knowledge base
- Development tools that assists in building and testing the knowledge base, mainly used by the knowledge engineers
- A working memory, which is the data area where partial results of the problem being solved are stored, and
- A user interface where the user interacts with and executes the ES: One of the most important parts of the user interface of the ES is the explanation facility

All ES Shells include these five components. Some ES includes one other component such as the knowledge acquisition component to impart the expertise of the domain expert to the system directly

(Beynon-Davies 1991). When using an ES shell, the shell design limits the way the resulting ES operates. Often the problem domain is more complex than what the shell can support. A complex problem need to be built from scratch using programming languages such as PROLOG, LISP or JESS. This option is time consuming, because a complete new ES system and structure needs to be designed. Figure 2-7 illustrates the architecture of an ideal expert system shell.

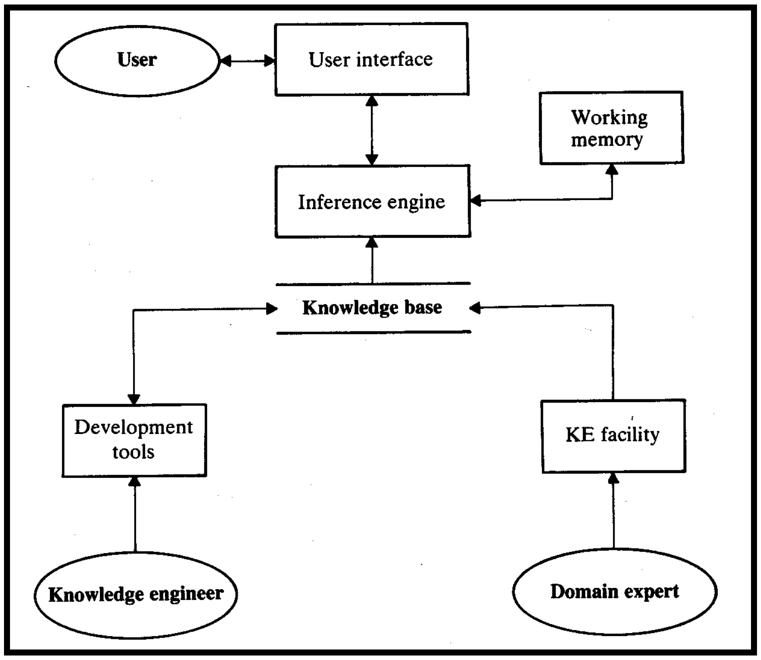


Figure 2-7 Expert shell architecture (Beynon-Davies 1991)

2.3.4 Knowledge acquisition and knowledge base construction

The quality of knowledge contained in the knowledge base determines the effectiveness of the ES. Knowledge engineering is the process of selecting the correct action recommendations and applying the best judgements for a specific situation. High degrees of complexities need to be coped with.

2.3.5 The future of ES

The term Expert Systems has been replaced by Intelligent Systems in publications.

2.3.6 Expert Systems and the Internet/Intranet

One of the justifications of building an ES is to provide expert knowledge to a large number of users. The widespread availability and use of the Internet and Intranets provide the opportunity for ES to be disseminated to mass audiences (Turban et al 2001).

ES can be transferred over the Web not only to human users but also to other computerised systems including DSS, robotics and databases. ES can even be constructed using the Web. Intranet-based GroupWare can facilitate the process of collaboration between builders, experts and knowledge engineers. Knowledge maintenance can also facilitate the use of the Net, which is useful to users

(Turban et al 2001). The Web can support the spread of multimedia-based Expert Systems. These systems also called *Intelimedia systems* support the integration of extensive multimedia applications and ES.

2.4 Knowledge-based Systems or Intelligent Support Systems

The paradigm of DSS is to *support* decision-making as shown in Figure 2-8 (p23). Knowledge-Based DSS (KB-DSS) integrate traditional DSS with the advances of ES. Traditionally DSS constitute data management, modelling, decision methodology and display of numerical data, while the advances of ES embrace symbolic reasoning and explanation capabilities (Klein & Methlie 1995).

The knowledge-based systems development methods in decision support evolved in a new generation of decision support tools known as Intelligent Decision Support Systems (ISS). Decision support systems can assist managers in making strategic decisions by presenting information and interpretations for various alternatives. Three important approaches in the development of current business DSS providing interpretation of knowledge are (Pal & Palmer 2000):

- Rule-based reasoning (RBR)
- Case-based reasoning (CBR), and
- Hybrid (a combination of RBR and CBR)

Each of the above-mentioned approaches focuses on enriching some of the aspects of the traditional knowledge-based business DSS.

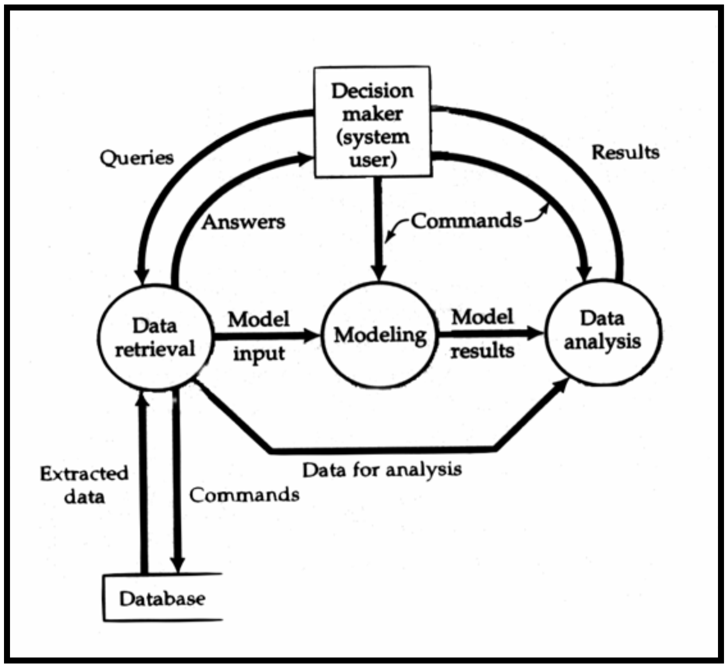


Figure 2-8 Data flow diagram of a generic DSS (Mallach 1994)

Klein & Methlie (1995) describe different directions in extending the DSS framework with ES:

- **Expert advice in a specific problem domain:** When assisting the user to define concepts, compute procedures, run decision models and present reports, a DSS usually does not provide the user with expert advice too. Adding an expert function will provide the DSS user with an expert assistant that could present conclusions or advice that may or may not be followed by the user. This capability requires a knowledge base of domain knowledge and a separate reasoning mechanism that offers expert assistance as part of the DSS.
- **Explanation of the conclusion of the expert:** A good DSS should improve the learning process of the user. The explanation facility of the ES will:
  - Cause the user to have more faith in the result and more confidence in the system.
  - Present the assumptions underlying the system explicitly, and
  - Enable quicker systems development, because the system will be easier to debug
- **Intelligent assistance to support the decision analysis methodology:** Decision analysis is a powerful aid in helping individuals to face difficult decisions. A knowledge base with methodological knowledge can assist the decision-maker to:
  - Define a decision model or an influence diagram
  - Assess a probability distribution, and
  - Assess value functions
- **Explanation of model results and/ or model behaviour:** A complex database may exist for even a simple problem. An expert system can explain the usual logical structure of the models themselves and the causal relationships amongst the variables in the models. Models may be quantitative or qualitative and may perform comparative evaluation of situations. An ES can explain the evaluation algorithm. It can comment on "what-if" scenarios and highlight trends and changes to variables.
- **Assistance when using statistical, optimising or other operations research techniques:** The expert system can:
  - Guide a novice user in using the tools properly
  - Help the user learn good strategies for using the tools, and
  - Discover domain knowledge from large databases
- **Guidance in using the DSS resources: developing intelligent user interfaces**. When a DSS becomes institutionalised in a company or organisation, the number of databases, decision models and reports can increase considerably. When time or money constraints cause difficulties for users to be trained, some kind of expert assistance to select the appropriate resources can be useful. The intelligent user interface can help the user select and use the resources of the system properly.
- **Assistance in formulating certain questions:** An ES can assist a user in formulating his request when the user is confronted with large databases and complex relations, and
- **Intelligence support during the model building process for a specific class of decisions:** An expert system can be used in the process of constructing analytical models. An intelligent component can

- Act as an expert modelling consultant
- Acquire the simulation model specifications
- Automatic document the models in domain terms, and
- Automatically write, compile and solve models

As the decision-making task is performed, expertise in the form of knowledge bases and reasoning and explanation capabilities are applied to specialised problems requiring expertise for their solutions (Klein & Methlie 1995).

2.4.1 The functions of a KB-DSS application

KB-DSS can provide (Klein & Methlie 1995):

- An interface to support man-machine co-operation during the problem-solving task
- Supporting access to relevant information during problem solving
- Support problem recognition
- Support problem structuring
- Support problem formulation and analysis
- An inference engine and knowledge based management systems to provide expert assistance to the user, and
- A reasoning algorithm included in the modelling subsystem

Various knowledge bases can assist the DSS in different domains related to the task to be supported by the application. A functional analysis of KB-DSS is given in Figure 2-9 (p26).
A typical interaction of user and the KB-DSS resources is shown in Figure 2-10 (p26).

2.4.2 KB-DSS architecture

A KB-DSS is a DSS with an ES or intelligent component. When adding this feature to Turban's (1995) DSS architecture (See Figure 1-1: p3), the result would be Figure 1-2 (p3). Figure 2-9 (p26) shows the various components of a KB-DSS. A KB-DSS environment is needed that facilitates communication between the expert module and the traditional modules of the DSS development tool. Full communication and integration is required between the models in the DSS and the knowledge base (KB) (Klein & Methlie 1995). The rules in the database need information from the variable values from the models in order to reach an informed conclusion. The transfer of information between the resources and the application file is given in Figure 2-11 (p27).

Klein & Methlie (1995) claim the KB-DSS environment gave way to new capabilities such as:

- Coupling a causal model's deep knowledge with the expert system's shallow symbolic knowledge, and
- Developing intelligent user interfaces that assist in selecting available resources or interaction with models

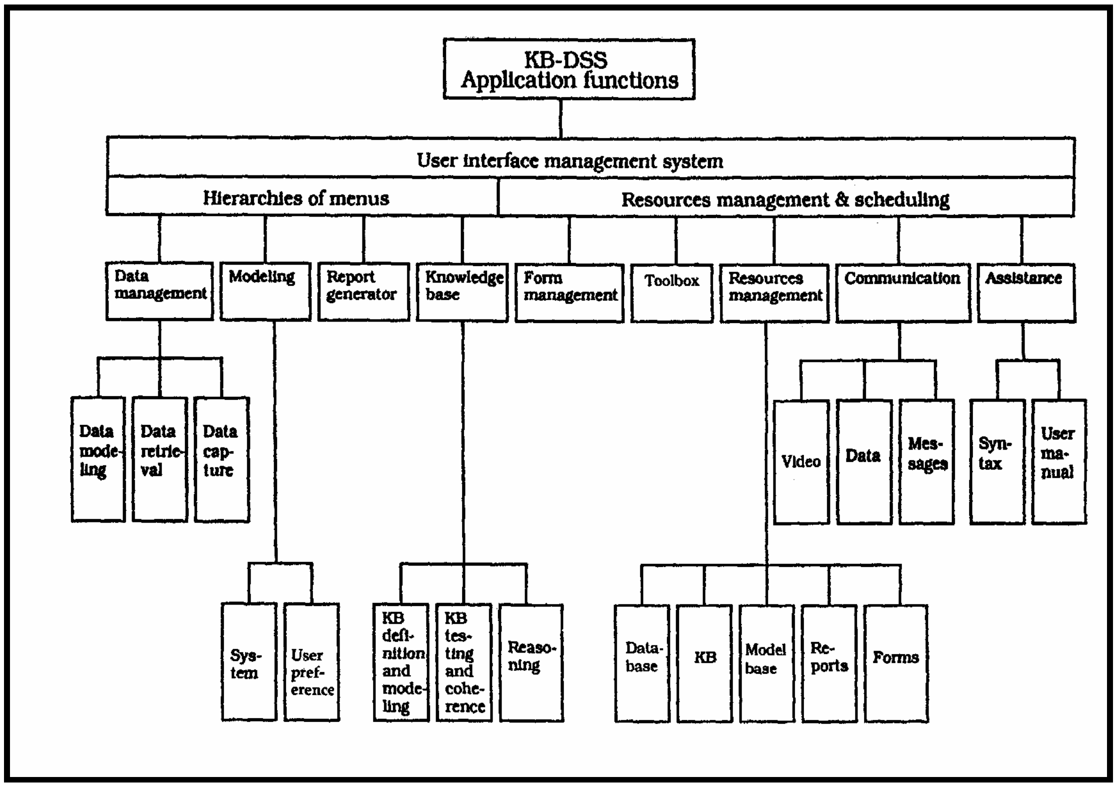


Figure 2-9 Functional analysis of a KB-DSS application (Klein & Methlie 1995)

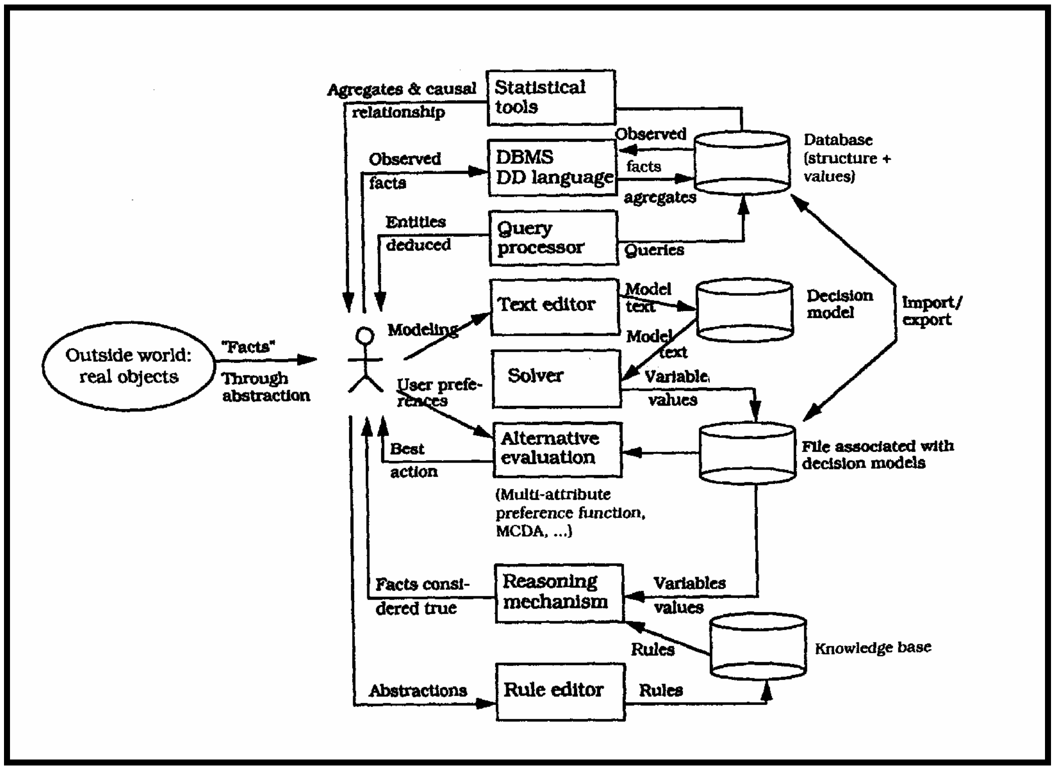


Figure 2-10 Interaction between the user and the KB-DSS resources (Klein & Methlie 1995)

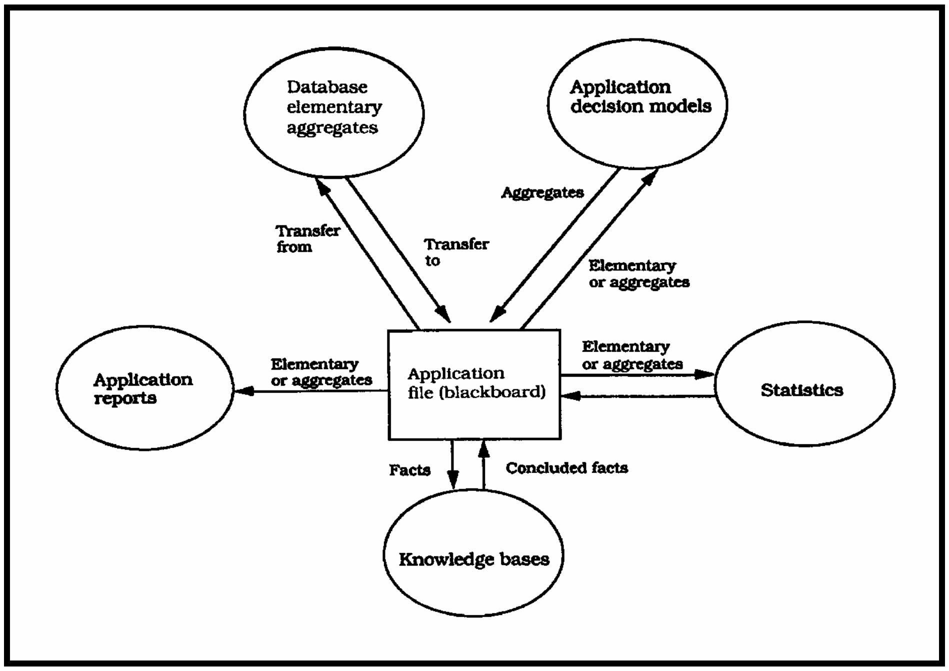


Figure 2-11 Transfer of information between resources and the application file (blackboard) (Klein & Methlie 1995)

2.5 Confirming the Intelligent Systems approach

Bielawski & Lewand (1991) summarise their characteristics of a suitable intelligent system as follows:

- The problem is cognitive (as opposed to computational):
  - It may require reasoning with uncertain or incomplete data, and
  - It does not require common-sense reasoning
- The scope of the problem is manageable:
  - It can be precisely articulated, and
  - Its domain is narrow and specific
- Resources for problem-solving exist:
  - At least one expert in the area exists
  - The expert is willing and able to contribute to the project, and
  - If more than one expert is involved, all must agree on an approach to the problem's solution, and
- The problem is worth solving if it:
  - Results in a saving of time
  - Enhances productivity
  - Reduces physical risk to workers
  - Propagates knowledge
  - Preserves endangered knowledge, and
  - Assures consistence

The advice problem seems to suit a knowledge-based approach, to be modest in scope and has experts willing to co-operate in the process. The completed system will benefit the university by:

- **Increasing productivity**: Departmental personnel will be freed of most of the queries from students regarding their subject choices, leaving them to focus on other aspects of their work
- **Propagating knowledge**: Students and newly appointed staff can obtain the expertise from the system in following certain scenarios
- **Preserving knowledge**: Knowledge attained by experience can be built into the knowledge component, and
- **Assuring consistency**: when advising students the intelligent system has no favouritism of any form or any inconsistencies. The criteria used are included in a knowledge base and the system accesses it to present the most suitable advice.

The advice problem has various components interacting with each other. The necessary information e.g. courses passed by the student may either be entered or obtained from existing information systems. As an integrated system, the advice system aims to provide concrete information and recommendations to the student by supporting him in his decision to choose the correct courses to enrol for in the coming year or semester.

2.5.1 Type specifying the Advice System

Mallach (1994) compares different information systems (IS) in Table 2-3. The characteristics of Transaction Processing Systems (TPS), Information Reporting Systems (IRS) and Decision Support Systems (DSS) are presented.

Table 2-3 Information System Characteristics (Mallach 1994)

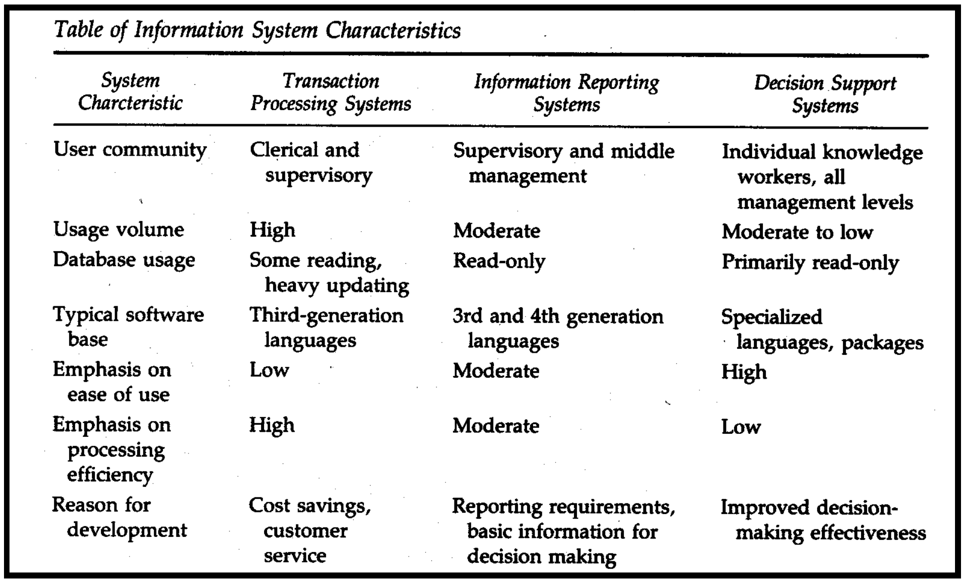


Table of Information System Characteristics

System Charcteristic	*Transaction Processing Systems*	*Information Reporting Systems*	*Decision Support Systems*
User community	Clerical and supervisory	Supervisory and middle management	Individual knowledge workers, all management levels
Usage volume	High	Moderate	Moderate to low
Database usage	Some reading, heavy updating	Read-only	Primarily read-only
Typical software base	Third-generation languages	3rd and 4th generation languages	Specialized languages, packages
Emphasis on ease of use	Low	Moderate	High
Emphasis on processing efficiency	High	Moderate	Low
Reason for development	Cost savings, customer service	Reporting requirements, basic information for decision making	Improved decision-making effectiveness

The different components comprising a context-specific DSS have been discussed in Paragraph 2.2.6 (p15). This information system's characteristics at this early stage seem to include the need for information from the university's database to assist the student in his/her decision-making. The aim of the system would be to improve the student's decision-making effectiveness. Because the knowledge workers are students, the emphasis on ease of use should be high. Since this program will only be used a few times by each student, the processing efficiency could be low. The usage volume would be relatively low and the database usage primarily read only. The typical software base would be specialised, probably specially developed. Students as individual knowledge workers would access this package to aid them in their decision of courses.

In a wider perspective, other types of information systems include (definitions by Mallach 1994):

- **Office Information Systems** (OIS), which improves the efficiency and effectiveness of handling information (words, images, schedules and so forth) in an office
- **Executive information syst**ems (EIS), from which the top management of an organisation can obtain information to guide its decisions (This makes them a type of DSS)
- **Personal information systems** developed and used by one individual (often known as end-user computing) to improve his personal productivity and effectiveness. This effectiveness can involve decision-making. A personal information system used for this purpose is a DSS too.
- **Work group information systems**, used to improve communication and co-ordination within members of a group who collaborate on a set of joint tasks
- **Expert Systems**, which follows rules similar to that of a human used to reach a recommendation or conclusion from available data. If the rules are similar and a human being follows it to reach a decision, an expert system can be a DSS also. , and
- **Strategic Information Systems** (SIS), through which an organisation can obtain a competitive advantage over its rivals, or prevent its rivals from obtaining a competitive advantage over it

Based on the above definitions, the advice system could be a DSS with an expert component. Further investigations will be conducted as to what type on information system would best suite the advice application. Mallach (1994) states: "A decision support system is an information system whose primary purpose is to provide knowledge workers with information on which to base informed decisions." He also distinguishes the following themes that are present in all DSS:

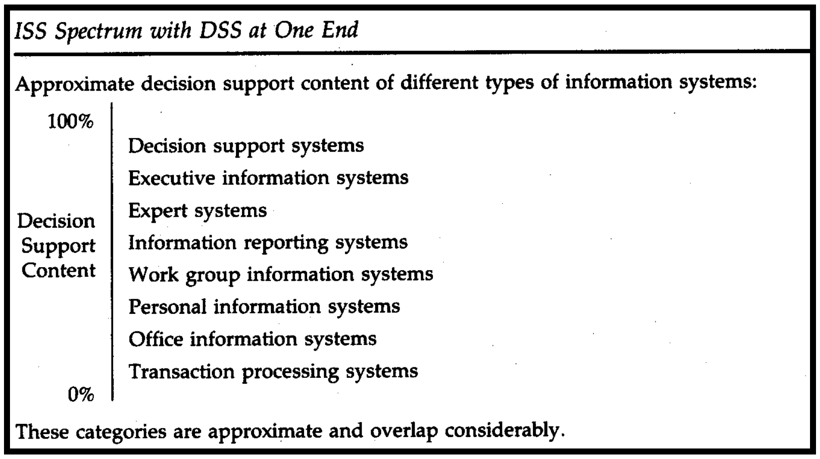
- DSS are information systems (IS). All the basic principles of IS apply to DSS
- Knowledge workers use DSS – most definitions call them managers
- DSS are used in making decisions and therefore impacts organisations using them, and
- DSS support does not replace people. Some human review is necessary. If not, the system will not be classified as a DSS.

Most DSS definitions include the following themes:

- DSS are used in unstructured or semi-structured decisions. The degree of structuredness of a decision was defined in Paragraph 5.1.2 (p74). Most decisions require human judgement.
- DSS incorporate some sort of database (collections of data). Decisions are based on information, often extracted from a database. , and
- DSS incorporate some models. Models represent reality. In DSS, models are used to investigate the impact a decision might have.

DSS may have secondary incidental features or by-products such as reports and other conventional data processing aspects. All types of information systems are related to decision support. A DSS is a system built with decision support as its primary intent. The spectrum of DSS and other information systems are presented in Figure 2-12.

Information systems may be involved in decision support in two ways: as an information system with decision support capabilities or as a decision support system that have decision support as its primary task. The advice system's primary purpose is to provide decision support to students (the knowledge workers).



ISS Spectrum with DSS at One End

Approximate decision support content of different types of information systems:

100%

Decision Support Content

Decision support systems
Executive information systems
Expert systems
Information reporting systems
Work group information systems
Personal information systems
Office information systems
Transaction processing systems

0%

These categories are approximate and overlap considerably.

Figure 2-12 IS decision support spectrum (Mallach 1994)

2.6 The future of DSS, ES and KB-DSS

For management to obtain maximum benefit from its investment in information systems, DSS, EIS, group DSS and ES can be combined into an integrated whole called a management support system (Mallach 1994), using new technologies as they emerge.

Including an intelligent agent (IA) component in a DSS can greatly increase its functionality (Turban et al 2001). An intelligent agent is a new technology with the potential to perform a set of operations on behalf of the user or another program with a degree of independence or autonomy. According to Turban et al (2001) an agent can "advise, alert, broadcast, browse, critique, distribute, enlist, empower, explain, filter, guide, identify, match, monitor, navigate, negotiate, organise, present, query, report, remind, retrieve, schedule, search, secure, solicit, sort, store, suggest, summarise, teach, translate, and watch".

Chapter 3 - Constructing a DSS

3.1 Building the DSS

In building a specific DSS (SDSS), the iterative design process seems to be the most appropriate because of the need for flexibility and the short development cycle needed by decisions and decision-makers (Sprague & Carlson 1982). Flexibility can be viewed as the ability of the SDSS to respond to changes in user decision-making processes as well as the ability to easily develop the specific DSS. An iterative design compresses the traditional levels of the system life cycle to generate repeated versions of the SDSS. It is facilitated in the use of a DSS generator, which reduces development time for the SDSS. The result of applying the iterative design process at all levels is an adaptive DSS capability.

The key aspects of iterative design are (Sprague & Carlson 1982):

- Focus on a sub-problem
- Focus on a small, but usable SDSS
- Plan for refinement/modification cycles, and
- Evaluate constantly

3.1.1 The user

This is the final component of a generic DSS that influences the way a final decision is reached. Differences exist in users' cognitive preferences and abilities and the way they arrive at a decision. The user is also known as the manager or decision-maker. Knowing who will use the DSS is important in the designing of it. Individuals can use a DSS for personal support, or they can individually use a DSS or a portion of a DSS in organisational or group support. A new dimension, namely how a group works together is introduced when decisions need to be made collectively. This complicated type of DSS is called a group DSS (GDSS).

3.1.2 Custom-made versus ready-made DSS

When a problem is non-routine and not structured, the DSS needs to be custom-made for the organisation. When similar functional problems exist in different organisations, a generic DSS can be built with the option of some modifications. Such a DSS is called a ready-made DSS. Most DSS are custom-made though.

3.1.3 DSS technology levels

Sprague & Carlson (1982) identified three levels of DSS technology: specific DSS (SDSS), DSS generators and DSS tools (also referenced by Turban 1995). DSS tools are used to construct DSS generators, which in turn are used to construct SDSS. The DSS tools may also be used to construct tools that are more complicated. Using a DSS generator saves time and money, making the DSS financially feasible.

♦ **Specific DSS**

Systems that actually accomplish the work are called a specific DSS (SDSS). A SDSS involves an application that allows a specific decision-maker or group of them to deal with specific sets of related problems. The case study (See Paragraph 4.4: p53) could be viewed as a SDSS, because it addresses a specific decision problem for a specific group of decision-makers.

♦ **DSS generators**

A generator is an integrated package of software that provides a set of capabilities to build a specific DSS quickly, inexpensively and easily (Turban 1995). A DSS generator is an integrated easy-to-use package with diverse capabilities ranging from modelling, report generation, graphical presentation to performing risk analysis. The ideal DSS generator may be a special-purpose language. The special-purpose language may be used to build a DSS application easily or as an integrated software system constructed around spreadsheet technology.

♦ **DSS tools**

This is lowest level also called the fundamental level of DSS technology and consists of software utilities or tools. These elements facilitate the development of a DSS generator or a SDSS. Examples include graphics, editors, query systems, random number generators and spreadsheets: elements used to build both SDSS and DSS generators. A specific DSS may be built directly from tools. DSS generators promise to create a platform from which SDSS can constantly be developed without much consumption of time and effort (Sprague & Carlson 1982).

3.1.4 Factors to consider when designing a DSS

According to Mallach (1994), one should consider the following before starting to design a DSS:

- One should first determine the purpose of the DSS in terms of the decision being made and the outputs it must supply
- One should determine any external sources that the DSS will communicate with and find any data flows to and from these sources
- Any internal data files needed should be determined. One should determine if the data in these files are obtained from external data sources and if it is, specify the external sources, and
- The major processes in the DSS should be determined. If one can understand all these considerations, you will understand your DSS as a system. One test of this understanding is being able to draw it as a flow diagram. A general schematic description of a DSS is shown as a data flow diagram in Figure 2-8 (p23).

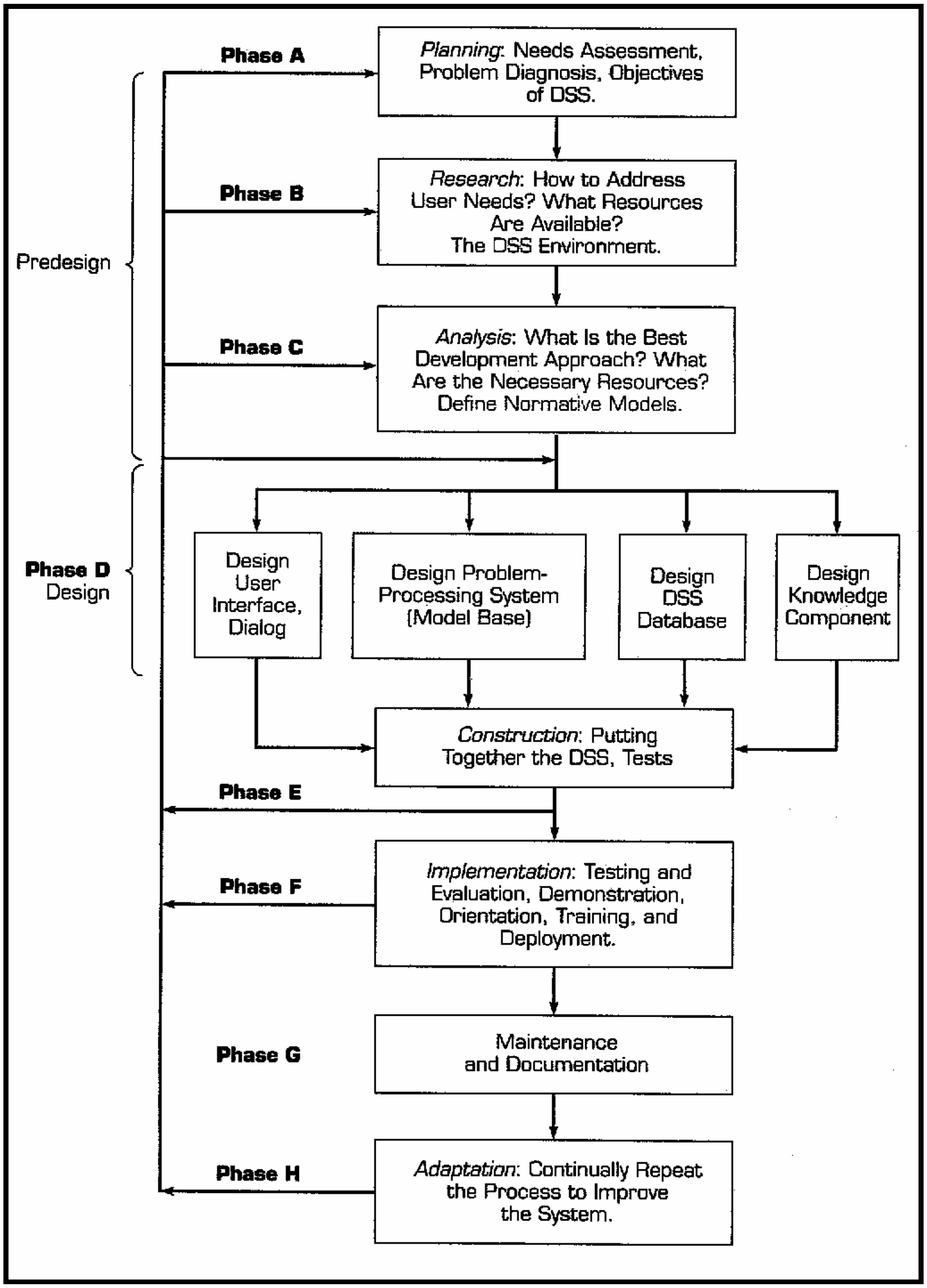

Phase A
Planning: Needs Assessment, Problem Diagnosis, Objectives of DSS.
Phase B
Research: How to Address User Needs? What Resources Are Available? The DSS Environment.
Predesign
Phase C
Analysis: What Is the Best Development Approach? What Are the Necessary Resources? Define Normative Models.
Phase D Design
Design User Interface, Dialog
Design Problem-Processing System (Model Base)
Design DSS Database
Design Knowledge Component
Construction: Putting Together the DSS, Tests
Phase E
Phase F
Implementation: Testing and Evaluation, Demonstration, Orientation, Training, and Deployment.
Phase G
Maintenance and Documentation
Phase H
Adaptation: Continually Repeat the Process to Improve the System.

Figure 3-1 Phases in building a Decision Support System (Turban 1995)

3.1.5 Creating a DSS Generator

The DSS generator should have three general capabilities (Sprague & Carlson 1982):

- It should be easy to use:
    - The generator should create a SDSS that is easy and convenient for non-technical people to use in an active and controlling way, and
    - The generator should be easy and convenient for the builder to use for building and modifying the specific DSS
- The DSS generator should provide access to a wide variety of data sources in a way that supports problem solving and decision-making for a variety of users, problems and contexts, and
- The DSS generator should provide analysis to support problem solving and decision-making for a variety of users, problems and contexts. The generator should provide suggestions on request.

3.1.6 Classical development life-cycle versus prototyping

A classical DSS development process, including all activities necessary for the construction of a complex DSS, is given in Figure 3-1 (p34). As mentioned earlier, the iterative design process seems to be a better alternative because of the flexibility as well as the short development cycle needed by decisions and decision-makers (Sprague & Carlson 1982). A prototyping process often clears misconceptions between builders and end-users. Mutual learning occurs as the DSS develops.

Prototyping provides the following advantages (Turban 1995):

- Short development time
- Short user reaction time (feedback from the user)
- Improved user's understanding of the system, its information needs and its capabilities, and
- Low cost

3.1.7 The development process of a DSS constructed by end users

When end-users are allowed to modify a SDSS to suit their decision needs or the decision needs of the group that they represent, it is better to follow a construction process from the end-users point of view. A suitable construction process developed from an end-users point of view is given by Turban (1995):

- Phase one – *Choosing the project or problem to be solved*: Departments involved are committed to the process of finding a suitable solution
- Phase two – *Selecting software and hardware*: Select suitable DSS software and hardware
- Phase three – *Data acquisition and management*: Acquire and maintain data in the knowledge base
- Phase four – *Model subsystem acquisition and management*: Build the model base: acquire and include relevant models in the model base
- Phase five – *Dialogue subsystem and its management*: Develop the user interface

- Phase six – *Knowledge component*: Perform knowledge engineering
- Phase seven – *Packaging*: The various software components of the DSS will be put together for easy testing and usage
- Phase eight – *Testing, evaluation and improvement*: Test the DSS with sample input and validate it to prove that the DSS is reliable
- Phase nine – *User training*: Train users in using the DSS
- Phase 10 – *Documentation and maintenance*: Produce documentation and maintain the DSS, and
- Phase 11 – *Adaption:* Adapt DSS to suit user needs

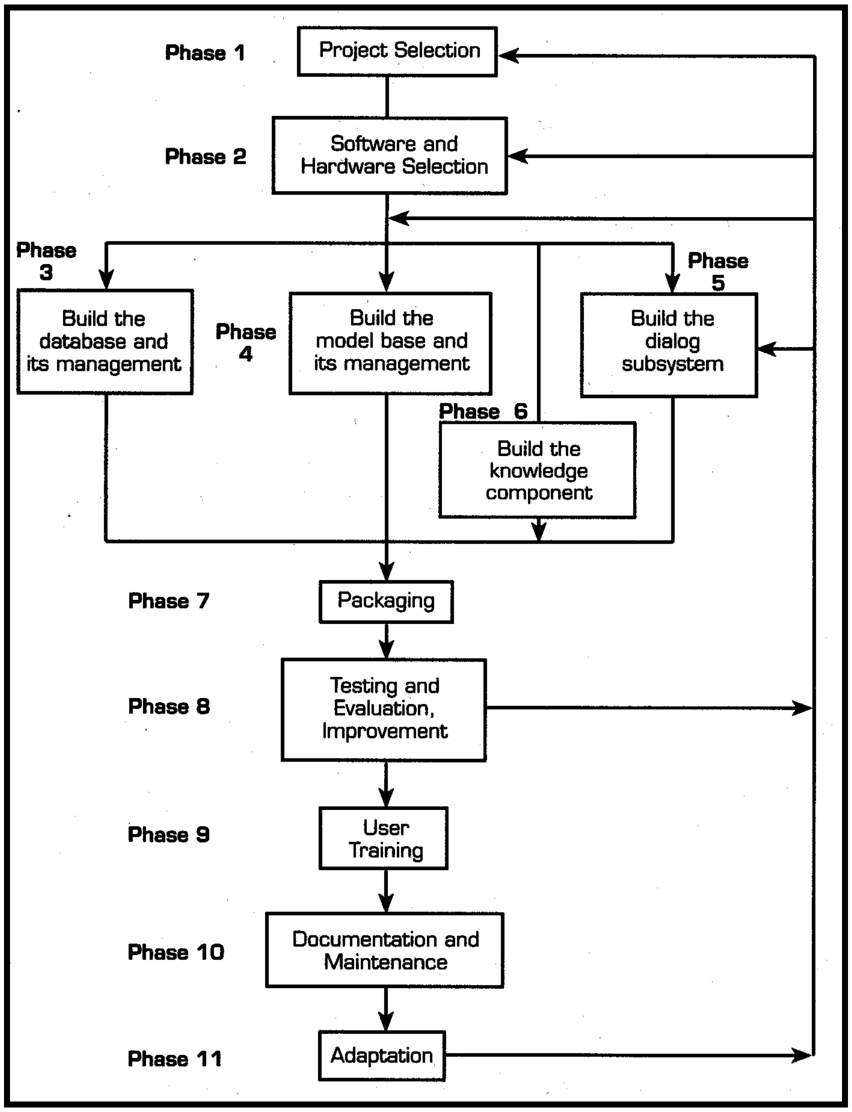

Phase 1
Project Selection
Phase 2
Software and Hardware Selection
Phase 3
Build the database and its management
Phase 4
Build the model base and its management
Phase 5
Build the dialog subsystem
Phase 6
Build the knowledge component
Phase 7
Packaging
Phase 8
Testing and Evaluation, Improvement
Phase 9
User Training
Phase 10
Documentation and Maintenance
Phase 11
Adaptation

Figure 3-2 The development process of a DSS constructed by end-users (Turban 1995)

3.2 Building an ES

Before starting an ES project many factors should be studied such as the main goal of the system; its constraints; its available support facilities; availability of human experts; user-imposed reliability; maintainability; solution needs; and application needs. Updating and maintaining the knowledge base and enhancing the capability of the inference engine are central to a successful ES (Raggad & Gargano 1999). When the ES is used, the effectiveness of the ES should be continually evaluated and monitored to test if the problem domain has not changed. If the problem domain changes, it can invalidate the recommendations given by the ES. This last aspect will be explored in more detail in the section on knowledge validations (See Paragraph 7.2.1: p130).

