A knowledge based system for the interpretation of site investigation information.

Andy Oliver

A Knowledge Based System for the Interpretation of Site Investigation Information

A thesis submitted to the

School of Engineering
University of Durham

for the degree of

Doctor of Philosophy

by Andy Oliver

November 1994

DECLARATION

I hereby declare that the work reported in this thesis has not been previously submitted for any degree. All material in this thesis is original except where indicated by reference to other work.

STATEMENT OF COPYRIGHT

Contents

List of Figures

List of Tables

Abstract

Geotechnical engineering involves the study of earth materials for construction purposes. A site investigation (S.I.) is an essential preliminary to the construction process, by which geological conditions, geotechnical parameters, and other relevant information which may affect the construction or performance of a civil engineering or building project, are acquired.

This thesis outlines work carried out at the School of Engineering, University of Durham, to apply Knowledge Based Systems techniques to the field of Site Investigation in order to ascertain their usefulness and applicability. To this end a Knowledge Based System (KBS) called SIGMA, System for the Interpretation of Site Investigation Information, has been developed. SIGMA has as its core a relational database to store all aspects of a site investigation, known as GeoTec. SIGMA also contains a number of knowledge bases which hold knowledge about the ground, geotechnical tests and correlations between geotechnical parameters.

SIGMA provides two aspects of interpretation (i) interpretation of design parameters from laboratory and field test results and (ii) the interpretation of ground conditions from borehole records. The first aspect of interpretation is the derivation of design parameters from laboratory or field tests. Values may also be assessed from other qualitative information, such as engineering descriptions of the ground, if quantitative data are not available. An important aspect of the assessment will be data validation by cross-checking of measurements to ensure consistency.

The second aspect involves the interpretation of the ground conditions between the points at which observations are being made (boreholes etc.). The system will

generate an interpretation from the borehole logs by tracing layers across a site and building up a picture of the ground conditions on a borehole to borehole level.

SIGMA's aim is to provide geotechnical engineers with a decision support system that assists them in coming to a decision on the choice of a particular value for a parameter or on a particular interpretation of the ground conditions. In addition SIGMA can provide an important data management role, storing, checking and manipulating the large quantities of data produced from a site investigation. Importantly, SIGMA leaves all the decision making to the geotechnical specialist using the system. Its role is to provide access to as much data as possible from which to form conclusions and to make available a wealth of relevant knowledge.

Acknowledgements

I would like to acknowledge the support and assistance of Dr David Toll throughout this research. Dave's enthusiasm and knowledge of the subject area has been a source of inspiration and motivation. His direction, support and general assistance made the production of this thesis markedly more manageable.

I would like to acknowledge SERC for funding the project and providing the opportunity and support for this work to be carried out.

The staff at the School of Engineering and Computer Science, University of Durham have made my time here fly by. Their humour, generosity and constant support has cheered me, supported me and made me proud to be here. It is slightly unfair to single out individuals, but it is also unavoidable. Peter Attewell, recently retired, proved to be the most thorough and professional academic I have yet to meet. His belief, his vision but most importantly his unique persona has made the time spent working with him enlightening and entertaining. Bernie and Steve in the Civils lab have always been available for advice and encouragement, but on a more general level abuse - thanks lads. Trevor Nancarrow is one of the departments scarest resources and yet was always available for help and advice - and I nearly forgot him! Ian Garrett deserves a special mention for his friendship and uncritical (!) advice throughout my work, but by the way Casper, I have this plan ...

Thanks to Wendy Lister and Janet Barclay for their sensible no-nonsense approach to secretarial work - two of the most serious and hardworking individuals as you're likely to find. Thanks to Olive for endless cups of tea and slices of lemon cake and thanks to everyone else I have forgotten.

Nikitas Vaptismas and Marina Moula were two colleagues and close friends that I have dearly missed since their return to Greece. Playing football without the delights of watching Nikitas is my excuse for my current loss of form. Working here simply

wasn't the same since they left, however Adonis Giolas has been an excellent friend and colleague in their absence. His seeming inability to become flustered or agitated has been a very calming influence in some of the more fraught stages of writing up!

This thesis is dedicated to Viv, the love of my life, who is the simple reason that this thesis has been completed. Her constant support, affection and honesty kept me going when it really seemed that I couldn't go on. I don't know how she has put up with me over the last year, but I only hope that I can repay her in the years to come. Thanks darling.

And finally, my parents. Two genuine and loving people - but I must declare a slight bias - who have given me everything that I have ever wanted in my life and to whom my debt of gratitude knows no bounds.

Oh, and thanks to the glorious Hartlepool United, for providing hours of entertainment in the depths of cold winters and really proving the saying that just when you thought things couldn't get any worse, they always can.

Chapter 1

Introduction

1.1 General

Geotechnical engineering involves the study of earth materials for construction purposes (Institute of Civil Engineers, 1991). The decision to develop a particular site cannot always be taken from a purely engineering viewpoint; geotechnical problems also require the assessment of geological conditions for their solution. A site investigation (S.I.) is an essential preliminary to the construction process, by which these geological conditions, geotechnical parameters, and other relevant information which may affect the construction or performance of a civil engineering or building project, are acquired (British Standards 5930, 1981). The term site investigation is used to encompass the whole process as a preliminary to construction. Throughout this thesis the term ground investigation (G.I.) is used to define those aspects of a S.I. that address themselves solely to sub-surface issues.

A site investigation is capable of producing a wealth of information and such data can be utilised by the geotechnical specialist to ascertain the subsurface ground conditions and values of required geotechnical parameters. The need to effectively manage and control this data has long been recognised, as the following quotation by G.S. Littlejohn in 1990 (Institute of Civil Engineers, 1991) reflects:

" Factual ground investigation data should be digitised by geotechnical specialists to a nationally agreed standard for ease of processing and transfer by computer. This should reduce significantly the time required to sort and assess the large amount of data generated by comprehensive ground investigations. "

The interpretation of this ground investigation data to produce a good understanding of sub surface conditions is one of the main tasks of the geotechnical specialist. This interpretation is carried out by applying experience gained from previous investigations, a knowledge of the site, published literature and the assistance of fellow experts to arrive at suitable hypothesises. As the quantity of the S.I. data increases so does the usefulness of any tools that can assist the engineer or geologist in carrying out this task. Knowledge Based Systems, KBS, can provide this assistance by offering a structured representation of domain knowledge, database access and routines that can assist the user in the carrying out of individual interpretation tasks. Domain knowledge is that knowledge which concerns a particular area - in this case the ground.

A KBS has been developed to assist the geotechnical specialist, which is known as SIGMA, System for the Interpretation of Geotechnical Information. SIGMA aims to provide a decision support role for the geotechnical specialist, allowing situations to be assessed in detail before design stage decisions are reached. SIGMA is described in detail in this thesis along with a discussion of the topics that effect the implementation, design and use of this and similar systems. At the core of SIGMA is the GeoTec database, a relational database capable of storing the data produced from a ground investigation.

SIGMA's role is twofold - to assist in the data management of a site investigation and to assist in the interpretation of the data produced from the investigation. The data management role of SIGMA revolves around the GeoTec database. Modelled on the Association of Geotechnical Specialists (AGS) Electronic Transfer of Geotechnical Data from Ground Investigations (AGS, 1992), the database has been implemented using the INGRES database (INGRES, 1990a). GeoTec can store all the data emanating from a ground investigation including multi-level test data. In addition

GeoTec has a structure to store the complex result of a parsed (broken down into constituent terms) layer description. This parsing is carried out by a module within SIGMA. Interfaces allowing access to the data via SIGMA, data checking routines and direct entry of AGS files all assist in the overall data management of a site investigation. In addition, being mounted on a Sun Sparc2 workstation, the database can be accessed by several users instantaneously from different locations, utilising the networking facilities of both INGRES and the Sun workstation.

SIGMA's second role is that of data interpretation. A parameter assessment module allows ground investigation data to be interrogated via SIGMA in order to identify the value of design parameters at specific locations. If the required parameter has not been recorded at a specific location, other measured data may be correlated to the required parameter in order to give the geotechnical specialist a better basis for their assessment. A borehole interpolation module allows borehole to borehole correlations to be carried out based on the parsed layer description data stored in the GeoTec database. Site wide marker beds are identified allowing a greater understanding of the sub-surface ground conditions.

SIGMA has been designed as a Decision Support System, that is it attempts to provide support to a geotechnical specialist in their decision making process. SIGMA does not have the functionality to change any data resident in the database without the specific authority of the user. Any interpretation results are displayed to the user in order to assist their decisions, not make that decision in their place.

1.2 An Overview of the Thesis

A definition of Knowledge Based Systems (KBSs) is given in Chapter 2, outlining the fundamental principles and main components. This is followed by an introduction to

Database Management Systems (DBMS) and Relational Database Management Systems (RDBMS). The manner in which RDBMS and KBS can be combined is then outlined by an examination of the varying linkage options. This leads on to a discussion of the emerging area of Knowledge Based Management Systems (KBMS). The chapter concludes with a discussion of the importance of ensuring the correct design of the interfaces to such systems as SIGMA and an introduction to the different methodologies available.

Chapter 3 outlines those Geotechnical KBS's that have specifically addressed themselves to site characterisation. A full literature review of **all** geotechnical KBS's was deemed outside the scope of this thesis. This is followed by a discussion of geotechnical databases, starting from the early punch card systems through to today's site specific Personal Computer (PC) based data management systems. The chapter is concluded with a discussion of the advantages of operating a national borehole database.

Chapter 4 covers the development tools that were used in producing SIGMA. The selection criteria for choosing a hardware platform are discussed along with the system finally chosen, the Sun Sparc2 workstation. This is followed by a description of the various types of software tools available for developing KBSs and a detailed illustration of the development environment finally chosen, ProKappa (IntelliCorp, 1991). The chapter concludes with a description of the RDBMS used for the development of the GeoTec database.

Chapter 5 gives an introduction and a basic description of the functionality of SIGMA. The dual roles of SIGMA, that is data management and data interpretation are discussed in detail. An overview of SIGMA is then given, covering the core role of the GeoTec database, the knowledge bases and the type and structure of the

knowledge stored therein. The processing and analysis modules are mentioned as an introduction to the following chapters in which they are covered in detail.

A full description of the GeoTec database is given in Chapter 6. The design history of is followed by a detailed description of the GeoTec database, its design methodology and final data structure. GeoTec's ability to store parsed soils description information and the ability to store multi-level test data is fully explained. The chapter concludes with a description of the mechanism by which the GeoTec database communicates with SIGMA.

Chapter 7 covers the data handling aspects of SIGMA. The most novel data handling function of SIGMA is its ability to parse a soil description and store this detailed information in the GeoTec database. The necessity for a parser as a component of SIGMA is discussed followed by a detailed examination of the design methodology and implementation of the parser. Its linkage to the GeoTec database, structure, object oriented functionality and rules lead onto a discussion of the operation of the parser illustrated with examples. SIGMA also contains interfaces to the GeoTec database and these are considered in the subsequent section along with the ability to import data in a standard format directly into the database. SIGMA also incorporates data checking routines which allow the user to verify data in the database and this is covered in the final section.

Chapter 8 covers the data analysis facilities of SIGMA. A detailed consideration is given to the parameter assessment module of SIGMA which allows the user to selectively examine data in the database and assess design parameters, either directly or via correlation routines. The borehole interpolation module is then discussed, from its object oriented approach through to an illustration of a session with the system. The borehole interpolation module is an implementation of earlier work at Durham

and as such the chapter covers its essential differences in approach to the original PROLOG system.

Finally a discussion of the system and those matters raised in earlier chapters is covered in Chapter 9. This covers the implementation of information systems to the geotechnical industry and their advantages. A review of the PROKAPPA software performance over the period of the project is also included. This is followed by suggestions for further work and final conclusions

Chapter 2

Knowledge Based Systems and Databases

2.1 Introduction

Knowledge based systems (KBS), are becoming an increasingly common component of the field of information technology. This chapter discusses attitudes to KBS, and compares them with the acceptance of computers in general. An outline of the fundamental structure of KBS's is given, along with a review of the main methodologies in use.

Throughout this thesis, and especially this chapter, the terms 'expert system' and 'knowledge based system' will be used interchangeably. Whilst there is a body of thought on the difference between these terms, (e.g. Mullarky, 1986) they are both used here to represent systems that incorporate the domain specific knowledge of experienced personnel in the field, or from published literature.

This chapter also introduces the concept of databases, their general structure and advantages over other data storage systems, followed by a discussion of the various data models utilised in their design. An overview of the historical evolution of database software is given, outlining their progress from basic punch-card file based systems through to the current Relational Database Management Systems, RDBMS. The combination of database and knowledge based system technology has led to the new field of Knowledge Based Management Systems which is briefly introduced.

The chapter is concluded with a discussion of the human-computer interface, as the importance of this interface can have a great impact on the acceptance of the technology. The principles of good interface design along with the differing tools available to the designer are discussed with reference to the design process for the SIGMA interface.

2.2 Knowledge Based Systems: An overview

Before a discussion of Knowledge Based Systems (KBS), a brief observation on their acceptance within the field of information technology as a whole is useful. Since the first computing devices were introduced to society, there have always been those people who would wish to be over enthusiastic about their potential, as there are also those who would decry them as dangerous machines. Both of these reactions, whilst understandable, are equally unhelpful; the former building up hopes and expectations to a false degree and the later giving rise to a reluctance to adopt a useful technology. As with most technological advances, the reality of progress falls somewhere between the two extremes. However a comparison of the language used to describe both early computers and KBS is interesting.

In 1946 Lord Mountbatten, speaking as President of the Institution of Electrical Engineers, stated (CSS, 1989):

"now that the memory machine and the electronic brain were upon us, it seemed that we were really facing a new revolution ..."

Talk such as this engendered an aura of mysticism and perhaps even fear and distrust around the new technology. What in fact had been developed was a technology that could carry out mathematical tasks on a scale hitherto unimagined, enabling complex

numeric tasks to be completed quickly and efficiently. There were some people who decried computers on the grounds that they would eventually subsume their human operators, that there would be robot slaves and the like. This imaginative talk, understandable in the dawning of a new technological era, looks stale in the light of what has actually occurred to date. There are no robots, no computers, capable of matching the performance of the human brain.

Compare Lord Mountbatten's comment with this claim made in an AI company brochure in the late 1980's:

" 'Expert Systems', sometimes known as Knowledge Based Systems', are capable of working like experienced and skilled staff ..."

Again, comments such as this leads to the assumption that the systems are capable of achieving tasks far exceeding their actual potential. This overstating of the abilities of KBS has engendered scepticism which may lead to public mistrust. Whitby (1988) proposed a code of conduct for those people working in the field of Artificial Intelligence, AI, one of the main recommendations being:

> All professional persons working in the field of AI should take all possible steps to ensure that their customers, other professions and the public are not misled to the degree of intelligence or competence possessed by AI systems. Descriptions and labels suggesting human attributes should be avoided where there is no technical justification for their use. Computer based labels such as 'data', 'computation', 'processing' and so on should be used in preference to human-based labels, such as 'expert', 'inference', knowledge' and 'intelligence'.

Whilst some of these terms have become a seemingly integral part of the language of KBS (*sic*), the main thrust of this missive is clear and, in the author's view a sensible course of action. It is hoped, that like computers, as the use of KBS becomes more widespread and their usefulness in helping people arrive at better informed decisions is seen, they will increasingly become an accepted part of information technology.

2.3 Fundamentals of a Knowledge Based System

2.3.1 Definition of a Knowledge Based System

KBS technology forms an area of research within Artificial Intelligence (AI), a branch of computer science concerned with simulating human intelligence in a computing machine. The term Knowledge Based Systems can be defined in many ways (Adeli, 1988; Maher and Allen, 1987), some of these descriptions are complex, some simple. Some seemingly do not distinguish between KBS technology and conventional programming techniques. A broad statement of definition for knowledge based systems is given by Gaschnig, 1982:

> "Knowledge based expert systems are interactive computer programs incorperating judgement, experience, rules of thumb, intuition and other expertise to provide knowledgeable advice about a variety of tasks".

One of the more meaningful ways to define KBS technology is to highlight where it differs from conventional programming, as shown in Table 2.1:

KBS	Conventional Programming
Symbolic processing	Numeric processing
Separated knowledge from control	Combination of knowledge and control
Ability to reason heuristically	Algorithmic processing

Table 2.1 - Comparison between conventional and KBS programming

Looking at each of these differences separately, a clearer definition of KBS can be obtained.

2.3.1.1 Symbolic Processing

KBS may utilise symbolic processing, whereby a symbol represents data, concepts or behaviour. The ability to process these symbols in a domain specific manner gives the KBS its flexibility. It allows the system developer to model complex structures to assist in the solving of a particular problem without having to operate within restrictive guidelines.

Conventional programming techniques operate in an environment where the structure of data is predetermined and essentially numerical. There are exceptions to this, for example C++ (Borland, 1993), but these exceptions can be seen as an illustration of the coalescence of computing methodologies.

2.3.1.2 Knowledge Separation

In conventional programming techniques there is very little separation between the data contained in the system and the control of the program itself. The two are dependent upon one another for the overall operation of the system. In KBS, the knowledge is distinctly separate from the control mechanisms, or inference engine. The knowledge, or data, can be stored in a structured format, for example knowledge bases or rulesets, and separate inference mechanisms operate on this data to produce results. Both the knowledge bases and the inference mechanism may be modified independently of each other.

2.3.1.3 Heuristic Reasoning

In conventional programming, data is generally provided and processed in an algorithmic manner. Repetitive or iterative processes are carried out on data of the correct form and type. KBS have the ability, as well as carrying out algorithmic processing, to operate on uncertain data or ranges of data. Data can be provided in many forms to the system which will attempt to deduce as much as it can from this input data, using the knowledge contained in the knowledge bases and the inference mechanisms provided.

This is not a rigorous definition, a knowledge based system might not adhere to all these principles but will, in the main, show these characteristics.

2.3.2 Main Components of a Knowledge Based System

In general, most KBS can be considered to consist of three main components:

Knowledge base

The component of a KBS that contains all the information associated with the domain to which the system is applied. This information may be documented definitions, facts, rules and heuristics. Knowledge bases may be organised hierarchically as knowledge trees or as sets of rules (rulebases) The knowledge should be able to be viewed and manipulated independently.

Context (also known as working memory or fact base)

The component of a KBS that contains all the information about the problem currently being solved. Its content changes dynamically and includes information that defines the parameters of the specific problem and information derived by the system at any stage of the solution process.

Inference mechanism

The component of a KBS that controls the reasoning process of the system. The inference mechanism uses the knowledge base to modify and expand the context in order to solve a specific problem.

In addition, a **user interface** is essential in allowing users to operate the system in a simple and easily followed manner using whatever control items and methodologies are required (section 2.9). In commercial systems it is not unusual for the development of the user interface to take up to 70% of the total development effort (Sutcliffe, 1988). A **knowledge acquisition module** may be considered to be a part of the user interface, allowing the users and/or system developers to enhance the scope and breadth of the knowledge bases within the system.