3.3 Building a KB-DSS

Building a KB-DSS involves the capabilities, functionality and structures of both DSS and ES with the emphasis on the support of DSS. Factors discussed in Paragraphs 3.1 (p32), 5.5 (p101), 6.3 (p121) and 6.4 (p122) are of relevance. Klein & Methlie (1995) propose a methodology for the KB-DSS implementation process in Figure 3-3 (p40). This design methodology is related to the learning process of users and supports flexible design strategies. It presents the essential characteristics and allows the designer to use and combine a variety of design strategies. It is possible for the user to start with the models, the database, the knowledge base, or by defining the application with displays of results. Klein & Methlie (1995) suggest starting with the user interface and the global logic of the application and its associated variables, adding the other sources as the designer wishes as a function of the application. The methodology for Klein & Methlie's (1995) KB-DSS implementation process includes:

♦ **Understanding the user's goals**

The usual user goals are to

- Recognise a problem situation
- Diagnose a problem
- Generate alternatives
- Compute criteria
- Evaluate alternatives, and
- Select one alternative

These goals can more generally be to improve the way the problem is presently solved or facilitate communication between individuals to ease the reaching of a solution.

♦ **Understanding and defining the problem boundaries**

Understanding and defining the problem boundaries will identify:

- The decision-makers
- The relationship of the decision-maker with the decision structure of the organisation
- The fixed and controlled decision boundaries as accepted and challenged by the decision-maker

- The problem that will be solved if an alternative is chosen
- The willingness and ability of other decision-makers and experts that co-operate and provide inputs to the analysis, and
- The dominant culture in the organisation

This step should lead to the creation of alternatives.

♦ **Understanding and defining actual decision processes**

It is very unlikely that a large number of users will use exactly the same decision processes. The sub-steps for identifying the decision process are:

- Describe the general task within which the decision process occurs, and
- Describe the persons involved in the decision process and the sub-tasks they accomplish:
  - Domain knowledge used for studying the problem
  - Problem diagnosis methodology and task knowledge
  - Alternative generation
  - Facts and documents used to obtain criteria
  - Computation method of criteria
  - Presentation of criteria, and
  - Constraints to be taken into account

♦ **Define a normative decision process for the problem**

The task of the designer is to analyse the decision processes, make a diagnosis and define an improved process.

♦ **Defining changes in the decision process**

Many users will use the KB-DSS. The adoption of a DSS is a social process. Therefore, all the users should assimilate the decision process improvement.

♦ **Selecting which part of the decision process to support**

The starting environment of the user should be defined.

♦ **Functional analysis of the KB-DSS**

The purpose of this step is to define the main functions and overall architecture or conceptual model of the KB-DSS. Usually a first list of reports, decision models, data structures, input forms and knowledge bases are needed. A first layout of the application user interface should be defined to demonstrate the resources to be used during the problem solving process.

♦ **Selection of a development environment**

Klein & Methlie (1995) rendered five possible development environments for designing DSS, ES or Knowledge-Based Decision Support Systems (KB-DSS):

- Standard third and fourth generation programming languages such as VB, DELPHI, PASCAL, BASIC, C, Java
- AI languages: such as LISP PROLOG, or Smalltalk
- Expert System shells – no DSS function supplied
- DSS development environments – no expert function needed, and
- KB-DSS development environments – integrating DSS and ES

Choosing the development environment depend on the functionality needed (Klein & Methlie 1995)

- Graphical user interface environment: A graphical object orientated environment allowing the user to use icons, menus and other graphical components. It allows for a global logic interaction between the application user and the application resources.
- Report generator: Allows the user to define reports interactively integrating various objects
- Modelling language: Present a modelling formalism to assist the decision-maker
- Form definition: Input should be entered using a variety of controls and report the explanation provided by the intelligent part of the system
- Database management system: A link to the database when necessary
- Knowledge base Management System: The key issue here is to match the knowledge presentation method and the knowledge to be presented
- Toolbox: Algorithms that are useful for the planned application such as in finance, statistics and forecasting , and
- Communications interface, local area network (LAN) and client/server architecture: Matching the communication needed with the hardware required, and

♦ **Design and implementation of the initial KB-DSS**

The steps proposed by Klein & Methlie (1995) include:

- Data analysis and modelling
- Form definition and input verification
- Decision model design and testing
- Report definition
- Knowledge base modelling and testing, and
- Overall user interface design and global application logic definition

According to Sprague & Carlson (1982), the iterative design process seems to be the most appropriate because of the need for flexibility and the short development cycle needed by decisions and decision-makers. Prototyping the steps would support the iterative design process. It is essential to test the KB-DSS thoroughly. The decision models and the knowledge bases need to be tested on completion with the users that took part in the design. A methodology for verification and validation is presented in

Paragraph 7.3 (p134). It is good practise to write the user manual during the testing and evaluation phase. The education of the users is largely dependent on their implication in the design, their level of expertise and the purpose of the KB-DSS.

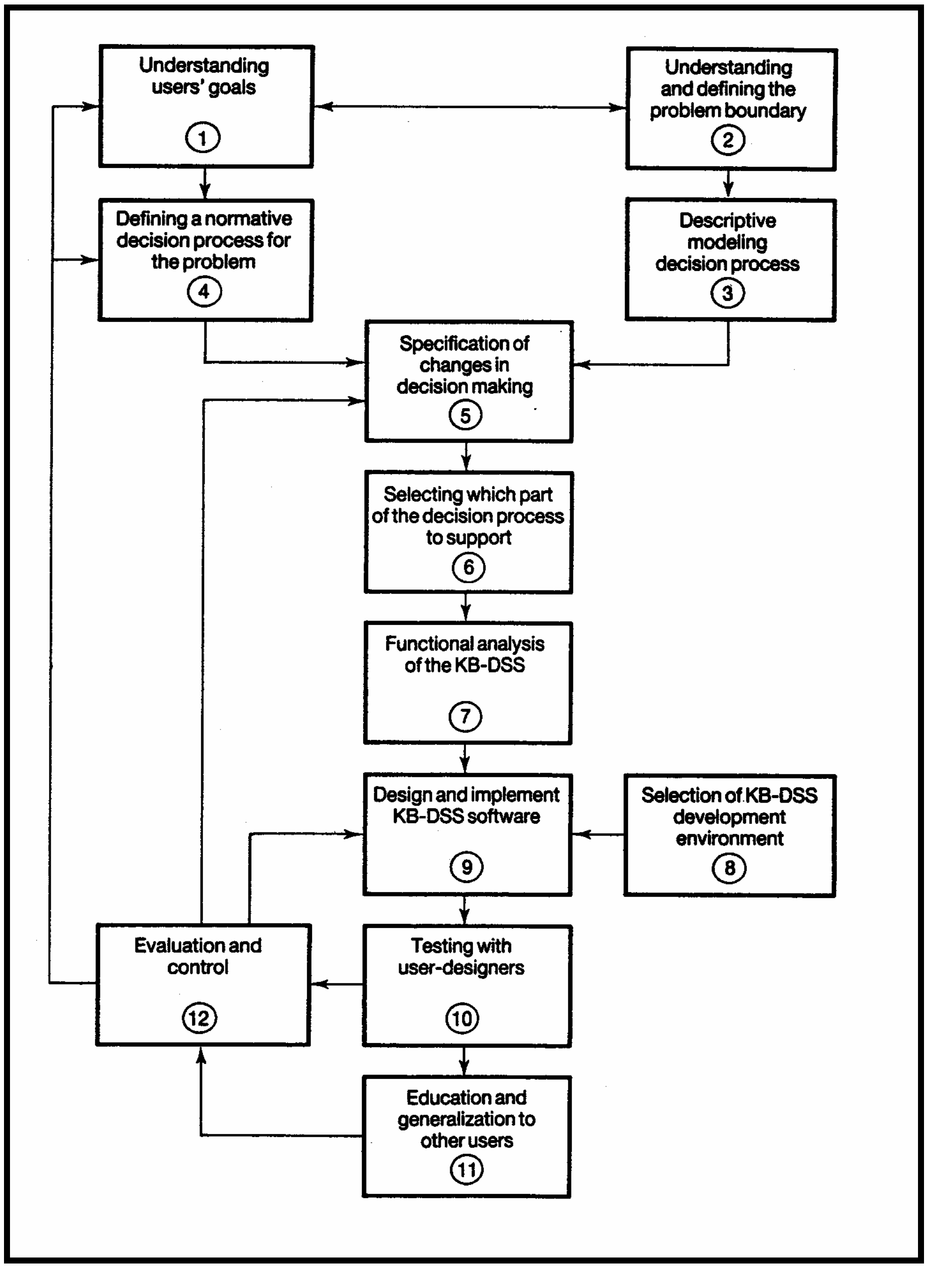

Understanding users' goals
1
Understanding and defining the problem boundary
2
Defining a normative decision process for the problem
4
Descriptive modeling decision process
3
Specification of changes in decision making
5
Selecting which part of the decision process to support
6
Functional analysis of the KB-DSS
7
Design and implement KB-DSS software
9
Selection of KB-DSS development environment
8
Evaluation and control
12
Testing with user-designers
10
Education and generalization to other users
11

Figure 3-3 Methodology for the KB-DSS implementation process (Klein & Methlie 1995)

Chapter 4 - Constructing the advice KB-DSS

The aim of this study is to provide a KB-DSS to students to select realistic relevant courses to support the completion of their current qualification in the most realistic time possible. The advice-giving knowledge component needs to be maintained by personnel that are course experts. A course expert is an expert on the relationships and rules that exist between the university's departments' study programs and courses. The first endeavour of this study will be to find a commercial software package or packages that will provide the desired functionality.

4.1 Selecting a suitable KB-DSS generator

A number of DSS and KB-DSS software packages were assessed to select an appropriate software package that allows domain experts to maintain a knowledge base of their knowledge. In the selection process, the focus was placed on the maintenance and creation of the rules as part of the knowledge component.

4.2 Software packages evaluated

The choice of as KB-DSS generator depends on the availability of a suitable knowledge component. The search for a suitable software package was not exhaustive. The student registration advice system was prototyped and presented as a solution after a suitable KB-DSS was discovered. No formal training was received on any of the software packages. The DSS software packages assessed are shown in Table 1. All software packages evaluated were DSS, KB-DSS or DSS generators. Not all of them contained an expert subsystem in their knowledge component. The software packages are listed in the order of experimentation (Table 4-1 (p42)). A short assessment of each of the above software packages is given below.

4.2.1 Criterium Decision Plus

This product assists decision-making in the form of setting goals and sub-goals, but does not contain an inference engine or expert subsystem. This option was not explored further.

4.2.2 ERGO

ERGO provided a DSS software sample (executable) choosing a colour printer. The tool to build an own application was not included in the demo package. This option was not explored further.

Table 4-1 Software Vendors and Decision Support packages

Product	Company	Web site:
Criterium DecisionPlus v 2.0	Infoharvest Inc.	http://www.infoharvest.com/
ERGO	Arlington	http://www.arlingsoft.com/
Expert Choice for Win	Expert Choice Inc.	http://www.expertchoice.com/
Statistica	Stasoft	http://www.stasoft.com/
Descision Pro	Vanguard	http://www.vanguardsw.com/
KnowMan Designer SR3	Intellix	http://www.intellix.com/
Business Rule Studio	Rule Machines Corp.	http://www.RuleMachines.com/
Visual Rule Studio	Rule Machines Corp.	http://www.RuleMachines.com/
AgentOCX	The Haley Enterprise	http://www.Haley.com/
Jess	Distributed Computing Sys	http://herzberg.ca.sandia.gov/jess
ILOG JRules 3.0	ILOG	http://www.ilog.com

4.2.3 Expert Choice for Windows

Expert Choice is based on the Analytic Hierarchy Process (AHP), a methodology for decision-making. It provides users with the tools to construct decision frameworks from both routine and non-routine problems and ways to include value judgements in these decision frameworks. This framework is a hierarchy, used to organise all the relevant factors to solve a problem in a logical and systematic way, from the goal to the criteria to the sub-criteria and down to the alternatives of a decision. The user must define the problem and enter all the relevant issues into the hierarchy. Expert Choice does not include an intelligent or expert component. This option was not explored further.

4.2.4 Statistica

Statistica includes an extensive set of mathematical models to assist a manager in the statistical assessment of a company's data during the decision-making process. The software analyses problems of a more computational nature. The advice problem has a cognitive nature and needs software with a strong intelligent or expert component. Statistica was not explored further.

4.2.5 Decision Pro

Decision Pro performs a wide range of tasks required in business decision analysis. It provides expert system development to automate routine decisions. The models can execute in a standard Web browser. Decision Pro allows one to build models that include logical rules as well as mathematical formulae. The rule-based models use a set of Boolean rules rather than numeric equations to calculate results. It either returns a True or False result (Binary Models) or classifies persons, places or things (Classification Models).

Decision Pro builds a logic tree of rules to be applied and this determines how the user is prompted for information. Decision Pro decides when specific questions should be asked by using the rules defined. It applies logical look-ahead to ensure that only pertinent questions are presented to the user. When finished, it returns a True or False value indicating for example whether the candidate qualifies for the loan or not. The user can easily see how the final decision was reached by following logical values down the branches. Decision Pro asks as few questions as possible and ensures that all questions are pertinently based on answers to previous questions.

When the rules change, adjustments must be made to the decision tree structure. The structure of how the various nodes are linked may also change. Data can be exchanged with other applications by reading and writing a common data file. One can only read and write ASCII text files. There are no facilities for processing binary files. As a decision tree must be updated, this option was not further explored.

4.2.6 KnowMan Designer SR3

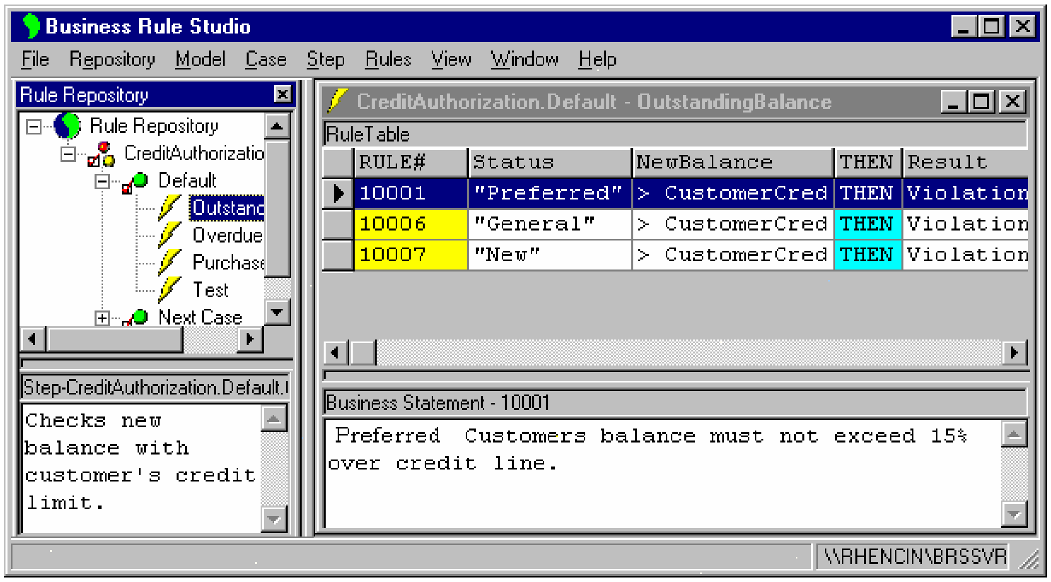
KnowMan Designer helps the user to collect, store and distribute knowledge. This knowledge can be collected from any expert and is stored in files called knowledge domains. Knowledge obtained from an expert can be formalised into a working knowledge domain and made accessible to others. The end users gain access to the knowledge domain by means of one of Intellix' viewer products.

When using the viewer products, a single result is derived from the input data provided. No reasons for the result provided are given. When additional knowledge needs to be added, it is recommended that a new knowledge domain be built. As the anticipated prototype will contain knowledge that needs to be altered and updated regularly, this option was not explored further.

4.2.7 Business Rule Studio (BRS)

BRS is a software tool used by corporate managers, enterprise experts and software developers who wish to define and manage business rules as a separate unit owned by management and corporate experts and deployed as part of the application software in co-operation with the software developers (Business Rule Studio 1998). These rules are stated using standard representations familiar to management such as a spreadsheet-like decision table. The business objects such as customer, order and product can tap into intelligence such as a credit limit specification according to the account history, time of the year, product demand and receivables supplied as rules. These rules make business more adaptable to changes and more responsive to customers. The rules and objects are mutually dependent on each other. The rule repository provides clear and concise business rule authoring, documentation and management without any of the unnecessary complexities of programmer syntax, compilation, installation and distribution.

BRS is not an expert system. It comprises of a Rule Author for capturing business policies in the form of objects and rules, a rule engine for executing these policies, a rule repository to manage and store the objects and rules, and a repository administrator for managing developer privileges and application deployment (Business Rule Studio 1998). A business rule is the combination of a set of constraints, the resultant actions and a business statement. The business statement is the message displayed to the client application user when that rule's constraints have been met. The tutorial provided links to a Visual Basic client application.


Business Rule Studio
File Repository Model Case Step Rules View Window Help
Rule Repository
Rule Repository
CreditAuthorizatio
Default
Outstand
Overdue
Purchase
Test
Next Case
CreditAuthorization.Default - OutstandingBalance
RuleTable
Step-CreditAuthorization.Default.
Business Statement - 10001
\\RHENCIN\BRSSVR

RULE#	Status	NewBalance	THEN	Result
10001	"Preferred"	> CustomerCred	THEN	Violation
10006	"General"	> CustomerCred	THEN	Violation
10007	"New"	> CustomerCred	THEN	Violation

Checks new balance with customer's credit limit.

Preferred Customers balance must not exceed 15% over credit line.

Figure 4-1 Developing the business rules (Business Rule Studio1998)

Management and enterprise experts can use the rule author component to create and maintain the rules within the designed core model structure established to interface with production software applications. The rule's interface in the form of a spreadsheet (See Figure 4-1) compares certain columns to specific values to determine cases when actions should be taken. The objective will be to set up a rule set with properties with relations between values of the same column to suite our case study. This seems cumbersome and may be due to the fact that BRS is not to an expert system. Thus, this software product seemed not to be a solution to the advice system.

4.2.8 Visual Rule Studio (VRS)

Another product from Rule Machines Corp., called Visual Rule Studio (VRS), may be a potential generator to develop expert systems, business rules and knowledge management applications. VRS might have a possible solution to our problem. The documentation states that VRS re-use the rules and knowledge in any client applications that support COM (Component Object Model) or OLE (Object Linking and Embedding) such as C, C++, Java, PowerBuilder, Delphi and many more.

The different sets of rules can be grouped together in a rule set. The rules are coded using VRS' Production Rule Language (PRL). It has an "if—then—" structure. The rule set is then compiled by VRS into a preferred type of executable: a standard exe a server DLL, an ActiveX control (OCX) or an ActiveX document. This compiled rule set is then referenced from within a client application such as

Visual Basic. The tutorial that accompanied the product demonstrates the functionality via a Visual Basic client application. According to the documentation, this can just as easily be done via the products as mentioned above as well as Internet or Intranet applications.

VRS uses backward chaining, forward chaining and hybrid or mixed chaining to solve complex business issues. The rule set can become part of the distributed exe application, or be distributed by a host, executing on a client or be centralised, running on a single host server supporting multiple client applications. VRS is a rule development environment for Windows NT, Windows 98 and 95 that may be deployed in an Internet/Intranet environment. VRS seems to be a viable option if the operating system used for the Advice Systems is one of the above-mentioned operating systems.

4.2.9 AgentOCX

♦ The Rete Algorithm

An inference engine that determines all applicable rules before applying them would intuitively perform linearly with the number of rules. The more rules that are added, the slower the inference engine would perform. The Rete algorithm was developed to provide a production system whose performance varied less than linearly. Given a significant number of rules, the Rete Algorithm performance is unaffected when adding more rules. There is no other algorithm, published or promised, that offers such performance (Agent OCX 1996).

♦ Functionality of AgentOCX

AgentOCX is an OLE control for use with Rapid Application Development (RAD) tools as well as conventional programming languages including C++. AgentOCX is a COM (Component Object Model) component that encapsulates the Eclipse Toolkit for Microsoft Windows 95 and NT within an OLE interface to embed Artificial Intelligence (AI) for business applications. AgentOCX provides rule-based programming not obtainable from other procedural or object-oriented programming tools. The Eclipse inference engine uses the Rete Algorithm that provides the only viable architecture for rule-based programming. Eclipse is among, the world's fastest inference engines and supports forward and backward chaining as well as other functionality, such as truth maintenance and case-based reasoning (CBR). Eclipse is available as a set of Dynamic Link Libraries (DLLs) for Windows 95 and NT. These DLLs export an Application Programming Interface (API) that allows an application to embed knowledge-based or expert systems implemented using Eclipse.

♦ Defining the rules using Eclipse

Unfortunately, the syntax specification of the rules in Eclipse is of such a nature that it will be difficult to be maintained by course experts. Haley Enterprise offers a rule editor called Authorete, but it is not available for evaluation purposes. AgentOCX together with Authorete seems to be worth exploring, but because the product must be purchased to investigate, this option was not explored further.

4.2.10 Jess

The Java Expert System Shell and scripting language (JESS) is compatible with all versions of Java starting with Java 1.1. Jess is an API library and is itself written in Java. Jess supports the development of rule-based Expert Systems, which can be tightly coupled to code written in the powerful, portable Java language (Friedman-Hill 1997). The Jess language is very similar to the language defined by the CLIPS expert system shell, which in turn is a highly specialised form of LISP. Many simple CLIPS programs will also run unchanged in Jess.

Jess can be used as a rule engine, which is a program that very efficiently applies a set of if-then statements (*rules*) to a set of data (the *knowledge base*) (Friedman-Hill 1997). Jess rules have a similar syntax to CLIPS for example:

```
(defrule DegreeComputerScienceDetermines
    (currDegree (degreeCode “02130002”) (studyProgram” 02133221”)
=>
    (assert (CourseDetails          (courseCredits 225)
                                    (max100LevelCredits 70)
                                    (min300_400LevelCredits 56)
                                    (maxCreditsOtherDept 22)
                                    (maxCreditsPerYear 56))
```

The documentation states that Jess’ syntax is identical to the syntax used by CLIPS. The above rule may be translated into pseudo-English as follows:

```
DegreeComputerScienceDetermines rule:
    if
        the student’s current degree is and his study program is
    then
        in order to obtain the qualification the student must have at least 225 credits
        of which 70 on  level 100, at least 56 on level 300 and 400, 22 credits from
        other departments, maximum 56 credits per year.
```

Jess maintains a collection of facts called a knowledge base. Similar to a relational database the facts must have a specific structure. In Jess, there are three kinds of facts: ordered facts, unordered facts, and “definstance” facts (Friedman-Hill 1997). Ordered facts are simply lists, where the first field (the head of the list) acts as a sort of category for the fact for example: (shopping-list eggs milk bread) and (person "Bob Smith" Male 35). A course example could be: (currDegree “02130002 “ “02133221”) where currDegree is the head of the list, 02130002 the degree code and 02133221 the code used for the study program e.g. for Computer Science. Ordered facts can be added to the knowledge base using the “assert” function. The “facts” command shows all the facts in the knowledge base and the “clear” command clears the knowledge base. The "retract" function removes a single specific fact from the knowledge base. A course example would be:

```
Jess> (reset)
  TRUE
Jess> (assert (currDegree "02130002" "02133221"))
<Fact-1>
 Jess> (facts)
 f-0   (initial-fact)
 f-1   (currDegree "02130002" "02133221")
 For a total of 2 facts.
Jess> (retract 1)
  TRUE
Jess> (facts)
 f-0   (initial-fact)
 For a total of 1 facts.
```

Unordered facts offer functionality similar to Java or JavaBeans objects that encapsulates fields. The fields are referred as slots (Friedman-Hill 1997). A course example could be: (currDegree (degreeCode "02130002") (studyProgram "02133221"). The "deftemplate" construct defines the slots. This functionality will be useful in the creation of the different types of rules and data involved in the advice process. A course "deftemplate" example:

```
(deftemplate currDegree "The student's current degree"
  (slot degreeCode)
  (slot studyProgram)
  (slot fme (type BOOLEAN))
  (slot studyYear (type INTEGER)),
```

would allow one to define fact like:

```
(assert     (currDegree (degreeCode "02130002")
             (studyProgram "02133221"))
```

When the knowledge base is reset, all the facts defined by the "deffacts" construct are asserted into the knowledge base, shortening a tedious process to assert all known facts.

The third type of fact, definstance, adds a specific Java or JavaBeans object to the knowledge base. The "defrule" construct supplied by Jess contains all the rules to apply to the knowledge base. A Jess rule is in the form of a 'if... then" statement as in similar rule-based expert systems. If the Jess rule engine is active, a Jess rule is executed whenever its if-part (the left-hand-side or LHS) is satisfied (Friedman-Hill 1997).

Jess uses the Rete Algorithm and both forward and backward chaining. The Rete Algorithm is an efficient method to reduce the number of iterations in a rule loop when performing pattern matching. Past test results across iterations of the rule loop are remembered and only new facts are tested against any rule LHSs. As a result, the computational complexity per iteration drops to something more linear in the size of the fact or knowledge base. Jess tries to determine whether a fact in the knowledge base, matching the pattern of the particular rule's LHS, exists. If it is found, the rule is placed in the agenda to be fired or executed. As the rule executes some facts in the knowledge base are altered in such a way that a other rules evaluate to being true or matched and are placed in the agenda to be fired. The contrary is also true, when the rule that is fired causes some rules on the agenda to evaluate to no longer pattern matched, the rule is automatically retracted. The above describes the process of forward chaining of inference.

In a backward chaining system, rules are still "if... then" statements, but the engine seeks steps to activate rules whose preconditions are not met. This behaviour is often called "goal seeking". Jess also supports backward chaining. To use backward chaining in Jess, you must first declare that certain fact templates will be backward chaining reactive using the "do-backward-chaining" function. When a rule matches a rule as backward chaining, the rule's LHS is rewritten as being in need to be matched.

Rules are uniquely identified by their names. Rules may contain wildcards, functions and variables. The variables need to be tested. Each variable may be submitted to any number of tests within the rule to be pattern bound with the knowledge base. Several of the most commonly used Jess functions are wrappers for methods in the jess.Rete class. Examples are run(), run(int), reset(), clear(), assert(Fact), retract(Fact), retract(int) and halt(). You can call these functions from Java when running an application or applet (Friedman-Hill 1997). Each individual jess.Rete object is an independent reasoning engine. A single program may include several independent engines (Friedman-Hill 1997). Jess can be used in a multi-threaded environment. The jess.Rete class internally synchronises itself using several synchronisation locks. The most important lock is a global lock on all rule left hand sides': only one assert or retract may be processing in a given jess.Rete object at a time. This restriction is likely to be relaxed in the future. The rules used by Java embedding Jess are saved in a CLIPS file. The syntax of the rules is of such specialised format that an editor-compiler should guard the correctness of the rules.

4.2.11 JRules

ILOG JRules is a general-purpose expert-system generator that combines rule-based techniques and object oriented programming to help one to add rule-based modules to one's business applications (ILOG 2000 (a)). The JRules source code is implemented in Java and works in a completely object-oriented manner. ILOG JRules directly infers from the Java objects of the application without any duplication. The design of the application and the Java classes are independent of whether you use ILOG JRules or not. JRules uses the Rete Algorithm (See Paragraph 4.2.9: p45) to perform pattern matching. The JRules engine can be used in all types of Java applications including applets running inside a browser (ILOG 2000 (a)).

Rules can be added to or be removed from the inference engine whenever needed. The rules can be added from numerous sources such as text files, strings or a stream from a URL (Uniform Resource Link). The rules are Java objects themselves that can be embedded in an application and can be manipulated. Java objects can integrate with ILOG JRules using the class ILRContext (ILOG 2000 (a)).

ILOG JRules has the following elements:

- A language
- An API (application programming interface – similar to Java)
- An inference engine
- A rules editor called the ILOG JRules builder with debugger and tracer, and
- Additional tools

ILOG JRules is written for Java developers and assumes that the application developers are familiar with the environment in which the rules are used. The rules themselves may however, be created and maintained by course experts using the rules editor or builder.

♦ ILOG JRules language, API and inference engine

The rules are written using a language of specific syntax and structure (ILOG 2000 (a)). The rules can be created and debugged using the rules editor or builder. The rule base is implemented in Java code (ILOG 2000 (a)) and is in the form of API's. The API's include the ILOG JRules libraries and inference engine used by the application and the rules in the rule base. The inference engine manages the interaction between the application and the rule base (20: p xiii) and uses the Rete Algorithm (ILOG 2000 (a)) mentioned in Paragraph 4.2.9 (p45).

♦ The ILOG JRules builder, additional tools and documentation

This graphical interface provides debugging and tracing capabilities. It consists of a GUI interface that enables the user to input rules in a simple and direct manner (See Figure 4-3: p51). When debugging the state of the inference engine, the Work Area (WA) or Memory (WM) and the contents of the agenda (rules enabled to fire) can be viewed (See Figure 4-2: p51).

The following additional tools exist:

- A faster application may be obtained by compiling the ASCII rules file into Java source code. Benchmark tests done on sample cases accompany ILOG JRules.
- A documentation generator similar to Javadoc facilities are provided

- Rules may access any JDBC compliant database using the relational database connectivity API, and
- A syntax checker may be invoked to find errors before attempting to execute the rules in an application. This part is particular useful for the case study (See Paragraph 4.4: p53) where the end users are not ILOG JRules programmers. Invoking the syntax editor will assist course experts in detecting errors when entering and maintaining rules.

A user's manual (ILOG 2000 (a)) is supplied with ILOG JRules. In addition the following documentation is provided:

- A reference manual of the ILOG JRules API's in Javadoc format
- A language reference manual, documenting the ILOG JRules language: Programmers familiar with the Java environment may create an ASCII rules file direct without using the ILOG JRules builder using this manual
- A builder's user guide, describing the builder GUI (Graphical User Interface) environment and syntactic editor for course expert rule builders
- A business rule language support, describing how to develop your own business language to builders to develop ILOG JRules rules to assist course expert rule builders/ administrators, and
- A business action language guide, which presents a sample language and describe how to extend and customise it

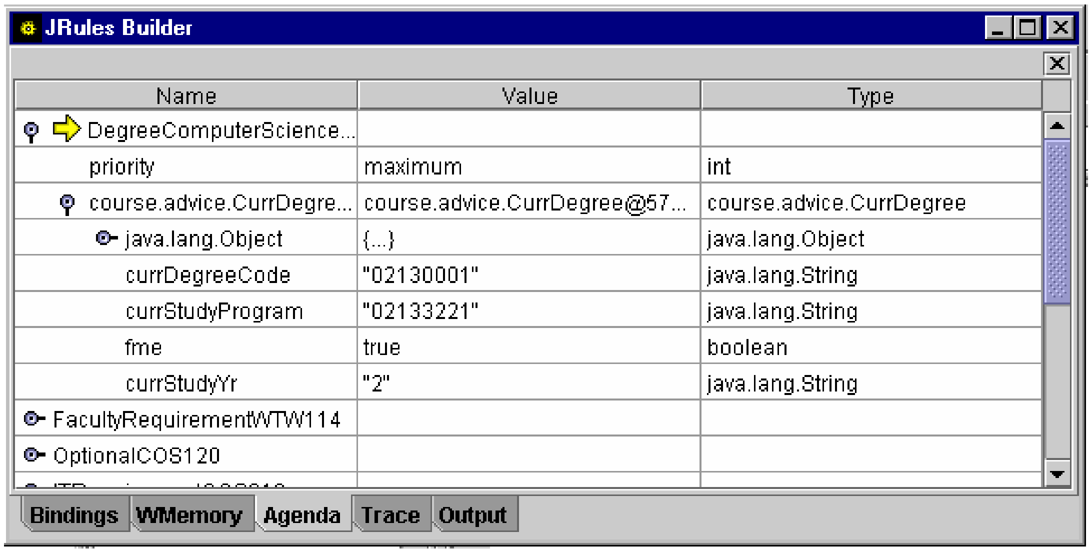
JRules seems to be adequate to specify a working set of rules. The builders provide functionality such as rule traces and the compiling of rules to assist the rule administrator or builder in maintaining rules in a simple and direct manner.

4.3 Comparison

In order to select an appropriate product (KB-DSS environment or generator), the following were set as criteria (See Figure 1-2: p3, for focus emphasis):

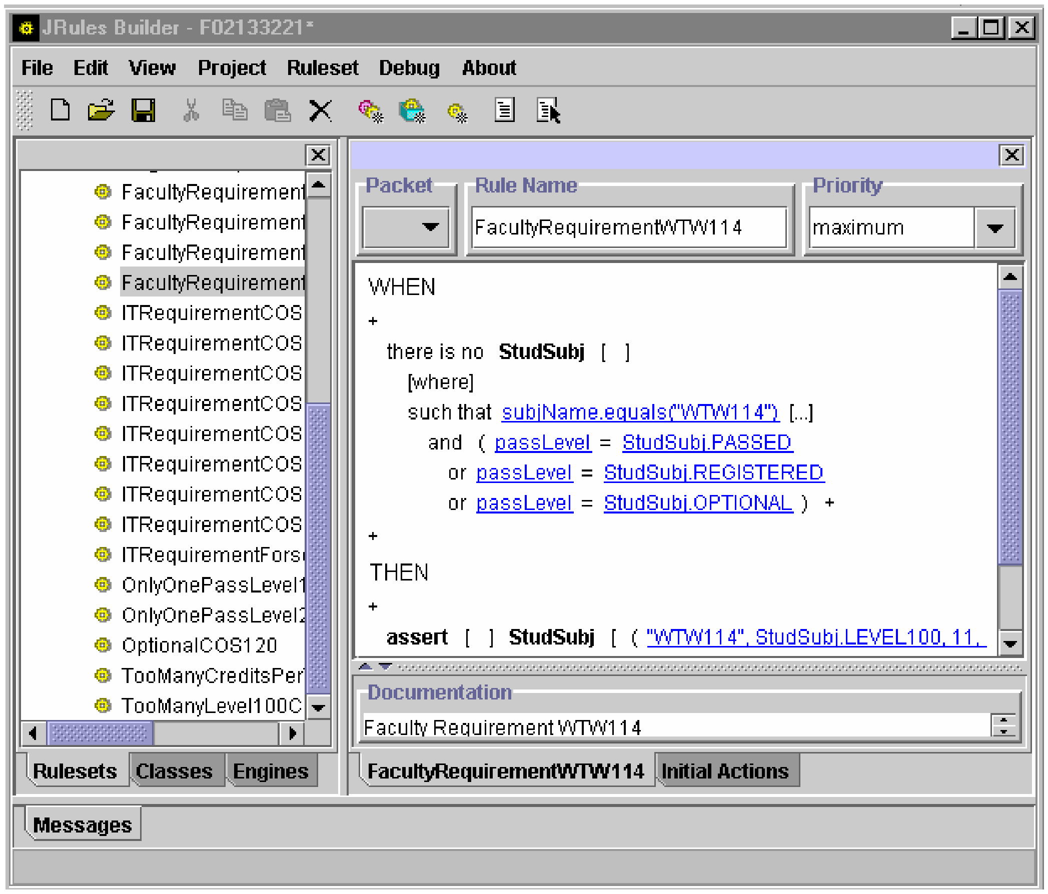
- The DSS or KB-DSS generator must include a knowledge component
- The knowledge component must include all relevant knowledge in a manner that course experts can maintain it
- Changes to the knowledge component should preferably not result in a new version of the KB-DSS/SDSS software application
- Possibility of deploying the software application on the Internet/Intranet will be an advantage, and
- The KB-DSS must be able to provide a reason for actions taken

All software packages evaluated were DSS, KB-DSS or DSS generators. Not all of them contained an expert subsystem in their knowledge component.



Name	Value	Type
DegreeComputerScience...		
priority	maximum	int
course.advice.CurrDegre...	course.advice.CurrDegree@57...	course.advice.CurrDegree
java.lang.Object	{...}	java.lang.Object
currDegreeCode	"02130001"	java.lang.String
currStudyProgram	"02133221"	java.lang.String
fme	true	boolean
currStudyYr	"2"	java.lang.String
FacultyRequirementWTW114		
OptionalCOS120		

Figure 4-2 ILOG Builder's Agenda



Rule Name: FacultyRequirementWTW114
Priority: maximum

```
WHEN
+
  there is no  StudSubj  [  ]
    [where]
    such that  subjName.equals("WTW114")  [...]
      and  ( passLevel  =  StudSubj.PASSED
        or  passLevel  =  StudSubj.REGISTERED
        or  passLevel  =  StudSubj.OPTIONAL  )  +
+
THEN
+
  assert  [  ]  StudSubj  [  ( "WTW114", StudSubj.LEVEL100, 11,
```

Documentation: Faculty Requirement WTW114

Figure 4-3 The ILOG Builder

Table 4-2 Summary of software selection criteria

Criteria / Software Package	Expert system, DSS or KB-DSS	Examples and suitable documentation provided	Demonstration version allows building of a course prototype	Reason for actions	Rules or knowledge maintainable by course experts	Release of knowledge changes	Web enable the application	Accept(A) or Reject (R) the software package
Criterium Decision Plus	DSS	Documentation, tutorial, examples	no knowledge base, setting goals and sub-goals in tree structure	tree, diagram	N/A	Rebuild goals and sub goals	N/A	R
ERGO	DSS	Example: choice of a printer, presentation	No	diagram	N/A	Recode	N/A	R
Expert choice for Windows	DSS	Documentation, tutorial, examples	goals and sub goals, tree structure	tree, diagram	N/A	Rebuild goals and sub-goals	N/A	R
Statistica	DSS	Documentation	Quantitative models, not assessed	N/A	N/A	N/A	N/A	R
Decision Pro	ES	Documentation, examples	Tree structure	tree	no	Tree structure change	yes	R
Knowman Designer SR3	ES	Documentation, examples	Knowledge domains	no	N/A	Knowledge domain change	no	R
Business Rule Studio (BRS)	DSS	Documentation, tutorials	No, spreadsheet with values. Not an ES.	N/A	Rules verification in the rules authoring tool	Business rule changes within the designed core model	yes	R
Visual Rule Studio (VRS)	KB-DSS	Documentation, tutorials for VB	Rules as ActiveX components used by a 4GL	yes, has a reason attribute	if..then structure in 4GL environment (No)	re-compilation necessary, re-deployment to application server: DLL creation	yes	R
Agent OCX	KB-DSS	Documentation, examples, tutorial	Embedded ECLIPSE from a 4GL	yes	Authorete™ *	Incremental compilation, use CLIPS files	yes	R
Jess	KB-DSS	Documentation, many examples provided	Yes, open source	yes	no specialised editor to check syntax or provide rule traces	Immediately: Replace CLIPS text file at URL	yes	R
ILOG JRules	KB-DSS	Documentation, extensive examples, tutorials provided	yes	yes	Various rule builders available: Platform specific	Immediately: Replace .ilr text file at URL	yes	A Rule builders specify rule builder platform

* Authorete™ supplied by Haley enterprises claims to include an easy maintainable rules component. Authorete was not available to be assessed.