2.3.2.1 Blackboard Model Architecture

A variation of the basic architecture described above is the **blackboard model**. This complex structure is based upon several independent knowledge sources, which can be viewed as knowledge bases, and the use of a hypothetical blackboard (Bundy, 1990). A good analogy of the methodology is several experts sitting around a blackboard, each contributing their own ideas and thoughts, and a chairman who organises their thoughts and attempts to produce correct hypotheses for a solution. In practise, a Monitor controls the flow of the hypotheses through the context, managing the contributions from the knowledge sources and attempting to produce a correct solution. These systems work on levels of hypothesis and are in general complex to design, lengthy in operation whilst producing meaningful and fully explained solutions.

2.3.3 Knowledge Representation

Within the framework of a typical KBS there are several methodologies of knowledge representation which may be applied (Mullarkey, 1987). These methodologies include:

2.3.3.1 Rule based

Rule-based representation schemes utilise a set of rules to store the domain knowledge, sometimes known as **production rules**. These rules take the form of **IF** (*situation, condition, pattern*) **THEN** *action* and the manner in which these rules are executed, or fired, is driven by the inference mechanism. The **IF** clause, or precondition, is matched against a series of facts held in the context of the system, and those rules that apply are fired, producing a new set of facts. These new facts can then be matched against other rule preconditions to achieve the solution to the domain problem. Rule based systems can be stand alone or a subset of a larger system.

2.3.3.2 Frame based

The term frame covers a variety of knowledge representation schemes, for example network or object, but generally the underlying concept is the same. These systems employ a representation of the knowledge of the problem concerned, either utilising slots on objects/frames, or nodes and their interconnections in a network. Alteration of data in certain slots may give cause to action in others, or independent modules, and knowledge may be inherited 'down' from frames precedent in the network.

2.3.3.3 Logic Based / Predictive Calculus

In logic-based systems knowledge is represented as assertions in logic. Logic based languages allow quantified statements and other well defined formulas as assertions. The flexibility of mathematical logic make these knowledge representation systems powerful, like Prolog (Marcellus 1989; Konigsberger and De Bruyn, 1990; Moula, 1993), however the difficulty in handling uncertainty make them unsuitable for some applications

2.3.4 Inference Mechanisms

The inference mechanism of a KBS is the engine of the system, the process by which the problem is attempted to be solved. The two main inference mechanisms are **forward chaining** and **backward chaining**, sometimes known as data driven or goal driven respectively. Forward chaining assumes an initial state of known facts, and progresses though the problem, utilising the knowledge in the system to a goal, or conclusion, state. Backward chaining assumes a goal state or hypothesis and reasons back utilising data or facts to support or discount the assumed hypothesis. **Mixed chaining**, can also be used as a valid problem solving technique and employs a mixture of both forward and backward chaining (hybrid approach).

2.3.5 Uncertainty within Knowledge Based Systems

KBS may also be required to deal with uncertainty in data and inference. Adeli (1988) has discussed various methods that have been employed to deal with uncertain or incomplete information in the knowledge base. The manipulation of uncertain and imprecise knowledge requires appropriate models of inference (Mullarkey, 1987; Benchimol *et al*, 1987).

Uncertainty in itself is a very uncertain area and prone to the subjectivity of the individual or group of individuals whom are assigning that uncertainty (Miles and Moore, 1994). To be able to assign a specific certainty to an event or set of facts, the person must be in full possession of all the facts that can affect that event. In addition, there must be no bias to any one characteristic of the problem, which in itself presents a very difficult property to measure. Gaining knowledge from experts or reference material is relatively a simple task when compared to assigning a certainty rating for that particular piece of knowledge.

The inclusion of uncertainty within a KBS should be decided upon at the earliest stage possible in order to construct the most suitable methodology for dealing with it. The exclusion of uncertainty by no means devalues the system and, depending on the domain, may even increase the systems practicability.

2.4 Software Tools for Developing a KBS

The tools which are available for developing a KBS can be divided into three main categories: a) General Purpose Programming Language (GPPL), b) General Purpose Representational Languages (GPRL) and c) Expert System Shells as described by Mullarkey, 1987. Expert System Development Environments might be added to the upper range of this spectrum, or as an addition to Expert System Shells.

The first category, General Purpose Programming Language (GPPL), includes the conventional procedural languages such as FORTRAN, C, Pascal etc. A number of KBSs have been developed in procedural languages since they offer easy portability among different types of computers and compatibility with numerous pieces of software available in these languages (Adeli, 1987). However, as these languages are mainly oriented towards numerical algorithmic computation they do not provide the most appropriate environment for the development of KBS. The production of non-deterministic systems is only achieved with difficulty, however procedural languages are suitable for producing rule-based systems. Noticeably, some of the more successful development environments are actually written in C.

In the second category, General Purpose Representational Languages (GPRL), symbol manipulation languages are included that have been developed for use in building KBS. These languages are symbolic, that is information is presented in a descriptive

form rather than strictly numerical. The most popular AI programming languages are LISP (LISt Processing) and PROLOG (PROgramming in LOGic).

LISP is the most widely-used language among AI researchers in the United States and was one of the first languages to be directed toward symbolic representation and list processing (Adeli, 1988). PROLOG is a symbolic programming language based on predicate logic. It allows information to be specified in a declarative style and includes a backward-chaining inference mechanism.

Another class of programming languages, the object-oriented languages, have recently been the subject of very active research work in AI (Benchimol, 1987; Adeli, 1988). An object-oriented language is a language which in principle handles only autonomous entities of a single type called objects. Each object is defined by data specific to it (its characteristics) as well as operations and computations that it is capable of executing when a message is sent to it. Objects are capable of inheritance, which can lead to models of a domain problem being programatically created.

Expert System Shells, which form the third category of tools, are software packages recently developed in order to aid in the rapid prototyping of application KBSs. They consist of two of the three main components of an expert system, i.e. an inference engine and a user interface. They usually provide one or more knowledge representation forms and inference mechanisms. Expert system shells provide greater ease of use than a straightforward AI programming language, but the overhead for this ease of use is less flexibility. Adeli (1988) and Benchimol *et al* (1987) describe some of the more popular expert system shells.

Expert System Development Environments can be viewed either as a separate level of tools, (Mullarkey, 1987), or as an extension of the Expert System Shell. They contain all the main components required to produce a KBS, within an environment which

also has software tools, editors, debuggers, workbenches and the like. Importantly, they usually provide a variety of knowledge representation forms and inference mechanisms, moving away from the traditional view of, for example, either a rule based system or a frame based system. This gives the development environments a flexibility unachievable with an AI programming language but that flexibility is constrained by the software environment. The developer may only develop systems that the environment is capable of producing, whilst using an AI language, greater variation within a more limited scope may be applied. Allwood *et al* (1987) draw attention to some experiences gained from evaluating a number of commercially available expert system shells and development environments.

Detailed analysis of the fundamental characteristics of KBSs, the available techniques for their development as well as their capabilities and potential applications are presented in the published literature (Maher, 1987; Adeli, 1988; Benchimol *et al*, 1987).

2.5 Databases

2.5.1 What are databases?

The word database can be used to refer to any large pool of data collected together at one location. Specifically, a database must be organised so that it can serve the data requirements of different applications, setting a standard data format. The term database relates to the physically stored data and the software required to allow that data to be stored (Bamford and Curran, 1987). A database can also be defined as a computer-based record keeping system i.e. a system whose overall purpose is to record and maintain information. The information contained can be anything that is deemed to be of significance to the organisation and/or application that the system is designed to serve, (Chahine and Janson, 1987). The Database Management System,

DBMS, provides the user with an interface to the physical data stored in the database and those routines to ensure data integrity and manipulation.

The history of database software dates back to the early 1960's when the first database systems were developed by individual companies to solve their particular problems. In the mid 1960's the first general purpose packages became available. Perhaps the most famous of such packages was developed by the General Electric Company (GEC) called the integrated data store (IDS), originally designed to run specifically on GEC machines. B.F.Goodrich saw the work that was being done at GEC and decided to implement IDS on the new IBM system 360 range of computers. John Cullinane entered into a marketing agreement with Goodrich to produce the Cullinet IDMS DBMS, the dominant force in database technology on IBM mainframe machines up to the 1980's

In 1969 a technical group operating under the auspices of CODASYL (Conference on Data Systems Language) produced the specification of a common database facilities which was strongly influenced by IDS and IDMS. The CODASYL model has been enhanced over the years to standardise the facilities of a range of DBMSs.

In 1970 an IBM scientist, Dr E.F. Codd, published an influential paper on database architecture (Codd 1970). Researchers at IBM used the material in Codd's early publications to build the first prototype relational DBMS called system/R. This was emulated at a number of academic institutions, perhaps the foremost example being the INGRES research team at the University of Berkeley, California (Stonebraker, 1986). During the 1970s and early 1980s relational databases got their primary support from academic establishments. The commercial arena was still dominated by IDMS-type databases. In 1983, IBM announced its first relational database for large mainframes - DB2. Since that time, relational databases have grown from strength to strength.

An analogy frequently used to aid in the definition of a database is that of a conventional office filing system, where documents are stored in folders of a specific drawer of a filing cabinet. When a particular document is required to be extracted, a separate external indexing system is consulted to identify the location of the document, i.e. a Roladex or equivalent, and then the document can be extracted directly. If documents covering a range of data groups i.e. a set of companies, is required, this grouping must be carried out manually, indexes consulted and then the individual documents extracted until the group criteria is satisfied. Once the filing cabinet is filled, the oldest documents are placed in an archive, leading to longer access times to extract data if it is required at a later date.