The availability of a demonstration version, the ease of use of the package and the examples supplied with the demonstration version are included as criteria for a suitable DSS/DSS generator in Table 4-2. The search for a suitable product was not exhaustive. The primary reason a software package was rejected is presented in red. ILOG JRules was found to have the most potential according to the criteria set. A student registration KB-DSS was prototyped using ILOG JRules 3.0.

4.4 An advice system for student registration

At the University of Pretoria under-graduate students have a choice between various courses in the process of obtaining a specific qualification. Courses have unit counts, levels and prerequisites attributed to them. Qualification and course rules guard the course choices students have to make. These rules differ from department to department and change from time to time. Students find it difficult to choose the best course alternatives for the current semester. The department of Biological and Agricultural Sciences find themselves mostly occupied assisting students in their course selection. These enquiries are time consuming, leaving the experts in course content and prerequisites very little time for their other duties.

Giving students access to some form of automated advice system involving computer software would alleviate the university's departmental course experts from numerous identical queries to answer only those queries that need their expertise.

4.4.1 The need for a Decision Support System

Recall that a DSS in short is a computer-based system that aids the process of decision-making (Finlay 1994). In this sense, a Decision Support System could be a viable option for the student advice system. Although various educational institutions may experience this same functional problem, their departmental and course structures might differ completely from that of the University of Pretoria. The different departments at the university share a common database and therefore most of the structures used in the departments would correspond. The aim is to develop a custom-made DSS (See Paragraph 3.1.2: p32 and 3.1.3: p32) that would be shared among the different departments at the university. It would be of a great advantage if this system could provide an explanation for the advice it renders.

Not only would it enforce the system's credibility and trustworthiness with the users, but it would also train the users (students) so that they can assist one another or encourage the use of this system among their fellow students. If this DSS is available on the Intranet of the university, it will be available to all students without delay. Students would have the opportunity to do "what-if" analysis and so determine the optimum period of study. The system never tires and never becomes irritable and is never in a hurry. If necessary the same query can be submitted until full understanding is obtained.

4.4.2 An intelligent system's approach

The SDSS needs a component that presents the student (decision-maker) with all the possible options and a recommendation of the best courses to enrol for. This recommendation calls for an intelligent or expert component as part of the DSS. Such a system is called a KB-DSS or ISS (See Paragraph 2.4: p23).

The different departments or fields of studies at the university have different sets of rules governing the process of course selection. These rules change often, causing the maintenance thereof using an automated conventional program almost incomprehensible, especially integrating the rules from all the different departments in one system. A solution is sought whereby departmental course experts could maintain their own rules separately (decentralised) from other departments as often as needed, suppressing the need to re-code and release a whole software application every time the rules change. Maintaining the rules should require little or no programming skills. A decentralised rule base would be a rule base stored on computers at multiple locations; however, a network does not interconnect the computers, so that domain experts at the various sites do not share rule bases (Adapted from decentralised database: Hoffer et al 2002).

An intelligent system reasons rather than computes a solution to a problem (See Chapter 6: p107 and Paragraph 2.4: p23). Knowledge or facts and experience are applied to the problem and one of many solutions concluded. The selection of the appropriate solution depends on the scenario and factors that surround the problem. Incomplete or uncertain data add to the complexity of the problem. In order to distinguish a suitable solution from an unrealistic solution, common sense may be applied. An intelligent system has a bit of this common sense included in its information bank and it may be referenced when necessary.

When a problem, such as the student advice scenario, has a restricted domain, the specifications of the problem can be communicated with ease. This expression of the specifications is an essential step in the intelligent system development (See Paragraph 3.1.4: p33). If such a system is of reasonable scope, it increases the probability of success to be reproduced by rules in a knowledge base (See Paragraph 6.1.1: p110). If a problem takes more than a few hours to be solved by a human, the mental processes and common sense used by humans to solve this problem would probably be overwhelming and impossible to be reproduced by a set of rules (Bielawski & Lewand 1991). Advising a student has a limited scope and should not involve hours of reasoning.

When the problem solution requires an explanation of how the system reached its conclusions, it confirms that an intelligent system's approach would be preferable (Bielawski & Lewand 1991). Advising a student involves reasoning and explanations. The focus of this study is to show that it is possible for a domain expert to supply and maintain a knowledge base of expert rules used in this Decision Support System for undergraduate students to obtain advice about the required and desired courses necessary in the current year of study. This dissertation focuses on the knowledge representation interface to be used by course experts that maintains rules used by a software application.

4.4.3 Requirements specification using a scenario-based approach

In order to derive a requirement's specification, a scenario-based approach was taken as specified by Haumer, Pohl & Weidenhaupt (1998). A current state analysis was performed to specify the functionality of the existing manual system. The Computer Science study program scenario, as outlined in the Rules and Syllabuses by the University of Pretoria (1999): Faculty of Science, was analysed as a use case to support the definition of the current-state model and to derive the change definition. According to Haumer et al (1998), the use of a scenario in the form of rich media, additional to the conceptual model, improves the quality of the requirements engineering process, leads to better understanding of the usage domain and enforces focused observation. Capturing the current system usage causes the abstraction process, which leads to the definition of the current-state model, to be more transparent and traceable, whilst explaining and illustrating the conceptual model. This process provides the basis to refine and detail the conceptual model during later phases.

Haumer et al (1998) focuses on the elicitation and validation of goals achieved by the current system and then outlines the interrelating goals and recorded observations. Their overall approach is shown in Figure 4-4 (p55).

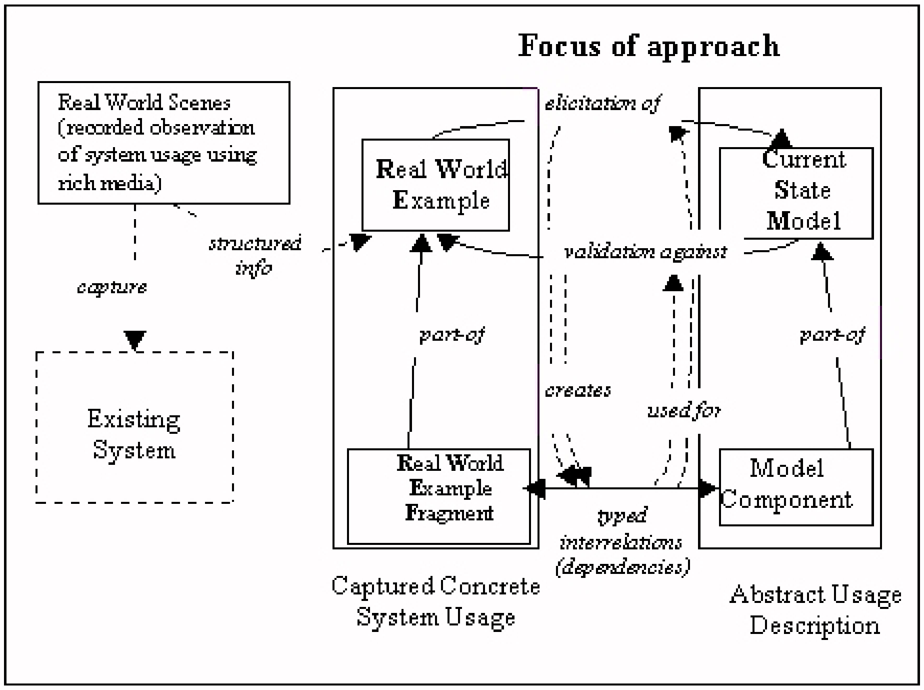

Focus of approach
Real World Scenes (recorded observation of system usage using rich media)
capture
structured info
Existing System
Real World Example
elicitation of
Current State Model
validation against
part-of
part-of
creates
used for
Real World Example Fragment
Model Component
typed interrelations (dependencies)
Captured Concrete System Usage
Abstract Usage Description

Figure 4-4 Interrelating Real World Example Fragments with model components (Haumer et al 1998)

Observations of the current system are captured and structured to reflect information about one system usage called a Real World Example (RWE). This approach uses dependency links to relate the goal to the parts of captured observations. Goals are chosen as the central concept for defining the current-state model. The RWE is used to validate the current-state model and to elicit new concepts. The fragment of the RWE that interrelates with the fragment of the conceptual model are elicited and validated. This approach is suitable in cases where the functionality of the old system has to be duplicated in the new system.

The goal and sub-goals in the form of rules and requirements were specified. These rules and requirements were put together to form the initial formal requirements specification. The main goal was established as being the acquisition of a qualification and the sub-goals the acquiring of the necessary courses to complete a specific qualification in the shortest possible time schedule allowed. The context of the custom DSS or SDSS is shown in Figure 4-5 (p56).

4.4.4 Derived problem definition

The rules and requirements that were derived as a result are summarised as follows:
To correctly advice the student, the student's results of courses attended needs to be considered. This should be an input to the advice system. According to a set of rules, possibly placed in a rules subsystem, some compulsory courses should be recommended to the student. A list of optional courses can be presented to the student to choose. At the end these choices should assist that the student complete his B-degree in the shortest period allowed. Some recommendations and alternatives should be presented to the student. The choices of the student should fit in with the timetable of the university – another input to the advice system.

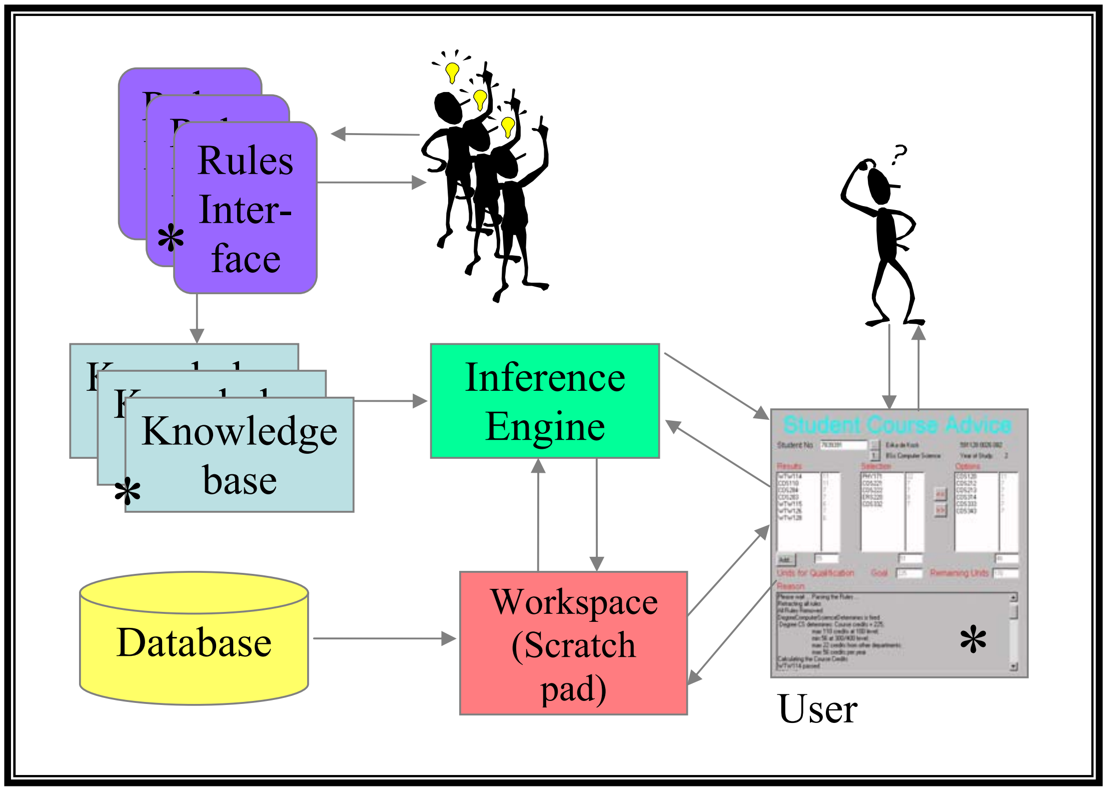


Figure 4-5 Context of the advice system

In this advice process, there should be components that:

- Recall the existing courses attended and passed by the specific student
- Specify the requirements of the specific degree and courses – possibly a set of separate rules for each study program
- Take these rules and infers advice to the student, matching the recommendations to the timetable of the university
- Present the advice to the student, and
- Offer a possibility to the course experts of the university to change the requirement rules

A summary of the rules derived from the scenario-based exercise using the Rules and Syllabuses of the Faculty of Science, University of Pretoria (1999) is as follows:
A student enrols for a specific B-degree. The degree has a name e.g. BSc, a unique code e.g. 02133221 and a study program name e.g. Computer Science associated with it. A degree consists of a number of courses that need to be included in the degree e.g. WTW114. Each of these courses has a number of credits assigned to it e.g. 11. Together with other course's credits, it will add up to a total number of credits needed to obtain the qualification. Every degree has a certain set of rules or prerequisites that needs to be adhered to, before a student can enrol (See Table 4-3).

Table 4-3 Pre-registration rule categories

Pre-registration rule categories:
Minimum performance of the prospective student obtained at subject level grade 12 for example:
– Mathematics HG: 50% or more, or SG: 60% or more
– Matriculation-score: 20 or more
University based rules e.g. obtaining a certain mark in a bridging course
Approval from the head of department or the dean of the faculty

Before a B-degree can be conferred on a candidate, a certain set of requirements should be adhered to (See Table 4-4). These requirements are categorised at degree level or at course level and may change in time. A separate component to maintain these requirement rules without prolonging the process before the rules are activated too much is thus a necessity.

Table 4-5 (p58) lists some of the types of rules specified for courses. Each course has a course code e.g. COS248. The first three characters are used to identify the subject e.g. Computer Science. The next character is used to show the level e.g. second year. This however is being invalidated by the anti-semester implementation. The last two characters are a unique number within the subject and the level to distinguish one course from another. These and other characteristics are shown in Table 4-6 (p58).

Whenever a student chooses a course that contradicts any of the rules, an explanation should be offered and the choice recalled. This study aims to investigate a possible solution to the above problem, focusing on computerised support for decision making to benefit the students and the personnel of the university. Chapter 4 (p41) will focus on the construction of such a system.

Table 4-4 BSc Computer Science requirement rules

<u>Degree requirements</u>:

Minimum number of credits at final year level e.g. at least 56

Maximum number of credits at first year level e.g. maximum 110

Maximum number of credits from departments other than the home department of the field of studies e.g. 22

A student has to pass at least half of his courses per year

No third year subject done at other universities will be accredited

Minimum credits to confer on the degree e.g. at least 225

Table 4-5 Rules governing the courses

<u>Rules governing the courses</u>:

Compulsory courses as specified by the faculty

– Specific courses e.g. WTW114

– A subset of courses e.g. the following eight courses and at least four of a list of six courses

Recommended by

Degree e.g. six courses

Choice

Other

4.4.5 Prototyping the requirements

In order to discover and verify the requirements (Whitten, Bentley & Dittman 2001) of the above-mentioned problem, a user interface prototype was developed using Delphi. Prototyping improves user-developer communication and is more likely to meet user needs. This prototype is used as a surrogate specification to visualise specifications that would be written on paper, with the option that this prototype can evolve in into the finished product. The prototype approach being a surrogate specification is also known as throwaway prototyping. Throwaway prototyping is normally less suitable for Decision Support Systems (DSS), because of the different usage patterns of DSS. This factor makes the second approach: to evolve the prototype into the finished product better suited for DSS. This approach is also called evolutionary prototyping or rapid application development. The nearly finished appearance of the interface may however mislead users that the system itself is nearly done. In this case, much investigation and development needs to be done to produce an interfacing rules subsystem suitable to be maintained by course experts.

Table 4-6 Course characteristics

<u>Each course has a</u>

Course description

Level e.g. 1st, 2nd, 3rd

Home department e.g. Computer Science, Information Technology, Mathematics

Presenting department

No of credits e.g. 11

Semester or Anti-Semester: This would be 1st or 2nd semester or anti-semester 2nd semester but also presented in the first semester

Courses pre-requisites: The pass level could be one of the following:

–GS-A Final mark of at least 40% e.g. COS110 GS

–( )-Exam Entrance e.g. (COS110)

–....-Passede.g. COS110

–TD-Approval of the Dean

–TDH-Approval of Head of Department

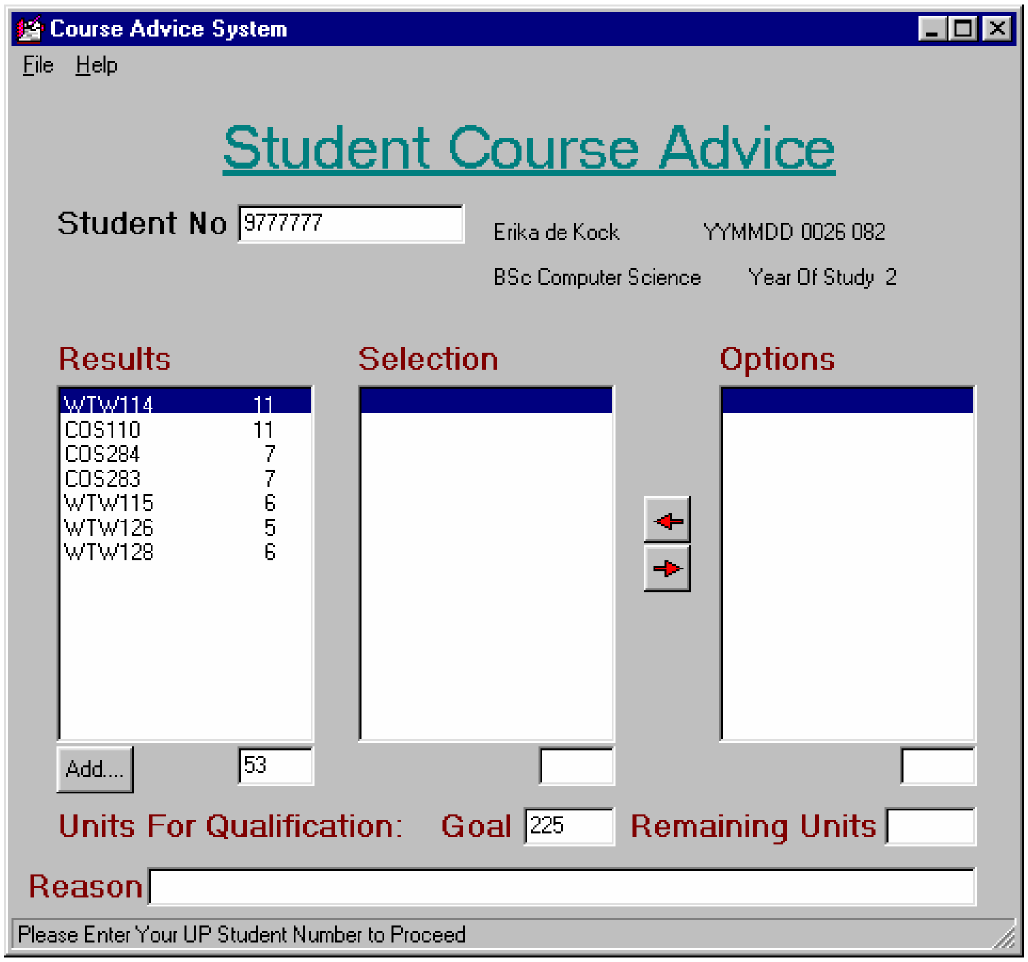
–Excludede.g. should not be taken if COS110 was completed by the student

List of courses that should be taken in conjunction (passed or simultaneous) with a specific course e.g. COS110

Delphi was chosen as the prototyping medium, because of its rapid application development features and the possibility that the rules subsystem still to be discovered, may interface with it using Object Linking and Embedding (OLE) or ActiveX controls. The prototype was developed without a database interface and without the rules subsystem. It was intended to, if satisfactory; evolve into a further prototype including a rules or knowledge-based subsystem. The focus of the system is intelligent subsystem's interface, its knowledge base and its knowledge editor. The intelligent system's control mechanism is assumed a working component not to be designed by this study. The student registration user interface prototype is presented in Figure 4-6 and Figure 4-7.

The advice process starts by entering a valid student number. An interface to the database extracts the student's details as well as all the courses the student has passed. This is the input to the intelligent subsystem. The intelligent subsystem then suggests, by consulting its knowledge base and control mechanism, subjects to register for and other possible subjects to choose from (See Figure 4-7). The student can then enter a "what-if" scenario by adding subjects to his acquired list and selecting or deselecting subjects he wants to register. He applies his personal interests and goals for his studies and then the system responds using the intelligent subsystem approving or rejecting the student's choices and supplying the necessary reason(s) for it.

The information requirements of the advice system were determined in Paragraph 4.4 (p53) and an Information Requirement Definition (IRD) prototyped to understand the user needs and objectives of the anticipated DSS/KB-DSS. Setting the objectives, understanding the user needs and formalising the problem is part of the first phase when designing a complex DSS (Turban 1995) (See Paragraph 3.1.6: p35).



Student No: 9777777

Erika de Kock YYMMDD 0026 082

BSc Computer Science Year Of Study 2

WTW114	11
COS110	11
COS284	7
COS283	7
WTW115	6
WTW126	5
WTW128	6

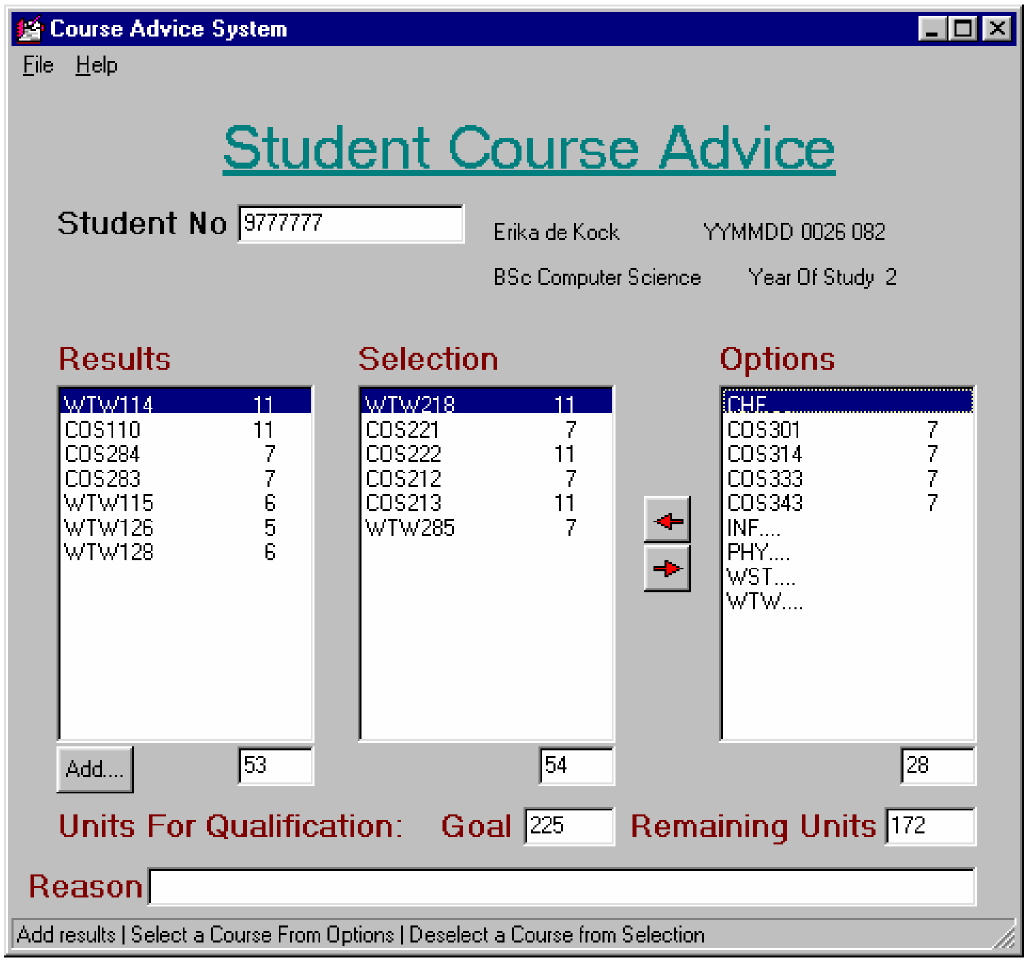
53

Goal 225

Please Enter Your UP Student Number to Proceed

Figure 4-6 Initial prototype of the student advice system in Delphi

A prototyping approach called the evolutionary process (Keen 1980, as referenced by Turban 1995) or iterative process (Sprague & Carlson 1982) was adopted (See Paragraph 3.1.7: p35) for the remainder of the phases. Prototyping combines the four processes of the traditional SDLC (analysis, design, construction and implementation) into a single step that could be repeated several times until a relatively stable and comprehensible system evolves. Recall that the iterative process seems to be the most appropriate (See Paragraphs 5.1.4: p75 and 3.1: p32) to build a DSS. For this reason, the KB-DSS prototype will be developed from an end-users point of view (See Paragraph 3.1.7: p35). If no suitable KB-DSS packages exist, another construction process such as the implementation process by Klein & Methlie (1995) in Paragraph 3.2 (p37) may be followed. Klein & Methlie’s (1995) methodology of implementing a KB-DSS will be used to assist Turban’s (1995) development process (See Figure 3-3: p40 and Paragraph 3.2: p37).


Course Advice System

Student Course Advice

Student No 9777777

Erika de Kock YYMMDD 0026 082

BSc Computer Science Year Of Study 2

Results		Selection		Options	
WTW114	11	WTW218	11	CHE....	
COS110	11	COS221	7	COS301	7
COS284	7	COS222	11	COS314	7
COS283	7	COS212	7	COS333	7
WTW115	6	COS213	11	COS343	7
WTW126	5	WTW285	7	INF....	
WTW128	6			PHY....	
				WST....	
				WTW....	
53		54		28	

Units For Qualification: Goal 225 Remaining Units 172

Reason

Add results | Select a Course From Options | Deselect a Course from Selection

Figure 4-7 The user interface prototype after consulting the rules subsystem

4.5 The application prototyped

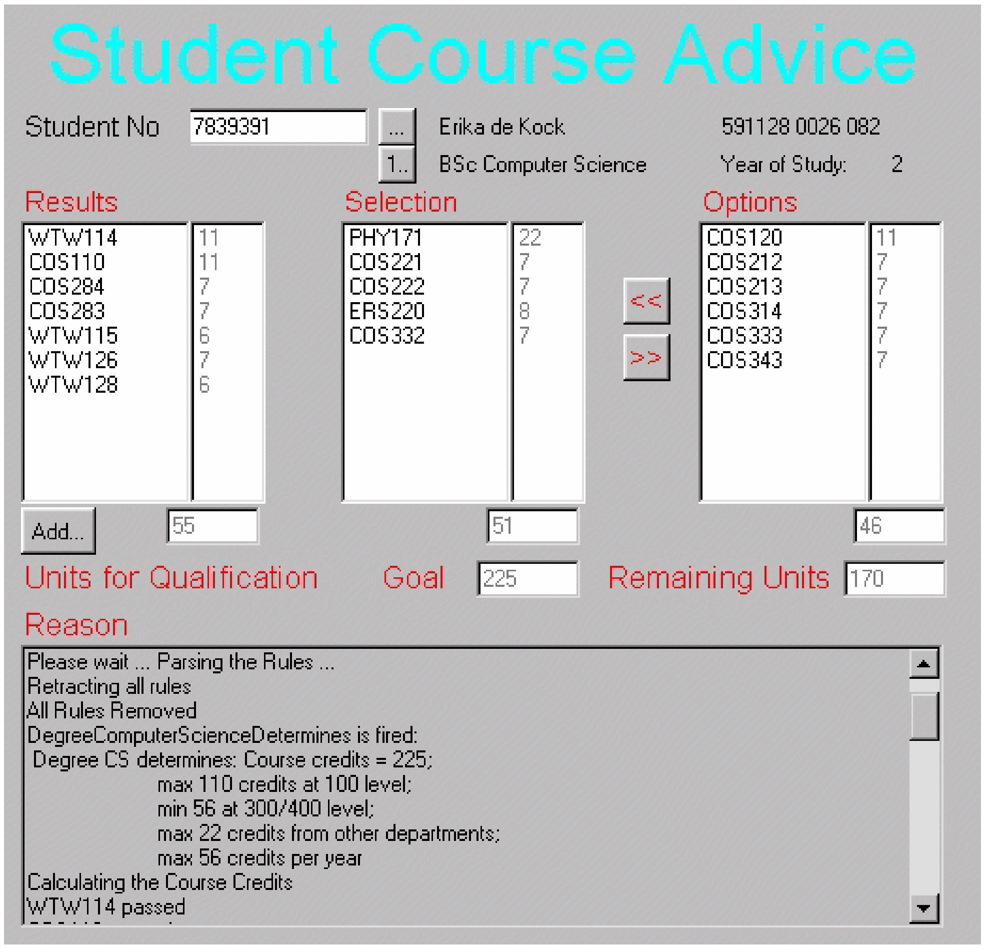
A prototype of the advice system was developed according to the examples shipped with ILOG JRules. In order to use the rule base in an application the ILOG libraries must be referenced in the coding. Commands such as retractAll( ) and fireRules( ) initiate the inference of rules within the prototype. Specific routines that realise as separate methods inside a particular rule class need to be specified. These routines manage the actions following the assertion and retraction of rules.

The prototyped application executes in two modes:

- Firing all suitable rules and displaying the explanations in a scrollable list box (… button), and
- Stepping mode firing one rule at a time and displaying the explanations rule by rule: At the same time the selection and options list boxes are updated with the recommendations (1… button)

The student may enter into a "what-if" scenario by selecting Options-list-box-courses to be Selected-list-box-courses. In the same way, students may choose to remove Selected-list-box-courses. In these actions, the students exercise their preferences. The expert component verifies the student's actions and supplies reasons why the student may select or may not select such a combination of courses to complete the required qualification. The application gives the student the ability to view future semesters as well as allowing the student to add his passed courses. When pressing the Add… button, the student may enter codes of courses passed (See Figure 4-9: p63).

On exit, the explanation list box clears, the rules fire again, using the mode chosen and the updated Selection, Options and Reasons are displayed (See Figure 4-10: p64). Students may enrol for a certain unit count of courses of related departments. This functionality is not included in this prototype, but can be invoked by displaying the different departments in the options list box. On selection of the department, a list of the other department's subjects can be added to the Options list box to be selected by the student. Rules governing this selection process should be added to the current rule set or alternatively another rule set invoked to explain and allow selection of these courses.



Student No: 7839391

Erika de Kock   591128 0026 082

BSc Computer Science   Year of Study: 2

Results		Selection		Options	
WTW114	11	PHY171	22	COS120	11
COS110	11	COS221	7	COS212	7
COS284	7	COS222	7	COS213	7
COS283	7	ERS220	8	COS314	7
WTW115	6	COS332	7	COS333	7
WTW126	7			COS343	7
WTW128	6				
	55		51		46

Units for Qualification   Goal 225   Remaining Units 170

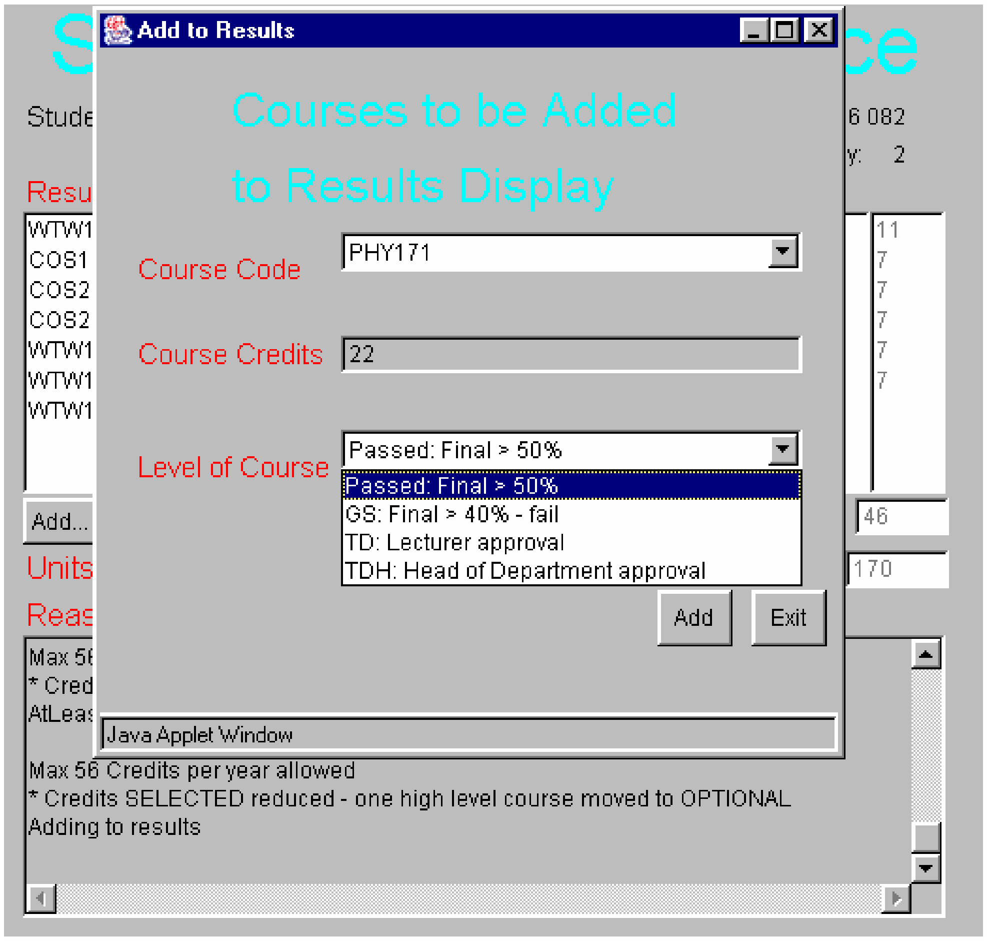
```
Please wait ... Parsing the Rules ...
Retracting all rules
All Rules Removed
DegreeComputerScienceDetermines is fired:
 Degree CS determines: Course credits = 225;
                    max 110 credits at 100 level;
                    min 56 at 300/400 level;
                    max 22 credits from other departments;
                    max 56 credits per year
Calculating the Course Credits
WTW114 passed
```

Figure 4-8 Working prototype of the advice system using ILOG JRules and Java

The student's final choice could be printed out, saved to a file, or e-mailed to him. A hard copy of the reasons could be provided as well. The student can enter into this "what-if" interaction as many times as he wishes. By matching his preferences to what is allowed by the rules he can make an optimum choice of courses for the next semester.

4.5.1 The rules prototyped using the ILOG JRules language

A prototype was developed using rules in the ILOG JRules language. The rules used by the prototype were set according to the specification for the system as presented in Paragraph 4.4.4 (p56) using the ILOG JRules language. An illustration of such a set of rules is given in Appendix A – Sample rules: F022331.irl. The Java source code for the prototype application is included on the CD. After a satisfactory rules file was developed using the ILOG JRules language and tested, the ILOG JRules builder was explored to determine if course experts would be able to maintain the rules.



Course Code: PHY171
Course Credits: 22
Level of Course: Passed: Final > 50%

```
Passed: Final > 50%
GS: Final > 40% - fail
TD: Lecturer approval
TDH: Head of Department approval
```

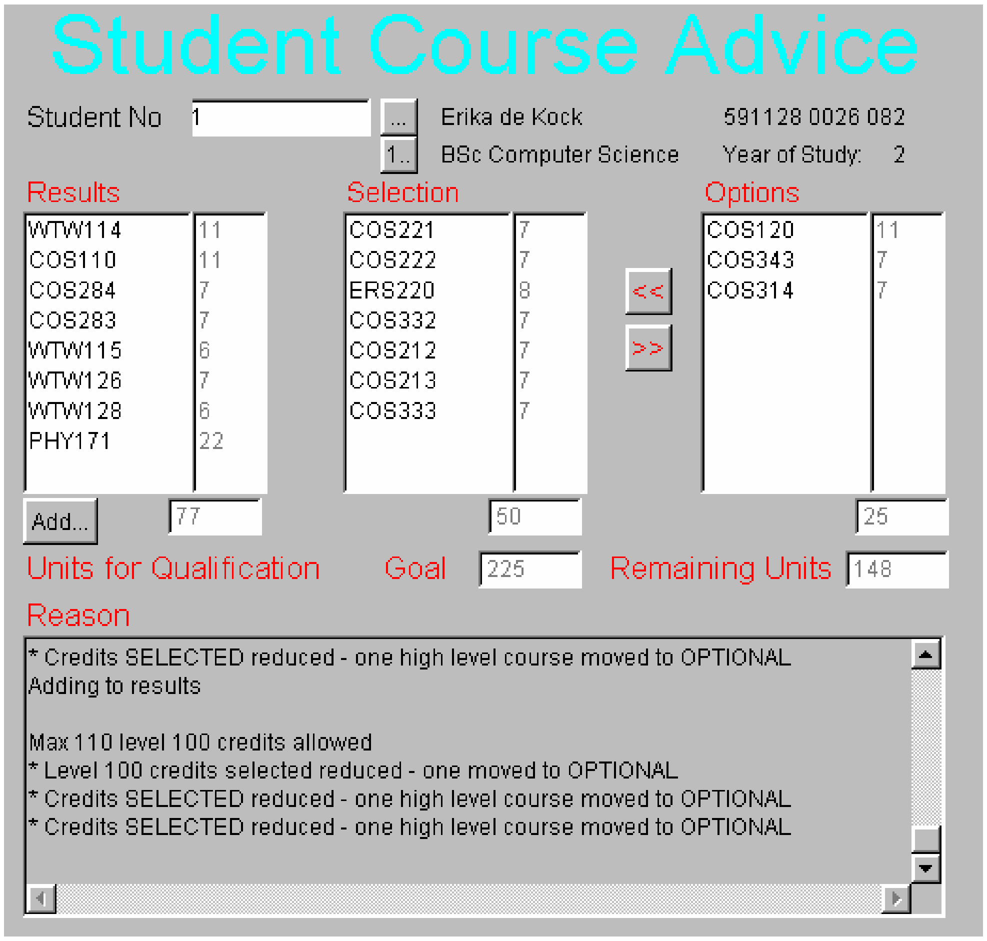
```
Max 56 Credits per year allowed
* Credits SELECTED reduced - one high level course moved to OPTIONAL
Adding to results
```

Figure 4-9 "What if" future semesters/years: Adding courses passed to results

4.5.2 The structure of a rule in the ILOG JRules language

An ILOG JRules rule has the following structure (ILOG 2000 (a)):

```
rule myRule  {
        priority high;
        [packet = abc;]
        when {  conditions …. }
        then  {  actions …… }
    }
```


Student Course Advice

Student No 1 ... Erika de Kock 591128 0026 082

1.. BSc Computer Science Year of Study: 2

Results

WTW114	11
COS110	11
COS284	7
COS283	7
WTW115	6
WTW126	7
WTW128	6
PHY171	22

Add... 77

Selection

COS221	7
COS222	7
ERS220	8
COS332	7
COS212	7
COS213	7
COS333	7

50

<< >>

Options

COS120	11
COS343	7
COS314	7

25

Units for Qualification Goal 225 Remaining Units 148

Reason

```
* Credits SELECTED reduced - one high level course moved to OPTIONAL
Adding to results

Max 110 level 100 credits allowed
* Level 100 credits selected reduced - one moved to OPTIONAL
* Credits SELECTED reduced - one high level course moved to OPTIONAL
* Credits SELECTED reduced - one high level course moved to OPTIONAL
```

Figure 4-10 Selection and Options after new "what-if" results

Rules have unique names or headers (ILOG 2000 (a)), and are defined by the keyword rule: `rule DegreeComputerScienceDetermines {… }`. The rule has a priority, optional packet, conditions and actions. ILOG JRules infers its objects directly from the Java classes without duplication. Relevant objects are placed in the Working Memory (WM) by the assert command. The working memory is the place where ILOG JRules stores all its currently active objects. The agenda is a place where selected instances of rules are stored to be executed by the inference engine. A rule instance is a rule instantiated due to active objects from the working memory and is produced by any combination of objects in working memory that pattern matches the specified rule. A rule instance is a dynamic concept, whilst a rule is a static concept.