In database systems, data can be stored in a formatted and easily accessible manner, the indexing is carried out by the system itself and most database systems can handle group selection. Large quantities of data can be stored and accessed, and if archiving is required the use of external backup media, such as magnetic tapes, floppy and Winchester disks, enable ease of access.

This simple analogy is often used to outline the function of a database, but it also serves to highlight the benefit of utilising information technology to store data, rather than using paper-based systems. The most important of these advantages are:

Fast data access.

Modern information technology allows thousands of data records to be grouped, sorted and displayed in a matter of seconds. The performance of high level database systems are not significantly affected by the quantity of data being processed. This ability is one of the mainstays of information systems, whereby the scope of the processing/analysis of data increases exponentially as the transition from paper-based to silicon based technology is implemented.

Improved data storage.

Paper can wear out and become unreadable and be easily damaged by fire or flood. Unless duplicates are taken, the actual piece of paper represents the one and only copy of whatever it contains. Also paper-based systems are handling dependent, that is the more they are accessed the more they wear out. With electronic data storage the availability of backup systems and the application of a thorough backup policy, the safety of the data should always be ensured. Electronic data storage media such as silicon disks and tape streamers have much greater longevity than their equivalent paper-based systems.

Electronic data transfer.

The use of electronic data storage allows data to be transferred to similar systems much faster than the corresponding paper based systems. With the advent of high speed telephone lines, satellites and fibre-optic cables, data can be transferred around the world in a matter of seconds. The continuity of data this offers, as stated in Chapter 5, presents real advantages for data management.

2.6 The Structure of a Database

The essential basic element of the database is the record. A record consists of a number of fields, or columns, that are related together and describe the same object. If a database is considered to be a table where all the information is stored, each record corresponds to a row of this table (tuple). Each record should be unique, so that when any manipulation operation is performed on the data, the exact record is accessed. This uniqueness is obtained by providing a key field or combination of fields, that will uniquely identity each record. The key fields have the property that, knowing their value, one can identify the values taken by other fields of the same record.

The main advantage of using a database is that it provides centralised control of its operational data. By providing such control of the data, several other advantages are obtained (Date, 1983).

1) Redundancy can be reduced.

In non database systems, each application has its own private files. This can often lead to considerable redundancy in stored data. When similar data resides in several different files, the updating procedure can become clumsy and processor inefficient. This inefficient processing counteracts one of the technology's main advantages over other systems, namely high speed processing.

2) Inconsistency can be avoided.

If a unique identifier is assigned to each record, no duplication of records will be allowed. This will eliminate the risk of having two inconsistent records for the same object. This occurs when, due to duplication, data in some file systems are not updated, leading to different values for the same data item.

3) The data can be shared.

Different databases and applications can share the stored data without having to duplicate this information for each object.

4) The data can be standardised.

By building a database, the information stored and its format can be standardised. This is particularly desirable in cases where data is being interchanged between different systems and where it is necessary to have the information stored in a valid and complete form.

2.7 Data Models

All of the database systems currently available are based on one of three recognised data models. These are the hierarchical, the network and the relational models (Benyon-Davies, 1991). The hierarchical model is a direct extension of commonly used file processing methods. Basically, data is organised hierarchically in relationships of ownership and stored in files. A single database can be made up of several separate files, each file concerned with a separate data group. This methodology, whilst still in use, has been superseded by the two later data models, mainly due to data redundancy. This redundancy is inherent within the hierarchical model, as data separation occurs at a fundamental stage in the design of the database and therefore the storage of similar data produces unnecessary duplication.

The network (CODASYL) model extends the hierarchical model, by introducing the concept of a network. Each entity or record within the system is joined to other relevant entities by a system of pointers. Like the hierarchical model, the data can be contained in sets of files, or directly onto a memory device such as a magnetic tape or disk, or the writable portion of computer memory. Whilst this model is far more efficient than the hierarchical model and reduces markedly the data duplication, it suffers from over complication. Complex and rational data structures can be defined and data integrity ensured but at a cost of a great deal of programming effort, leading to the situation where a great deal of experience is required to navigate one's way through the database.

Finally, the relational model organises data into one uniform representation. Everything in a relational database is represented in the form of two-dimensional tables related together by common attributes. This simple yet flexible orthogonal nature of the relational model allows for complex and efficient structures to be defined, whilst allowing simple design tools to aid the systems designer. The

relational model has become the most frequently used data model since its introduction in the 1980's, surpassing both the hierarchical and network models.

A Relational Database Management System, RMDBS can be considered a pool of shared resources that can be used to access and maintain a relational database. The RDBMS acts as an interface between the end-users, application programmes and the database itself, allocating storage, providing security and handling all the demands of traditional file-based processing.

These three data models are the basis for the majority of all database packages currently available. Some custom written systems are available that do not strictly adhere to any of the models, but these have tended to arise from a specific application and then been commercialised as an afterthought. A great deal of research is looking into object oriented databases, OODB's, whereby the DBMS actually resides within, and is part of, an object oriented environment. Some applications, for example Computer Aided Design (CAD) and hypermedia systems, require a flexibility that the fixed relational model is unable to supply (Ishikawa *et al*, 1991). OODB's seem to provide a solution to this flexibility but as yet the technology is still undergoing development (Kim, 1990).

With the growth in the use in Personal Computers (PCs), many database packages have been created specifically to operate on this platform. Some of these, DBase or Clipper for example, have progressed to become a standard for PC database systems and have been implemented on other platforms, e.g. UNIX, VAX. These systems utilise a ASCII file as the database, comprising a header, the data itself and an end of file marker. The header contains the information which informs the database software how many fields of data there are in a row, a simple format specifier and/or its location. A separate index file is associated with each database allowing fast data

retrieval and key fields can be defined within the database on which sorting may be carried out.

Several database files may be linked together to form a relational type data structure, key fields being used to link the structure together. These PC systems offer a great deal of flexibility, as the data structures need not conform to any one format. Moreover the performance of the more recent additions to the market, FoxPro for example, is comparable with workstation based systems.

However, this traditional file based approach can result in serious limitations in the final software. The most important of these is uncontrolled redundancy, as previously mentioned in section 2.6.1. In addition, when large quantities of data are being manipulated, the limitations of file based systems become apparent, that is the processing speed is dependent upon data volume. PC based database systems, whilst having flexibility, do not have the multi-user and security capabilities offered by the more sophisticated workstation or mainframe based database products. The ability to have several users interrogating a database consisting of 30 plus tables is simply not achievable with many PC based systems.

The maintaining of industry wide standards is not assisted by the flexibility offered by these file based systems. The decentralisation of both systems design and operation may lead to the different names and formats being used for the same data item, making the sharing of data impractical. As standardisation and centralisation becomes the aim of many information based systems, this flexibility may be seen as detracting from file based systems versatility. Sophisticated RDBMS's, perhaps in conjunction with PC based sub-systems, provide the stable environment for the application of industry wide standards and efficient data management.

2.8 Databases and Knowledge Based Systems

As the design and implementation of Knowledge Based Systems (KBS) continues, the introduction and gradual integration of databases into the field has added to the technology as a whole. The utilisation of databases to store, and in some form process, knowledge is of increasing use and has led to a new area of research, namely Knowledge Based Management Systems, KBMS.

Before a discussion of KBMS technologies it is useful to look at the various methods for linking together a database with a KBS or expert system, (Al-Zobaide and Grimson, 1984; Jarke and Vassiliou, 1984). These can be broken down into the following:

1) An enhanced database system.
2) An enhanced KBS.
3) An interdependent KBS and database.
4) A higher order synthesis. Direct KBMS.

The first two types are examples of an evolutionary approach to the integrating of database and KBS. This approach treats databases and/or KBS as starting points and moves in an evolutionary fashion towards the goal of a KBMS. However, there is a strong argument in employing the known strengths of each tool and allowing them to communicate down a common data channel to solve the sort of "intelligent" tasks required by the application, that is interdependent *expert and database systems*. The final choice, that of a higher order synthesis of the two technologies, is the most revolutionary approach to the problem. As yet, no systems of this nature exist, although the theory has been defined.

2.8.1 Enhanced Database Systems

By definition this method uses an existing DBMS and is complemented by the addition of a deductive component. This addition can be carried out in three different ways, namely:

a) Embedding. Deductive routines are embedded within the DBMS itself, and act as another facility, or command, of the DBMS (see figure 2.1a).

b) Filtering. User and application program queries are directed through a deductive component before being processed by the DBMS. In this way, the deductive component acts as an interface between the user/application programme and the DBMS (see figure 2.1b).

c) Interaction. The DBMS interacts with the deductive component rather than the user or application programme (see figure 2.1c).

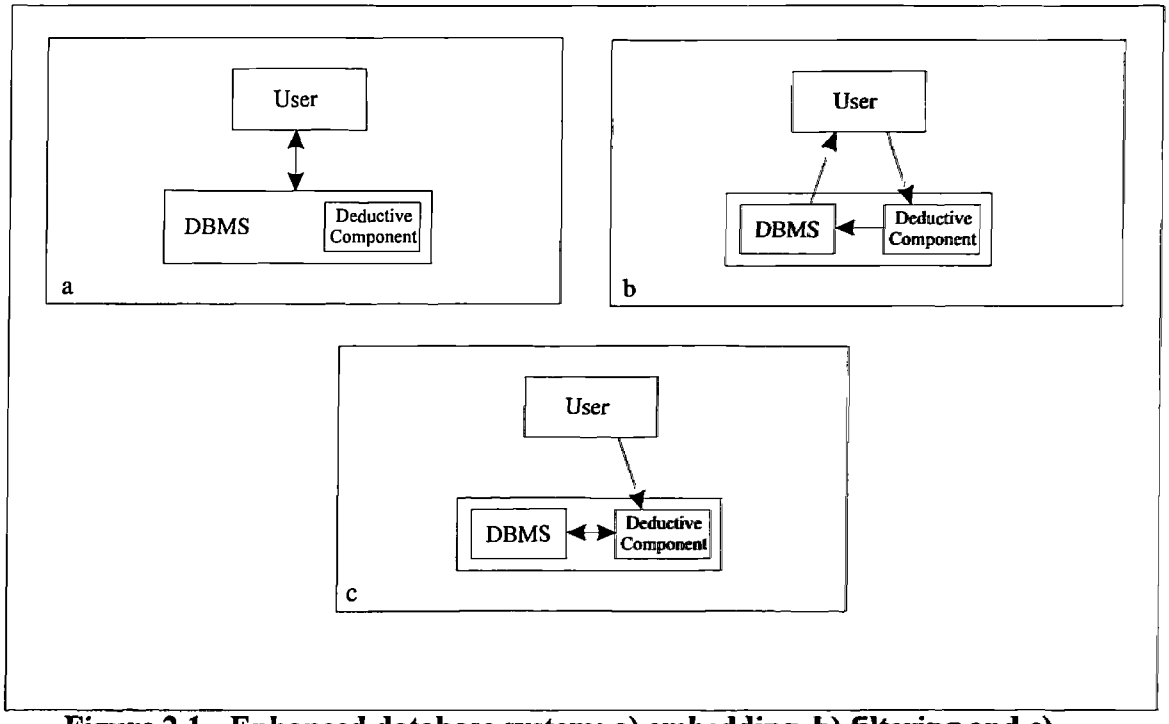


Figure 2.1 - Enhanced database system: a) embedding, b) filtering and c) interaction

This approach relies heavily on direct access to the commands and protocols of the DBMS, as well as a reliance on programmers and developers with experience in the field. For these reasons this methodology is used by database led projects with a requirement for only a specific type of deductive component, for example integrity constraints within real time systems and mechanisms for handling incomplete data within the DBMS environment.

2.8.2 Enhanced KBS

The extension of an KBS to incorporate database facilities can be achieved with either internal or external enhancement (Jarke and Vassiliou, 1984). With internal enhancement, the programming language in which the KBS is written is extended to allow database facilities, in effect giving the KBS its own DBMS (see figure 2.2). With systems written in such languages as PROLOG this is achievable and preferable (Walker, 1984). However for larger shells or development environments the computational effort required to produce a DBMS subset is prohibitive.

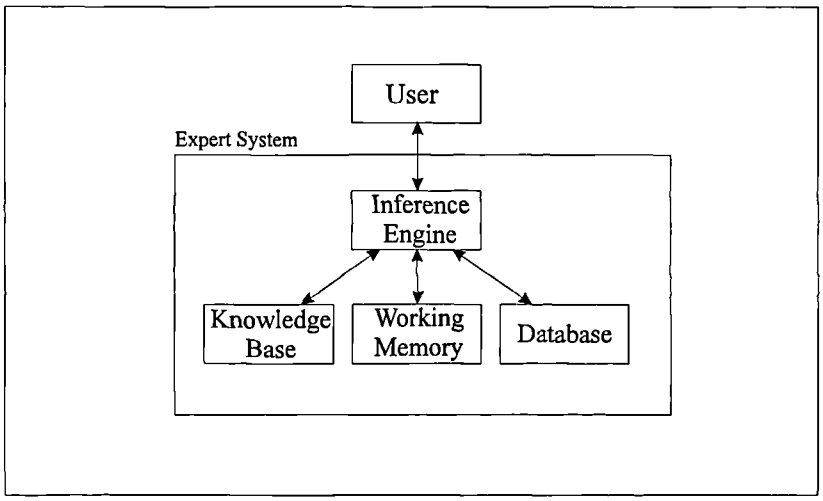


Figure 2.2 - Enhanced KBS: internal enhancement

External enhancement requires the inference engine of the KBS to be provided with direct access to a general purpose, external DBMS. There are two approaches to how this access is gained, namely:

1) Loose coupling. In loose coupling, there is no dynamic link between the database and the KBS (see figure 2.3). The data is delivered to the KBS as a snapshot prior to execution of the system and when this data has been processed, new data may be requested from the DBMS (Missikoff and Wiederhold, 1986). This methodology works efficiently in those situations where batch data retrieval is appropriate. However, due to the distinct separation of the deductive and data retrieval phases, the system has no method in which to store data dynamically. Secondary mechanisms have to be set up to allow memory paging. In addition, if the database that is being used is updated whilst the KBS is in operation, inconsistencies may occur.

2) Tight coupling. In tight coupling, data is retrieved from the database as and when required during the execution of the KBS (see figure 2.3). The DBMS therefore acts as a slave to the KBS. This overcomes many of the problems associated with loose coupling, but only at a cost of impaired performance due to the system overhead required to maintain a dynamic link to the database (Benynon-Davies, 1991). However, in a development environment, this cost can be seen to be justifiable.

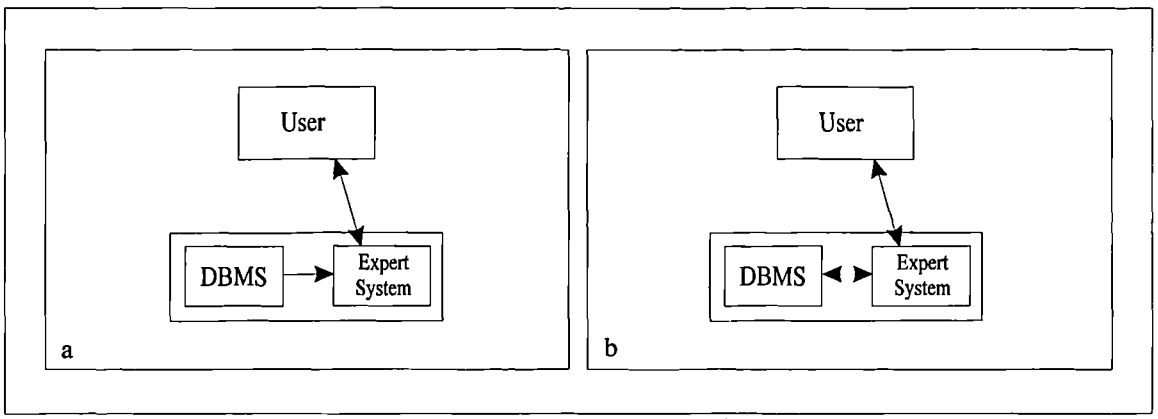


Figure 2.3 - Enhanced KBS: external enhancement a) loose coupling and b) tight coupling

2.8.3 Interdependent KBS and Databases

Allowing both the database and the KBS to stand independently whilst communicating through a common data channel allows both systems to be used as stand alone units or in conjunction with one another (see figure 2.4) . This approach is most suitable where existing systems are required to be enhanced by addition of another. The major problem with this interdependency revolves around the decision as to where the overall control of the system interaction and processing resides. If the control is distributed between the two systems, with interaction via message passing, there is an inevitable problem of data integrity and redundancy.

A more suitable solution is to have distributed processing but with control residing in an independent subsystem, managing the interaction between the database and the KBS (see figure 2.4). This more stable solution is better suited to real-time applications whereby both systems may be interacting at several levels simultaneously, however with a considerable economic and resource overhead.

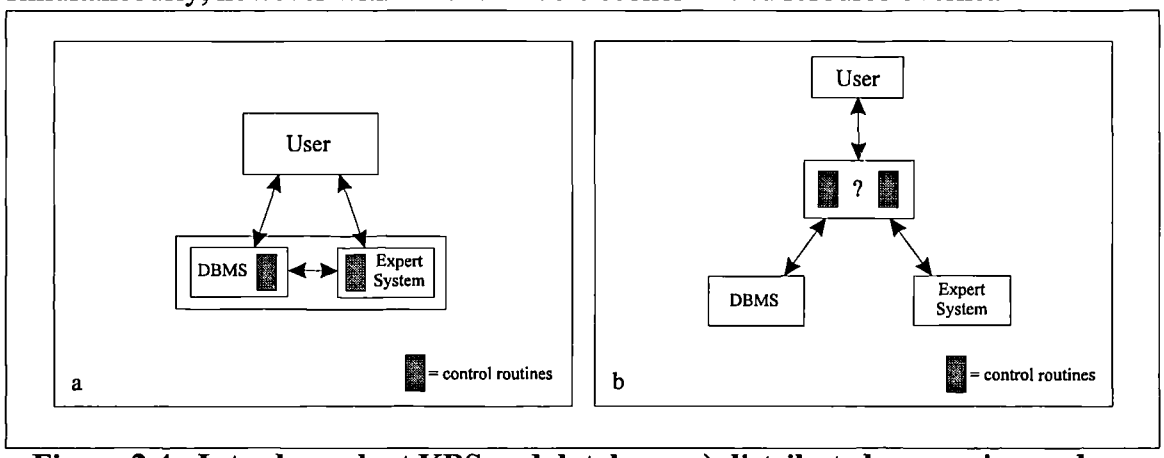


Figure 2.4 - Interdependent KBS and database: a) distributed processing and control and b) independent subsystem

2.8.4 Knowledge Based Management Systems

The approaches described above are all evolutionary approaches to the problem of building a system to manage knowledge, that is the bringing together of database and KBS. There is a body of thought that have suggested that a true knowledge based management system is unlikely to come form the wedding of existing technologies.