♦ Conditions

To determine a certain condition of a rule, the process of pattern matching is performed. The process determines the existence of one or more objects in working memory that matches the attribute values in the condition part of the rule. When a match is found an instance of the rule is placed in a place called the agenda. A negative pattern identified by a tilde (~) sign gives rise to a successful match if there are no objects in the working memory that match the desired pattern (ILOG 2000 (a)). Whenever a successful match is performed, an instance of the rule is placed in the agenda.

More than one condition may be specified in a rule. Each condition must be on a separate line. All patterns of a rule must be matched before an instance of the rule is placed in the agenda. If one pattern is not successfully matched, then no rule instance is placed in the agenda and none of the actions of the rule performed (ILOG 2000 (a)). When a successful match occurs, the object may be marked with a variable (?x:) to be referred to in further patterns or in the actions of the rule. Each condition operates as a filter that selects a subset of objects. Variables may be used to reference a specific attribute value.

- **Actions**

Actions are operations that change the working memory. Two elementary actions are "assert" and "retract". Assert inserts an extra object into the working memory, while retract removes a particular object from the working memory (ILOG 2000 (a)).

- **Priority**

The priority part determines the order in which ILOG JRules' inference engine executes the different active rule instances. Two types of priorities exist: static and dynamic. Static priorities have specific values including a few predefined values such as maximum, high, low and minimum. Dynamic priority is used to alter the priority between instances of the same rule using variables defined in the condition part of the rule (ILOG 2000 (a)).

- **Packets**

Rules may be optionally grouped together using the keyword packet. A packet may be removed or added from the rule base and provides a means to activate and inactivate groups of rules (ILOG 2000 (a)).

- **Rule sets**

A rules set may contain both isolated rules and rules in packets. Each department will ideally have its own rule set of rules to maintain.

4.5.3 The inference process of ILOG JRules rules

Current objects are placed in the working memory, whilst matching rule instances are placed in the agenda. When an object is inserted into the working memory (action assert or initialise performed) the active rules are pattern-matched with all the objects and the relevant rule instances placed in a place called the agenda (ILOG 2000 (a)). The relevant rule(s) in the agenda are capable of being fired. To fire the rule(s), "fireRule( )" or "fireAllRules( )" has to be executed. These commands execute the action portions of the relevant rule instances and remove the fired rule instance from the agenda. The contents of the working memory are changed and the satisfied rule instances placed in the agenda to be fired. Objects may be inserted into, removed from or updated in the working memory.

When several rule instances exist in the agenda, ILOG JRules' inference engine (ILOG 2000 (a)) has to decide which rule to fire using selection criteria such as priority, recency and order of rule names or lexicographic order. Firing a rule introduces a change to the working memory by means of actions (inserting, removing or updating objects) that might invalidate a rule instance that is already placed in the agenda but not yet fired. The invalidated rule instance automatically disappears from the agenda without firing it. This is also known as the truth maintenance system.

The working memory and the agenda evolve constantly and together these two containers constitute what is referred to as the ILOG JRules context. A context serves as an interface between the Java application and the ILOG JRules inference engine (ILOG 2000 (a)). A context associates a rule set with a working memory and implements the inference engine that controls them from the application.

4.5.4 The ILOG JRules Builder and syntactical editor

ILOG JRules assumes the rule programmer/builder is familiar with the environment in which ILOG JRules will be used. The Java Virtual Machine (JVM) needs to be installed together with the ILOG JRules software (ILOG 2000 (b)). A tool called ILOG JRules Builder exists that enables a user to create, modify and execute rules in a graphical environment. Rules created with this builder can be saved in a project file for later use. An environment called the syntactic rule editor allows both developers and users to write and edit rules in a more natural language, using menus to enter language statements and expression values.

♦ The ILOG JRules Builder

Rules may be written in different editor modes. Three rule editor modes exist (ILOG 2000 (b)):

- JRules syntactic mode
- JRules text mode, and
- Business Action Language syntactic mode

A Business Object Model (BOM) configures the rule editor and specifies the generation of executable rules to be used by an application. This is where an own business rule language may be specified. The builder provides a tracing and debugging tool where the contents of the agenda, the working memory, variable bindings, outputs and traces of the rules and active objects may be viewed. ILOG JRules may be launched in three different modes: offline, synchronised and advanced. When using the synchronised mode, the rules can be tested or debugged in conjunction with the application that uses it.

Upon entering the builder, a new project (.irp suffix) is created by default. The project is the root container for all objects and contains references to rule sets (.irs suffix) and BOM files (.bom suffix). Only one project may be edited at a time. One or more rule sets may be associated with the project. A rule set will contain rules written according to a Business Object Model (BOM). A rules set contains a set of rules that may be grouped in packets. Rule files in the JRules syntax (.irl suffix) may also be loaded when opening a rule set (ILOG 2000 (b)). An example .irl file is included in Appendix A – Sample rules.

An empty BOM file is created when a new project is created. The Builder allows Java classes to be imported and configured in the BOM. Classes and class members may be hidden, removed or edited for use in the rule editor window. When a class is added to the BOM, all referenced classes are also added automatically marked as not visible not to be seen by the syntactic rule editor. The syntactic rule editor only uses the visible classes and class members. Setting a class or class member as not visible removes it from the possible choices in the rule syntactic editor. Whenever a class is removed, the builder removes all its subclasses and the members that reference this class. The column 'Display Name' may be edited to enter a name that will be used instead of the class element name with certain rule languages. The class list editor displays a filtered view of the BOM. Only visible classes and members are displayed. The syntactic editor uses only this filtered classes and members.

To add a new rule to a rule set using the rule Editor the rule language should be selected. Three choices are available (ILOG 2000 (b)):

- Text: the JRules standard language with textual view of the rule editor
- JRules: the JRules standard language with syntactic view of the rule editor, and
- Business Action Language: any custom-developed business rule language with the syntactic view of the rule editor: Any rules written in languages other than JRules need first to be transformed to JRules syntax before the rules are executable by the JRules engine.

The rule editor provides two views for editing: textual (See Figure 4-11) and syntactic (See Figure 4-12). The textual view is only available for rules written in the JRules language. The textual view offers the standard features of a text editor (ILOG 2000 (b)). At any time, the syntax of a single rule or the rule set may be checked. If the syntax is not correct, the rule error panel will list the errors. If the rule set does not contain any errors, it may be executed.

Execution includes several operations. It checks the rule set, create an engine or connect to an existing engine, send the rule set to the engine, reset the engine and fire all the rules (ILOG 2000 (b)). The rule set and all its rules may not be edited during execution.

Debugging commands exist to allow inspection of the working memory, the agenda and the variable bindings. The history of all events that occurred in the engine is accessible in the trace panel. The output panel displays the messages that have been sent to the output stream associated with the engine. Rule breakpoints, class breakpoints and object breakpoints may be set to evaluate all the relevant mentioned panels.

```
when  {
        ?x:CurrDegree(currDegreeCode.equals("02130001");currStudyProgram.equals("02133221"));
  }
 then    {
       assert CourseDetails(225,70,56,22,56);
  }
```

Figure 4-11 Using the textual mode of the syntactical editor

```
WHEN
+
       there is a CurrDegree [ ] called ?x +
              such that currDegreeCode.equals("02130001")
                     and currStudyProgram.equals("02133221") +
+
THEN
+
       assert [ ] CourseDetails ( 225, 70, 56, 22, 56 )
      [ so that ]
+
```

Figure 4-12 Using syntactic mode of the syntactical editor

After a satisfactory set of rules are compiled and tested the resultant rules file (.irs suffix) could be generated as a text file to be invoked as external rules by the application that uses the rules. The file data/F02133221.ilr in Appendix A – Sample rules is an example of such a file. The creation of this text file is the focus of this dissertation. This process of setting, editing and debugging the rules needs to be of a manner that course experts would be able to maintain their department's rule set with little or no assistance, taking ownership for the rules in the rule base.

Whenever a rule base is set and tested, it can be placed in a central place. This rule base then becomes the current active rule set as soon as the next enquiry by a student is made. The application need not even be terminated. Each department/faculty would have its own set of rules. The next enquiry by a student would query the database, obtain his department of faculty, derive the applicable rule file name to fuel the inference engine and invoke the newly created version of the rule file.

♦ Example rules from the advice system

In Figure 4-13 (p69), an example rule for a required course that has a pre-requisite, is given. The syntactical editor view of the rule is presented. Once the rule is created, it is easy to create similar course rules for the qualification. The course expert that is knowledgeable in courses and credits will recognise the course codes and credit values. The course expert knows that COS222 is an Information Technology requirement for the degree BSc Computer Science. The prerequisite COS110 has to be passed before the student can register for COS222. COS222 is presented in both semesters and the student can optionally choose to do COS222 in a later semester. COS222 is a second level course and contributes seven credits to the qualification.

```
Rule ITRequirement5
WHEN
+
    there is a StudSubj called ?x
        such that subjName.equals("COS110")
            and passLevel = StudSubj.PASSED
+
    there is no StudSubj
        such that subjName.equals("COS222")
            and ( passLevel = StudSubj.PASSED
                or passLevel = StudSubj.REGISTERED
                or passLevel = StudSubj.OPTIONAL )
THEN
+
    assert StudSubj ( "COS222", StudSubj.LEVEL200, 7,
StudSubj.REGISTERED, StudSubj.BOTH_SEMESTERS )

+
```

Figure 4-13 A rule from the student advice prototype for a required course that has a prerequisite course

A copy of such a rule can be changed to reflect the new course and its prerequisite e.g. "COS110" can be replaced by "ERS220"; "COS222" by "ERS320" and "7" credits by "8" credits. The new rule can be named CompulsoryCourseERS320 to include ERS320 and its prerequisite in the qualification for BSc Computer Science. The rule is frame-based (See Paragraph 6.2.2: p113). A frame called StudSubj exists with different slot values. The slot values can either be of a specific range of values or numeric and is created to support the rule.

If the pre-requisite COS110 is passed and the rules fired in the application, this course will appear in the Selection-list. The Selection list shows the recommended subjects the student should register for (Figure 4-8: p62). Explanations for the recommendations are given in the Reason-list box. "Information Technology Requirement COS222: Prerequisite COS110 passed" would be the explanation of the ES component for this course recommendation. The textual mode of the same rule is given in Figure 4-14 (p70).

Figure 4-15 (p70) shows a textual mode rule for a maximum of 100 credits allowed for level one courses. Slots should exist that contains the number of credits for a specific level. Another slot should contain the student's collective course credits for all passed and registered courses at a specific level. Before using additional frames and slots, e.g. CourseDetails and TestLevel100, for collections they need to be coded by a programmer as part of the application. Once the frames and slots are coded, course experts can maintain the rules.

```
When   {
       ?x: StudSubj(subjName.equals("COS110");(passLevel==PASSED));
       not StudSubj(subjName.equals("COS222");(passLevel==PASSED)
                          ||(passLevel==REGISTERED)||(passLevel==OPTIONAL));
 }
 then   {
assert StudSubj("COS222",LEVEL200,7,REGISTERED,BOTH_SEMESTERS);
 }
```

Figure 4-14 The textual mode of an example rule for the student advice system

```
When     {
         CourseDetails(?a:level100Max);
         ?c: collect ( new TestLevelSubj(LEVEL100))
         StudSubj(level==LEVEL100;(passLevel==PASSED)||(passLevel==REGISTERED);
                    ?y:noOfUnits)
         where (totalNoOfUnits() > ?a);
    }
   then      {
          assert(?c);
    }
```

Figure 4-15 Textual mode rule for a maximum number of 100 credits for level 100

♦ The syntactic rule editor

The syntactic rules editor of the builder allows the writing and editing of rules in a more natural language than using the textual editor. This editor allows non-JRules language programmers to write and edit rules. Menus provide a user-friendly way to enter statements and expression values. This editor may be used to specify rules in an in-house developed business rule language too. To develop an appropriate set of rules, the given sample language was found to be inadequate. The sample language needs to be extended before a set of usable rules can be created to assist the prototyped application. In particular, rules that combined objects into collections were difficult to specify using the sample language. The functionality to specify it using the JRules syntax existed.

To provide such a self-developed business rule language, a Business Object Model has to be defined. The development of such a language (business action language by specifying a business object model) may be the content of an additional study. Setting up this specification needs the expertise of programmers familiar to the JRules environment. This could provide an even more user-friendly way to specify rules and expression values to be used by domain experts. The business language may also be used as an independent Java Bean (ILOG 2000 (b)).

A new Business Object Model is a viable solution, but outside the scope of this study. It may be a valid topic for further research in the area. The following discussion will focus on the editor and Business Object Model provided by JRules. The aim is to determine if the provided model and editor is sufficient to be used by the different departments to set up the rules needed to automate the students' enquiries on electing possible courses.

♦ The textual mode

The condition part of the JRules rule as discussed in Paragraph 0 (p63) is substituted with 'there is a' or 'there is no' followed by a selection of the applicable class. The 'where' clause incorporates fields of the class and binds them to local variables and 'such that' write tests on field values (ILOG 2000 (b)). When collections are evaluated in a condition, the token 'the selection of' may be used. By selecting the token '+', any number of tests may be incorporated in a single rule. The '+' token is also a delimiter between actions in the action part of the rule. Possible actions include 'assert', 'retract', 'update', 'modify', 'apply', 'bind', 'execute', 'if', 'while' and 'timeout'. The token 'so that' is used to assign values to the object fields of the selected class. Rules or portion of rules may be inserted, deleted, copied, cut and pasted to facilitate the specification of similar rules for example prerequisites for certain courses.

♦ Business Action Language

The business action language is a sample business rule language supplied with JRules, using a natural and readable syntax (ILOG 2000 (b)). Only a subset of the JRules language is implemented and is therefore limited in its application. These rules are translated into executable JRules rules before use in an application. This language can be used to define an own customer business action language by specifying a Business Object Model (BOM). As stated before, the definition of such a language is beyond the scope of this study. Mapping the BOM to the application classes can specify a whole vocabulary to be used by the rule administrators.

4.5.5 Evaluating the creation of rules using the rule editors

Objects can be grouped into collections and tested against certain conditions. These rules are the toughest to set and would need the assistance of a programmer/builder. Examples are TooManyCreditsPerYear, and TooManyLevel100CreditsMoveLastOneAdded. Appendix A contains sample rules in the different modes used by the advice SDSS prototype. The syntactic rules editor of the builder allows the writing and editing of rules in a more natural language than using the textual editor. This editor allows non-JRules language programmers to write and edit rules. Menus provide a user-friendly way to enter statements and expression values.

Adding similar rules and changing existing rules is quite easily achieved. Many of the rules will be of the same type e.g. prerequisite courses, total number of credits for a level, total number of credits for the qualification, etc. These rules could easily be maintained by changing the credit count, adjusting the prerequisites, adding new pre-requisites. The rule administrator, when needed, can create similar rules. The rule administrator would need to copy an existing named rule as a new named one and change the field names and values. The field values and the course codes can be changed to reflect the new prerequisite. If any changes need to be applied to existing rules, the field values can be replaced. If any of the rules are no longer valid, they can be removed from the rule set or package. Course expert rule administrators can maintain these types of rules using either the textual and syntactic modes of the builder.

When, however a new type of rule, different in structure to any other rule in the rule set, a builder/programmer's assistance will be needed. If a collection type rule, such as, "Only a certain amount of units for specific course level or attribute allowed", is needed, it will be safer to allow a builder/programmer expert to code or assist in coding such a rule. When altering the knowledge base, the rules need to be verified. Problems (See Paragraph 7.2.2: p130) that can be detected are:

- Consistency problems such as redundant rules, conflicting rules, subsumed rules and circular rules, and
- Completeness problems such as missing rules and gaps in the inference chains

These logical problems may not necessarily cause reasoning faults.

Using the debugging commands supplied with the builder environment can perform rule traces. It can also assist in performance validation. The debugging commands allow inspection of the working memory, the agenda and the variable bindings (ILOG 2000 (b)). The history of all events that occurred in the engine is accessible in the trace panel. The output panel displays the messages that have been sent to the output stream associated with the engine. Rule breakpoints, class breakpoints and object breakpoints may be set to evaluate all the relevant mentioned panels. The rule administrator needs to be trained in using this debugging environment to determine the validity of the rule base.

It is possible to set up the rules file (.ilr) using an ordinary text editor. The rules are typed and saved in an .irl file. The syntax of these instructions is checked on execution of the DSS and a system failure will result if this .ilr file is incorrect. Rule editing using an ordinary text editor is suitable for the programmer/builder only. The rule administrator can maintain the rules using the syntactic builder.

♦ Summary

A programmer/builder should set up the initial rules sets and any subsequent rules of a new type. Rule administrators can be allowed to maintain the rule set by adding similar rules using copy and paste and by changing existing field values of rules. The rule administrator needs to be trained to debug the rule base, verifying the completeness and consistency of the rules as described in Paragraph 7.2.2 (p130). Rules should be tested, validated and compiled before using it with the active SDSS. JRules has an expert system as an integral component that can supply recommendations and explanations to the user, who is the student.

The knowledge component is in a limited sense maintainable by course experts. Course experts can easily copy, paste and change rules as discussed. Once trained in the use of the software, the interface would be logical and useful to the experts, especially when the copying of rules and the changing of field values are required to maintain the rule base. The knowledge component interfaces with the SDSS/KB-DSS application classes and is available to departmental experts. Departmental experts can access the rules decentralised or independent of the users and other experts. Once the rule base is verified, it can be released and copied to the JRules server and released into production. The JRules Builder 3.0 uses Remote Method Invocation (RMI) to communicate with the JRules engines.

Chapter 5 - A broader perspective to Decision Support Systems

5.1 Introduction

This chapter presents a broader background into what DSS are and from where they evolved. It also presents definitions of the terms used in Chapter 2: p6. As stated before, the term Decision Support Systems (DSS) is viewed as a context-free expression. It means different things to different people. There is no universal accepted definition of DSS (Turban 1993). DSS attempt to automate several tasks of the decision-making process (Turban et al 2001). Before viewing DSS, decisions and the process of making decisions are investigated.

5.1.1 What is a decision?

A decision is a reasoned choice amongst alternatives (Mallach 1994; Simon 1960). Decision-making is the process of choosing among alternative courses of action, to attain a goal or goals (Turban 1995). Problem solving and decision-making are not viewed as synonyms in the literature. Problem solving is the overall process of closing the gap between reality and a more desirable situation. Problem solving includes the aspect of making decisions. One view is to consider all four phases or steps of Simon's (1960, 1977) model (See Paragraph 2.2.1: p7) as being part of problem solving, and step three as decision-making. Another view is to consider steps one through to three as decision-making, ending with a recommendation. Problem solving additionally includes implementation of the recommendation (Turban 1995).

5.1.2 Types of decisions

Well-structured problems are repetitive and routine. They are easily solved and may be presented by a standard model. The poorly structured problems are new, novel, non-recurrent and difficult to solve (Shim et al 2002). **Semi-structured** decisions have some structured aspects. Most organisational decisions are of this type (Mallach 1994).

5.1.3 The scope of decision-making

Anthony (1965) categorised managerial activities into strategic planning, management control and operational control. Other authors refer to this as the scope of the decision-making process (Mallach 1994). A **strategic** decision will affect the whole company or a major part of it. It will affect the company's objectives and policies. Strategic decisions are generally made by the upper management level of the company. It affects the company for an extended period of time (Antony 1965; Mallach 1994). Strategic decisions tend to be concerned with the levels of resources needed to achieve organisational goals and involve long-term relationships between the organisation and its environment (Finlay 1994).

A **tactical or managerial control** decision will affect how the organisation works for a limited period. These decisions take place within the context of the previous strategic decisions. Middle management is normally involved in tactical decisions (Mallach 1994). Tactical decisions concern activities that have a longer time-span than operational activities. They are primarily concerned with the most appropriate effective use of the resources already available in the company (Antony 1965; Finlay 1994).

An **operational** decision affects activities taking place in the company right now. The tasks, resources and goals of these activities have already been set by other strategic or tactical decisions. It involves the day-to-day well-established procedures (Finlay 1994) or the execution of specific tasks (Antony 1965). The operational decisions made do not have any impact on the future of the company. Lower management or non-managerial personnel are normally involved in operational decisions. Procedures used in operational decisions are classified as routine and information is unlikely to surprise the decision-maker (Mallach 1994).

5.1.4 The decision-making process

Each decision is characterised by a decision statement, a set of alternatives and a set of decision-making criteria. These always exist, though we are not always aware of them (Mallach 1994). Simon (1960) proposed decision-making as a four-phase model: (1) intelligence, (2) design, (3) choice and (4) implementation (Turban et al 2001: Figure 2-1: p8) also called review (Finlay 1994). Intelligence is comprised of the search for problems; design involves the development of alternatives; and choice consists of analysing the alternatives and choosing one for implementation (Shim et al 2002). There is a continuous flow of activities from intelligence to design to choice, but at any phase there may be returned to a previous phase (See Figure 2-1: p8).

Finlay (1994) suggests a slightly modified and extended model to Simons's (1960) model, called problem tackling, consisting of three phases: structuring, understanding and action in seven stages. His model is shown in Table 2-1 (p8). He recognised a backtracking and iteration between his stages.

♦ The intelligence phase or problem detection and definition

The need for a decision is determined in the intelligence phase. Simon (1960) expresses the intelligence phase as "searching the environment for conditions calling for decision". Finlay (1994) defines it as the searching for problems. He calls this the problem detection stage in his model. The decision-maker decides what to decide or formulates the problem that needs a decision. This formulation is called a decision statement or problem statement (Mallach 1994) or the problem definition stage in Finlay's (1994) model. Finlay (1994) states that the problem will be defined once an objective and the associated obstacle have been defined.

A decision statement states what we are trying to decide. A clear decision statement is important to intelligent decision-making. It keeps the decision-maker's thinking focussed clearly on the main subject and away from irrelevant side issues. Every Decision Support System development project should start by gaining a clear understanding of the decisions to be made with the help of the proposed system (Turban 1993).

The term intelligence in the intelligence phase means the gathering of information without knowing the outcome of the decision (Mallach 1994). The modelling process starts here. Turban et al (2001) defines it as the stage where reality is examined and the problem identified and defined. The intelligence phase may involve activities such as problem classification, problem decomposition and determination of problem ownership (Turban 1993).

- **Problem classification**

Two important factors to keep in mind are the scope of the decision and its degree of structuredness. An important classification is according to the degree of structuredness: **structured** vs. **unstructured** (Mallach 1994) or programmed vs. non-programmed (Simon 1960).

Information is the raw material for intelligence. The manager or decision-maker might need assistance to interpret the information in order to include it in his problem situation or scenario (Finlay 1994). Data, information, intelligence and the process whereby these are formed are all aspects of knowledge. Finlay (1994) presents a knowledge model that build a decision-maker's intelligence in order to place the decision-maker in a better position to plan and control his decisions (See Figure 5-1: p77). The scenario sets the context of the evaluation of a decision outcome e.g. worst case, best case and most likely case. This, to a large degree, sets the evaluation criteria used in the choice (Turban 1993). This model is applicable to any type of cognitive system, not just individual and group planning and decision-making (Finlay 1994).

Managers need information almost continuously. Finlay (1994) claims that different information requirements are necessary for the different levels of problems or decision scope. Figure 5-2 (p77) shows the different requirement characteristics at the different levels of decision scope. Most of the dimensions in Figure 5-2 are self-explanatory. The hardness of the information requirements relates to the objectivity associated with the source. It is often the case that hard data is quantitative and written, while soft data tends to be qualitative and verbal.

- **Problem decomposition and ownership**

Problems may be broken down in simpler sub-problems that could be solved individually. This may help to solve the original more complex problem. It is important to establish the ownership of the problem by establishing if the problem is capable of being solved within the organisation or if it is an uncontrollable factor (Turban 1993). The intelligence phase ends with a problem statement (Mallach 1994).

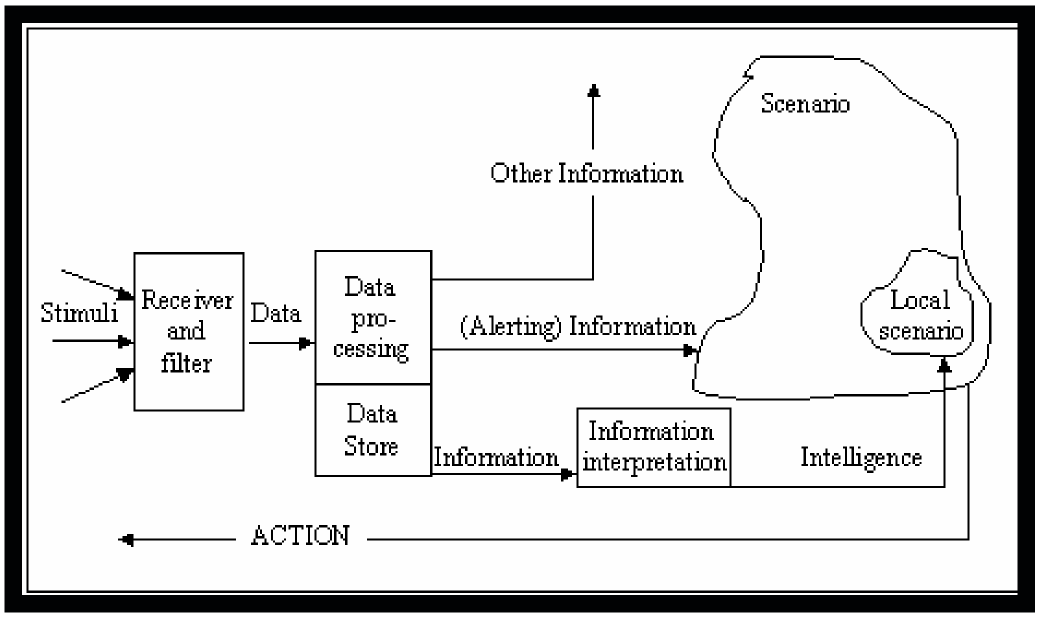
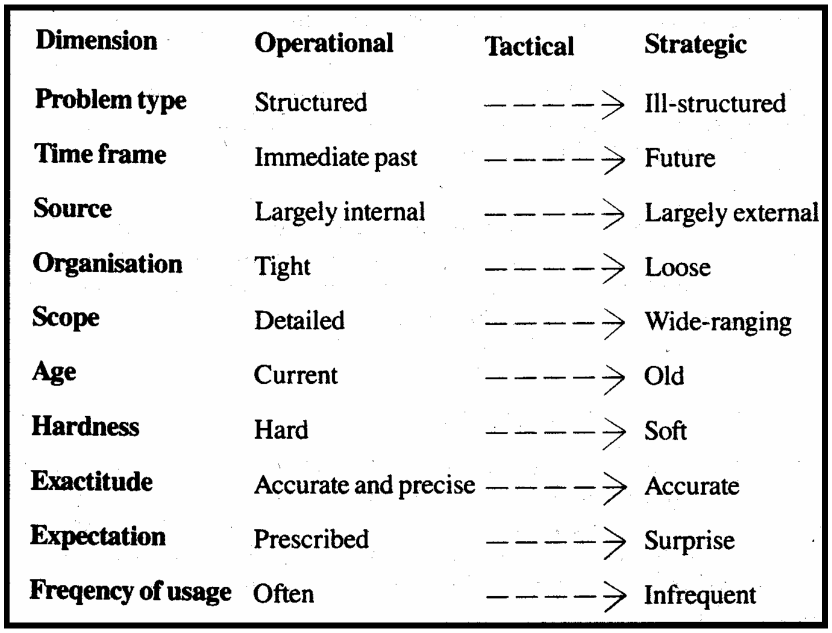


Figure 5-1 A knowledge model of planning and control (Finlay 1994)



Dimension	Operational	Tactical	Strategic
Problem type	Structured	————>	Ill-structured
Time frame	Immediate past	————>	Future
Source	Largely internal	————>	Largely external
Organisation	Tight	————>	Loose
Scope	Detailed	————>	Wide-ranging
Age	Current	————>	Old
Hardness	Hard	————>	Soft
Exactitude	Accurate and precise	————>	Accurate
Expectation	Prescribed	————>	Surprise
Freqency of usage	Often	————>	Infrequent

Figure 5-2 Levels of information Requirements (Finlay 1994)

♦ The design phase

The design phase may involve a great deal of research to determine alternatives or available options that exists (Mallach 1994). Simon (1960) describes this phase as “inventing, developing and analysing possible courses of action”. Turban et al (2001) defines this as the stage where a model is constructed and validated and criteria set for the evaluation of potential solutions that are identified. Setting decision-making criteria would assist decision-makers in optimising a decision. Decision-makers can often not define their decision-making criteria precisely. The criteria however exists, even if the user cannot specify them, and should be stated for the decision in question as an outcome of this phase. Creativity should be encouraged in the choice of alternatives (Mallach 1994).

It is in the design phase where models could be built in order to explore alternative solutions (Finlay 1994). Reality is simplified by using models. It involves activities such as understanding the problem. In this phase, a model of the problem situation is constructed, tested and validated. Modelling involves the conceptualisation of the problem and its abstraction to a quantitative and/or qualitative form. In case of a mathematical model, the dependant and independent variables are identified and their relationships established in the form of equations (Turban 1993).

Decision-making includes zones of certainty, risk and uncertainty (Turban 1993). Decision-making under certainty assumes that complete information is available and that the outcome will be deterministic. This occurs most often with structured problems in short time horizons (up to one year). A decision made under risk is also known as a probabilistic or stochastic decision situation. The different types of simulation (See Paragraph 5.4.2: p91) are given, where several possible outcomes may be considered for each alternative, each with a given probability of occurrence (given or estimated). The degree of assumed risk can be calculated. Expected values are calculated and the best alternative chosen. In cases where decisions are made with uncertainty, there are several outcomes for each alternative. The probability of occurrence of the possible outcome is not known and cannot be estimated. This type is more difficult to evaluate. Modelling involves the decision-maker's attitude toward the risk.

♦ The choice phase

In the choice phase all alternatives are searched, evaluated and one chosen as recommended solution (Simon 1960). The solution is tested "on paper" and once the proposed solution seems feasible, the decision can be implemented (Turban et al 2001). The chosen decision is to be carried out. Only if the recommended solution is successfully implemented, the problem is considered solved. The boundary between this phase and the design phase is frequently unclear because of the stating of the alternatives and the evaluation thereof.

The search of an appropriate course of action that will solve the real problem could span several approaches depending on the criteria set. Normative models may use either the analytical (optimal solution) or a complete exhaustive enumeration model. For descriptive models, a limited number of alternatives are used either blindly or by using heuristics. Analytical techniques use mathematical formulas to derive an optimum solution (used for structured problems) or to predict a certain result. Algorithms may be used to assist in increasing the efficiency of the search (Mallach 1994).

When a description of a desired solution is given, a search may be conducted using a goal and search steps. **Blind search** and **heuristic search** may be considered (Turban 1993). **Blind search** is arbitrary and not guided. Two types exist: **complete enumeration**, in which case all alternatives are considered; and **incomplete**, which continues until a good enough solution is found. Blind search is not practical for large problems, because too many nodes must be visited before a solution is found. The limits on the amount of time and computer storage are practical limits to this option. **Heuristics** are decision rules regarding how a problem should be solved (Turban 1993). Rules of thumb are usually developed as a result of trial-and-error. Step-by-step procedures are followed until a satisfactory solution is found. Such a search is much faster and cheaper than a blind search.

Evaluation is the final step that leads to recommendation. Evaluation can be done at the hand of **multiple goals** and **sensitivity analysis** such as **trial-and-error**, **"what-if"-**analysis and **goal seeking** (Turban 1993). In the evaluation of **multiple goals**, it is often necessary to analyse each alternative and its potential impact on several goals. **Computerised models** are used extensively to support multiple goal decision- making (Turban 1993). **Critical Success Factors** (CSF) is a diagnostic technique to identify factors critical to the evaluation of the recommended solution. Selecting an appropriate DSS software package to assist the decision-maker would require these decision-making skills. **Sensitivity analysis** attempts to help decision-makers when they are not certain about the accuracy or relative importance of information. The decision-maker does not know the impact of the input on the model. Sensitivity analysis may provide **standard quantitative models** such as **linear programming**. It is powerful because of its ability to establish ranges and speedily setting of limits. **Trial-and-error** involves the changing of inputs several times to discover better and better solutions. This experimentation appears as **"what-if"** s and **goal seeking** (Turban 1993).

In a **"what-if"** scenario, the results depend on the input data and model the assessment of uncertain futures. Assuming the appropriate user interface, the impact of input data on the model can be analysed. "What-if" analysis can be executed with **Expert Systems**. A revised recommendation is shown that can be compared with the previous one (Turban 1993). In conventional systems, it is difficult to test "what-if" scenarios because of its prewritten routines. When deciding, the "what-if" and **goal seeking** options are easy to execute and provide many opportunities for flexibility and adaptability (Turban 1993). **Goal seeking** analysis checks the inputs necessary to a desired level of a goal (output). It represents the backward solution approach. Some computer packages include a break-even point in their analysis. **Sensitivity analysis** is important, because it can be used to improve confidence in the model (Turban 1993).

♦ The implementation phase

If the proposed solution seems reasonable, it may be implemented. Successful implementation results in solving the original problem. Failure results in returning to the previous phases (Turban et al 2001). The decision-making process is not as easy as it seems. It is often an iterative process (Figure 2-1: p8) where each of the phases is revisited during the decision making process.

5.1.5 Human decision-making processes

Dean (1991), as referenced by Mallach (1994), categorises the many methods by which decision-makers decide along three dimensions:

- Rationality: the ability to collect and analyse information objectively and make a final choice according to the objectives
- Politically: the ability to make decisions in a group within the organisation's goals and power, when different goals exist among the members of the group: It is characterised by compromise and should aim at a win-win outcome. , and
- Flexibility: the ability to make decisions that break the mould of tradition and structure

Table 5-1 Preferred decision-making techniques for personality types: (Dean 1992 in Mallach 1994)

Decision-making style	**Preferred technique**
Left-brain	Analytical and quantitative techniques
Right-brain	Unstructured and spontaneous procedures concerning the whole rather than its parts such as Brain-storming, emergent trend projection
Accommodating	Has dominant styles but adopt to required the alternate decision-making style
Integrated	Combines left- and right brain, taking advantage of their symbiosis filtering the information analytically (left-brain) while intuition helps decision-makers contend with uncertainty and complexity, constantly verifying the appropriateness of the decision.

As stated before, humans are not entirely rational. Different psychological personality types exist influencing the decision-maker's approach to decisions and their preferred support when deciding (Mallach 1994). Another important factor that determines the type of preferred support is whether decisions are to be made by an individual or a group. Psychological types also affect how well people work together in teams. Sauter (1999) further differentiates between four decision-making styles: left-brain, right brain, accommodating and integrated. Table 5-2 shows the preferred technique for each of the decision-making styles.

Table 5-2 Different decision-making styles and their preferred technique to decision-making (Sauter 1999)

Decision-making style	**Preferred technique**
Left-brain	Analytical and quantitative techniques
Right-brain	Unstructured and spontaneous procedures concerning the whole rather than its parts such as Brain-storming, emergent trend projection
Accommodating	Has dominant styles but adopt to required the alternate decision-making style
Integrated	Combines left- and right brain, taking advantage of their symbiosis filtering the information analytically (left-brain) while intuition helps decision-makers contend with uncertainty and complexity, constantly verifying the appropriateness of the decision.

5.2 Decision-making and models

Modelling simplify reality and help to conceptualise decision alternatives (Mallach 1994).

5.2.1 The benefits of modelling

The benefits or advantages of using modelling are: (Turban et al 2001; Mallach 1994)

- The cost of modelling is much lower that the cost of experimentation conducted with a real system
- Models allow for years of operation to be simulated in seconds of computer time
- Manipulating the model is much easier than manipulating the real system: Experimentation is easier to conduct and does not interfere with the daily operation of the organisation
- The cost of making errors during trial-and-error experimentation is much lower when using models than the real system
- Modelling allows the calculation of risks in specific actions: Experimentation can be done in areas that involve considerable uncertainty.
- Mathematical models allow the analysis of solutions with very large or even an infinite number of alternatives
- Models enhance and reinforce learning and support training
- It is easier to access and manipulate a model when viewing alternatives than applying alternative options to the real world: Several decision options can be evaluated via computer models

- It is easier to collect data from a computer model than from an actual system e.g. production bottlenecks: Data is easily collected as a by-product of a running model. In a real system, the data needs to be specially collected and recorded. , and
- A model compresses time and yields results more quickly than the real world: That which takes years to achieve in the real world can be simulated and made available to decision-makers in minutes

5.2.2 Classifying models

Modelling can be done with various degrees of abstraction and is classified into four groups: **iconic**, **analogue**, **mathematical** and **mental** by Turban et al (2001). **Iconic** (scale) models, the least abstract type, are physical replica of systems. It is usually based on a smaller scale than the original. Examples are models of aeroplanes, cars, bridges or production lines. An **analogue** model does not look like a real system, but behaves like a real system. The shape of the model differs from that of the actual system. Examples include organisational structure charts showing authority or responsibility relationships; maps showing mountains or water using colours; machine or house blueprints; and thermometers.

Mathematical (quantitative) models are the most abstract and may contain four types of variables: result variables, decision variables, uncontrollable variables and intermediate variables. Some systems possess complex behaviour and are best represented as abstract models using mathematics. A **mental** model of a situation provides a description of how a person thinks about a situation. This model includes beliefs, assumptions, relationships and workflow as perceived by an individual. Mental models are important in environmental scanning where it is necessary to determine which information is important. Mental models are subjective and frequently change, so it is difficult to document them. They are not only important for decision making, but also for computer-human interaction.