They maintain that a search for a higher order synthesis is required. In other words, that an approach is required that embraces under one umbrella the facilities of both KBS and database systems.

A number of initial proposals have been made in this area, for example semantic data modelling (Mylopoulos,1989), object-oriented databases (Navathe, 1989; Ullman, 1989) and first order logic (Galliere and Minkler, 1978). In particular, the combination of logic with database technology, the formation of deductive database systems, is an area of continued interest. The work carried out by Clocksin and Mellish (1981), Walker (1984) and Kowalski (1985), in showing that the use of such languages as PROLOG can be applicable in the formation of logic based database systems, has been a significant contribution to the field.

The KBMS approach may be the most appropriate and thorough manner for the synthesis of technologies to appear, but its further application and theories are beyond the scope of this thesis.

2.9 The importance of Interface Design to Knowledge Based Systems

In designing practical KBS that can be used in geotechnical engineering it is important to plan the way in which the system will interact with the end user. Interface design has been an issue in the information technology field and associated industry since the early 1970's, (Martin, 1973) and its importance to the acceptance of new technology has long been recognised. People have realised and complained for a long time that computer systems are difficult to use, obtuse and jargon-riddled. By and large, users had to put up with this state of affairs because computer programmers took no notice of their complaints. The rise of human-computer interaction, or interface, as an active discipline correlates well with the rise of the microcomputer

and of late the Personal Computer (PC), possibly due to the availability of computers and their software to the population at large. As systems have become more information intensive the core nature of systems has moved away from being a super powerful calculator and towards that of an efficient information processor and the importance of the interface has increased.

Computing systems are becoming increasingly interactive. As they do, the amount of code required to manage the input and output, that is the interface, has risen to accommodate this transfer of human/machine data. Knowledge based systems are very information intensive and as such it is very important not only to get the interface right but also to ensure its efficiency. It is estimated that most commercial decision support and information systems have between 70 and 80 per cent of their code devoted to interface handling (Sutcliffe, 1988).

If poor interfaces are used the consequences can be far ranging, from simple errors to system rejection. Increased mistakes in data entry and system operation are expensive both in the time lost in correcting them and in errors that go uncorrected. Incorrect data residing in databases and knowledge bases can have a compounded error effect, see Section 2.6. Badly designed interfaces can also to lead to user frustration, manifesting itself in low productivity or under utilisation of the system, as the users tend to avoid usage due to the unfriendliness of the task. This may also be a cause of additional data errors being introduced to the system

If the interface is over designed then the performance of the overall system will decline. Machine resources will be devoted in the main to interfacing rather than the software's main task. This is both inefficient and ineffective. In the extreme case, users will simply not use a system, either utilising replacements or organising tasks so as to avoid the system. It would be unfair to attribute this to purely bad interface design, poor system requirements analysis or machine quality could also be to blame.

2.9.1 Principles for user interface design

In order to fully understand the requirements of a good interface, much research work has been carried out in the field of user psychology (Card *et al*, 1983: Christie *et al*, 1987; Lindsay and Norman 1977; Sukaviriya 1993). This has given an insight into how the human acts during interaction, areas that should be avoided and areas that should be encouraged. These have been subdivided into several principles that should be adhered to in the design of the interface, although rigorous enforcement is not necessary or possible, they are merely guidelines.

Consistency - similarity of patterns and in presentation of information. Consistency reduces human learning load and increases recognition by presenting familiar patterns. The human mind is excellent at pattern matching.

Compatibility - New designs should be compatible with, and therefore based upon, the users previous experience. Obviously with the introduction of a new technology to an area such as geotechnical engineering this may not always be possible.

Economy - Interface designs should reduce the number of operations required by the user to a minimum and lessen the work of the user whenever possible.

Adaptability - Interfaces should be able to adapt to different levels of user, from speed of operation through to the skill level of particular users. When this is not possible the interface should be clear and concise, not laborious for the experienced user yet clear for the novice.

Guidance not control - Interfaces should guide the user through a set of tasks and inform and instruct in the process. The interface should function at the users' pace according to the users' command and should not attempt to control the user.

Structure - Interfaces should be designed to reduce the complexity of a given framework. Information should be presented and organised so that only relevant information is passed to the user in a simple manner.

The first three of these principles - consistency, compatibility and economy - can be grouped together as a measure of the efficiency of the interface, that is how easy is an interface to learn and use. The more approachable and friendly the interface the quicker will be its acceptance and thereby the acceptance of the technology. The later two - structure and guidance and control - are important in gauging the correct approach of the interface. Throughout the design of the interface for SIGMA it is these two groups of principles that have been used as guidelines.

Type	Advantages	Disadvantages
Question and Answer	Easy to use Easy to learn Easy to program	Unsophisticated Slow
Menus	Easy to use Easy to learn Easy to program	Limited choice per menu Can be slow
Icons	Very easy to learn Language independent Easy to use	Requires graphical hardware Requires specialist software Uneconomic on system resources
Form Filling	Quick and easy to use Easy to learn	Unsophisticated Mainly suitable for data processing
Command Language	Quick to use Sophisticated Extensible	Difficult to learn Difficult to program Requires a level of expertise
Natural Language (includes voice)	No learning required Natural communication	Difficult to program Requires knowledge base Verbose input Can be ambiguous

Table 2.2 - Types of Interface Available

Whilst operating within these guidelines there are still a wide range of choices to be made in the type of interface that is to be employed, these are summarised in Table 2.2 (Sutcliffe 1988).

The vast majority of interfaces utilise several of these types, thereby gaining their cumulative advantage. As the ingress of windows based software into the PC and workstation market has increased, so their acceptance and usefulness has grown accordingly. This acceptance in itself helps to fill the principle of consistency, with the familiarity encouraging use and helping to reduce a fear of the system.

Much use should be made of feedback in the interface; if there is a delay whilst processing the user should be informed. Where applicable and possible Help information should be available to the user and in the case of an error, messages should be given to the user advising on any action to be taken.

There is no doubt that the acceptability of any computing system, and especially in areas of new technology, that the look and feel of the interface to the user has a strong impact on its acceptance (Easdon, 1981). The use of prototype systems to demonstrate at an early stage how the system will look is a very useful device. The important feedback these demonstrations can provide can give a direction to the overall design of the interface, with many changes being implemented as a result of the demonstrations and ensuing discussions.

2.10 Conclusions

As KBS become more widespread throughout the field of information technology they are coming to be seen as a useful addition to the range of tools available to the engineer. Their ability to process symbolically as well as numerically increases their versatility and their ability to store knowledge in a structured format allows for easy access. Important to the acceptability of such systems is the design of interface which are consistent with the environment in which they are to be used.

There are several facets to KBS technology, differing knowledge representation schemes, inference mechanisms and overall methodologies and this again increases their flexibility. Some procedural General Purpose Programming Languages (GPPLs) have strict limitations to their overall domain. Whilst within their limitations they are flexible. However, if methodologies are attempted that surpass these limitations, processing problems can occur.

Databases offer a centralised data store that can act independently or in conjunction with other systems. The more data is present within a system, the more advantages can be gained from the introduction of data structures through the medium of databases. The relational data model has been shown to be the most suitable for modelling real world systems, producing a clear orthogonal structure.

The combination of both KBS and database technologies has lead to the new area of Knowledge Based Management Systems (KBMS) which may lead to the bonding of logic and database technology. This could provide a powerful environment for the deductive analysis of data.

Chapter 3

Geotechnical Applications of Knowledge Based Systems and Databases

3.1 Introduction

As stated in the previous chapter, Knowledge Based Systems (KBS) are becoming an increasingly common component of the field of information technology. Their applicability within the field of geotechnical engineering is briefly discussed in this chapter followed by a review of those KBS involved with site investigation/characterisation.

A major component of SIGMA is GeoTec, a ground investigation database. Other geotechnical databases are reviewed along with a discussion on the applicability and implications of a national standards applied to both data storage and transfer.

3.2 Knowledge Based Systems in Site Investigation

Site investigation is the process by which geological, geotechnical and other relevant information, which might affect the construction or performance of a civil engineering or building project, is acquired (Clayton *et al*, 1982). Details of a site investigation structure, aims and procedures can be found in the BS5930 and Weltman and Head (1983).

Geotechnical engineering is the area of civil engineering most recognised for the use of expert knowledge, for not only is it concerned with calculation and numeric analysis but also with ideas, concepts, judgement and the deployment of experience which

cannot be represented numerically (Moula, 1993). This is neatly summarised by Peck (quoted by Tomlinson, 1986) when he states:

"If the techniques of soil testing and the theories had not led to results in accord with experience and field observations, they would not have been adopted for practical, widespread use. Indeed, the procedures are valid and justified only to the extent that they have been verified by experience."

Geotechnics has, therefore, been seen as an area of civil engineering most suitable to the application of KBS technology. There are numerous development and prototype systems and much literature has been published to review and quantify these (Santamarina and Chameau, 1987; Adeli, 1987; Adams et al, 1989; Moula *et al*, 1994; Ibrahim, 1994). Accordingly, this review is restricted to only those systems that are directly within the scope of this work, that is site investigation and characterisation.

SITECHAR (Norkin, 1985; Rehak et al, 1985) is a proposed KBS component of a geotechnical site characterisation workbench, the purpose of which is to 'provide advice on data interpretation and on inferring depositional geometry and engineering properties of subgrade materials. The other components of the workbench are databases to store the site data, graphics to produce 'alternative stratigraphic images' and network workstations to carry out the numerical and algorithmic processing.