Mallach (1994) classifies models into **graphical**, **narrative**, **physical** or **symbolic**. A data flow diagram is an example of a **graphical** model. A **narrative** model describes a system in a natural language such as English. A **physical** model is a smaller or idealised representation of the real system. These three models are generally not part of a Decision Support System. The fourth type of model: **symbolic** or mathematical is usually used by Decision Support Systems. It is also called a mathematical model or information-based model.

Table 5-3 (p83) lists some structured management science problems and modelling tools available to solve these problems (Turban et al 2001). Standard models cannot solve managerial problems that are not structured. Such problems are usually classified as tactical or strategic and require the use of a DSS.

Table 5-3 Representative Structured Management Science Problems and Tools (Turban et al 2001)

Problem	Tool
Allocation of resources	Linear and non-linear programming
Project management	PERT, CPM
Inventory control	Inventory management models, simulation
Forecasting results	Forecasting models, regression analysis
Managing waiting lines	Queuing theory, simulation
Transporting and distributing goods	Transportation models
Matching items to each other	Assignment models
Predicting market share and other dynamically orientated situations	Markov chain analysis, dynamic programming, simulation

5.3 Understanding Decision Support Systems

Decision Support Systems exists to help people make decisions. DSS do not make decisions by themselves (Mallach 1994). As stated before DSS attempt to automate several tasks of the decision-making process. Modelling is the core of DSS (Turban et al 2001).

A system is a group of interacting connected components with a purpose (Mallach 1994). The components may be systems in themselves as they interact in fulfilling a common goal or purpose. A system is a collection of objects such as people, resources, concepts and procedures intended to perform an identifiable function or to serve a goal (Turban 1993). DSS is a type of information system that uses and applies several other types of information systems. A DSS may interact with other systems, having information crossing its boundary inward, outward or in both directions. It integrates databases, networks, and programs with user interfaces. A DSS attempts to deal with systems that are fairly open. An open system is very dependent on its environment and/or other systems (Turban 1995). Systems that are fairly open tend to be more complex in nature because of the impact on and from the environment. During analysis, it is necessary to check these impacts (Turban 1995). Virtually every information system has decision support aspects.

5.3.1 Diversity of Decision Support Systems

The concept of intelligence in DSS is important. Finlay (1994) claims that the biggest difference between types of DSS, is between systems that provide information, and those that provide intelligence. Information enables the formation of a better scenario, simply providing the decision-maker with more information only provides the decision-maker with the raw material for intelligence. The decision-maker may need help in the form of intelligence to interpret his scenario. A knowledge model may include data, information, intelligence as well as the process whereby these are formed.

Some DSS have the capability to explore situations that do not yet exist. Analysing such situations require a model or abstraction of reality rather than reality itself. This modelling capability is an important characteristic of a DSS (Turban 1995) and ensures lower cost of experimentation as well as minimising errors and compression of time amongst other advantages. The modelling component of DSS will be discussed in Paragraph 5.4.2 (p91).

5.3.2 Working definitions of DSS

According to Turban (1995), the above definitions do not provide a consistent focus, but each of them rather tries to narrow the populations of DSS systems in different ways. To narrow the population is indeed a proper function of a definition, but the definitions should not be so converse. Collectively they ignore the central issue in DSS namely the support and improvement of decision-making.

After considering the various conceptual frameworks originating from Management Science, Computer Science, Ergonomics, Decision Analysis and Decision Research, Finlay (1994) concludes that DSS should encompass systems that include both management information (MIS) and provide intelligence termed Management Intelligence Systems (MINTS). He specifies that DSS should not be restricted to systems used in person or interactively or concurrently by the decision-maker. He therefore retains his definition of DSS as being a computer-based system that aids the process of decision-making. He adds that the problem structure would be meaningless unless viewed in context of the perceptions of the decision-maker.

Taylor (1999) states that a DSS differs from a Management Information System (MIS) that the manager or decision-maker typically acts as an internal component in a DSS and as an external component in a MIS. The manager interacts with the CBIS to reach a decision through an iterative dialogue process. The iterative capabilities are expressed via the “what-if”-analysis as the manager or decision-maker experiments with the system. Shim et al (2002) states that DSS are typical computer technology solutions that can be used to support complex decision-making and problem solving.

5.3.3 Classifications of Decision Support Systems

♦ The hierarchy of Decision Support Systems

Alter (1980) compared different types or levels of DSS (See Paragraph 2.2.5: p15). **File drawer** systems allow immediate access to data items. When retrieving a specific desired piece of information the decision-maker is in a position to act. This is the simplest DSS and often of great value. **Data analysis** systems allow the manipulation of data. Almost all data in the file drawer category also have a file analysis capability. It is able to perform operations such as conditional retrieval, and elementary arithmetic summaries of selected data. **Analysis information** systems provide access to a series of databases and small models. Analysis is performed by an information system across various files and even external sources.

Accounting models calculate the consequences of planned actions based on accounting definitions. It is a model with no uncertainty, where the calculations of each period only depend on other data of that period. While the accounting model itself does not incorporate any uncertainty, its inputs may not be known precisely. **Representational** models estimate the consequences of actions based on models that are partially non-definitional. This model reflects uncertainty, often noted in individual or collective human behaviour, or represents the dynamic behaviour of systems over time. These models are widely used to forecast the effect of a decision.

Optimisation systems provide guidelines for action by generating the optimal solution consistent with a set of constraints. Alternatives are enumerated or laid along mathematical axes, and the best alternative chosen. **Suggestion** systems perform mechanical work leading to a specific suggested decision for a fairly structured task. Two models are available: a descriptive system and a prescriptive system. A descriptive system may suggest a decision to the decision-maker. This is practical if the decisions are highly structured. Such a DSS is called a suggestion system. A prescriptive model is used to mimic the reasoning process of a human expert in reaching a decision. Human expertise is codified into a reasonable number of rules.

The first three classifications almost entirely focus on their database. They are referred to as **data-oriented**, also called data retrieval systems. The last four categories concentrate more on the model of the business system and less on the database. The databases in these DSS are often small, self-contained and constructed solely for use by the model. Such a DSS is called **model-oriented**, also called Extrapolatory Systems (Finlay 1994). The last two levels: optimisation and suggestion models are often purely based on **process** models. In such cases, it is preferable to refer to them as process-oriented DSS. This focuses the attention on the fact that the DSS mimics the human decision-making process.

Some DSS are further characterised as being an **institutional** rather than an **ad hoc** DSS. An **institutional** DSS is usually used regularly in the organisation and used by more than one person. An **ad hoc** DSS is developed for one-time use only, often used by a single individual. Finlay (1994) bases his classification of DSS on the degree of difficulty to be experienced by the decision-maker in use of the DSS, which in turn is dependant on the degree of complexity and uncertainty the decision-maker perceives, the last being the major factor for classification (Finlay 1994). Figure 5-3 (p86) and Figure 5-4 (p87) show the relationships between DSS.

Finlay (1994) further states that emphasis within DSS changes depending whether the DSS deals with information (using MIS) or with intelligence (employing MINTS). MIS is generally context independent opposed to the use of MINTS that heavily relies on the context. MIS is concerned with efficiency (doing the thing right) while MINTS is concerned with effectiveness (doing the right thing). A comparison between MIS and MINTS with respect to emphasis is given in Table 5-4. (p87) Most of the items are self-exploratory.

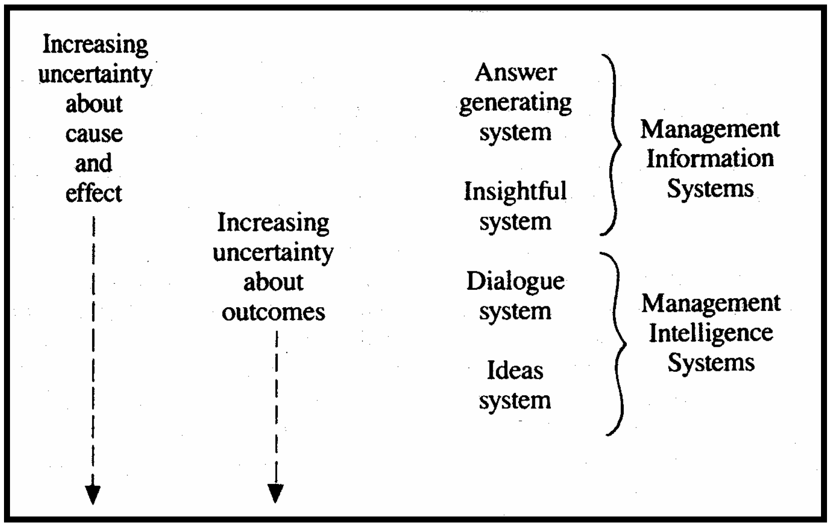

Increasing uncertainty about cause and effect
Increasing uncertainty about outcomes
Answer generating system
Insightful system
Dialogue system
Ideas system
Management Information Systems
Management Intelligence Systems

Figure 5-3 Relationships between Decision Support Systems (Finlay 1994)

MIS may be viewed as two broad types: providing information about the past: facts also termed data retrieval systems and those providing information about the future using extrapolation or relationships based on past data also termed Extrapolatory systems. MINTS may also be viewed as two broad types: the first termed preference determination systems where decision-makers choose between options without formally considering cause and effect and a second type called Scenario development systems. Scenario development systems pose great uncertainty in both the preferred outcomes as well as in the perceived cause and effect relations. In this case, the decision-maker uses the DSS to build his overall view of the world: his scenario. Figure 5-4 (p87) shows the extended relationships between DSS.

♦ Types of management information systems

The first broad type of MIS is data retrieval systems. Data retrieval systems can be divided into three subclasses:

- File Drawer systems: providing rapid, ad-hoc access to pre-structured data
- Data Analysis Systems: providing straight forward analysis, reports and graphical output of predefined situations; and
- Executive Information Systems: a recent evolved type of data retrieval systems that provides selected and summarised information and trend forecasts in a suitable form to senior executives

The second broad type of MIS according to Finlay (1994) is Extrapolatory Systems. Many management science techniques are employed by these systems. . Finlay's (1994) practical approach classifies Extrapolatory Systems into three categories:

- Definitional systems that comprise of definitional relations pre-defined between variables such as systems within the accounting arena

- Causal systems that identify cause-and-effect relationships: possibly derived from statistical data from the past and used with assumed future data values to obtain a result. The system builder needs to be a mathematical modelling expert. Models such as linear programming would be required in the system. , and
- Extrapolatory systems distinguish probabilistic systems that include probabilistic features within the system. The data models need to include probabilistic and/or statistical distributions. Examples of this type include simulation and statistical forecasting. These systems require considerable mathematical and statistical expertise from the DSS builder.

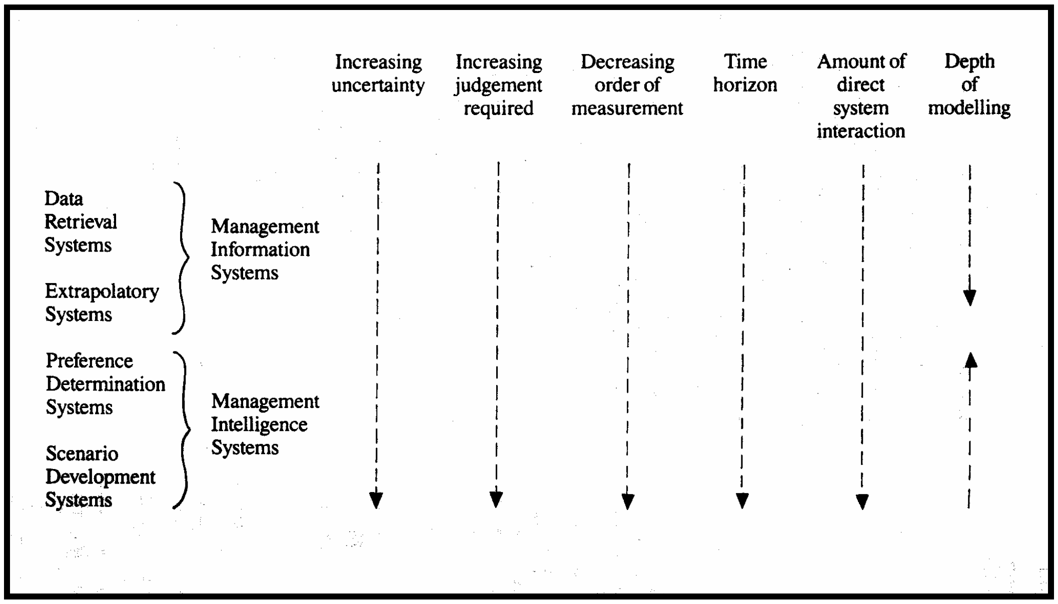

Increasing uncertainty
Increasing judgement required
Decreasing order of measurement
Time horizon
Amount of direct system interaction
Depth of modelling
Data Retrieval Systems
Extrapolatory Systems
Management Information Systems
Preference Determination Systems
Scenario Development Systems
Management Intelligence Systems

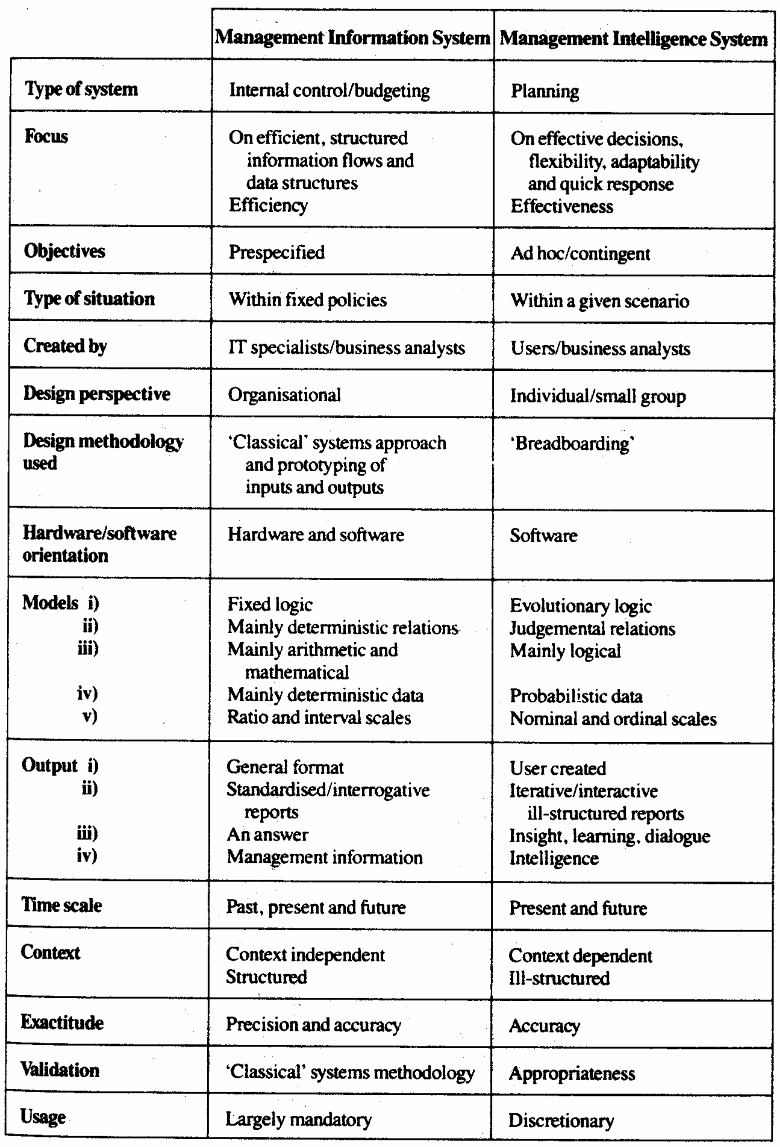
Figure 5-4 Types of DSS (Finlay 1994)

The second type of DSS, MINTS also includes two broad types:

- Preference determination systems, and
- Scenario development systems

The first type requires an exhaustive list of feasible options and associated criteria, and can be categorised based on the predominant decision making technology employed: types based on decision trees and types based on multi-attributed criteria. Decision trees graphically illustrate a series of decisions in a clear and convenient way. It combines probabilistic features with an exhaustive list of possible options and outcomes. Multi-attribute DSS requires several criteria to be specified. The interaction with this criteria and options open to the problem owners will produce a matrix using judgement. Preference methodology exposes the structure of the problem situation. Subjective input from the users and dimensions of uncertainty can be incorporated. The available tools in this DSS type support group decision-making.

Table 5-4 The emphasis associated with types of DSS (Finlay 1994)



	Management Information System	Management Intelligence System
Type of system	Internal control/budgeting	Planning
Focus	On efficient, structured information flows and data structures Efficiency	On effective decisions, flexibility, adaptability and quick response Effectiveness
Objectives	Prespecified	Ad hoc/contingent
Type of situation	Within fixed policies	Within a given scenario
Created by	IT specialists/business analysts	Users/business analysts
Design perspective	Organisational	Individual/small group
Design methodology used	'Classical' systems approach and prototyping of inputs and outputs	'Breadboarding'
Hardware/software orientation	Hardware and software	Software
Models i)	Fixed logic	Evolutionary logic
ii)	Mainly deterministic relations	Judgemental relations
iii)	Mainly arithmetic and mathematical	Mainly logical
iv)	Mainly deterministic data	Probabilistic data
v)	Ratio and interval scales	Nominal and ordinal scales
Output i)	General format	User created
ii)	Standardised/interrogative reports	Iterative/interactive ill-structured reports
iii)	An answer	Insight, learning, dialogue
iv)	Management information	Intelligence
Time scale	Past, present and future	Present and future
Context	Context independent Structured	Context dependent Ill-structured
Exactitude	Precision and accuracy	Accuracy
Validation	'Classical' systems methodology	Appropriateness
Usage	Largely mandatory	Discretionary

The second broad type of MINTS is scenario development systems. The emphasis here is on tools that enhance human cognitive and interactive processes. Two subtypes are identified: cognitive mapping systems suitable to facilitate the codification of the decision-maker's or group of decision-maker's concepts. Analysis of these types identifies where resolution needs to be sought. The second subtype is an idea generation system that supports sessions such as brainstorming to call for relatively uninhibited responses from the participants.

5.3.4 Characteristics and capabilities of DSS

The more beneficial characteristics and capabilities according to Turban (1995) are summarised on the next page (p 89). Most DSS have some of the listed features. The features are graphically represented in Figure 2-3 (p 13).

The characteristics and capabilities of DSS according to Turban (1995) are:

1. DSS support brings together human judgements and computerised information in a semi-structured or unstructured situation. The problem cannot be solved by using a computerised system only.
2. Support is provided for all levels of management
3. Support is provided to individuals as well as to groups. Less structured problems tend to require involvement of various individuals.
4. DSS provides support to several interdependent and/or sequential decisions
5. A DSS supports all levels of decision-making: intelligence, design, choice and implementation
6. A DSS supports a variety of decision-making processes and styles e.g. the individual's decision style
7. A DSS is adaptive over time. DSS should be able to adapt to changing conditions. Basic elements should be capable of being added, changed, combined, rearranged and adjusted to provide fast responses to unexpected situations.
8. A DSS is easy to use specially focussing on non-computer people. Users must feel 'at home' with the system. Ease of use implies an interactive mode.
9. A DSS attempts to improve effectiveness of decision-making including accuracy, timeliness and quality
10. The decision-maker has complete control over all steps of the decision-making process. The system supports but does not replaces the decision-maker. The computer's recommendations can be overwritten at any time.
11. A DSS leads to learning and so initialises a process of developing and improving the DSS
12. A DSS is relatively easy to construct. End users should be able to construct simple systems by themselves.
13. A DSS usually utilises models. The modelling capabilities enable experimenting with different strategies under different configurations to provide new insights and learning. , and
14. An advanced DSS is equipped with a knowledge component to solve difficult problems.

Turban et al (2001) adds another characteristic:

- A DSS allows the easy execution of sensitivity analysis. Sensitivity analysis is the study of the impact changes that one part of the model has on other parts of the model, also called "what-if" analysis. Sensitivity analysis also promotes the discovery of necessary inputs to obtain a desired level of output also called goal seeking.

5.3.5 The benefits of a DSS

In Paragraph 5.2.1 (p81), the benefits of modelling were discussed. Modelling is only one of the components of a DSS (Turban 1993). Most companies use DSS to improve an aspect of their decision-making operation. A DSS assists decision-makers to decide faster with less chance of error, thus improving the decision-maker's efficiency. The benefits most DSS provide are (Turban 1995; Mallach 1994):

- DSS have the ability to support the solutions of complex problems
- DSS can expedite problem solving by providing information to decision-makers about similar decisions made in the past, thus providing increased consistency of similar decisions in the future. The technology of ES can contribute to the consistency of DSS by recommending decisions in an unemotional way. This is one of the many close relationships between Expert Systems and Decision Support Systems.
- DSS provide fast responses to unexpected situations
- DSS have the ability to try several different strategies under different configurations
- DSS provide new insight in learning or training. The ES component can be designed to provide this type of benefit. Most ES offer an interface that allows users to ask why the system made a particular recommendation and receive an answer in non-technical terms. Expert System users, after seeing many of these explanations, will understand the reasoning of experts in reaching the recommendation. Learning has then taken place, because these users would be able to make better decisions without the system.
- DSS facilitate communication: "What-if" analysis can be used to satisfy sceptics and improve teamwork
- DSS facilitate interpersonal communication: One such way is as a tool of persuasion, using to illustrate a particular action to be taken in future, called *offensive* use, or using it to illustrate that a particular action was taken properly in the past, called *defensive* use. When viewed in a broader organisational context, these decisions could influence the group. *GroupWare* is a new form of DSS designed to accommodate the way a group reaches decisions, for instance using electronic mail or bulletin boards. There are also various other ways of electronic conferencing.
- Using DSS as a routine application can eliminate the cost of wrong decisions
- DSS facilitate objective decisions by improving managerial effectiveness and allowing managers to perform a task in less time and/or with less effort. This results in more time for analysis, planning and implementation. , and
- DSS increase organisational control: The DSS can constrain the individual's decision to conform to organisational norms, guidelines or requirements. A level of consistency can be ensured across organisational units. An individual's decisions could also be reported to his manager and then used to assess the productivity of the individual. This aspect has to be used very cautiously, because this might encourage the individual to make "safe" decisions not in the organisation's best interest, or at the worst, damage morale. This use of DSS raises legal and ethical privacy issues.

5.4 Architecture of a DSS

5.4.1 Data management subsystem

Data used by a DSS may be acquired from various sources called: **Internal**, **external** and **personal** (Turban 1995) as well as from **commercial** databases and by **collecting raw data** (Turban et al 2001).

Internal data refer to databases inside the company, some of them maintained by other information systems used by the company. **External** data refer to sources outside the company used around the globe that may range from commercial databases to data collected by sensors and satellites. If relevant, this data may interface with the DSS. In many DSS applications, data come from a **data warehouse**. A data warehouse includes DSS-relevant data extracted from different sources and organised as a relational database (Turban et al 2001). **Personal** data refers to the users own expertise. This includes opinions for example what competitors are likely to do.

Some **external** data flow to an organisation on a regular basis through electronic data interchange (EDI) or via other company-to-company data channels. Much data are also accessible via the Internet in the form of home pages of vendors, clients and competitors. Information can be downloaded from these sites. Online publishers sell access to specialised databases, newspapers, magazines, bibliographies and reports in a timely manner and at a reasonable cost. Several thousand of these services are currently available. **Raw data** can be collected manually or by instruments and sensors. Regardless of how this data are collected, data need to be validated. The quality and integrity of the data are critical for a DSS. Therefore, safeguards on data quality are designed to prevent problems (Turban et al 2001).

Problems observed in large DSS include data not correct, data not timely, data not measured or indexed properly, too much data needed, and needed data non-existent. The data issue should be considered in the planning stage of the system life cycle of the DSS. If too many problems are anticipated, the system should not be undertaken.

If the creation of the DSS involves the creation of a separate DSS database, the DSS builder will need to design and prepare the necessary data. Data may be organised using a relational, hierarchical, network or object orientated database model. The databases may be accessed via networks using technologies like client-server. Many companies develop enterprise-wide databases. Relational Database Management Systems (RDBMS) are better suited for DSS because their records do not contain predefined links to associated records in other files. This provides them with greater flexibility retrieving the data. Another advantage is the use of a standard interface called Structured Query Language (SQL). All the major RDBMS vendors use SQL. It provides database connectivity (ODBC). ODBC is a programming interface that enables applications to access data in a DBMS that uses SQL as a data access standard.

5.4.2 Model management subsystem

An important characteristic of DSS is the inclusion of a modelling capability (Turban 1995). The DSS analysis is executed on a model of reality, rather than reality itself. As stated before, a model is a simplified representation or abstraction of reality, which has advantages such as lower cost of experimentation, compression of time, manipulation of the model itself, lower cost of error, reinforcement of learning and enhanced training. It is limiting to take reality to mean that which presently exist. Some DSS tools exist to explore situations that do not yet exist. To include such tools as models, reality should include that which could come about in future (Finlay 1994).

♦ DSS models

According to Mallach (1994), DSS uses the fourth type of model: a **symbolic** or *mathematical* model also called an *information-based* model (See Paragraph 2.2.6: p15). Reality is represented by data, which can be processed or interpreted as information. The data elements used in this model normally consists of values containing True/False, character strings or numerical values - any type that computers and computer programs can deal with. The symbolic model incorporates procedures and formulas to manipulate the model's data elements. The values of new data elements are derived. The model may use external values received from the database, the user or other information systems.

In DSS, models are used to predict the outcome of decision choices made (Mallach 1994). DSS represent reality by information about reality. A DSS can incorporate several different types of mathematical models. The DSS builder is often faced with the dilemma of which models to include in the DSS and whether to build new models, to use ready-made ones or to modify existing models. Until recently, mathematical modelling formed the core of almost all DSS. Many applications nowadays use logical relationships other than mathematical ones as the basis of their DSS. Mathematical-based DSS however still constitute the majority of applications (Finlay 1994; Turban et al 2001). A tendency exists to complement mathematical modelling using computer graphics in Decision Support Systems. Visual simulation combines iconic, analogue and mathematical modelling.

♦ The model management subsystem of a DSS

The models in the model base can be divided into four categories: **strategic**, **tactical**, **operational** and **model building blocks and subroutines. Strategic** models are used to support top management's strategic planning and tend to be broad in scope to support developing corporate objectives, select plant locations and perform environmental impact analysis. The matching of the organisation with its environment is the primary consideration (Finlay 1994). The raw material for production of intelligence is loosely organised information. Inputs are from a variety of sources such as external databases, external rumours, formal internal reports and formal discussions. High precision of data is generally unobtainable and the concern is accuracy. Strategic planning tends to be periodic with intervals seldom less than months. A well-managed organisation has broad contingency plans to put into practice, should the unexpected happen.

Tactical models are mainly used by middle management to assist in the allocation and control of the organisation's resources such as sales promotion planning and plant layout specification and are usually applicable to one subsystem like the accounting department. **Operational** models support the day-to-day activities of the organisation like approving a loan, supplying advice to students and inventory control. The model normally uses internal data that is accurate and precise. Operational models are buffered from direct influences of the organisation's environment.

In addition, the DSS model base may include **building blocks** such as a random number generator, present value computational routine or regression analysis. These routines may be used on their own or as components of larger models. The models in the model base may be classified by functional areas, such as financial models or production control models, or by discipline, such as statistical or management science models. The number of models in a model base may vary from a few to several hundred. Although some models in the model base are standard, it is frequently necessary to write a model. This can either be done with high-level languages such as COBOL or fourth-generation languages (4GL) or special model languages.

Models, similar to data, need to be managed. Such management is done with the aid of **model base management software** (MBMS) (Turban 1993). The MBMS is capable of interrelating models with the appropriate linkages to the database. This software should ensure that the following exists (Turban 1995):

- Control: The DSS user should have various options of control. The system should support both fully automated as well as the manual selection of models applicable to the intended application. This will enable the user to exercise problem-solving skills at a pace most comfortable for the user. It should be possible for the user to enter subjective information. The information necessary need not be complete. Users should be able use sensitivity analysis such as "what-if" and goal seeking in their experimentation.
- Flexibility: The DSS should be able to switch between modelling approaches during a session
- Feedback: The model should provide sufficient feedback to the user to indicate to the user the state of the problem-solving process at any stage
- Interface: The DSS user should feel comfortable with the specific model. Information provided by the user should be kept to the minimum and optional if possible.
- Redundancy reduction: This can be accomplished by the use of shared models, and
- Increased consistency: Multiple users must be able to use the same model or data

To provide the above the MBMS design must allow the user to

- Access and retrieve existing models
- Exercise and manipulate existing models
- Store existing models, and
- Construct new models

The models are catalogued using a model directory that is similar to a database directory. Model management should control activities such as model execution – controlling the actual running of the model, and model integration – combining the operations of several models, when needed. It must be possible to analyse and interpret the results obtained from such a model.

♦ **Different classifications of models in the model base**

Mallach (1994) also classifies models as being either **system** or **process** models. A mathematical model is an information-based representative of an actual system. The **system** model models the system that needs to be studied, and the **process** model models the process humans follow in making a decision about the system. **System** models would include the correct formulas and algorithms used to solve the problem. It would also include many types of mathematical models to suite the different types of decisions to be made by the DSS.

Process models form the basis of most Expert Systems (Mallach 1994). It would include rules of thumb used by experts in a particular field to determine a specific outcome. When applied, a process model reaches the same conclusion in much fewer steps, and is therefore be more efficient. Process models do not include the deep knowledge necessary to adapt it easily to unforeseen circumstances because its rules of thumb assume certain relationships between the present and the new. When any unforeseen circumstances arise, a system model would adapt the formula values to the given situation and provide the correct answer.

Process models can be classified as being **descriptive, prescriptive** or **predictive** depending on the source of their input data. Both types are based on the same model. A **descriptive** model describes what a system did in the past or what the system is doing in the present. A **prescriptive** or predictive model describes what the modeller expects to do in the future using assumed or statistically described future data. **Descriptive** models are being classified as **static** or **dynamic** (Mallach 1994; Turban 1995). Static models show values of a system in balance. Dynamic models show a system's changes, with cause-and-effect relationships, over a period. A series of cause-and-effect connects one period with the next. A static model may model either a static system or a snapshot of a dynamic system. A **static** model describes relationships among data values at any instant of time. Static models take a single time interval, long or short in duration. Any data element remains stable once calculated or processed. The result will not change unless the decision-maker changes the input.

In a **dynamic** model data values change over time. The change is essential in the model and is time dependant. What happens at one instant in time affects the modelled system at future instants in time. It shows trends and patterns over time. A collection of static models can appear to show quantity varying over time. The difference lies in the presentation of the data over time. If more than one time period of results is shown and formulas exist in the model to effect growth or change over the period, the model is a dynamic one. For some decisions the changing of values does not matter and therefore a static model or series of static modelling would suffice. Dynamic models mimic the behaviour of a system over time, allowing the examination of behaviour and thus focussing on the optimising of the system (Mallach 1994). Dynamic models can be further subdivided into **continuous** and **discreet-event** models. Simple continuous models can be studied via calculus and differential equations. Complex systems are solved easier numerically. Continuous models progress smoothly from one value to the next.

Discreet-event models model systems in which the state of the system changes instantaneously from one value to another. A precise instant exists at which a value changes such as an order that arrives or a product that is shipped. Discreet-event models suit most business planning needs and are classified further into **deterministic** and **stochastic** models. A model is classified as **deterministic** if its outputs are fixed for a given set of inputs and **stochastic** if the outputs reflect an element of statistically defined **uncertainty**. The familiar spreadsheet is a useful tool in dealing with **deterministic** models. A **stochastic** model's output, for a given set of inputs, varies randomly over a range of possible outcomes. When this model is applied several times, many different answers will be obtained clustered around a mean. The variance of this mean will depend on the steps taken during the modelling process.

The decision to characterise a model as deterministic or stochastic depends on what is put inside the model and what is put outside the model (Mallach 1994). To manage **uncertainty** in the model, it is necessary to agree on the boundary of the system. **Accounting** models and **representation** models correspond closely to deterministic and stochastic models. **Certainty** models yield optimum solutions. Many financial models are constructed under assumed certainty (Turban 1995). If insufficient information is available a problem can be treated as an **uncertain** problem. If more information is acquired the problem can be treated under calculated or assumed **risk**. Decision-making under risk assumes that the decision-maker has knowledge of some of the probabilities that are involved with the decision variables (Lawrence & Pasternack 2002). Several techniques can be used to deal with risk analysis such as **probabilistic simulation**, **time (in-) dependant simulation** and **visual simulation**.

Probabilistic simulation assigns probabilities to one ore more independent variables. Two subcategories exist: **discrete** distribution where discrete values are assigned a probability and **continuous** distribution where the values are statistically distributed with a mean and a deviation. Probabilistic simulation is conducted with the aid of a technique called Monte Carlo (Turban 1995).

♦ The seven groups of DSS models

Many DSS incorporate models of various types. Models of a given type tend to have many characteristics in common. Mallach (1994) compares DSS models according to the following:

- System versus Process
- Static versus dynamic
- Continuous versus discreet-event, and
- Deterministic versus stochastic

Turban (1995) also assigns the above characteristics to DSS models, but categorised the models further into seven groups:

- Complete enumeration – few alternatives
- Optimisation via algorithm
- Optimisation via analytical formula

- Simulation
- Heuristics
- Other descriptive models, and
- Prescriptive models

Models have procedures and formulas to manipulate their data elements or to derive other data elements. An external database or other external data sources may be used to derive these values. A useful characteristic of these models is the fact that the model remains valid when data changes (Mallach 1994). The following paragraphs provide summarised explanations of the seven groups as given by Turban (1995).

▪ **Complete enumeration**

A finite and small number of alternatives are modelled by decision analysis. Two types are distinguished: **single goal** and **multiple goals**. Decision tables or decision trees are two techniques used in single goal decision analysis. There are quite a few techniques in multiple goals. The objective in both cases is to select the best alternative after listing them and assessing each one's forecasted contribution towards their goal(s). The decision table is a mathematical model. Two factors that can affect the model are uncertainty and risk. In uncertainty the probabilities of each state of nature is unknown. In the case of risk, these probabilities are known. Sufficient information is gathered about the case to assume the risk. An alternative to the decision table is the decision tree. A decision tree shows the relationships of the problem and can deal with situations that are more complex.

▪ **Optimisation via mathematical algorithm**

Mathematical programming provides a relatively unbiased approach in solving the problem. Linear programming is the most known mathematical technique used in optimisation. Other programming and network models that exist are non-linear programming, goal programming and distribution problems. The objective is to find the best solution from a large or infinite number of alternatives. Suitable problems usually display the following characteristics and necessitate some assumptions:

- A limited number of economic resources are available for allocation (for example labour, capital, machines, and water)
- The resources are used in the production of products or services
- Two or more ways exist to user the resources, each called a solution program
- Each product or service yields a return in terms of a stated goal
- Several constraints limit or restrict the allocation
- The returns from the various resources are measured with a comparable common unit
- The return of a specific allocation is independent from other allocations
- The total return is the sum of the return of the different services or products
- All data are known with certainty, and
- The resources are to be used in the most economic manner

▪ **Optimisation using analytical formulas**

Several inventory models exist in this category. The objective here is to find the best solution using a single line formula. The technique: "Optimisation using a mathematical algorithm", may be used too. Several other inventory models exist including statistics, financial analysis, and accounting and management science models.

Statistical and financial functions are built into many DSS generators (See Paragraph 3.1.5: p35). There are several hundred management science packages ranging from inventory to project management. Several DSS generators include optimisation and simulation capabilities. A DSS generator can invoke these models by issuing a single command. A DSS generator can also interface with powerful quantitative methods in stand-alone packages. Pre-programmed quantitative models can be accessed via templates. Some of these models can be building blocks for other models e.g. the regression model can be part of the forecasting model that supports the financial planning model. SQRT calculates the square root and can be part of the inventory model.

▪ **Simulation**

Simulation is a technique for conducting experiments. Simulation usually imitates reality opposed to models that represent reality (Turban 1995). This means there are fewer simplifications of reality in simulation models than in other models. Output values of the decision given specific input variables are observed. Simulation is a descriptive rather than a normative tool. There is no automatic search for an optimal solution. Simulation predicts or describes a system under certain circumstances. The best one amongst the alternatives can be selected. The process often exists of many repetitions of an experiment. Simulation is usually called for if the problem to be solved is too complex for numerical optimisation techniques. An accurate simulation model requires an intimate knowledge of the problem. The model is built from the decision-maker's perspective and in his decision structure. The simulation model is built for one particular problem and will, typically, not solve any other problem.

The decision-maker can experiment with different input variables to determine which are important. He can experiment with different alternatives to determine which variable set is the best. It allows the decision-maker to ask "what-if" type of questions. Decision-makers that employ a trial-and-error approach to problem solving, can do it faster and cheaper with less risk. A great amount of time compression can be attained, giving the decision-maker an idea of the long-term effects of various policies in a matter of minutes.

Some disadvantages of simulation are:

- An optimal solution cannot be guaranteed
- Constructing a simulation model is frequently a slow and costly process
- Solutions and inferences from a simulation study are often not transferable to other problems, and
- Simulation is sometimes chosen instead of analytical solutions that can yield optimal results

The methodology of simulation consists of a number of steps using a real world problem as the input:

- Problem definition: The real-world problem is examined and classified. Reasons why simulation is necessary should be specified. The system's boundaries are specified.
- Construction of a simulation model: Gather the necessary data, and optionally use a flowchart to describe the process. Write the computer program.
- Testing and validating the model: The model must imitate the system
- Design the experiments: If the model has been proven valid, the experiment is designed. Accuracy and cost is two important contradictory objectives.
- Conducting the experiment: The rules may be terminated and the results presented where applicable
- Evaluation of the results: The final step is the evaluation and analysis of the results. Statistical tools and sensitivity analysis may be used, and
- Implementation: The implementation of simulation results involves the same issues as any other implementation

There are several types of simulation. They include:

- Probabilistic simulation: One or more independent variables are probabilistic. Two sub-categories exist: discrete distributions and continuous distributions. Discrete distributions involve a limited number of events or variables that can take on a finite number of values. Continuous distributions have an unlimited number of possible events that follow density functions such as the normal distribution.
- Time dependent and time independent simulation: Time independent refers to a situation where it is not important to know when the event occurred. Other problems may require that the precise time of arrival is known e.g. waiting line problems. , and
- Visual simulation: It represents the results in a graphical way and is one of the more successful new developments in problem solving

▪ **Heuristic programming**

The simulation process may be lengthy, complex and even inaccurate. In such situations it is sometimes possible to arrive at a satisfactory solution rapidly and less expensive by using heuristics. Heuristics is the finding of a "good enough" solution of a complex problem using rules. Techniques include heuristic programming and Expert Systems. Heuristics are most often used for solving ill-structured problems, but they can also be useful in providing satisfactory solutions to well-structured complex problems. Alternatives can be generated using heuristics. Generating alternatives is done manually in most DSS; however, this activity can be automated. It is necessary to predict the future outcome of each alternative (Turban 1993). The main difficulty in using heuristics is that they are not as general as algorithms. Therefore, they can only be used for the specific situation for which they were intended. The result may still be a poor solution.

It is better to employ a heuristic model when:

- The input data are inexact or limited
- Reality is so complex that the optimisation model is oversimplified
- A reliable exact model is not available
- The computation time of optimisation is too excessive
- It is possible to improve the efficiency of the optimisation process by using heuristics
- Problems are being solved frequently and use up a lot of computer time
- Complex problems that are not economical for optimisation, and
- When symbolic rather than numerical processing is involved

Some advantages of heuristic models are that they are easier to understand and therefore easier to implement. They can produce multiple solutions that can help in the training of people to be creative and set up rules for complex problems, save formulation time and save computer execution time.

▪ **Other descriptive models**

Other descriptive models involve non-quantitative models expressed in terms of rules or formulas. These models may be done separately or in combination with quantitative models such as financial models and waiting lines to find solutions to "what-if" scenarios by the use of a formula.

▪ **Prescriptive models**

The main objective of this type of model is to predict the future for a given scenario. Representative techniques include Markov-analysis and forecasting models. Forecasting predicts the values of model variables at some time in the future. Two types of forecasts are distinguished by Turban (1995): **short run** (up to one year), where forecasts are used in deterministic models and **long run** where forecasts are used in both deterministic and probabilistic models.

Often multiple factors, most of which are uncontrollable, are involved in the difficult task of forecasting. Formal forecasting methods can be divided into four categories:

- **Judgement** models, based on subjective estimates and expert opinion rather than hard data used where historical data are limited or non-existent
- **Counting** methods, involving a kind of experimentation or surveys of the market based on hard historical data commonly divided between time-series and causal methods
- T**ime-series analysis** in which historical data are used to predict future events, and
- **Association or causal methods**, that includes more variables than time-series methods and use sophisticated statistical methods in presenting the alternatives

5.4.3 Dialogue subsystem or user interface

This subsystem is the key to successful use of the DSS. Various interface modes exist which determine how information is displayed and used. Dialogue styles, dialogue modes and conversation formats all contribute to the ease of use of the DSS. Styles such as menu interaction, command language, question and answer, form interaction, natural language and object manipulation are techniques used to assist the dialogue subsystem of a DSS. Graphics are especially important for problem solving, because it helps decision-makers visualise data, relationships and summaries.

Graphical User Interfaces (GUIs) are direct manipulation systems in which the user has direct control of the visible objects such as icons or buttons to replace complex command syntax. A wide variety of graphics such as: text - in the form of titles and descriptions, time-series charts, bar and pie charts, scatter diagrams - showing the relationship between two variables, two- or three-dimensional maps, room layouts, hierarchy charts, sequence charts, motion graphics and desktop publishing are all viable options when designing user interfaces. A subset of the above options can be used to enhance the specific DSS, whether the DSS produces reports or just visualise problems and potential solutions.

User interfaces can be enriched by the use of interactive multimedia. One new class of multimedia is called hypermedia and includes several types of media such as text, graphics, audio and video elements as tools to navigate knowledge and data and capture results. Associated knowledge can be linked with hypertext allowing the user to control navigation to various components. Virtual reality, presenting a 3-D user interface, offers rich opportunities for powerful interactions. The implementation of 3-D is difficult and expensive. In virtual reality, a person "believes that what he or she is doing is real", even though it is artificially created (Turban et al 2001).

Several problems could be minimised or even eliminated if the user could communicate to the computer using the user's own native language. The computer should be smart enough to interpret this input regardless of its format. Natural Language Processing (NLP) is part of Artificial Intelligence (AI). Two techniques are used in NLP: key work search (pattern matching) and complex language processing (syntactical and semantic analysis). Keyword analysis search for selected words or phrases. Once found, the program responds with the associated set of responses. Language processing is still an emerging technology. Because communication involves language in context, it is difficult for the computer to understand what is exactly meant when free-format commands are issued.

Heterogeneity exists among users and usage patterns of DSS (Turban et al 2001). Different users have different cognitive preferences and abilities, different ways of arriving at a decision and therefore need different types of support in making a final decision (Paragraphs 2.2.1: p7 and 5.1.5: p80). Some users are skilled in using DSS and need a different user interface than other users that are less experienced.

5.4.4 Knowledge management support for decision making

C.W. Holsapple (2001) in his editorial summary observes the following with respect to knowledge management support for decision-making:

- Decision-making is a knowledge-intensive activity with knowledge as its raw materials, and work-in-progress, by-products and finished goods

- Computer-based DSS employ various KM techniques to represent and process knowledge of interest to decision-makers, including descriptive knowledge (e.g. data, information), procedural knowledge (e.g. algorithms), and reasoning knowledge (e.g. rules)
- Knowledge management is concerned with the representation and processing of knowledge by humans, machines, organisations and societies
- An aim of knowledge management is to ensure that the right knowledge is available on the right forms to the right entities at the right times for the right costs, and
- Proficiency in knowledge management is increasingly important to the competitiveness of decision-makers as we rapidly move into the global knowledge society

ES form the basis of the knowledge component and are discussed in detail in Paragraphs 2.2.6 (p15), 2.3 (p19) and Chapter 6 (p107).

5.5 Integrating the DSS in the organisation

The degree of user involvement, the general implementation strategies and the organisational politics may differ among organisations and DSS. A plan that guides the integration process should be developed that consists of education, installation and evaluation (Sprague & Carlson 1982). The purpose of the integration process is to reduce the risk of failure. Alter (1980) identified eight risk factors. When present, these factors increase the probability of DSS failure (See Table 5-5: p101). The integration processes can help reduce each risk. Evolving prototypes can be installed on the Intranet for the use of the company. Constant evaluation to determine the value of the DSS should determine the life span of the DSS.

User education involves training on how to operate the DSS. It involves training the user to:

- Solve problems using the DSS
- Adapt the DSS to new problems, and
- Use special features of the DSS

Table 5-5 DSS risk factors and integration process (Sprague & Carlson 1982)

Risk Factor	**Integration Process to help reduce risk**
Nonexistent or unwilling user	Education, evaluation
Multiple users or implementers	Education, installation
Disappearing users, implementers or maintainers	Education, installation, evaluation
Inability to specify purpose or usage pattern	Education, evaluation
Inability to predict and cushion impact	Evaluation, education
Loss or lack of support	Education, installation, evaluation
Lack of experience with DSS	Education, installation, evaluation
Technical problems and cost effectiveness	Installation, evaluation

5.6 A more comprehensive view of organisational decision-making

With the increasing popularity and use of the Internet and telecommunications technologies, it can be expected that organisations will increasingly become more global, complex and connected. Mitroff & Listone (1993) as referenced by Courtney (2001) state that managers will require radically different reasoning skills facing such an environment. Much broader cultural, organisational, personal, ethical and aesthetic factors need to be considered in decision-making. Decision Support Systems should be capable of handling softer information in a broader context than the mathematical models and knowledge-based systems of the past.

This is an enormous challenge, but imperative if DSS is to remain relevant in the future. Figure 5-5 presents a more comprehensive view of organisational decision-making (Courtney 2001). The perspective of the problem formulation phase need to include multiple and varied perspectives. The problem formulation should include organisational (O), personal (P) and technical (T) perspectives. In addition, ethical and aesthetic factors should be considered too.

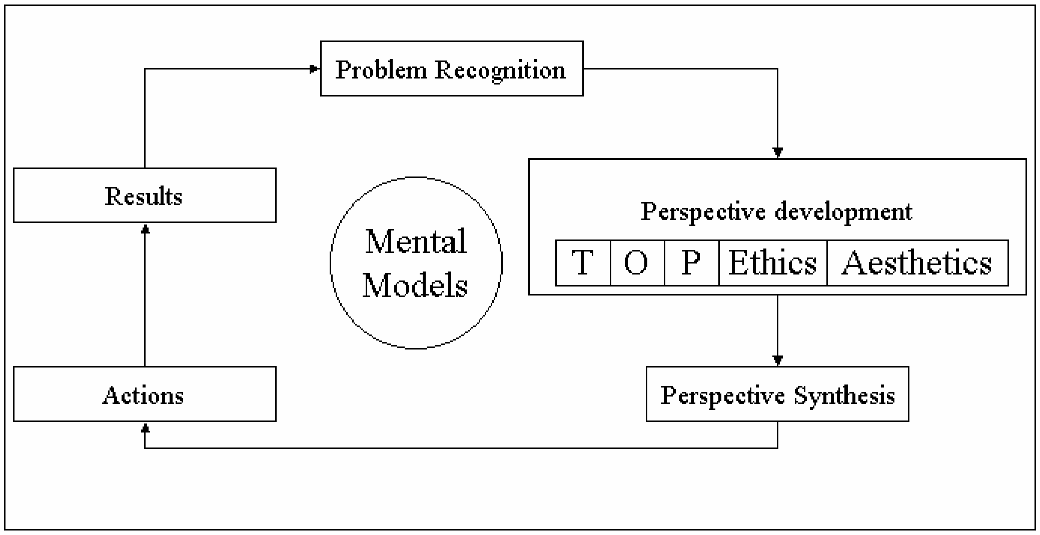


Figure 5-5 A new decision paradigm for DSS (Courtney 2001)

At the heart of defining the problem through to solving the problem, lays the perspective of the problem. The technical perspective has dominated the process in the past and involves the development of the databases and models of the DSS. Individuals are complex and varied in decision-making styles. The organisational and personal perspectives may not be quantifiable, especially the ethical and aesthetic concerns. Given the same external information in an unstructured complex situation, no two people may come to the same conclusion (Courtney 2001). The individual's background, training, experience, values and ethics may differ and thus cause the individual to reach a unique decision. Broader forms of analysis such as group sessions may become even more appropriate in the future.

5.7 The future of DSS

Shim et al (2002) calls the database capabilities, the modelling functionality and the interface design components of the DSS the classic tool design components of DSS. They add tools such as data warehouses, on-line analytical processing, data mining and web-based DSS as tool developments for future DSS. These tools together with collaborative support systems, virtual teams, and knowledge management optimisation-based DSS and active decision support are important topics in the development of the DSS concept for the next millennium.

5.7.1 Data warehouses

A data warehouse is a subject-orientated, integrated, time-variant, non-volatile collection of data used in support of management decision-making processes (Inmon & Hackathorn 1994). Barquin (1996) states: "Data warehousing is the process whereby organisations extract meaning from their informational assets through the use of data warehouses".

5.7.2 On-line analytical processing (OLAP)

OLAP is a set of tools that is been developed to provide users with multi-dimensional views of their data and to easily analyse the data using a graphical interface. Multi-dimensional analysis is often used as a synonym for OLAP (Hoffer, Prescott & McFadden 2002). OLAP tools help analyse the historical data in a data warehouse. OLAP tools enables analysts, managers and executives to gain insight into data through fast, consistent, interactive access to a variety of possible views of information that has been transformed from raw data to reflect the real dimensionality of the enterprise as understood by the user.

5.7.3 Data mining

Data mining is knowledge discovery or database exploration using a sophisticated blend of techniques from traditional statistics, artificial intelligence and computer graphics (Weldon 1996). Data mining tools find patterns in data and infer rules from them. Given databases of sufficient size and quality, data mining technology can generate new business opportunities by providing automated prediction of trends and behaviours and provide automated discovery of previously unknown patterns in the data of the database (Turban et al 2000).

Data miners use several tools and techniques such as neural computing, intelligent agents and association analysis (Turban et al 2000). **Neural computing** is a machine learning approach by which historical data can be examined for patterns. Agents sense the environment and act autonomously without human intervention ranging from those with no intelligence (software agents) to learning agents that exhibit some intelligent behaviour. **Intelligent and software agents** (Turban et al 2000) are computer programs that help the user save significant time when they

- Conduct routine tasks
- Search and retrieve information (Search, match and filter using search engines)

- Support decision making, and
- Act as domain experts

Association analysis is an approach that uses a specialised set of algorithms to sort through large data sets and expresses statistical rules among items (Turban et al 2000).

5.7.4 Web-based DSS

Web-based DSS are computerised systems that deliver decision support information or decision support tools to a manager or business analyst using a web browser such as Netscape or Internet Explorer.

♦ Decision support using the Intranet

The Internet is a public and global communication network that provides direct connectivity to anyone over a Local Area Network (LAN) or Internet Service Provider (ISP). Users need effective and efficient search engines to navigate the vast scope of public and advertising information provided in the Internet (Turban et al 2000). The Intranet is a corporate LAN that uses Internet technology and is secure behind a company's firewalls. A firewall is a network node consisting of both hardware and software that isolates a private network from a public network.

Sridhar (1998) views the capability of the Intranet to facilitate access to distributed information as one of the strengths of decision support using this mechanism. The technology underlying the Intranet is fast evolving as new web browsers, supporting collaborative multimedia applications, are developed. Intranets are becoming more popular. He presents illustrative examples and gives classifications of Intranet-based decision support types in order to understand the role of the Intranet in decision support. A major concern, using the Intranet, is the possibility of compromising security, and this needs to be addressed. In his taxonomy, Sridhar (1998) considers the various dimensions of the major decision support components: data, modelling and the user interface.

The dimensions Sridhar (1998) considers for the component are as follows:

- Data
  - Data source: Data may be obtained from internal or external to the organisation
  - Nature of data: Data may be structured or unstructured , and
  - Type of access: Data may be accessed locally or remotely
- Modelling
  - Model source: Models may be obtained internal or external from the organisation
  - Type of access: Models may be accessed locally or remotely, and
  - Type of execution: Models may be executed locally or remotely from the user's computer, and
- User Interface
  - Number of users: Single or multiple users may be involved in the decision process

- Type of use: Multiple users involved in an synchronous or asynchronous use, and
- Place of use: Multiple users involved in different locations (distributed) or in the same location (non-distributed).

The browser may access data and execute models remotely or locally using a remote or local interface at a remote or local place. Sridhar (1998) states the benefits of using an Intranet DSS are greater in terms of facilitating remote access rather than local access. The use of intelligent agents may further facilitate quick retrieval of desired information or models. When using a distributed environment, the communication links between the components of the DSS may experience delays depending on network traffic and bandwidth availability. He expects that these delays, because of the rapid advances in computer and network technology, would decrease and this is therefore unlikely to be perceived as a concern.

♦ Customer Decision Support Systems (CDSS)

The Customer Decision Support System is a Web-based, marketing model that establishes a link between a firm and its customers and provides assistance to the decision-making process (O'Keefe & McEachern 1998). They classify CDSS as a second-generation, web-based, marketing system that includes electronic publishing (first generation marketing systems), but also takes advantage of interfaces and gateways to databases and models to provide richer systems. Decision support is provided for a part of the customer decision process such as information search, product evaluation, purchasing of products and after-purchase evaluation. The service is provided for existing and potential customers using electronic agents to generate needs such as notifications of new publications or price changes such as used by online bookstores.

♦ Open interchange of decision models

A need exists for customers or users to share models and data. In order to achieve this, a modelling tool or environment has to know about other tools and environments in the market and build bridges to them in order for customers to share models and data. In many cases, bridges may not exist or models may not scale well (Kim 2001).

Using an exchange standard in open architecture could be designed to improve the shortcomings of a closed architecture. Each vendor need only implement the standard to leverage access to the other tools. A standard syntax creating data and exchanging data structures is important for this type of integration. XML (Extensible Mark-up Language) provides such a framework and allows the participation of sharing in a web-enabled collaborative environment. XML is a subset of the standard generalised mark-up language (SGML) (Kim 2001). SGML allows documents to be self-describing, through the specification of tag sets and structured relationships between the tags. This specification is referred to as the Document Type Definition (DTD). XML is simple, extensible, interoperable and open. XML's rigid set of rules assists humans and machines to read XML documents. XML are built on a core set of basic nested structures needing very little implementation effort. XML allows users to create their own DTD and by allowing the addition of standards that adds styles linking and referencing abilities to the core XML set of capabilities. XML can be used on a wide variety of platforms and interpreted with a wide variety of tools. It is freely available on the Web and anyone can parse a well-formed XML document and validate it if the DTD is provided (Kim 2001).

Kim (2001) proposes a structured mark-up language called OOSML (Object-Oriented Structured Mark-up Language) based on a conceptual modelling framework: Object Oriented Structured Modelling (OOSM), an object-orientated extension of Structured Modelling (SM). OOSML can be used for the representation and management of decision support models within the DSS community. OOSML is an XML application that supports object-orientated concepts such as object-oriented modular structure, models as entities and specialisation using structured mark-ups. The structure of the decision models is defined using XML tags and this causes the finding, manipulating and using of models much easier. The structured delivery of data enables open interchange between servers and clients and potentially between servlets themselves. Users can manipulate and analyse OOSML models using the syntax and semantics of OOSML. Models can be downloaded from a web server using a web browser. OOSML-aware agents need to be implemented to integrate the models with environments.

5.7.5 Summary

In future new hardware and software technologies will make DSS both easier to develop and easier to use.

Chapter 6 - Expert Systems and knowledge acquisition

An expert system's major objective is to provide expert advice and knowledge in specialised situations (Turban 1995). ES is a sub-discipline of AI (Turban et al 2001). For an ES to reason, provide explanations and give advice, it needs to process and store knowledge. Knowledge in the form of facts and rules are kept in a knowledge base.

6.1 Expert System definitions

ES definitions by Turban et al (2001) and Olson & Courtney (1992) were presented in Paragraph 2.3.1 (p19). Raggad & Gargano (1999) state: "ES emerged, as a branch of AI, from the effort of AI researchers to develop computer programs that could reason as humans". Goodall (1985) distinguishes two approaches when defining an expert system:

- The human/AI oriented approach, and
- The technology oriented approach

Goodall's (1985) *AI approach* defines an expert system as a computer system that performs functions similar to those normally performed by a human expert. He extends this definition to include the nature of an expert system, in other words, he includes how ES behaves, in the *technology approach*:

> "An expert system is a computer system that uses a representation of human expertise in a specialist domain in order to perform functions similar to those normally performed by a human expert in that domain".

In formulating the above definition, Goodall (1985) focused on the implementation of two types of ES:

- ES performing at a suitable 'high' expert level, and
- ES that performs tasks which no human expert has, using perhaps a very complex set of rules to control a chemical process at a speed and in circumstances (e.g. temperature, time, location) which no human could possibly match. Such a system will most surely outperform the best human expert.

The value of ES as a sub-discipline of AI (Turban et al 2001), can be appreciated more when comparing AI to natural intelligence and conventional programs. Kaplan (1984) as referred to by Turban et al (2001), states several commercial advantages of **AI over natural intelligence**:

- AI is more permanent. Employees take knowledge with them when they leave their place of employment, or worse: an employee may even forget his knowledge. As long as the computer systems and programs remain unchanged, the knowledge remains the same.

- AI offers ease of duplication and dissemination. Transferring knowledge between persons may be a lengthy process. Some expertise is never transferred. Knowledge embodied in a computer system can be easily moved to another computer.
- AI can be less expensive than natural intelligence. Buying computer services can cost less than having a human performing the same task.
- AI is a computer technology and thus consistent and thorough. Natural intelligence is erratic because people are erratic and perform inconsistently. , and
- AI can be documented. Decisions made by a computer can easily be documented by tracing the activities of the system. Natural intelligence is difficult to document. A conclusion reached may be difficult to be recreated or recalled later. Some of the assumptions or even the reasoning process may be forgotten.

Natural intelligence has several advantages over AI:

- Natural intelligence is creative: In an AI system, knowledge has to be carefully constructed, while knowledge is inherent to all human beings
- Natural intelligence uses sensory experienced directly: Users using AI, use sensory experiences indirectly
- Natural intelligence enables people to recognise relationships between artefacts, sense qualities and spot patterns that explain how various items interrelate, and
- A wide context of experiences is used when natural intelligence solves an individual problem, while AI typically gains its power by having a very narrow focus

ES differs from conventional programs too. Raggad & Gargano (1999) and Turban et al (2001) summarise the **characteristics of conventional programs** as follows:

- Conventional Computer Based Information Systems (CBIS) *processes data and information* as a direct conceptual resource
- Conventional programs are based on *algorithms* or mathematical formulas and sequential procedures that lead to a solution using data to solve problems
- The *objective* of CBIS is to create and efficient solution by designing an efficient algorithm
- CBIS are characterised by *repetitive* processing
- The processing system or *control* mechanism of the system and the knowledge of the subject are *intertwined* as part of the sequence of instructions
- *Knowledge of the subject is built into the processing mechanism* or control of the system. The subject knowledge or business rules form part of the sequence of instructions. All business rules are part of the program code. The knowledge is stored as control embedded in the program instructions. The control is the sequence in which the transactions are performed. The programmer fixes these steps in advance. It is called the algorithm of the program.
- To change the business rules or domain *knowledge* a programmer needs to edit the program and insert or change the statements that represent the information *in the correct place*

- The program or system used *does not explain which rules* were used during processing of data to produce the output
- The program is specialised to deal with specific information. A *complete system* of programs and input are *necessary* to achieve something useful.
- *One missing essential step causes the program(s) to malfunction*. The data that are processed should be complete and certain to achieve an appropriate result or valid output.
- *Error detection* of the domain knowledge is performed *when executing the program* and noticing faulty output
- CBIS frequently uses *algorithms* in its search approach ,and
- CBIS has no *reasoning* capability except for a fixed process in the algorithm of the program, and
- *Updating and maintenance of data and information are usually difficult*

When with conventional programs, **the characteristics of ES** are:

- The main conceptual source of ES is *knowledge*. ES can expand to include a knowledge acquisition component that processes data and information into rules. Data and information are indirect conceptual resources.
- ES does not represent its knowledge in algorithmic-like procedures. AI is based on *symbolic* processing of knowledge. Knowledge is represented by symbols such as letters, words or numbers that represent objects, processes and their relationships. The knowledge base contains facts and concepts and relationships among them. Processes are used to manipulate the symbols in the knowledge base to generate advice or recommendations to solve problems.
- The *objective* of ES is to create an efficient knowledge representation to support the inference mechanism
- ES keeps the *control separate* from the rules: the inference engine is responsible for most of the control. The knowledge base contains the rules or knowledge.
- *Knowledge is expressed as data* and not encoded in the control of the ES. This has the advantage that knowledge can be returned to the user as explanations of the ES conclusions.
- The *order* of the rules in the knowledge base *does not matter*
- ES *provides an explanation facility*
- The rules that comprise the knowledge may be built bit-by-bit, a rule at a time. ES can *tolerate imperfect, incomplete or uncertain business rules* or domain knowledge.
- The ES will *produce results* even if the knowledge base is incomplete and uncertain. This is because of the separation of the knowledge and control components.
- ES frequently uses *heuristics* in its search approach
- ES has a *reasoning* capability, and
- *Maintenance and updating of data are relatively easy*. Changes can be made in self-contained modules.

6.1.1 When to use an Expert System

ES is appropriate for a limited number of problem domains. The task should be well bounded and should not include a large number of event combinations. A general rule of thumb is that if the average employee can solve the problem in a few minutes it is not worth solving it by an expert system.
ES is appropriate (Raggad & Gargano 1999; Mallach 1994):

- If great costs are involved when making the decision
- When applied to repetitive problem domains, the task must be performed often
- When a big difference exists between the best solution and the worst solution
- When test data is easy available to test and validate the ES
- When there is a general agreement on the system's conclusions. Errors in the input should be tolerated in the system output. Experts must agree on the solutions.
- When recognised expertise is available and a necessity during the development of the ES
- If the problem is clearly specifiable in the area of expertise i.e. what the system is supposed to do before it is done, the desired system can be developed
- If the problem is identifiable with a human expert, that is based on human expertise. Experts must perform the task substantially better than non-experts. The human expertise must be scarce.
- If the problem domain is well bounded: for example defining criteria to determine the subject matter within the system from the matter outside of the system: The task is reasonably stable and within an acceptable narrow domain.
- If solving the problem is based on knowledge rather than common sense reasoning: ES is based on knowledge and naïve systems on reasoning.
- The solution has to be possible, justifiable and concisely generated (using only a few hundred rules), and
- The knowledge has to be of a cognitive style and independent of common sense. Physical activities are ruled out.

Goodall (1985) provides a comprehensive list of suitable types of tasks appropriate using ES in solving problems that includes:

- Interpretation
- Prediction
- Diagnosis
- Debugging
- Design
- Planning
- Monitoring
- Instruction, and
- Control

Turban et al (2001) calls the above-mentioned types the generic categories of Expert Systems. Most ES systems will include one of the above problem areas as a dominant characteristic, but may include some aspects of the others.

6.1.2 Advantages and disadvantages of using ES

Developing an ES is time consuming for the expert, whose knowledge is captured, the knowledge workers, who will eventually make use of the system; as well as the programmers, who will be involved in the building of the ES. ES is limited in the sense that it focuses on a narrow problem domain and can not therefore be applied outside of the specific problem domain. For this reason it involves intensive development cost.

According to Goodall (1985), Mallach (1994), Turban (1995) and Turban et al (2001) the advantages of using an expert system (a system with intelligent reasoning) above a conventional information system (for example a system without intelligent reasoning) are:

- It improves the quality of the system. The program will function even with incomplete and uncertain data from the very first time it executes previously non-programmable tasks. This is an advantage in domains where knowledge changes frequently. It provides consistent advice whilst reducing the error rate. More rules can be added to the existing knowledge as knowledge about the domain increases.
- It can handle uncertainties expressed as probabilities. The way it is handled depends on the inference engine of the particular ES.
- It explains the logic behind its recommendations, making the knowledge explicit
- It can be used as a training vehicle for users who lack the expertise
- It provides monetary savings once implemented. ES does not cost money when not being used. Costs can be cut as the human expert's availability is not needed all the time.
- Experts are freed enabling them to focus on tasks requiring their expertise
- It can codify and preserve knowledge of the specific problem domain. It offers a solution to scarce expertise where few experts exist for a task or where the expert is about to retire or leave the job.
- It increases programmer productivity. The author of the knowledge base will have access to more advanced techniques and tools than the system's eventual user. Separating the expert's knowledge into modular rules reduces the change of development errors and improves the maintainability of the system.
- It can discover new knowledge
- It can provide increased output and productivity. ES works faster than humans do. Increased outputs mean fewer workers and reduced costs. ES can be replicated as needed.
- It eliminates the need for expensive equipment used by human experts for monitoring and control
- It makes knowledge and information accessible, and

- It can outperform human experts because of the fact that ES:
    - Make fewer errors
    - Do not become tired or bored and never sleeps.
    - Will not overlook a solution
    - Can handle large volumes of data
    - Can respond more rapidly
    - Can function in hostile environments such as deep-sea drillings and reactor control, and
    - When the knowledge of several experts is integrated, it can outperform any one expert

Another advantage of ES is that it is a tool for manipulating knowledge. It can:

- Discover new knowledge e.g. discovery of a new prime number
- Codify and explicitly state old knowledge by forcing the author of the knowledge base to formalise the expertise in rules or other structures. This resultant knowledge can be used by anyone who needs it. Tools are available to codify knowledge. One such tool is Teiresias.
- Assist in teaching once knowledge is codified, and
- Help analyse knowledge

Mallach (1994), Turban (1995) and Turban et al (2001) give the drawbacks of ES as:

- The domain of expertise is usually narrow. An ES is designed for a specific purpose and is not useful for another purpose.
- ES cannot apply common sense, only its rules
- Experts do not realise when they reach their limits. They may recommend inappropriate actions. The experts' performances "falls off a cliff" rather than degrade gradually.
- ES may be costly to develop because of the time of human experts and other people involved in the process
- Knowledge is not always readily available
- Expertise is hard to extract from humans
- Lack of trust by end-users may be a barrier to ES use
- Knowledge transfer is subject to a host of perceptual and judgemental biases
- The work of rare and expensive knowledge engineers is required, and
- Most experts have independent means of checking whether their conclusions are reasonable

6.2 Architecture of ES

6.2.1 The process of ES

Transferring expertise from an expert to the computer and then to the user involves four activities: knowledge acquisition, knowledge representation, knowledge inference and knowledge transfer (Turban et al 2001). Knowledge is acquired from experts and other documented sources and then represented or organised as rules or frames. These rules or frames are electronically stored as a body of knowledge or a knowledge base. This knowledge base forms one of the distinct parts of an ES. The reasoning mechanism that draws conclusions from the knowledge attained in the knowledge base and the knowledge obtained from the user forms the other distinct part of the ES. This reasoning mechanism is known as the inference engine. The inference engine results in advice or recommendations for novices. When understood by the novice, possibly by accessing the explanation given by the ES, knowledge has been transferred to the user. Figure 2-6 (p21) shows the different components of an ES and their interaction.

6.2.2 The components of ES

An expert system consists of various major components (Goodall 1985):

- A dialogue structure
- A knowledge base, and
- An inference engine

Turban et al (2001) adds:

- A blackboard
- An explanation sub-system, and
- A knowledge refining system

The **user interface** provides the conversation of the system with the user. It is responsible for posing the questions to the user, reading the user's reply and explaining the rules used to reach a conclusion. A survey of typical ES suggested that on average (Goodall 1985):

- Eight percent of the code is used for the inference engine
- 22% of the code is used for setting the knowledge base, and
- 44% of the code is used conversing with the user

The **explanation capability** is one of the most important features of an ES and it realises in the user interface. Turban et al (2001) views the explanation facility as a separate component of the ES that can trace and explain the ES' behaviour (See Figure 2-6: p21). The body of knowledge (also called the **knowledge base**) and the **inference engine**, the reasoning mechanism of the ES, form the two fundamental parts of the ES. These two subsystems provide the ES' main functionality. The **knowledge base** represents domain specific knowledge in a specific form. The **inference engine** deduces facts or draws conclusions from the knowledge base based on the user input and the facts from the knowledge base and/or other sources. The inference engine and the knowledge base exist as two separate modules of the ES that work closely together.

♦ The user interface

The knowledge worker (also called the user or decision-maker) gains access to the ES via the **dialogue structure** provided by user interface. The knowledge worker provides the ES with input and receives the system's explanation of how it reached its conclusion. Output to the user usually comprises of taking text from the knowledge base and slotting them into a few predefined sentence formats.

When allowing the user to use **natural language** in obtaining input, a misunderstanding could result because of the ambiguous characteristic of natural language. The use of natural language in the dialogue with the user relies on the meaning of the user's utterances. The meaning of what exactly is meant lies in examining the surrounding context and is prone to be misunderstood. The naïve user is likely, by the system's ability to read natural language, to be fooled into thinking that the ES absolutely understands what he (the user) means. The best way to solve this problem is for the ES to generate all possible interpretations of any sentence that could be ambiguous, feed it back to the user, and enquire of the user which interpretation he (the user) really meant. The easiest way to understand the user is not to allow conversation to be in any natural language, but rather via a **predefined syntax**, requesting **preformatted specific input**. Questions posed to user and explanations of conclusions that are reached can be given by taking the text provided by the author of the knowledge base and slotting it into a few predefined sentence formats.

♦ The knowledge base

The power and effectiveness of the ES is equal to the quality of the knowledge it contains. The knowledge has to cope with high degrees of complexity and apply the best judgement. The acquiring of expert knowledge is crucial and involves the gathering of information about a domain usually from an expert. This information is incorporated in a computer program stored as a knowledge base. Obtaining knowledge from humans can be a difficult task. Hayes-Roth et al (1983), as referenced by Turban (1993) and Klein & Methlie (1995), views **knowledge acquisition** (See Paragraph 6.4: p122) or **knowledge engineering** as being composed of five stages:

- Identification or definition of the problem and the major characteristics of the problem
- Conceptualisation or the choice of an appropriate representation medium such as rules and the acquiring of the knowledge associated with the certainty of the knowledge
- Designing of an architecture or deciding the formalism or acquisition methodology e.g. rules and the deciding the process extracting knowledge from the experts
- Implementation, building or the programming of knowledge and the development of a prototype, and
- Testing and refining of the knowledge base by subjecting it to examples, examining the validity of the knowledge.

The *methods* of knowledge acquisition can be divided into *manual*, *semi-automated* and *automated*. The primary manual approach is interviewing, ranging from completely unstructured to highly structured interviews. Automated knowledge acquisition uses an induction system with case histories and examples as input to derive the knowledge base. It eliminates the role of the knowledge engineer and expert (Turban 1993). Automatic knowledge acquisition is also known as machine learning. Knowledge collected must be analysed and coded prior to its representation in the computer. Automated knowledge acquisition methods are easier to validate and verify. On average fewer knowledge mistakes occur when acquiring knowledge from multiple experts because of the enhanced synergy among experts. Difficulties such as meeting time scheduling and resulting compromising solutions in opinion conflicts may arise.

Knowledge representation is important and crucially affects the ease and speed with which the inference engine can use it. Knowledge representation implies a systematic means of encoding what an expert knows about a knowledge domain in an appropriate medium (Goodall 1985). A number of knowledge acquisition techniques have been developed. Turban (1993) discussed a variety of techniques. The selection of a technique depends on the types of knowledge that should be retained in the knowledge base. Knowledge can be classified as **surface knowledge**, to put declarative and procedural knowledge into heuristics to solve a problem quickly; or **deep knowledge**, which involves fundamental knowledge of domain including the definition, axioms, general laws, principles and causal relationships upheld by the knowledge. Surface knowledge is the basis for most common ES using production rules. Production rules are widely used and quite efficient in diagnosis problems. They are used to encode rules of thumb also called heuristics or knowledge used by humans (Turban 1995). Knowledge can be formulated using formal theories or normative models.

Knowledge representation is the systematic means of encoding knowledge of the human expert in an appropriate medium (Beynon-Davies 1991). Knowledge can be represented as:

- Predicate calculus or formal logic
- Business applications in the form of production rules
- Semantic networks, which organise knowledge through nodes in a graph rather than data structure and represent relationships between the facts by links between the nodes, and
- Frames or structured objects that use data structures to store all structural knowledge of a specific object in one place

Logic itself is not a way for computers to store knowledge, but proves to be a vital tool to think about how computers store knowledge. Logic is part of mathematics and can be used in various forms to reason about the correctness of computational representation and inference (Goodall 1985). The forms of logic include:

- Programming languages such as PROLOG (programming in Logic)
- Pro-positional logic or calculus that consist of building blocks such as elementary sentences, joined by ‘and’, ‘or’ and ‘not’. The internal structure of the elementary sentence is irrelevant. , and
- Predicate logic with its basic building block objects and relations such as “is-a” and “has-a” between them to build statements. The relations is called predicates and deals with the logical operators “and”, “or” and “not”.

The above provide a theory to formalise the study of reasoning, determining valid knowledge representations. It is used to prove the correctness or determine that certain types of inference are correct or incorrect.

The forms of knowledge representation often used in ES are:

- Rules
- Semantic nets
- Frames, and
- Cases

Rules are often called production rules and the program that reasons with rules a production system, especially when the inference is data-directed forward chaining. Most ES represent knowledge as rules and therefore the knowledge base is often referred to as the rule base. The reason for its popularity is that almost every piece of knowledge can be written as a rule. Many ES exist that requires rules as input and tempts knowledge workers to express knowledge as such. Rules are natural and the only way to codify some knowledge. Rules are a simulation of the cognitive behaviour of human experts. It represents knowledge, but also represents a model of actual human behaviour. Rules are easy for a human expert to read, understand and maintain. If the knowledge is expressed as data and not encoded in the program's control mechanism, it can be returned to the user in the form of explanations. Production rules involve simple syntax that is flexible and easy to understand. They are quite efficient in diagnosing problems of the form:

- if (condition)
- then (conclusion)

Each production rule in a knowledge base implements a chunk of expertise and when fed to the inference engine, as part of a set, should synergistically yield a better result than any of the rules individually. In reality, rules are highly independent and adding a new rule may contradict other rules or cause other rules to be revised. Rule systems can broadly be classified into simple, all rules on the same level and available to every search cycle, and structured rule-base systems where searches are limited to segments of the rule base, thus improving the efficiency or the search. A rule set is a named collection of individual rules pertaining to a distinct aspect of a problem and helps in comprehending and maintaining the rule base. This structure is a kind of meta-knowledge that is imposed on the knowledge base. Each sub problem could have its own rule set (Klein & Methlie 1995).

Semantic Nets is a popular and easy-to-understand way of representing non-rule knowledge. Semantic networks organise knowledge through nodes in a graph rather than a data structure. Links or arcs present relationships between the named nodes. The links or arcs represent relationships such as is-a, has, is, own, needs and reflects the association between concepts. An ES that stores knowledge as a semantic net incorporates an inference engine devoted to operations like inheritance. Such an inference engine will consist of two parts. One part will deal with rules by forward chaining, backward chaining or some other method. The other part will handle net operations matching relevant links in the net to deduce facts.

Objects can be described by a number of features or attributes, many of which stay constant from one instance to the next. A **frame** is a piece of structured data about typical characteristics of an object, act or event. The knowledge is more descriptive than procedural. Similar to rules, frames must be able to deal with uncertainty and missing values. Frames may have default values and slot-filling procedures associated with the slots, to cope with missing values. Frames enable reasoning about object features such as inheritance and the occurrence of exceptions. The reasoning process starts by identifying a frame as applicable to the current situation. Matching the set of frames against the facts available selects an appropriate frame. The use of frames is relevant to non-monotonic logic. Non-monotonic reasoning expresses reasoning with default attributes.

Case-based reasoning is a process that uses similar problems to solve the current problem. It consists of two steps:

- Find those cases in memory that solved problems similar to the current problem, and
- Adapt previous solutions to fit the current problem

The case library forms an extra important component in case-based reasoning. The inference cycle using CBR consists of:

- Retrieving solutions
- Adapting solutions, and
- Testing solutions

The critical step is to find and retrieve a relevant case from the case library. Cases are stored using indexes. The stored case contains a solution, which is then adapted by modifying the parameters of the old problem to suit the new situation resulting in a proposed solution. The solution is tested and if found successful, added to the case library (Klein & Methlie 1995). Knowledge acquisition is easier in case-based reasoning because of the granularity of the knowledge. Knowledge is presented in precedent or resultant cases.

Beyond the knowledge representation language (rules, semantic nets, frames, cases), the knowledge engineer needs further aids such as tools to edit the knowledge base; inference tracers to assist in error detection; and analytical tools to find, update and consistently check the represented knowledge or attributes (Klein & Methlie 1995).

♦ The inference engine

The mechanism that performs the search and reasoning in rule-based systems is called the **inference engine**. The inference engine is activated when the user initiates the consultation session. The inference engine finds the rules that match the given facts, selects which rule to execute and executes the rule by adding the deduced fact to the Working Memory (WM). The inference engine uses pattern matching to select the qualifying rules. The choice of which rule to fire is done by conflict resolution. The most commonly used conflict resolution strategy is the first found strategy. The first applicable rule is executed (Klein & Methlie 1995) or fired by applying rule deduction or using formal logic. A new fact is concluded and added to the working memory and new patterns found that match the new fact. This sequence of steps and the linking of facts and patterns and rules are known as chaining (Klein & Methlie 1995).

The inference engine deduces facts or draws conclusions from the knowledge base based on the user input and the facts from the knowledge base and/or other sources. Three basic techniques are identified when inferring facts or conclusions from the knowledge base:

- Forward chaining
- Backward chaining, and
- Hybrid chaining: using both forward and backward chaining

New knowledge can be inferred from the existing knowledge in the knowledge base. The specification of how the reasoning is done is the core of the ES. This programming task is independent of the subject knowledge in the knowledge base component. The reasoning method treats facts as arbitrary symbols. The same inferencing method can be used by different kinds of knowledge (Goodall 1985). Various algorithms can accomplish this task. One such algorithm used by ES, is the **Rete Algorithm** (See Paragraph 4.2.9: p45)

Forward chaining match the current state with the rules' antecedents or conditions in the knowledge base. If the condition is true, the inference engine adds the conclusion to the list of known facts. Known facts imply other facts. This technique is also known as data driven. It can be very inefficient, especially if many rule conditions match the data provided by the user. The data provided by the user is examined and new knowledge is concluded or asserted and responded to.

The second technique, **backward chaining**, also known as goal driven or directed, is more efficient if the number of goals is limited and involves the acquiring of information from the user in the form of questions to draw specific conclusions. This style is used to confirm diagnostic tasks and works backwards from what needs to be proved to the facts that might affect it. It is a goal-directed backward chaining of rules to try to prove a hypothesis made by the system. Sometimes stating the conclusions and trying to prove them are unhelpful because none of the multiple goals exist. If this is the case, forward chaining should rather be considered

When a large problem domain is involved, a more efficient program is yielded when the two techniques are used in combination - **hybrid** chaining. To model the strategies used by a human expert, an ES has to employ complex inference that almost always include both forward and backward chaining with potentially many variations. One inference engine will not suit all possible tasks solved by an ES. A typical rule-based system represents heuristic knowledge: that is short cuts in the reasoning procedure. This process uses shallow knowledge.