The system's architecture is based in the blackboard model (Section 2.3.2.1) that provides a general structure for a complex problem-solving technique. Individual rule-based knowledge modules, or knowledge sources, are used for solving particular parts of the problem, namely: identification of trends and geometry, soil matching by description, assessment of geology and geomorphology and searching for Marker Beds.

The knowledge is divided into the following classes: the strategic and tactical planning, the micro- and macro-level inferences and the inference scripts. The strategic planning processes select the procedures used to characterise the site, pursue several alternatives in parallel, monitor the progress and redirect the system's "focus of attention". Tactical level processors work for and under the strategic level processors, in order to prove hypotheses, fill the details of the characterisation, access and filter data and drive the production of the graphical output.

Micro- and macro-level inferences represent different hierarchical levels of problem abstraction, namely geometric trend recognition, matching soil descriptions, defining proximity etc., and verifying hypotheses on site geomorphology, lithology, geology, identifying marker beds, respectively. The inference scripts are used to represent the steps in characterising different types of sites and performing various parts of the overall process. These knowledge classes are not necessarily distinct, each class can potentially operate at every level.

The overall control of the system is provided through a single co-ordinating Knowledge Based supervisor, which may be operational over several processors simultaneously. The inference engine, which supports both forward-chaining and backward-chaining, controls the communication and interaction between the blackboard and the knowledge modules. This proposed system provides a very clear implementation of how geotechnical engineers carry out their work in the real world. However, the level of computational complexity required to operate the system combined with the depth and breadth of work required to assemble the independent knowledge sources (particularly producing a structure to allow them to interact at a meaningful level) may cause implementation problems.

CONE (Mullarkey, 1986) is a development prototype KBS for the interpretation of raw Cone Penetrometer (CPT) data. The system takes the raw CPT data as input and carries out a validity scan. A classification of the soil types, including profiling of the layers and the inference of the shear strength of sands and clays, is then attempted by the system. Soil type classification is based on the use of two soil classification systems (Dutch and Douglas & Olsen classification systems) plus a third, which is a fuzzy set representation of the raw data from the Douglas & Olsen classification system (Mullarkey and Fenves, 1986). The shear strength of sands and clays is estimated using both empirical and rational based methods.

Both linguistic data (soil classification) and numerical data (shear strength), along with the incorporated uncertainties (vagueness and statistical variability respectively), are represented as fuzzy sets with respect to the linguistic variable. The soil type for example, is represented as a three element fuzzy set (sand, silt, clay) along with the corresponding numerical values indicating the membership of each element in it. The appropriateness of a soil classification system, the accuracy of the system in respect to certain soil types (Belief), and the relative importance of the inferred information (Weight) are expressed as linguistic variables, again through the implementation of fuzzy sets. The Belief and Weight are used as fuzzy set modifiers incorporating the uncertainty in a certain piece of information (soil type, or shear strength).

The system has been implemented using OPS5 rules and LISP functions. The system, that is classified as a development prototype, has been validated using published cases and proved to be fairly reliable (80% accuracy). However, these case studies were very simple soil descriptions, for example no terms such as *slightly* or *very* were allowed, and further development of a parser would be required to produce more meaningful results. A typical run of CONE may take up to 1.5 hours on a lightly loaded Dec-20, depending on the length of the CPT log.

SOILCON (Siller, 1987) is a development prototype KBS for helping the engineer in deciding the level of geotechnical investigation required for a specific project. The system matches the requirements of a proposed structure with the level of information known about a site and the amount of information required to reduce the risk involved with the subsurface to an acceptable level. The system starts by querying the user for preliminary project and site data. Based partly on the responses to the preliminary questions, higher level questions are then posed to the user, until finally, and depending on the existing available information, the appropriate level of site investigation is recommended. The system contains information for 24 investigation techniques, ranging from very preliminary, such as topographical maps, to more sophisticated, such as the pressumeter. The complexity of the investigation proposed from the system is in direct relation to the amount of site data available. The system was developed using the M1 expert system shell. It provides a backward-chaining control strategy interfacing with a production rule knowledge base. The major shortcoming of the system is the inability in dealing with quantitative geometrical descriptions. The size of the project can only be defined as small, medium, or large, while the foundation geometry is given as shallow, or deep. Despite these limitations the system provides a good example of rule based reasoning applied to the choice of a site investigation.

A simple KBS for site investigation is presented by Alim and Munro (1987). It offers guidance on soil identification based on visual and physical observations of soil characteristics. Given the soil and loading conditions the system provides judgement about the most likely foundation type and then identifies possible foundation problems. Finally based on the above information the most suitable sampling and drilling technique is recommended. The system incorporates a backward-chaining inference engine interfacing with a production rule knowledge base, and handles uncertainty and imprecise knowledge using fuzzy sets. This expert system was written in micro-PROLOG using the PROLOG expert system shell APES. The complete

system exists in six separate files, which are fully compatible with each other and can be used both independently or by loading them all into memory at once. The system suffers from a very simplistic approach to the problem, being limited to only utilising basic textbook knowledge.

SITECLAS [Wong et al, 1989] is an expert system used for site classification according to the Australian Standard AS2870.1. The system was designed to explore two points:

1) The potential of SUCAM, an expert system shell. This shell, written in TURBO PROLOG and running on an IBM Personal Computer (PC) or compatible, can be used to create rule-based backward-chaining systems.

2) The potential of applying expert system technology to geotechnical engineering.

The input required involves information about natural ground or fill, site location, site history and type of footing (if available). The main components of the system consist on a knowledge base, a fact base, an inference engine, a user interface, an explanation facility and modules for different functions. The knowledge base stores the knowledge about a subject in the form of IF-THEN and IF-THEN-ELSE rules, procedures, tables and comments. The fact base stores the specified problem statements and goals, input facts and conclusions, providing the advantage of being able to modify the input facts without starting a new consultation. The backward-chaining inference engine also incorporates additional functions for question generation, explanation, table checking, executing procedural commands, extra reasoning control etc. The system is deterministic in the sense that it does not deal with uncertain, imprecise, or conflicting knowledge The developers of the system have left any non-deterministic solutions to the engineers using the system.

The knowledge in SITECLAS was extracted from the Australian Standard AS2870.1, (which provides a flow chart describing the procedure for site classification) and two very experienced engineers. The knowledge elicited from the above sources was implemented in the system as approximately ninety rules, created in groups, so that it is easier to understand the relationships between them. The system was validated for five different sites and the results were compared with the results from two experienced engineers, and found to be in generally good agreement.

The project also came up with several useful conclusions as to the potential of expert system technology in geotechnical engineering, namely:

1) Effort is required to ensure that the technology is accepted.

2) Custom made tools, those specifically aimed at the geotechnical engineer, aid in the acceptance of the technology, by giving them access to the facilities that they require to solve 'real world' problems.

3) Assisting the engineer to make his/her decision was primary, not making the decision on their behalf.

LOGS [Lok, 1987] is an expert system that treats information from several boring logs and provides the user with two-dimensional subsurface profiles. It is a rule-based forward-chaining system written in the OPS5 and Common LISP languages, and has been implemented in the Knowledgecraft^{TM} environment. The system includes geological and geomorphological knowledge for deposits of glacial origin in a specific region (Kane County Illinois). The knowledge was provided by an expert geologist and by publications furnished by the Illinois State Geological Survey (ISGS). It was implemented in the system in the form of rules and is divided into micro- (soil classification, identification of marker beds) and macro-level (determination of overall trends and specific characteristics of a site) knowledge. Heuristics are incorporated in the system in the form of production rules that use the knowledge base to make inferences. The system tries to identify marker beds, lenses

(wedge-shaped deposits having one terminus within the site), and lentils (strata whose boundaries exist within the confines of the site) consisting of till, lacustrine or outwash which are the major geological conditions observed in Kane County area. An interesting point in the system's inference engine is that a soil can be identified as a continuous layer, based on the geological knowledge of the site even if this is not observed in some of the boring logs, provided that no conflicting data exists. The current version of LOGS comprises approximately 350 rules and future improvements are identified to be three dimensional interpretation and calibration against the judgement of experts.

Smith and Oliphant (1991), describe a prototype KBS for civil engineering site investigation. The prototype system was developed using the Leonardo Development System, Level 3 shell, which runs on an I.B.M. compatible P.C. supporting MS-DOS. The expertise was represented in the knowledge base as rulesets, objects, and object frames. The main ruleset is the central component of the knowledge base. Every application starts with the execution of a rule in the main ruleset. Execution is controlled by the inference engine and the goal of the main ruleset. The goal is the object (variable) whose value is obtained through the knowledge base. The inference engine executes rules in a systematic order, using mostly a backward-chaining strategy, in order to obtain the desired value.

Each object, in Leonardo, has an associated object frame, which in turn consists of a number of parameters (slots) set to specific values during the development of the knowledge base. Object frames can also contain rulesets. The prototype features a systematic data input facility that helps minimise oversights or omissions of relevant data. The data obtained from the planning stages of different site investigations are of a similar form, so it was possible to create multiple choice menus as a means for getting data from the user. The information obtained is used by the system to provide suggestions to the user for the following stage of site investigation, the subsoil

exploration (possible locations of boreholes, trial pits, etc. and suitable types of soil testing). The variability of data returned from the subsoil exploration stage was handled by writing external executable programs. The information obtained at this stage is used for the creation of a 2-D visual representation of the soil layers. The strength characteristics of the various soil strata are used by the system to make crude recommendations for suitable foundation types for the ground conditions present. The prototype system is user friendly, can be used as a learning tool, has a cost saving capability, and provides the facility for future expansion.