Model-based reasoning is based on fundamental or deep knowledge of the problem domain. The relationships modelled could be either structural or behavioural. Structural relationship models are used for advice systems and behavioural relationship models for Decision Support Systems that do scenario analysis such as a "what-if" analysis. The behavioural model consists of causal relationships that can be quantitative or qualitative (Klein & Methlie 1995). In model-based reasoning, the reasoning follows the fundamental relationships, the cause-and-effect paths using a deeper knowledge. The reasoning styles are directed by what need to be proved (goal directed) versus by what data are available (data directed). When pursuing the goal directed style and several rules exist that prove some conclusion the rule with the best change of proving it, is selected. This is done by conflict resolution, for example a more specific rule is chosen above a less specific rule.

To simplify the process each rule can be assigned a number to reflect how useful it is. The number could be a guide to select one rule from several that proves the same conclusion. Rules that deal with uncertainty or imperfect data or are referred to as approximate rules, are called **probabilistic reasoning**. An integral part of the working of the ES is how the inference engine determines when to enquire the appropriate information from the user. Probabilistic reasoning is a method used by an ES to draw inferences from the domain and problem knowledge where the knowledge or its implementations or both are less than certain.

Three ways exist to represent uncertain knowledge:

- A single subjective probability is assigned to the proposition, possibly derived using Bayesian inference. Bayes' theorem provides a way of computing the probability of a particular event given some set of observations. The main fact here is not the derivation of the value, but the association of the specific proposition with a single value. Criticism against this approach include that the single value will not reveal much about the rules precision and yet another states that this single value combines the evidence for and against this proposition without reflecting how much of each there is. An alternative approach is to provide an independent measure of belief and an independent measure of unbelief that can be combined into a single certainty factor.
- Fuzzy logic is another technique that can handle inexact data, and
- A final method is the use of a few quantitative qualifiers to express the levels of probability. Possible values may include unlikely, possible, likely, impossible or certain

The knowledge base can include meta-rules. A meta-rule is a rule about the rules and potentially affects the sequence of all other rules called. Meta-rules make the knowledge base more complex and more difficult to understand e.g. the use of different rule sets. Two types of **facts** exist: those received from the user in the form of input data (known facts) and those deduced from the input data (inferred from primitive data). Knowledge can also be obtained from other sources such as databases (DB). Much research is connected to Expert Systems and databases to refrain from transcribing some of the information into rules.

In acquiring information from databases, difficulties may be experienced such as:

- Connected DB may not be able to provide answers to all the queries ES would like to put to it. The ES needs to distinguish between questions viable to put to a DB and that not.
- The ES and the DB may not be able to run in the same extensive memory needed for an ES and a DBMS on one machine, and
- Queries to the DB need to be phrased in a special DB query language. The ES should be able to formulate the query and interpret the reply.

The rules connect items of knowledge about a problem. Some ES concentrate more on facts and less on rules. Many pieces of information may have more than one attribute. Such knowledge is said to be structured, compound or multi-attributed. All ES languages should provide for it in some way.

▪ **Inferencing using production rules associated with objects**

The following describes the interaction between the knowledge base and the inference engine:
A production system typically consists of a rule base (knowledge base) and a rule interpreter (inference engine) that decides when to apply the rules to the data, goals and intermediate results in the working memory. The Working Memory (WM) holds the Object-Attribute Values (OAV) used to drive or fire the rules. The OAV triggers some rules in the WM by satisfying their conditions.

▪ **Inferencing using frames**

Non-monotonic reasoning expresses reasoning with default attributes designed for frames. The process retracts rules when proven wrong. Non-monotonic reasoning is a mathematical tool by which default information holds until this information becomes inapplicable.

♦ Blackboard subsystem

Blackboards are not an alternative to frames and nets for storing knowledge. It is rather a complex method used in complex systems to record choices, guesses and decisions about what to do next. It is used in situations where more than one source of information is active. All knowledge sources have access to a shared database. It involves an architecture that allows the independent knowledge sources to communicate through the central device: the blackboard.

When a knowledge source is activated, it examines the current state of the blackboard and implies its knowledge to either create a new hypothesis or change an existing one. The scheduler determines which knowledge source can contribute to the solution in the blackboard next. Hearsay-III - a speech-understanding project; ExperTax - a tax planning system, and FINSIM are examples of blackboard systems (Klein & Methlie 1995). Turban et al (2001) describes the blackboard as a kind of database that an area of working memory set aside for the description of the current problem, the input data and the intermediate results.

♦ An explanation sub-system

A by-product of the inference engine is that it is able to provide a trace of rules (subject knowledge) used and thus provide an **explanation** of its conclusions. This explanation can be presented to the ES user in the dialogue component. The explanation sub-system provides an area where the behaviour of the ES can be accounted for (Turban et al 2001). It provides answers to questions such as:

- How was a certain conclusion reached?
- Why was a certain question asked?
- What is the plan to reach the solution? , and
- Why was a certain alternative rejected?

♦ A knowledge refining system

Human experts can analyse their own performance, learn from it and improve it for future consultations. Having the program evaluate the reasons for success or failure and learning from it would greatly enhance ES. This functionality is not available in many commercial Expert Systems yet, but is being developed in experimental ES (Turban et al 2001).

6.2.3 The process of ES

The process of ES (See Figure 2-6: p21) can be divided into two parts (Turban et al 2001): the system **development** process in which the ES is constructed and the **consultation** process which describes how advice is rendered to the users. The **development** process starts with the knowledge engineer acquiring the knowledge from the expert. This acquired knowledge is then programmed in the knowledge base as facts about the subject and knowledge relationships in terms of if-then rules.

The **consultation** process involves the user who starts the process by requiring advice from the ES. The ES provides a conclusion and explanation, using its inference engine. The engine searches the knowledge base for an appropriate action by matching the input the user provides to the facts and rules in the knowledge base. To execute this task the inference engine uses a work area or temporary databases called the blackboard. All intermediate results are stored here. An explanation of actions taken by the inference engine can be provided to the user. Finally, the knowledge in the knowledge base may be refined by repetitive consultations. This knowledge refining system is not available in many Expert Systems now, but is being developed as experimental ES.

6.3 ES and expert shells

An expert shell is an ES without the domain-specific knowledge (Beynon-Davies 1991). Mallach (1994) refers to a shell as a pre-packaged inference engine. Construction time can be reduced by using an ES shell. Advantages of using a shell include (Beynon-Davies 1991):

- An expert shell can assist in rapid prototyping. The programmer needs only construct the knowledge component. The structure of the ES exists, enabling the developers to concentrate on the substantive content rather than the form of the ES. A shell helps to reduce the level of skill required by the developers by effectively supplying some of the required expertise.

- A shell imposes prior structure, enabling developers to concentrate on substantive content rather than form, and
- A shell reduces the required skill level of Expert Systems builders

6.4 Knowledge acquisition and knowledge base construction

The knowledge that is contained in the system determines the effectiveness of the ES (See Paragraph 2.3.4: p22). Knowledge engineering may contain the following steps (See Paragraph 6.2.2: p113):

- Definition of the problem
- Acquisition of expert knowledge
- Design of an architecture, and
- Testing and refining of the program

Acquiring expert knowledge is a crucial component of knowledge engineering. This phase is difficult and time consuming. It is the process of gathering the relevant information about a domain, usually from an expert. Usually a computer program is compiled to incorporate all the knowledge of the domain. A number of knowledge acquisition techniques have been developed. As mentioned before knowledge can be classified as surface knowledge or deep knowledge. Surface knowledge involves the presentation of declarative and procedural knowledge into heuristics to quickly solve the problem. Surface knowledge is the basis for most ES and uses production rules. Deep knowledge involves fundamental knowledge of the domain including definitions, axioms, general laws, principles and causal relationships of the domain.

The human expert, from whom the knowledge is acquired, needs to be a person that can effectively cope with the problem domain. Selection of the person is based on reputation. The expert should be available and willing to co-operate in the process of developing the ES especially when acquiring the knowledge and testing the system. The expert must be able to communicate his expertise.

Validity of knowledge in any system is crucial. If the knowledge inferred proves to be faulty or erroneous in any way, the wrong decisions made by the experts will be made more speedily than when the process was a manual one. The sources of errors could include:

- Not much is known about the problem domain; too little knowledge is available. To solve the situation more domain expertise need to be obtained. , and/or
- The environment of the decision might be changing in which case the captured knowledge needs to be maintained

Relational database systems manage knowledge in the form of facts but are quite different when compared to knowledge bases. To understand ES and knowledge base construction, Beynon-Davies (1991) compares the two technologies.

6.4.1 Comparing a Knowledge-based System (or an ES) to a Relational Database System

Beynon-Davies (1991) states that knowledge can be represented as both facts and rules. Facts declare relationships between objects. Rules define the process by which new facts are generated from old facts. Rules are mechanisms to manage the information explosion in the attempt to represent reality. Beynon-Davies (1991) declares a database as a collection of structured data shareable between different parts of an organisation's information system, having properties such as program-data independence, data integration, data integrity and separate logical and physical views of the same data using a specific underlying data model. Hoffer et al (2002) calls these different models the database architecture. The different styles of database management characterised by the way data are defined and structured are called the database architecture. A database management system supports one of five architectures: Hierarchical, network, relational, object-orientated or multi-dimensional database models. Beynon-Davies (1991) compares knowledge-based systems to the relational model of databases when discussing the integration of expert and database technology. The relational model consists of tables representing the information stored on objects or entities and the relationships between them.

Important characteristics of relational databases are:

- *Data* are stored in a component called a *database*
- To use one data structure called a *table* or relation
- To handle large *collections of facts* representing declarative knowledge or relationships between objects
- Rules can be represented in databases only in a *declarative* manner, storing values in a separate table reporting the relationship. This may result in *many tables* to represent all the different relationships that exist between objects. To limit the number of tables some knowledge is *embedded* in the high-level language programs that accesses the database called the *front-end.*
- Rules as *procedural* knowledge are represented as constraints or additional tables mainly to govern *data integrity*
- A *high-level language* code interacting with the database contains and processes the procedural knowledge
- A database is a *limited Knowledge-based System*. It is a subset of what is normally meant by knowledge.
- A RDBMS (Relational Database Management System) is *program-data independent*. Data can be maintained independently of the programs that use it.
- Addition of a rule involves modifying the *structure* of the database (addition of a table structure) – not a simple matter
- *No inferencing* except when explicitly programmed
- Data must be valid. Rules govern the logical consistence of the database.

- A database uses a tool called the *database management system* (DMBS) to build database systems
- The DBMS (Database Management System) is built upon a *data model* or architecture, and
- The *database management system* (DBMS) is an integral part of the database and handles manipulation of large collections of facts

In contrast with DBMS, the characteristics of ES are:

- *Knowledge* (facts and rules) are kept as a separate component called a *knowledge base*
- The data structure depends on the knowledge representation chosen e.g. *production rules*, *structured objects* or *frames* and *predicate calculus* and others
- ES handle large *collections of rules* representing procedural or derivation of new facts from existing facts
- Rules as *declarative* knowledge are added independently from other rules or the process that uses it
- Rules as *procedural* knowledge are stored as data mainly used by the process to *derive new facts* from existing facts
- Knowledge can be represented as both *facts and rules*
- The knowledge base's *general-purpose processor* activates the procedural knowledge
- ES are *knowledge-process independent*. Rules are maintained independently of the process itself.
- Addition of a rule involves adding or deleting a *condition* – a relatively simple matter
- ES separate *inference* and knowledge. Once the inference process is in place, the knowledge worker can focus on the knowledge itself. ES is sometimes referred to as a knowledge-based system because knowledge is kept in a separate component as data.
- Knowledge may be incomplete
- ES use a tool called an *expert shell* to build Expert Systems
- An expert shell is built on some *formalism for knowledge presentation*, and
- The inference mechanism as an *integral part* of ES handles the activation of large collections of rules

A database system is an example of a limited Knowledge-based System (Beynon-Davies 1991). A knowledge base handles large amount of rules (procedural and declarative knowledge), while a database handles large amounts of facts or declarative knowledge. The procedural knowledge of a database is found in the associated external high-level language interacting with the database. The procedural knowledge (rules) are stored as data in a Knowledge-based System and activated by an inference process integral to the Knowledge-based System.

6.5 Summary

Expert Systems are based on two fundamental principles: the appropriate representation of domain knowledge and the control of the domain knowledge. A standalone expert system stores a few facts and relies on the user to pass information to the system when needed to perform certain deductions. The combination of Expert Systems and database technology could benefit both technologies. Models have been developed for data base systems that would benefit the storage of knowledge in a knowledge base. DBMS technology could contribute Expert Systems by giving them the ability to access large collections of facts and also apply features such as concurrency control, data security and optimised access to knowledge base items (Beynon-Davies 1991).

A deductive component could enhance database technology by providing the rules component of an ES (Beynon-Davies 1991). A frame is roughly the equivalent to a row in a relational database. A slot is roughly the notion of an attribute in a relation. Frames and slots are artefacts used in ES rules to represent knowledge. A frame (See Paragraph 6.2.2: p113) use slots and is a knowledge presentation used in Knowledge-based Systems.

The writer concludes in agreeing with Turban (1995) that an ES component is ideal to assist a decision-maker in an area where expertise is required.

Chapter 7 - Evaluating the prototype

7.1 Evaluating DSS

Evaluation of a DSS should be built into the development process. It should begin before any technical phases (analysis, design, development, testing and implementation) and should continue beyond the life of the DSS. It is the control mechanism for the entire iterative design procedure (Sprague & Carlson 1982). The evaluation process keeps the cost and effort of the DSS in line with its value. With constant evaluation, the system can die when the need for it is over or it proves not to be valuable. DSS evaluation can also help to quantify the impact of decision-making processes on organisational goals. Sprague & Carlson (1982) consider what to measure, how to measure it and presents a general model for evaluation. DSS evaluations should be considered as planned experiments designed to test one or more hypotheses.

Table 7-1 Examples of measures for DSS Evaluation (Sprague & Carlson 1982)

Productivity measures
1. Time to reach a decision
2. Cost of making a decision
3. Result of the decision
4. Cost of implementing the decision

Process Measures
1. Number of alternatives examined
2. Number of analysis done
3. Number of participants in the decision-making
4. Time horizon of the decision
5. Amount of data used
6. Time spent in each phase of the decision-making
7. Time lines of the decision

Perception measures
1. Control of the decision-making process
2. Usefulness of the DSS
3. Ease of use
4. Understanding of the problem
5. Ease of "selling" the decision
6. Conviction that the decision is correct.

Product measures
1. Response time
2. Availability
3. Mean time to failure
4. Development costs
5. Operating costs
6. Maintenance costs
7. Education costs
8. Data acquisition costs

Sprague & Carlson (1982) divide the possible evaluation measurements into four categories:

- Productivity measures to evaluate impact of the DSS on decisions
- Process measures to evaluate the impact of the DSS on decision-making
- Perception measures the evaluate the impact of the DSS on decision-makers, and
- Product measures that evaluate the technical merit of the DSS

See Table 7-1 (p126) for examples of measures of DSS evaluation. The initiating DSS' impact can be evaluated or the target system (the decision, the organisation, the product on which the impact is) can be measured (Sprague & Carlson 1982). Sprague & Carlson 1982 compare eight methods to measure and evaluate the DSS. These methods are not mutually exclusive and claim not to be system and hypothesis independent. Each is a systematic method of evaluating DSS impact.

7.1.1 Event logging

Events may indicate the DSS impact: log events before and after implementation of the DSS to gain information on actions, opinions, newspaper articles, items from memos, dates etc. Judgement is required in selecting the events to be recorded, and the set of events is not pre-definable. The recording may be on a continuous or before-and-after basis. The evaluation is simply an analysis of recorded events. This method is most useful when:

- Quantitative measures cannot be used
- Time series of effects is of interest, and/or
- Multiple effects are being considered

7.1.2 Attitude Survey

This method attempts to measure opinions based on a questionnaire issued to individuals. A large body of psychological and behavioural research can be used. For the best results, the questionnaire should be administered at various intervals during the development and use of the DSS and the results compared. An attitude survey is most useful when:

- Target system consists of people, and
- Perception measures are being used

7.1.3 Cognitive testing

Cognitive testing utilises methods developed by social psychologists and is a variation of the attitude survey. It is based on explicit theories of behaviour and more formal evaluation procedures. Behaviour patterns, preferences, processes or understanding of the DSS is evaluated. The results of the tests are often difficult to interpret. Cognitive testing will have to be done by experts.

7.1.4 Rating and weighing

This is a highly structured method for a composite numerical evaluation. This methodology involves the development of a set of parameters such as accuracy, timeliness and usability. Individuals rate each parameter evaluating the relative importance of each parameter by assigning a specific parameter to a certain score or weight by using a scale of one to 10. The score of each individual may be averaged or totalled. When analysing the results the evaluator should consider that the numbers attached to the parameters are not precise and may not be accurate. If the ratings and weights are reliable, this method is the easiest to interpret. If not, the results will be misleading.

7.1.5 System Measurement

The performance of the target system is measured in an attempt to quantify the effect of the DSS.
If the target system is the DSS, the evaluation should be product measures for the DSS. The measurements may be collected automatically by the DSS or other sensors, through questionnaires, interviews or observations, or extracted from documents. The method of evaluation is usually a before-and-after comparison of measurements.

The analysis often utilises statistical techniques; data collection; data sampling; and the results are usually easy to interpret. This method unfortunately has a narrow range of applicability because it requires that the characteristics of the target system have quantified performance measures. System measurement is the most reliable evaluation method.

7.1.6 System Analysis

This technique involves a formalised, qualitative technique for describing the impact of the target system on multiple aspects. This evaluation involves the technique used in the analysis preceding the development of any system and may involve description of the system in terms of procedures, information flows, data stored, personnel activities, data used in report and decisions depending on the impact being investigated. Tools that are often used are interviews, document reviews, observation of data collection techniques, flow charts, organisational charts, input-output matrices and decision tables. The evaluation may contain a description or the comparison of descriptions.

This method involves both quantitative and qualitative measures. It is the best method to evaluate the target system's impact on procedures and organisational structures. The biggest advantage is that the baseline can be gathered during the analysis that precedes the DSS design. The results are usually difficult to interpret because this method does not provide a numerical measure of the system impact. An example of this technique may be to document the requirements for data used in a report and compare it with the content and retrieval capability of the DSS.

7.1.7 Cost-Benefit Analysis

This method utilises some of the data collection techniques of the system measurement and the system analysis methods. This method is more often used in feasibility studies for DSS than for evaluation of the DSS. Comparing the ratios before and after the DSS implementation can enhance this analysis. Cost-benefit impact evaluation will have the most meaningful results for managers, but may be particularly difficult to compile because the benefits and costs of a DSS may be socially and not quantifiable. The benefits of the system may appreciate rather than depreciate over time because of the accumulation of data and experience, and because of the difficulty to allocate computer systems costs among users. Because of these problems, the method may reduce to a comparison of direct costs. Cost-benefit analysis is the only way to evaluate economic impact and it should not be ignored or limited to a comparison of direct costs and benefits.

7.1.8 Value Analysis

Keen (as referenced by Sprague & Carlson 1982) proposes a value analysis evaluation technique. It is an approach similar to the cost-benefit analysis but based on benefits first and costs second. It is the benefits that are of importance to decision-makers. Calculations of the costs are not necessary if the benefits are not acceptable. The method attempts to reduce risk by requiring prototyping as part of the evaluation method. Prototypes are relatively low-risk and an inexpensive way to obtain relatively accurate evaluation data. The method evaluated DSS as a research-and development (R&D) effort rather than a capital investment. An R&D evaluation tends to encourage innovation rather than the return on investment.

Keen (as referenced by Sprague & Carlson 1982) describes value analysis as a series of four steps:

- Establish the operational list of benefits the DSS must achieve to be acceptable
- Establish the maximum cost that one is willing to pay to achieve the benefits
- Develop the prototype DSS, and
- Assess the benefits and the costs.

The advantage of this method is that it is simple and integrated with prototyping. It may not include all the relevant measures and is a less rigorous method than the cost-benefit or system analysis techniques. Value analysis seems very close to the intuitive approach that many managers use to evaluate DSS.

7.1.9 Combining methods

The choice of the evaluation method will depend on the DSS and the impacts investigated. The evaluator and the environment too, have an impact on the evaluation method chosen. A combination of evaluation methods will result in a better evaluation because of this effect and the fact that some problems exist with all evaluation methods. System measurement, system analysis and cost-benefit analysis use similar data collection techniques and can therefore easily be combined.

7.2 ES quality and validation

ES users start as novices that do not know much about the knowledge domain. They do not know what the system can deliver (Raggad & Gargano 1999). Users become loyal because of the value they receive continually learning from the ES. One of the reasons ES fail is that ES owners' measurements, analyses and maintenance stress user satisfaction and fails to emphasise value creation (Raggad & Gargano 1999). Value creation is the key to ES success. For an ES to create value for the user continually, it has to evolve by having new knowledge added to the knowledge base regularly. The novice user will learn the rules of the ES with time. Ragged & Gargano (1999) prove by induction that the end-user will become so literate that invoking the ES for future sessions will become infeasible. The users will only become more dependent on the system if the system learns with a higher learning rate than the end-users.

7.2.1 Knowledge and knowledge base validation

Knowledge validation refers to the checking of the accuracy of the expert system. This is achieved by extensive testing of the quality of knowledge as well as the changes in the environment that inappropriate past expert rules.

The value of the ES depends on the validity of its knowledge base. The validity or the knowledge base depends on the consistency of the reasoning scheme of the inference engine (Raggad & Gargano 1999). For the ES to be useful, it must have knowledge depth. Knowledge depth is the ability to extend existing knowledge and infer new knowledge. Contrary to the DSS-user, the ES-user is usually not familiar with the knowledge domain and this creates a risk using the ES. The ES-user who lacks a great deal of background in the problem domain hopes that the system contains the knowledge needed to solve the problem. If the entire knowledge domain of the ES cannot be represented, the defection risk is greater since the probability of missing information is high.

When the ES generates a message that may be anticipated by the decision-maker, there is no added value of information to the decision, and invoking the ES is not justified. Recommendation anticipation is measured by the difference between the value of the decision given by the ES' recommendations and the value of the decision based on prior evidence (Raggad & Gargano 1999).

7.2.2 Knowledge base verification

The power and effectiveness of the ES is equal to the quality of the knowledge it contains. To ensure quality of knowledge the knowledge base needs to be verified. Logical verification of rules can detect many problems in the knowledge base. The potential problems can be grouped into (Klein & Methlie):

- Consistency problems: caused by redundant rules, conflicting rules, subsumed rules, unnecessary if conditions and circular rules. , and
- Completeness problems: caused by missing rules because of unreferenced attribute values, missing combinations of attribute values, control faults and gaps in the inference chains.

The above logical problems may not necessarily cause reasoning faults. Klein & Methlie (1995) summarise various verification tests that verify rules in a knowledge base. Some of the techniques that verify the knowledge base for consistency and correctness are given below.

- **Consistency checking techniques**

Redundant rules:

Two rules are redundant if they succeed in the same situation and have the same conclusions. The condition parts of the rules are equivalent, and one or more of the conclusions of the two rules are the same. Redundant rules do not necessarily cause logical problems. If stored in adequate parts of the knowledge base, it may even affect performance positively. Redundant rules may however create maintenance problems. One of the rules may be maintained when the other is left unchanged.

Conflicting rules:

Two rules are conflicting if they succeed in the same situation but with conflicting solutions. Rules may be different, having equivalent conditions but different conclusions and yet do not conflict at all.

Subsumed rules:

One rule is subsumed by another, if the two rules have the same conclusions, but applies additional constraints on the situation on which it will succeed. Whenever the one rule succeeds, the other one also succeeds.

Unnecessary IF conditions:

Two rules contain unnecessary IF conditions if the rules have the same conclusions, and the IF condition of the one rule is in conflict with an IF condition in the other rule and all other IF conditions are equivalent. , and

Circular Rules:

A set of rules is circular if the chaining of the rules in the set forms a cycle. The system will move into an infinite loop. A dependency chart may help in the detecting of circular rule chains. The circular rule chain may not always be direct and apparent. An algorithm can easily be constructed to detect any implicit circular patterns among rules.

- **Techniques checking for completeness**

Missing rules:

Rules may be missing because of:

- Unreferenced attribute values, and/or
- Missing combinations of condition attribute values

Illegal attribute values:

If a rule refers to an attribute value that is not in the set of legal values, an error occurs, and

Dead ends:

A dead end occurs when a conclusion of a rule does not match the goal of an IF condition of another rule. Dead ends may be caused by terminology errors e.g. where synonyms are used.

♦ Validation

A knowledge base may be logically correct without being valid. Validation has to do with how well a model or measurement conforms to what is being modelled or measured. It tests if the system is a true representation of reality (Klein & Methlie1995).

Three types of faults may be encountered during the evaluation process:

- Factual faults: An assertion does not correctly represent the fact
- Inferential faults: A rule does not correctly represent the domain knowledge. The result is an incorrect conclusion drawn by the system. , and
- Control faults: The rules are correct, but have undesirable control behaviour

The validation is a three-step procedure:

- Run the program or problems
- Identify the faults, and
- Modify the program (Rules and control strategies)

Validity refers to relevance, meaningfulness and correctness. Three types of validations exist: content, construct and criterion related validations. **Content validation** judges each item for its presumed relevance to the property being measured. **Construct validation** refers to the validation of the models: the knowledge base, the reasoning strategies and the analytical relationships. Construct validation is dependant on the way a knowledge base is built. If the models are built around hypothesised relationships, each relationship can be validated. If built on a theory such as the theory of problem solving, constructive validation can be done by the expert's judgements of the constructs established. One such way is by a ***rule trace*** approach. The system is run on a specific problem, the system behaviour traced as the rules are tried, rejected or fired.

Criterion related validity is studied by comparing test scores with one or more criteria or measurements of the attribute to measure. Criterion-related validity is concerned with the predictive ability of the system. Criterion-related validation can be performed ***empirically*** by measuring the system's success. Statistics on overall performance may be easy, but to identify the right and wrong behaviour of individual rules is difficult.

▪ Rule traces

Running rule traces on a problem causes a search tree to develop. The rule trace is the path through this tree. It is an account of rules that have been tried, those that did not succeed and those that did. Two types of rule trace analysis can be done: the intuitive and the analytical. In the intuitive approach, a rule trace is presented to the expert who will comment on the conclusions and the reasoning. There is no explicit external reference to which the system's performance can be compared. The reference is in the expert's head. There is no systematic and controlled way of correcting the identified faults.

Changes to the program may affect other faults in other cases, cases that might have been handled well previously.

Using a more precise conceptual framework and formal procedures may result in a formal analysis to identify and correct faults. A critical point in this analysis is to determine the rules and the sequence in which they should have fired. This trace is called the ideal trace. The ideal trace can be taken as input or be determined by using problem solving and inference techniques. The ideal trace is compared with the actual trace of the rules to locate the first point at which the trace differs. The comparison determines the rule that should have fired and could not do so because it was insufficiently or incorrectly constrained or missing (Bundy et al 1985, as referenced by Klein & Methlie 1995).

It is sufficient to concentrate on errors of commission. Correcting these errors will eventually correct errors of omission. If the rule of commission is corrected, it will cause the faulty rule to not fire any more. If any other errors of commission exist, they too will be corrected to cause the most preferred rule to fire. Several modification techniques associated with rule trace comparisons exist:

- Reordering of rules to correct the control faults
- Adding extra conditions to the antecedent of a rule (also called specialisation or discrimination), and
- Concept learning techniques

▪ **Empirical performance validation**

This approach tests the predictive ability of the system by comparing its conclusions with one or more external variables or criteria. One criterion to use is to test the expert conclusions on a set of test cases and empirically state how well the system's conclusions match with these. The critical point is to have enough representative coverage for validation – to gather enough cases for typical decision outcomes. Special validation techniques can be used based on sensitivity analysis procedures where few cases exist.

A second critical point is that the conclusion must be precise for accurate and useful comparisons. A conclusion drawn by the system must be classified as either correct or incorrect. Judgement substitutes statistical analysis. Weiss & Kulikowski (1984) as referenced by Klein & Methlie (1995) describe the process of performance validation as iterations through the following steps:

- Obtain the performance of rules on a set of stored cases
- Analyse the rules, and
- Revise the rules

The first step is to produce a performance summary for all stored cases. Several cases may give the same conclusions. Measuring the number of cases where the model's conclusion match with that of the expert, gives a performance measure called the true positive. The model may also infer a conclusion falsely. This performance measure is called the false positive. If all true positives were identified and some extra false positives exist, it calls for strengthening of the antecedent conditions of one or more rules of the inference chains leading to the conclusion. This is called specialisation and makes the rule's conditions more difficult to satisfy and therefore to execute or fire. If no conclusions were identified as false positive and some positive conclusions were not identified, it calls for generalisation or weakening of some of the antecedent conditions of one or more rules. The rule's conditions must be weakened so that it can be fired in more of the test cases. Statistics about each specific rule can be gathered and the performance analysed. On viewing these results, a revision of the rules called rules refinement can be carried out.

Errors of commission can be corrected by specialisation, that is adding constraints to the antecedents of the rules and so limiting the situations in which it will be fired. Rules that should have fired could be generalised making the antecedent conditions less restrictive to fire more easily. The above is an example of rule refinement. By applying generalisation, the number of correctly concluded cases may be increased, but this may result in increasing the number of misconcluded cases. Specialisation may reduce the number of misconcluded cases, but have the potential to reduce the target ratio.

7.3 Testing and evaluation of the KB-DSS

When a prototype is running, design specifications are tested, revised and new specifications added to accomplish needs that were not initially known. Development moves through a series of cycles before the system is finally ready for operation. This is a spiral operation having a cyclic pattern. Testing and evaluation are well-known tasks in model building and computer programming. Data, knowledge and models need to be validated and the quality of the system assessed. In testing and evaluating the system, norms are needed to test the actual behaviour of the system. Three kinds of norms exist:

- Theoretical norms: norms described by the normative decision theory
- Empirical norms: norms such as those prescribed by the expert model for problem solving, and
- Subjective norms: qualitative assessments of the system performance, either by users or experts

Theoretical and empirical norms validate the system and subjective norms evaluate the system. Verification and validation of knowledge-based systems are closely related to maintenance and learning. The typical building process of the knowledge-based systems is incremental. New knowledge is added to existing knowledge and may affect the existing knowledge. Inside the knowledge base, three kinds of knowledge can be changed (TEIRESIAS system by Davis & Lenat 1982, as referenced by Klein & Methlie 1995):

- Inference rules can be added to, modified or deleted from the knowledge base
- Concepts (objects, attributes and values) can be deleted, added or changed, and
- Control strategies can be changed

Klein & Methlie (1995) deal with the verification and validation of knowledge-based reasoning systems; discussing techniques to verify the logical correctness of knowledge bases including problems of completeness and consistency; discussing what and how to validate; and evaluating performance. These issues were discussed in Paragraph 7.2.2: p130.

7.3.1 A framework for testing and evaluation

This framework includes DSS features and ES features under the paradigm of decision support (Klein & Methlie 1995). ES and DSS differ in several aspects. Testing and evaluating these two systems are two very different tasks. DSS is difficult to evaluate, because of the diversity of DSS. A DSS is a set of resources, data and models that are placed at the disposal of the decision-maker. The individual decision-making's behaviour determines the usage of the system. Various decision-makers may use a DSS in different ways. There is no single criterion by which a KB-DSS can be evaluated. In ES, the predictive ability can be a measure of good or bad.

Table 7-2 A framework for KB-DSS (Klein & Methlie 1995)

Validation	
1	Data input
2	Knowledge base: concepts and relations (e.g. rules)
3	Reasoning: strategies
4	Results: conclusions
System performance	
1	Efficiency
2	Data entry
3	Output format
4	Hardware
5	Usage: how much, when
6	Man-machine interface
Task performance aspects	
1	Decision-making: time, alternatives, analysis and participants
2	Decision quality
3	User perception: trust, satisfaction, usefulness and understanding
Business opportunities	
1	Costs: system development, operating and maintenance
2	Benefits: income and reduced costs
3	Values: service, competitive advantage and training
Evolutionary aspects	
1	Knowledge maintenance
2	Response time to change demands
3	Functionality of the development tool

The ultimate objective of DSS is to improve the effectiveness of decision-making. The effectiveness of decision-making is still unproven using measurements. Various DSS evaluation techniques are discussed in Paragraph 7.1: p126. A KD-DSS can be verified and validated, by testing and evaluating its knowledge-base parts.

Table 7-2 (p129) shows a summary of the test and evaluation framework for a KB-DSS as proposed by Klein & Methlie (1995). Knowledge base verification as well as knowledge base validation was discussed in Paragraph 7.2.2 (p130). Both are essential when validating the KB-DSS.

7.3.2 Performance evaluation

In this section, measurements are determined to evaluate the functionality of the KB-DSS. Evaluation requires an external reference to which the system can be compared. Because of the evolutionary nature of DSS, the ill structuredness of the DSS tasks, and the individual decision making behaviour, it is difficult to establish such a reference. No one criterion is likely to be sufficient to evaluate a KB-DSS. Klein & Methlie (1995) present categories of aspects of criteria to evaluate a KB-DSS:

- System performance
- Task performance aspects
- Business opportunities, and
- Evolutionary aspects

♦ System performance

Measures such as response time, availability, reliability, data entry, dialog, usage time, users and quality of the system support (documentation, training and more) can be obtained by observations, event logging and attitude surveys as discussed in Paragraph 7.1 (p126).

♦ Task performance aspects

These aspects are concerned with the functionality of the computer program in relation to the decision task. A distinction needs to be made between decisions and outcomes. A good decision does not necessarily lead to a good outcome. The quality of the decision depends on the correctness of the actions taken by the decision-maker. Task performance is measured by the quality of the decision more than the quality of the outcome. It is measured in terms of time spent to make the decision, alternatives measured and information searched. Some qualitative measures such as trust, satisfaction and understanding can be done.

♦ Business opportunities

The amount of resources used for a decision in terms of cost, and the effects of commitment in terms of benefit such as income and reduced costs, can be measured. Methods that can be used to measure business opportunities are cost/benefit analysis and value analysis (See Paragraph 7.1: p126).

♦ Evolutionary aspects

These measurements are more qualitative than quantitative and judgement is primarily done by evaluating aspects such as the system's ability to change, the functionality of the development tool and the availability of a system's builder.

7.4 Evaluating the prototype

In Paragraph 7.1 (p126), examples of measures of DSS evaluation were described. These measures are listed again in Table 7-3. It was mentioned before that the evaluation of the DSS should be built into the development process (See Paragraph 7.1: p126). It was also mentioned that the choice of the evaluation method will depend on the DSS and the impacts investigated (See Paragraph 7.1.9: p129). Recall that Sprague & Carlson (1982) divide the possible evaluation measurements into four categories:

- Productivity measures to evaluate impact of the DSS on decisions
- Process measures to evaluate the impact of the DSS on decision-making
- Perception measures the evaluate the impact of the DSS on decision-makers, and
- Product measures to evaluate the technical merit of the DSS

Since the advice-system is a prototype, and not in production yet, it would not be possible to evaluate the DSS' impact on the decision-makers and students. After implementation, a questionnaire could be given to decision-makers as well as the experts that usually assisted them before the DSS to determine the impact and success of the DSS.

Table 7-3Examples of measures for DSS Evaluation (Sprague & Carlson 1982)

Productivity measures
1. Time to reach a decision
2. Cost of making a decision
3. Result of the decision
4. Cost of implementing the decision

Process Measures
1. Number of alternatives examined
2. Number of analysis done
3. Number of participants in the decision-making
4. Time horizon of the decision
5. Amount of data used
6. Time spent in each phase of the decision-making
7. Time lines of the decision

Perception measures
1. Control of the decision-making process
2. Usefulness of the DSS
3. Ease of use
4. Understanding of the problem
5. Ease of "selling" the decision
6. Conviction that the decision is correct.

Product measures
1. Response time
2. Availability
3. Mean time to failure
4. Development costs
5. Operating costs
6. Maintenance costs
7. Education costs
8. Data acquisition costs

Measures such as the number of alternatives examined, the number of analysis done, the number of participants, the length in time the decision took and the amount of data can be automatically calculated from the application and the statistics displayed on the web page presenting the user interface. This can be used as ongoing evaluation to determine the success and relevance of the project. The explanations of the recommendations should convict the user that the decision is correct. This would support the perception measures. To prove the effect and usefulness the user will have to complete a questionnaire.

ILOG JRules reports the number of rules fired and the time the rules took to fire can be calculated. These values can be retained and statistics compiled. When in production, these measures could be an indication of a problem in the system and should be constantly evaluated. The decision recommendations of the KB-DSS and the end-result of the decision made by the decision-maker using the KB-DSS can be recorded and re-evaluated by the experts from time to time to verify the rules and the process.

A product measure such as response time could also be captured automatically. Response time depends on access to the database, the network, the rule set and the time the rules took to fire. Access to the database is done once, at the start of the student session, and then used by the rules in the knowledge base when necessary.

The KB-DSS could be cost-evaluated by determining all the relevant costs involved to produce and maintain a KB-DSS on the Intranet of the university. The main decisive factor in measuring the success of the DSS would be the decreasing number of queries from students that need assistance in selecting the best courses for the current semester. By keeping statistics, it can be proved that the system was worthwhile. Another way of evaluating this KB-DSS is to consider the criteria set in Paragraph 4.1 (p41). These criteria were:

- The DSS must include an inference engine in their knowledge component to provide reasons for actions taken for example it must include an expert subsystem with an inference engine
- The knowledge component must include all relevant knowledge in a manner that course experts can maintain it
- Changes to the knowledge component should preferably not result in a new version of the DSS/SDSS software application
- Possibility to deploy the software application on the university's Intranet will be an advantage, and
- The DSS must be able to provide a reason for actions taken

Table 4-2 (p52) listed these criteria as met by ILOG JRules. The criteria re-evaluated are:

- ILOG JRules includes an inference engine in its knowledge component. It is implemented by using the Rete algorithm. JRules is an expert shell that provides an empty knowledge base and an inference engine. It offers the capability of a user interface that may contain the other components of a DSS or KB-DSS.

- In Paragraph 4.5.5 (p71), the rule editors were evaluated. It was concluded that the rule administrator would need some initial assistance in compiling the rules. He would then be able by selection, copying, deleting and pasting, to maintain and test rules after he received some training. The training should focus on the verification of the rules. When a new type of rule is needed, it would be best if the rule analyser consult the builder/programmer for assistance.
- Each department would have its own rules in a separate rule set file (.irl). Depending on the degree the student is pursuing the correct .irl file would be invoked. When updated and replaced by the department the next student enquiry when getting his data from the database will invoke the updated file immediately. The maintenance of the rules, except for structural changes to the objects used (addition of new fields etc.), should be fairly painless provided that the rule base was tested sufficiently by the rule administrator before submitting it into production. Setting and testing of the rule bases are done in a decentralised way. Each department functions independently of the others in setting and testing their rule base. When satisfied, their .irl file is exported and used with the application in production e.g. the Intranet where the students would have easy access to the application.
- The DSS application can run as an applet on the university's Intranet, and
- Reasons for recommendations, approvals and rejections are given as part of the user interface

7.5 Summary

Although the rule administrator will not be able to create his complete rule base using the current prototype, he can do rule maintenance and be responsible for his current set of rules. The administrator will be part of the testing and setting of the rules and the more experienced he becomes the less involved in the process the builder/programmer would be. The administrator will take the responsibility of the rules in his rule base, as set by him or with the assistance of the builder/programmer, or suffer the consequences when having to assist students when the system behaves unexpectedly. When the rules in the KB-DSS are verified as complete and consistent, the experts in the departments would be alleviated of the multitude of queries and allowed to focus on those situations where their expertise are really needed.

Chapter 8 - Conclusions and final remarks

The hypothesis of this study was set as:
"It is possible to provide an interface to a knowledge base where domain experts can develop and maintain their knowledge that serves as an input to the expert or intelligent component of an Intelligent Decision Support System."

The objectives were partially met:

1. An Intelligent Decision Support System (IDDS) or Knowledge-Based Decision Support System (KB-DSS) was prototyped using ILOG JRules that has the following features or capabilities:
   - The **expert is able to maintain** his knowledge in the knowledge base. ILOG JRules have different builder launchers displaying rules in different modes as interfaces to the knowledge base. Depending on the skill of the rule administrator or course domain expert, the modes will be suitable to specify and maintain the expert's knowledge. A programmer/builder is needed to create the objects shared by the application and the rules of the knowledge base as well as the user or decision-maker's interface.
   - The knowledge is kept as objects and **decentralised** rules. The rules can be developed independently of other experts, in a decentralised way, and is stored in a separate rule set that is not dependant on the rule sets of other experts, to be invoked when needed. A specific **DSS simulation model** was created to assist the decision-makers: the students, to reach a well-informed decision.
   - The KB-DSS can run as an applet on the **web** and provides the decision-maker with multiple **alternatives** to choose from, presenting explanations and suggestions. This contributes to the **effectiveness** of the DSS by constantly **explaining** its actions.
   - A **data component** can extract details from the university's database supplying information to the DSS intelligent component.
2. Maintenance of the knowledge in the knowledge base will not cause a new release of the IDDS application and thus ensures productivity, providing changes to the rules did not involve structural changes to the objects used in the rules. Once a new update has been done, a new student will invoke the latest updated file.

Domain experts can maintain their knowledge in a knowledge base, providing that the objects and attributes used in the rule base were specified by a builder/programmer. Having the expert maintain his own rule sets, decentralised from the other experts and separately from production, places the responsibility of the rules where it belongs: with the expert. At the same time, the expert is alleviated from numerous identical queries to focus on situations where his expertise is really needed. As the expert captures his knowledge by additional rules in the knowledge base, he will find more and more time to focus on the non-routine tasks that requires his expertise , and when he leaves the company, the knowledge will remain, resulting in a win-win situation.

Decision Support and Expert Systems are matured fields of study with limited developments. Some developments were discussed in Paragraph 1.1 (p1). The advice system could possibly be developed as an intelligent agent supplying recommendations to other systems used by the university where selection of new courses for the current semester is appropriate. A learning component can adjust the advice system's knowledge base by comparing its suggestions with the final confirmed decision of the decision worker. A business object model (BOM) can be developed to be used with the builder interfaces provided with ILOG JRules to specify an even simpler business action language linked with the objects used in the environment. Specifying this BOM could result in a simpler, more natural way of expressing the rules that would allow the rule administrator as a course-skilled domain expert to develop his own rules without the assistance of the builder/programmer.

As technologies develop, Decision Support and Expert Systems or Intelligent Systems can be revisited to improve the decision-making processes by including these newer technologies.

Bibliography

1. AGENT OCX Programming Guide. c1996-1998. Internet: http://www.haley.com CD-ROM: THE PDFs\ AgentOCXprogrammersGuide.pdf. Access: 6 August 1999.

2. ALTER, S.L. 1980. Decision support systems: Current practice and continuing challenges. United States of America: Addison-Wesley publishing company.

3. ANTHONY R.N. 1965. Planning and Control Systems: A Framework for analysis. 10th reprint: 1979. United States of America: President and Fellows of Harvard College.

4. BARQUIN, R. 1996. On the first issue of the journal of data warehousing. The journal of data Warehousing, 1 (July): 2-6

5. BEYNON-DAVIES, P. c1991. Expert database systems a gentle introduction. UK: McGraw-Hill Book Company (UK) Limited.

6. BIELAWSKI, L. & Lewand, R. c1991. Intelligent systems design. Integrating Expert Systems, Hypermedia, and Database Technologies. United States America: John Wiley & Sons, Inc.

7. BUI, T. & Lee, J. 1999. An agent-based framework for building Decision Support Systems. Decision Support Systems, 25 (1999): 225-237.

8. BUSINESS RULE STUDIO. c1998. Internet: http://www.RuleMachines.com/ \Rule Machines\Business Rule Studio\Docs\: Rule Machines Corporation. Access: 12 July 1999.

9. COURTNEY, J.F. 2001. Decision making and knowledge management in inquiring organizations: toward a new decision-making paradigm for DSS. Decision Support Systems, 31 (2001): 17-38.

10. EARLE, N & Keen, P. 2000. From .com to .profit: inventing business models that deliver value and profit. San Francisco: Jossey-Bass Inc.

11. FINLAY, P. c1994. Introducing Decision Support Systems. Great Britain: Blackwell Publishers.

12. FRIEDMAN-HILL, E.J. 1997. Jess, the Java expert system shell. Unlimited Release. Internet: http://herzberg.ca.sandia.gov/jess. Access: 27 April 2000.

13. GERVASI, V & Nuseibeh, B. 2002. Lightweight validation of natural language requirements. Software - Practice and experience, 32 (2002), February 2002: 113-132.

14. GOODALL, A. 1985. The guide to expert systems. Oxford and New Jersey: Learned Information.

15. GORRY, G.A. & Scott-Morton, M.S. 1971. A framework for management information systems. Sloan management review, 13 (1), Fall 1971: 55-70.

16. HAUMER, P., Pohl, K. & Weidenhaupt, K. 1998. Requirements Elicitation and Validation with Real World Scenes. IEEE Transactions on Software Engineering, December 1998, 24(12): 1036-1053.

17. HOFFER J.A, Prescott M.B. & McFadden F.R. 2002. Modern database management. 6th Edition.
New Jersey: Pearson Education, Inc.

18. HOLSAPPLE, C.W. 2001. Editorial: Knowledge management support of decision-making. Decision Support Systems, 31 (2001): 1-3.

19. HOLSAPPLE, C.W. & Joshi, K.D. 2001. Organizational knowledge resources. Decision Support Systems, 31 (2001): 39-54.

20. ILOG JRules 3.0 User's Manual. c2000. (a) Internet: http://www.ilog.com JRules30\docs\pdf\jruleuser.pdf. France: ILOG JRules S.A. Access: 9 August 2000.

21. ILOG JRules 3.0 Builder User's Guide. c2000. (b) Internet: http://www.ilog.com JRules30\docs\pdf\jrulesbuilder.pdf. France: ILOG JRules S.A. Access: 9 August 2000.

22. ILOG Business Rules development tools white paper. 2002. Internet: http://www.ilog.com Devtools.pdf. Access: 2 December 2002.

23. INMON, W. H. & Hackathorn, R.D. 1994. Using the data warehouse. New York: John Wiley & Sons.

24. KIM, H. 2001. An XML-based modelling language for the open interchange of decision models. Decision Support Systems, 31 (2001): 429-441.

25. KLEIN, M.R. & Methlie, L.B. 1995. Knowledge-based Decision Support Systems with Applications in Business. 2nd Edition. England: John Wiley & Sons.

26. LAWRENCE, J.A. (jr.) & Pasternack, B.A. 2002. Applied management science: modelling, spreadsheet analysis, and communication for decision-making. 2nd Edition. United States of America: John Wiley & Sons, Inc.

27. MALLACH, E.G. c1994. Understanding Decision Support Systems and Expert systems. United States of America: Richard D. Irwin, Inc.

28. O'KEEFE, R.M. & McEachern, T. 1998. Web-based customer Decision Support Systems. Communications of the ACM, 41(3), March: 71-80.

29. OLSON, D.L. & Courtney, J.F. (jr.) c1992. Decision support models and expert systems. United States of America: Macmillan Publishing Company.

30. PAL, K. & Palmer, O. 2000. A decision-support system for business acquisitions. Decision Support Systems, 27(2000): 411-429.

31. RAGGAD, B.G. & Gargano, M.L. 1999. Expert system: defection and perfection. Logistics Information Management, 12(5): 395-406.

32. SAUTER, V.L. 1999. Intuitive decision-making. Communications of the ACM, 42(6), June: 109-122.

33. SHIM, J.P., Warketin, M., Courtney, J.F., Power, D.J., Sharda, R. & Carlsson, C. 2002. Past, present and future of decision support technology. Decision Support Systems, 33(2002), February: 111-126.

34. SIMON, H.A. 1960. The new science of management decision. New York: Harper & Brothers Publishers.

35. SIMON, H.A. 1977. The new science of management decision. United Stated of America: Prentice-Hall International, Inc.

36. SPRAGUE, R.H. (jr.) & Carlson, E.D. 1982. Building effective decision support systems. Second reprinting 1986. Grolier Incorporated. New Jersey: Prentice-Hall.

37. SRIDHAR, S. 1998. Decision support using the Intranet. Decision Support Systems, 23(1998): 19-28.

38. TAYLOR, B.W. III. 1999. Introduction to Management Science. 6th edition. United States of America: Prentice Hall.

39. TURBAN, E. c1993. Decision Support and Expert Systems: Management Support Systems. 3rd edition. New York: Macmillan Publishing Company.

40. TURBAN, E. c1995. Decision Support and Expert Systems: Management Support Systems. 4th edition. New York: Macmillan Publishing Company.

41. TURBAN, E., Lee, J., King, D. & Chung, H.M. c2000. Electronic Commerce: A Managerial Perspective. International Edition. United States of America: Prentice Hall.

42. TURBAN, E., McLean, E. & Wetherbe, J. 2001. Information Technology for Strategic Advantage. 2nd Edition. New York, Chichester, Weinheim, Brisbane, Singapore, Toronto: John Wiley & Sons, Inc.

43. UNIVERSITY OF PRETORIA: Rules and Syllabuses 1999: Faculty of Science. 1999. Pretoria. Universityof Pretoria.

44. WELDON, J.L. 1996. Data mining and visualization. Database programming and design. 9(May): 21-24.

45. WHITTEN, J.L., Bentley, L.D. & Dittman K.C. 2001. Systems analysis and design methods. 5th
Edition. New York: Irwin/McGraw-Hill.

Appendix A – Sample rules

External rules file generated from the syntactic editor used as input to the application

- **data/F02133221.ilr**

```
//===================================== -*- Java -*- ========================
// Prototype for the student course advice system:  Sample rules that should be maintained
//=========================================================================
import course.advice.*;


/**Degree CS determines:  Course credits = 225;
                max 110 credits at 100 level;
                min 56 at 300/400 level;
                max 22 credits from other departments;
                max 56 credits per year  */
rule DegreeComputerScienceDetermines
{
   priority = maximum;
   when
     {
  ?x:CurrDegree(currDegreeCode.equals("02130001");currStudyProgram.equals("02133221"));
     }
   then
     {
  assert CourseDetails(225,70,56,22,56);
     }
};
/**Faculty Requirement WTW114 */
rule FacultyRequirementWTW114
{
  priority = maximum;
   when
     {
        not
StudSubj(subjName.equals("WTW114");(passLevel==PASSED)||(passLevel==REGISTERED)
           ||(passLevel==OPTIONAL));
     }
   then
     {
        assert StudSubj("WTW114",LEVEL100,11,REGISTERED,BOTH_SEMESTERS);
```

```
      }
};
/**FacultyForseRequirement WTW114 */
rule FacultyRequirementForseWTW114
{
   priority = maximum;
    when
      {
         not
StudSubj(subjName.equals("WTW114");(passLevel==PASSED)||(passLevel==REGISTERED));
         ?x: StudSubj(subjName.equals("WTW114");(passLevel==OPTIONAL));
      }
    then
      {
         retract ?x;
         assert StudSubj("WTW114",LEVEL100,11,REGISTERED,BOTH_SEMESTERS);
      }
};
/**Information Technology Requirement COS110 */
rule ITRequirementCOS110
{
  priority = 10000;
    when
      {
         not
StudSubj(subjName.equals("COS110");(passLevel==PASSED)||(passLevel==REGISTERED)
         ||(passLevel==OPTIONAL));
      }
    then
      {
  assert StudSubj("COS110",LEVEL100,11,REGISTERED,BOTH_SEMESTERS);
      }
};
/**Information Technology Requirement Forse COS110 */
rule ITRequirementForseCOS110
{
  priority = 10000;
    when
      {
          not
StudSubj(subjName.equals("COS110");(passLevel==PASSED)||(passLevel==REGISTERED));
          ?x: StudSubj(subjName.equals("COS110");(passLevel==OPTIONAL));
      }
    then
      {
          retract ?x;
  assert StudSubj("COS110",LEVEL100,11,REGISTERED,BOTH_SEMESTERS);
      }
};
/**Optional COS120 */
rule OptionalCOS120
{
  priority = 1000;
    when
      {
          not
StudSubj(subjName.equals("COS120");(passLevel==PASSED)||(passLevel==REGISTERED)||(passLe
vel==OPTIONAL));
      }
    then
      {
  assert StudSubj("COS120",LEVEL100,11,OPTIONAL,BOTH_SEMESTERS);
      }
};
/**Information Technology Requirement COS212:  Prerequisite COS110 passed */
rule ITRequirementCOS212
{
  priority = 900;
    when
      {
          ?x: StudSubj(subjName.equals("COS110");(passLevel==PASSED));
          not
StudSubj(subjName.equals("COS212");(passLevel==PASSED)||(passLevel==REGISTERED)||
                    (passLevel==OPTIONAL));
      }
    then
      {
  assert StudSubj("COS212",LEVEL200,7,REGISTERED,BOTH_SEMESTERS);
      }
```

```
};
/**Information Technology Requirement COS213:  Prerequisite COS110  passed*/
rule ITRequirementCOS213
{
  priority = 900;
   when
     {
        ?x: StudSubj(subjName.equals("COS110");(passLevel==PASSED));
        not
StudSubj(subjName.equals("COS213");(passLevel==PASSED)||(passLevel==REGISTERED)||
                (passLevel==OPTIONAL));
     }
   then
     {
  assert StudSubj("COS213",LEVEL200,7,REGISTERED,BOTH_SEMESTERS);
     }
};
/**Information Technology Requirement COS221:  Prerequisite COS110 passed */
rule ITRequirementCOS221
{
  priority = 900;
   when
     {
        ?x: StudSubj(subjName.equals("COS110");(passLevel==PASSED));
        not
StudSubj(subjName.equals("COS221");(passLevel==PASSED)||(passLevel==REGISTERED)||
                (passLevel==OPTIONAL));
     }
   then
     {
  assert StudSubj("COS221",LEVEL200,7,REGISTERED,BOTH_SEMESTERS);
     }
};
/**Information Technology Requirement COS222:  Prerequisite COS110 passed */
rule ITRequirementCOS222
{
  priority = 900;
   when
     {
        ?x: StudSubj(subjName.equals("COS110");(passLevel==PASSED));
        not
StudSubj(subjName.equals("COS222");(passLevel==PASSED)||(passLevel==REGISTERED)||
                (passLevel==OPTIONAL));
      }
    then
      {
  assert StudSubj("COS222",LEVEL200,7,REGISTERED,BOTH_SEMESTERS);
      }
};
/**Information Technology Requirement COS283:  Prerequisite COS110 passed */
rule ITRequirementCOS283
{
   priority = 900;
    when
      {
        ?x: StudSubj(subjName.equals("COS110");(passLevel==PASSED));
        not
StudSubj(subjName.equals("COS283");(passLevel==PASSED)||(passLevel==REGISTERED)||
                (passLevel==OPTIONAL));
      }
    then
      {
  assert StudSubj("COS283",LEVEL200,7,REGISTERED,BOTH_SEMESTERS);
      }
};
/**Information Technology Requirement COS284:  Prerequisite COS110  passed */
rule ITRequirementCOS284
{
    priority = 900;
    when
      {
        ?x: StudSubj(subjName.equals("COS110");(passLevel==PASSED));
        not
StudSubj(subjName.equals("COS284");(passLevel==PASSED)||(passLevel==REGISTERED)||
                (passLevel==OPTIONAL));
      }
    then
      {
  assert StudSubj("COS284",LEVEL200,7,REGISTERED,BOTH_SEMESTERS);
      }
```

```
};
/**Information Technology Requirement COS301 : Prerequisite COS120 passed*/
rule ITRequirementCOS301
{
    priority = 500;
    when
      {
         ?x: StudSubj(subjName.equals("COS120");(passLevel==PASSED));
         not
StudSubj(subjName.equals("COS301");(passLevel==PASSED)||(passLevel==REGISTERED)||
                (passLevel==OPTIONAL));
      }
    then
      {
  assert StudSubj("COS301",LEVEL300,7,REGISTERED,BOTH_SEMESTERS);
      }
};
/**Compulsory Course PHY171:  Prerequisite WTW114  Passed or Registered*/
rule CompulsoryCoursePHY171
{
  priority = high;
    when
      {
         ?x:
StudSubj(subjName.equals("WTW114");(passLevel==PASSED)||(passLevel==REGISTERED));
         not
StudSubj(subjName.equals("PHY171");(passLevel==PASSED)||(passLevel==REGISTERED)||
                (passLevel==OPTIONAL));
      }
    then
      {
  assert StudSubj("PHY171",LEVEL100,22,REGISTERED,BOTH_SEMESTERS);
      }
};
/**Compulsory Course ERS220:  No Prerequisites*/
rule CompulsoryCourseERS220
{
   priority = 900;
    when
      {
        not
StudSubj(subjName.equals("ERS220");(passLevel==PASSED)||(passLevel==REGISTERED)||
              (passLevel==OPTIONAL));
      }
    then
      {
  assert StudSubj("ERS220",LEVEL200,8,REGISTERED,BOTH_SEMESTERS);
      }
};
/**Compulsory Course ERS320:  Prerequisite ERS220  Passed*/
rule CompulsoryCourseERS320
{
    priority = 500;
    when
      {
        ?x: StudSubj(subjName.equals("ERS220");(passLevel==PASSED));
        not
StudSubj(subjName.equals("ERS320");(passLevel==PASSED)||(passLevel==REGISTERED)||
              (passLevel==OPTIONAL));
      }
    then
      {
  assert StudSubj("ERS320",LEVEL300,8,REGISTERED,BOTH_SEMESTERS);
      }
};
/**AtLeast4Courses COS314:  One of at least 4 - Prerequisite COS110  Passed*/
rule AtLeast4CoursesCOS314
{
    priority = 500;
    when
      {
        ?x: StudSubj(subjName.equals("COS110");(passLevel==PASSED));
        not
StudSubj(subjName.equals("COS314");(passLevel==PASSED)||(passLevel==REGISTERED)||
              (passLevel==OPTIONAL));
      }
    then
      {
  assert StudSubj("COS314",LEVEL300,7,REGISTERED,BOTH_SEMESTERS);
      }
```

```
};
/**AtLeast4Courses COS324:  One of at least 4 Prerequisite COS222*/
rule AtLeast4CoursesCOS324
{
   priority = 500;
   when
     {
        ?x: StudSubj(subjName.equals("COS222");(passLevel==PASSED));
        not
StudSubj(subjName.equals("COS324");(passLevel==PASSED)||(passLevel==REGISTERED)||
               (passLevel==OPTIONAL));
     }
   then
     {
  assert StudSubj("COS324",LEVEL300,7,REGISTERED,BOTH_SEMESTERS);
     }
};
/**AtLeast4Courses COS332:  One of at least 4 - Prerequisite COS283  Passed */
rule AtLeast4CoursesCOS332
{
   priority = 500;
   when
     {
        ?x: StudSubj(subjName.equals("COS283");(passLevel==PASSED));
        not
StudSubj(subjName.equals("COS332");(passLevel==PASSED)||(passLevel==REGISTERED)||
               (passLevel==OPTIONAL));
     }
   then
     {
  assert StudSubj("COS332",LEVEL200,7,REGISTERED,BOTH_SEMESTERS);
     }
};
/**AtLeast4Courses COS333:  One of at least 4 - Prerequisite COS110  Passed */
rule AtLeast4CoursesCOS333
{
   priority = 500;
   when
     {
        ?x: StudSubj(subjName.equals("COS110");(passLevel==PASSED));
        not
StudSubj(subjName.equals("COS333");(passLevel==PASSED)||(passLevel==REGISTERED)||
                (passLevel==OPTIONAL));
      }
    then
      {
  assert StudSubj("COS333",LEVEL300,7,REGISTERED,BOTH_SEMESTERS);
      }
};
/**AtLeast4Courses COS341:  One of at least 4 - Prerequisite COS212  Passed */
rule AtLeast4CoursesCOS341
{
    priority = 500;
    when
      {
        ?x: StudSubj(subjName.equals("COS212");(passLevel==PASSED));
        not
StudSubj(subjName.equals("COS341");(passLevel==PASSED)||(passLevel==REGISTERED)||
                (passLevel==OPTIONAL));
      }
    then
      {
  assert StudSubj("COS341",LEVEL300,7,REGISTERED,BOTH_SEMESTERS);
      }
};
/**AtLeast4Courses COS343:  One of at least 4 - Prerequisite COS110  Passed */
rule AtLeast4CoursesCOS343
{
    priority = 500;
    when
      {
        ?x: StudSubj(subjName.equals("COS110");(passLevel==PASSED));
        not
StudSubj(subjName.equals("COS343");(passLevel==PASSED)||(passLevel==REGISTERED)||
                (passLevel==OPTIONAL));
      }
    then
      {
  assert StudSubj("COS343",LEVEL300,7,REGISTERED,BOTH_SEMESTERS);
      }
```

```
};
/**CourseApproved1:  Head of Department/lecturer approval*/
rule CourseApproved1
{
   priority = 1000;
   when
     {
        ?a: StudSubj(?x:
subjName;(passLevel==TD)||(passLevel==TDH);?y:level;?z:noOfUnits;?s:semester);
        not StudSubj(subjName.equals(?x);(passLevel==PASSED)||(passLevel==REGISTERED)||
                (passLevel==OPTIONAL));
     }
   then
     {
        retract ?a;
  assert StudSubj(?x,?y,?z,REGISTERED,?s);
     }
};
rule OnlyOnePassLevel1
{
   priority = low;
   when
     {
?x: StudSubj(?y: subjName; passLevel==PASSED);
       ?z: StudSubj(subjName.equals(?y);(passLevel==REGISTERED)||(passLevel==OPTIONAL));
       }
   then
     {
       retract ?z;
     }
};
rule OnlyOnePassLevel2
{
   priority = low;
   when
     {
?x: StudSubj(?y: subjName; passLevel==REGISTERED);
       ?z: StudSubj(subjName.equals(?y);passLevel==OPTIONAL);
     }
   then
```

```
     {
      retract ?z;
     }
};
/**
Max 110 level 100 credits allowed */
rule FacultyRequirementTooManyLevel100Credits
{
  priority = maximum -5;
   when
     {
        CourseDetails(?a:level100Max);
        ?c: collect ( new TestLevelSubj(LEVEL100))

StudSubj(level==LEVEL100;(passLevel==PASSED)||(passLevel==REGISTERED);?y:noOfUnits)
        where (totalNoOfUnits() > ?a);
     }
   then
     {

      assert(?c);
     }
};
/**
Max 56 Credits per year allowed */
rule FacultyRequirementTooManySELECTEDSubjects
{
  priority = maximum -10;
   when
     {
        CourseDetails(?a:creditsPerYear);
        ?c: collect ( new TestLevelSubj(REGISTERED))
          StudSubj(?x:level;(passLevel==REGISTERED);?y:noOfUnits)
        where (totalCreditsPerYear() > ?a);
     }
   then
     {
      assert(?c);
     }
};
```

```
/*** Credits SELECTED reduced - one high level course moved to OPTIONAL */
rule TooManyCreditsPerYearMovedHighestLevelToOptional
{
 priority = maximum -10;
   when
    {
      ?g: StudSubj(?b: subjName;?c: level; ?d: noOfUnits;passLevel==REGISTERED; ?e: semester; ?f:
itemNo);
      not StudSubj(passLevel==REGISTERED; ?c < level);
      CourseDetails(?a: creditsPerYear);
      ?x: TestLevelSubj(total > ?a;collectionType==TestLevelSubj.REGISTERED);
    }
   then
    {
      retract ?g;
      assert StudSubj(?b,?c,?d,OPTIONAL,?e);
      update refresh ?x;
    }
};
/*** Level 100 credits selected reduced - one moved to OPTIONAL */
rule TooManyLevel100CreditsMoveLastOneAdded
{
 priority = maximum - 5;
   when
    {
      ?g: StudSubj(?b: subjName;?c:level;level==LEVEL100; ?d:
noOfUnits;passLevel==REGISTERED; ?e: semester; ?f: itemNo);
      not StudSubj(passLevel==REGISTERED;level==LEVEL100; ?f < itemNo);
      CourseDetails(?a:level100Max);
      ?x: TestLevelSubj(total > ?a;collectionType==TestLevelSubj.LEVEL100);
    }
   then
    {
      retract ?g;
      assert StudSubj(?b,?c,?d,OPTIONAL,?e);
      update refresh ?x;
    }
};
```

The syntactical mode of the syntactic editor: rules from data/F02133221.ilr

```
Rule DegreeDetermines
WHEN
    there is a CurrDegree called ?x
        such that currDegreeCode.equals("02130001")
            and currStudyProgram.equals("02133221")
THEN
    retract ?x
    assert CourseDetails ( 225, 70, 56, 22, 56 )


Rule FacultyRequirement
WHEN
    there is no StudSubj
        such that subjName.equals("WTW114")
            and ( passLevel = StudSubj.PASSED
                or passLevel = StudSubj.REGISTERED )
    there is a StudSubj called ?x
        such that subjName.equals("WTW114")
            and passLevel = StudSubj.OPTIONAL
THEN
    retract ?x
    assert StudSubj ( "WTW114", StudSubj.LEVEL100, 11, StudSubj.REGISTERED,
StudSubj.BOTH_SEMESTERS )

Rule ITRequirement1
WHEN
    there is no StudSubj
        such that subjName.equals("COS110")
            and ( passLevel = StudSubj.PASSED
                or passLevel = StudSubj.REGISTERED )
    there is a StudSubj called ?x
        such that subjName.equals("COS110")
            and passLevel = StudSubj.OPTIONAL
THEN
    retract ?x
    assert StudSubj ( "COS110", StudSubj.LEVEL100, 11, StudSubj.REGISTERED,
StudSubj.BOTH_SEMESTERS )


Rule Optional1
WHEN
    there is no StudSubj
        such that subjName.equals("COS120")
            and ( passLevel = StudSubj.PASSED
                or passLevel = StudSubj.REGISTERED
                or passLevel = StudSubj.OPTIONAL )
THEN
    assert StudSubj ( "COS120", StudSubj.LEVEL100, 11, StudSubj.OPTIONAL,
StudSubj.BOTH_SEMESTERS )


Rule ITRequirement2
```

WHEN
    there is a **StudSubj** called ?x
        such that subjName.equals("COS110")
            and passLevel = StudSubj.PASSED
    there is no **StudSubj**
        such that subjName.equals("COS212")
            and (passLevel = StudSubj.PASSED
                or passLevel = StudSubj.REGISTERED
                or passLevel = StudSubj.OPTIONAL)
THEN
    assert StudSubj ("COS212", StudSubj.LEVEL200, 7, StudSubj.REGISTERED, StudSubj.BOTH_SEMESTERS)

Rule ITRequirement3

WHEN
    there is a **StudSubj** called ?x
        such that subjName.equals("COS110")
            and passLevel = StudSubj.PASSED
    there is no **StudSubj**
        such that subjName.equals("COS213")
            and (passLevel = StudSubj.PASSED
                or passLevel = StudSubj.REGISTERED
                or passLevel = StudSubj.OPTIONAL)
THEN
    assert StudSubj ("COS213", StudSubj.LEVEL200, 7, StudSubj.REGISTERED, StudSubj.BOTH_SEMESTERS)

Rule ITRequirement4

WHEN
    there is a **StudSubj** called ?x
        such that subjName.equals("COS110")
            and passLevel = StudSubj.PASSED
    there is no **StudSubj**
        such that subjName.equals("COS221")
            and (passLevel = StudSubj.PASSED
                or passLevel = StudSubj.REGISTERED
                or passLevel = StudSubj.OPTIONAL)
THEN
    assert StudSubj ("COS221", StudSubj.LEVEL200, 7, StudSubj.REGISTERED, StudSubj.BOTH_SEMESTERS)

Rule ITRequirement5

WHEN
    there is a **StudSubj** called ?x
        such that subjName.equals("COS110")
            and passLevel = StudSubj.PASSED
    there is no **StudSubj**
        such that subjName.equals("COS222")
            and (passLevel = StudSubj.PASSED
                or passLevel = StudSubj.REGISTERED
                or passLevel = StudSubj.OPTIONAL)
THEN
    assert StudSubj ("COS222", StudSubj.LEVEL200, 7, StudSubj.REGISTERED, StudSubj.BOTH_SEMESTERS)

```
Rule ITRequirement6
WHEN
    there is a StudSubj called ?x
        such that subjName.equals("COS110")
            and passLevel = StudSubj.PASSED
    there is no StudSubj
        such that subjName.equals("COS283")
            and ( passLevel = StudSubj.PASSED
                or passLevel = StudSubj.REGISTERED
                or passLevel = StudSubj.OPTIONAL )
THEN
    assert StudSubj ( "COS283", StudSubj.LEVEL200, 7, StudSubj.REGISTERED,
StudSubj.BOTH_SEMESTERS )

Rule ITRequirement7
WHEN
    there is a StudSubj called ?x
        such that subjName.equals("COS110")
            and passLevel = StudSubj.PASSED
    there is no StudSubj
        such that subjName.equals("COS284")
            and ( passLevel = StudSubj.PASSED
                or passLevel = StudSubj.REGISTERED
                or passLevel = StudSubj.OPTIONAL )
THEN
    assert StudSubj ( "COS284", StudSubj.LEVEL200, 7, StudSubj.REGISTERED,
StudSubj.BOTH_SEMESTERS )

Rule ITRequirement8
WHEN
    there is a StudSubj called ?x
        such that subjName.equals("COS120")
            and passLevel = StudSubj.PASSED
    there is no StudSubj
        such that subjName.equals("COS301")
            and ( passLevel = StudSubj.PASSED
                or passLevel = StudSubj.REGISTERED
                or passLevel = StudSubj.OPTIONAL )
THEN
    assert StudSubj ( "COS301", StudSubj.LEVEL300, 7, StudSubj.REGISTERED,
StudSubj.BOTH_SEMESTERS )

Rule CompulsoryCourse1
WHEN
    there is a StudSubj called ?x
        such that subjName.equals("WTW114")
            and ( passLevel = StudSubj.PASSED
                or passLevel = StudSubj.REGISTERED )
    there is no StudSubj
        such that subjName.equals("PHY171")
            and ( passLevel = StudSubj.PASSED
                or passLevel = StudSubj.REGISTERED
```

```
                    or passLevel = StudSubj.OPTIONAL )
THEN
    assert StudSubj ( "PHY171", StudSubj.LEVEL100, 22, StudSubj.REGISTERED,
StudSubj.BOTH_SEMESTERS )
```

Rule CompulsoryCourse2

```
WHEN
    there is no StudSubj
        such that subjName.equals("ERS220")
            and ( passLevel = StudSubj.PASSED
                or passLevel = StudSubj.REGISTERED
                or passLevel = StudSubj.OPTIONAL )
THEN
    assert StudSubj ( "ERS220", StudSubj.LEVEL200, 8, StudSubj.REGISTERED,
StudSubj.BOTH_SEMESTERS )
```

Rule CompulsoryCourse3

```
WHEN
    there is a StudSubj called ?x
        such that subjName.equals("ERS220")
            and passLevel = StudSubj.PASSED
    there is no StudSubj
        such that subjName.equals("ERS320")
            and ( passLevel = StudSubj.PASSED
                or passLevel = StudSubj.REGISTERED
                or passLevel = StudSubj.OPTIONAL )
THEN
    assert StudSubj ( "ERS320", StudSubj.LEVEL300, 8, StudSubj.REGISTERED,
StudSubj.BOTH_SEMESTERS )
```

Rule AtLeast4Courses1

```
WHEN
    there is a StudSubj called ?x
        such that subjName.equals("COS110")
            and passLevel = StudSubj.PASSED
    there is no StudSubj
        such that subjName.equals("COS314")
            and ( passLevel = StudSubj.PASSED
                or passLevel = StudSubj.REGISTERED
                or passLevel = StudSubj.OPTIONAL )
THEN
    assert StudSubj ( "COS314", StudSubj.LEVEL300, 7, StudSubj.REGISTERED,
StudSubj.BOTH_SEMESTERS )
```

Rule AtLeast4Courses3

```
WHEN
    there is a StudSubj called ?x
        such that subjName.equals("COS222")
            and passLevel = StudSubj.PASSED
    there is no StudSubj
        such that subjName.equals("COS324")
            and ( passLevel = StudSubj.PASSED
                or passLevel = StudSubj.REGISTERED
```

```
                        or passLevel = StudSubj.OPTIONAL )
THEN
        assert StudSubj ( "COS324", StudSubj.LEVEL300, 7, StudSubj.REGISTERED,
StudSubj.BOTH_SEMESTERS )
```

Rule AtLeast4Courses5

```
WHEN
        there is a StudSubj called ?x
                such that subjName.equals("COS283")
                        and passLevel = StudSubj.PASSED
        there is no StudSubj
                such that subjName.equals("COS332")
                        and ( passLevel = StudSubj.PASSED
                                or passLevel = StudSubj.REGISTERED
                                or passLevel = StudSubj.OPTIONAL )
THEN
        assert StudSubj ( "COS332", StudSubj.LEVEL200, 7, StudSubj.REGISTERED,
StudSubj.BOTH_SEMESTERS )
```

Rule AtLeast4Courses7

```
WHEN
        there is a StudSubj called ?x
                such that subjName.equals("COS110")
                        and passLevel = StudSubj.PASSED
        there is no StudSubj
                such that subjName.equals("COS333")
                        and ( passLevel = StudSubj.PASSED
                                or passLevel = StudSubj.REGISTERED
                                or passLevel = StudSubj.OPTIONAL )
THEN
        assert StudSubj ( "COS333", StudSubj.LEVEL300, 7, StudSubj.REGISTERED,
StudSubj.BOTH_SEMESTERS )
```

Rule AtLeast4Courses9

```
WHEN
        there is a StudSubj called ?x
                such that subjName.equals("COS212")
                        and passLevel = StudSubj.PASSED
        there is no StudSubj
                such that subjName.equals("COS341")
                        and ( passLevel = StudSubj.PASSED
                                or passLevel = StudSubj.REGISTERED
                                or passLevel = StudSubj.OPTIONAL )
THEN
        assert StudSubj ( "COS341", StudSubj.LEVEL300, 7, StudSubj.REGISTERED,
StudSubj.BOTH_SEMESTERS )
```

Rule AtLeast4Courses11

```
WHEN
        there is a StudSubj called ?x
                such that subjName.equals("COS110")
                        and passLevel = StudSubj.PASSED
        there is no StudSubj
        such that subjName.equals("COS343")
            and ( passLevel = StudSubj.PASSED
                or passLevel = StudSubj.REGISTERED
                or passLevel = StudSubj.OPTIONAL )
THEN
    assert StudSubj ( "COS343", StudSubj.LEVEL300, 7, StudSubj.REGISTERED,
StudSubj.BOTH_SEMESTERS )
```

Rule CourseApproved1

```
WHEN
    there is a StudSubj called ?a
        where subjName is called ?x
            and level is called ?y
            and noOfUnits is called ?z
            and semester is called ?s
        such that ( passLevel = StudSubj.TD
                or passLevel = StudSubj.TDH )
    there is no StudSubj
        such that subjName.equals(?x)
            and ( passLevel = StudSubj.PASSED
                or passLevel = StudSubj.REGISTERED
                or passLevel = StudSubj.OPTIONAL )
THEN
    retract ?a
    assert StudSubj ( ?x, ?y, ?z, StudSubj.REGISTERED, ?s )
```

Rule OnlyOnePassLevel1

```
WHEN
    there is a StudSubj called ?x
        where subjName is called ?y
        such that passLevel = StudSubj.PASSED
    there is a StudSubj called ?z
        such that subjName.equals(?y)
            and ( passLevel = StudSubj.REGISTERED
                or passLevel = StudSubj.OPTIONAL )
THEN
    retract ?z
```

Rule OnlyOnePassLevel2

```
WHEN
    there is a StudSubj called ?x
        where subjName is called ?y
        such that passLevel = StudSubj.REGISTERED
    there is a StudSubj called ?z
        such that subjName.equals(?y)
            and passLevel = StudSubj.OPTIONAL
THEN
    retract ?z
```

Rule FacultyRequirement1

```
WHEN
    there is a CourseDetails
        where level100Max is called ?a
    the collection of StudSubj called ?c stored in ( new TestLevelSubj(TestLevelSubj.LEVEL100) )
        where noOfUnits is called ?y
        such that level = StudSubj.LEVEL100
            and ( passLevel = StudSubj.PASSED
                or passLevel = StudSubj.REGISTERED )
        verifies totalNoOfUnits() > ?a
THEN
    execute
        so that System.out.println(((" totalNoOfUnits = " + ?c.totalNoOfUnits()) + " > ") + ?a)
        and ?context.assert(?c)
```

Rule FacultyRequirement2

```
WHEN
    there is a CourseDetails
        where creditsPerSemester is called ?a
    the collection of StudSubj called ?c stored in ( new
TestLevelSubj(TestLevelSubj.REGISTERED) )
        where level is called ?x
            and noOfUnits is called ?y
        such that passLevel = StudSubj.REGISTERED
        verifies totalCreditsPerYear() > ?a
THEN
    execute
        so that ?context.assert(?c)
        and System.out.println(((" totalCreditsPerYear = " + ?c.totalCreditsPerYear()) + " > ") + ?a)
```

Rule TooManyCreditsPerYear

```
WHEN
    there is a StudSubj called ?g
        where subjName is called ?b
            and level is called ?c
            and noOfUnits is called ?d
            and semester is called ?e
            and itemNo is called ?f
        such that passLevel = StudSubj.REGISTERED
    there is no StudSubj
        such that passLevel = StudSubj.REGISTERED
            and ?c < level
    there is a CourseDetails
        where creditsPerSemester is called ?a
    there is a TestLevelSubj called ?x
        such that total > ?a
            and collectionType = TestLevelSubj.REGISTERED
THEN
    retract ?g
    assert StudSubj ( ?b, ?c, ?d, StudSubj.OPTIONAL, ?e )
    update refresh ?x
```

Rule TooManyLevel100Credits

```
WHEN
    there is a StudSubj called ?g
        where subjName is called ?b
            and level is called ?c
            and noOfUnits is called ?d
```

```
                and semester is called ?e
                and itemNo is called ?f
            such that level = StudSubj.LEVEL100
                and passLevel = StudSubj.REGISTERED
        there is no **StudSubj**
            such that passLevel = StudSubj.REGISTERED
                and level = StudSubj.LEVEL100
                and ?f < itemNo
        there is a **CourseDetails**
            where level100Max is called ?a
        there is a **TestLevelSubj** called ?x
            such that total > ?a
                and collectionType = TestLevelSubj.LEVEL100
THEN
        retract ?
        assert StudSubj ( ?b, ?c, ?d, StudSubj.OPTIONAL, ?e )
        update refresh ?x
```