Halim et al [1991], describe a KBS for probabilistic site characterisation developed to facilitate the planning of site exploration with emphasis on assessing anomaly statistics. The system has been implemented to perform three major sub tasks. The inference of the prior estimates of soil and anomaly characteristics, which is mainly done by using production rules combined with any additional information provided by the user; selection of the most effective exploration program, using Baysian techniques to update the prior estimates of soil and anomalies characteristics for the different exploration schemes considered by the user, and incorporating the calculated costs; and finally reliability evaluation of the proposed geotechnical system. The system was developed using the knowledge engineering environment KEE. The general problem solving strategy of the system uses a data-driven forward-chaining inference mechanism. The knowledge representation scheme in the system is an integration of frame-based and rule-based representations, allowing both procedural and event-driving programming. Future development of the system involves the inclusion into the system's knowledge of the capability to update the soil properties based on the site exploration results.

Carpaneto and Cremonini (1991), describe an expert system framework for the automation of geotechnical design characterisation. The system is based on an existing expert system (Righetti and Cremonini, 1988) employed for stratigraphic soil

characterisation. The framework consists essentially of various data bases, containing information about the site to be analysed, a knowledge base, containing the domain rules, and an inference engine able to process and to interpret the available data. The procedure leading to the characterisation of a site takes place in four phases. During the Input Phase, information from boring logs, penetration testing (if available), laboratory testing, along with heuristic knowledge about the site (site patterns etc.) is retrieved from the databases and is used to derive a first trial stratigraphy, and recognise the principal constituents of the tested soil samples. The data stored in the working memory during the Input Phase is then cross-checked against rules stored in the knowledge base and possible conflicts are identified and treated accordingly. The Comparison Phase also includes the improvement of the initial complex configuration by combining adjacent layers of the same soil type and the computation of the initial configuration certainty. The Reduction Phase is the next step, where the construction of a "best solution" set is carried on by generation of a logical solution tree. The new configurations are recorded in order of decreasing certainty factors. In the Output Phase, the best solutions detected for the borehole stratigraphy and the corresponding design parameters are processed to provide an appropriate display of the results.

Further developments to the system will focus on improved data base management and graphics software in the system, but more importantly on attempting to enable the system to deal with sites where preliminary data is unknown.

A KBS was developed by Davey-Wilson (1991) for soil shear strength analysis. The system uses soil descriptions as input in order to infer their angle of friction in degrees, to a maximum accuracy of $1^{o} \pm 1^{o}$. The user is queried about the particle size distribution, the grain size, the in-situ density and homogeneity. The more detailed the answers, the higher the precision of the result. The same system is also used for educational purposes to simulate the execution of the laboratory shear box test with step by step interaction with the user, linking geotechnical theory to practice.

Gillette (1991) presents a Computerised Adviser on Soil Strength (CASS), a KBS to assist in the selection of shear strength parameters for use in stability analysis. After preliminary data has been entered by the user, the system attempts to advise on the shear strength parameters ϕ and c, to make recommendations about the strength representation in the analysis, to advice on soil behaviour and give warnings about possible problems. CASS was written using the rule-based expert system shell Personal Consultant Plus (PC+) and runs on an AT-class PC with extended memory. The conclusions are reached using a backward-chaining inference mechanism. Checks on the consistency and validity of the input information are also performed by the system.

CESSOL (Magnan, 1992) is a KBS for planning a site investigation. Based on information about the site, the type of construction envisaged and the type of data required, CESSOL can give qualitative advice on the type of investigation needed, and what sort of testing would be required. It can also give quantitative advice on the number of boreholes and piezometers and amount of testing required. CESSOL was implemented on a PC using LISP with a window style interface. Development started at the University of Savoie and then by the company CRIL and the Laboratoires des Ponts et Chaussées (LPC). A knowledge base of 117 rules contains a large amount of experience from LPC. Magnan suggests that the system was developed due to the enthusiasm of the instigators rather than any need for the system in geotechnical practice. The knowledge is over detailed and too high level for it to be useable by 'ordinary' geotechnical engineers.

SAGITAIRE (Vergobbi *et al,* 1992) is a KBS for processing site investigation data. It can be used to merge data from soil descriptions, classification data from laboratory testing and results from insitu tests to form a final borehole log. It has knowledge bases for identification of soil types (based on the Unified Soil Classification Scheme)

and for processing results from the Cone Penetration Test (CPT). Having identified layering from these different sources, it then tries to simplify the log by eliminating insignificant beds and attemps to identify the major formations. The system was developed in order to aid in analysis of offshore foundations. The authors reported that a commercial version would be available by the end of 1992. SAGITAIRE has been implemented using C++ (object oriented C), CLIPS a rule based shell for inferencing and GPhigs for graphics. It runs on a Unix workstation and uses the X Windows user interface. It can access external databases using DB++ and also makes use of external programs written in Fortran.

Winter and Matheson (1992) and Thomas *et al* (1992) outline a system being developed for assisting in the planning of a site investigation. The system contains knowledge about the different phases and stages of an investigation. An activity log of an investigation can be produced for comparison against a list of mandatory and advisory procedures contained within the system. The system can therefore be used to highlight omissions in the way that an investigation has been carried out which could impair its effectiveness. It is intended for use on trunk road projects. The system was developed using the Leonardo shell running under MS-DOS on a PC (286 or above). The system uses a heirarchy of menus and windows. The rule structure and also the menu heirarchy can be viewed graphically to locate the user within the system.

Ibrahim (1994) has produced a system which incorporates many aspects of a site investigation, including the elicitation of the domain knowledge, representation of heuristic knowledge and the design of a rudimentary geotechnical knowledge based management system. The system also provides for correlation of lithological boreholes, estimation of foundation parameters and a graphical interface. The system has been produced using the Leonardo shell and has been implemented on an IBM compatible PC. The geotechnical knowledge based management system utilises a

simple linked flat file structure which does not comply with the AGS data exchange standard (AGS, 1992).

3.3 Geotechnical Databases

Geotechnical databases have advanced significantly from the early days when developers used conventional languages such as FORTRAN to write data storage and access routines. These early systems used punch card technology to provide data storage covering a wide range of geotechnical areas.

Buller (1964) is credited with the first geotechnical database in use with a system that stored records for the Department of Mineral Resources in Canada. This systems was a rudimentary attempt at an electronic data store which was cumbersome in its operation. To search the database could take repeated passes through various sources, however the concept of a local geotechnical data store had been established.

Rhind and Sissons (1971) implemented a database for the storage of Drift borehole records in Edinburgh, ustilising a combination of numerical and free form text storage. This approach allowed layer description and their associated depths to be stored in an easily accessible manner. The liberal use of abbreviations and mneumonics allowed the data to be compressed whilst retaining the essential meaning. However, as with all the punch card systems the database proved to be so cumbersome to use that its full potential was never exploited.

Cripps (1978) attempted to set up a borehole database that had a natural language interface, that is the database could be queried in pseudo-English. This punch card system could retrieve the details of borehole logs in many formats. Whilst the system proved effective there was no attempt at a data structure other than that of the original

input data. Laterly Day (1984) utilised a similar methodology to Cripps implemented on an interactive microcomputer, allowing for more sophisticated data retrieval and manipulation.

Early database systems provided the storage required by the user however their specialisation rendered them difficult to use by other interested parties and they also suffered from lengthy processing times inherent with their contemporary technology. Since the late 1980s Database Management Systems (DBMS) and procedural languages have been used to implement more sophisticated databases that can be applied to a range of users.

Such a system is Geoshare (Raper and Wainwright, 1987), a geotechnical database implemented over a seven year period using the CODASYL Database Management System running on an ICL 2988 mainframe computer. The aim of Geoshare was to highlight how beneficial centralised data storage could be to the geotechnical community and as such concentrated on efficient data retrieval, manipulation and searching. Data entry was restricted to pre-formated data menus yet within individual data fields free form English could be used. The system proved a successful prototype, yet by contemporary standards the system was cumbersome in use. The research highlighted some interesting aspects to geotechnical data storage. The variation and verbosity of layer descriptions could lead to difficulty in storage and processing of such data. The requirement for access to such systems by skilled and non-skilled personnel alike was identified as an important facet as was the need to produce practical systems - systems that the geotechnical community would actually use.

The application of microcomputer technology to the production of borehole logs led to geotechnical data being stored electronically in data files. Whilst not strictly a database, it is through processes such as this whereby the advantages of electronic

data storage have lead to the foundation of sophisticated geotechnical databases. Howland and Polanski (1985) produced a system to produce borehole logs, written using TA-BASIC and implemented on a Triumph Adler Alphatronic microcomputer. Data was stored in a sequential file system, allowing for relatively simple data manipulation. Data was entered via menu driven routines, a novel approach being the use of self-validation as the data was being input. Simple rules within the BASIC program allowed invalid input statements to be simply corrected. The output of the system was a purely textual borehoe log. The work also identified areas where geotechnical data management could improve the operation of geotechnical companies, for example the automated inclusion of geotechnical data into the invoicing systems

Chaplow (1986) produced a borehole log system that allowed graphical as well as textural data to be presented, utilising a structured data file as a central data store. This data file was of a fixed format and comprised 10 data fields. This data could be edited or updated at any time in the future and utilised removeable floppy disks as the file store. The description of the layer was coded, that is each term within the description was allocated a one or two letter code. Chaplow created a complex coding system to allow the full range of vocabulary to be utilised, however coding systems designed to assist computer systems tend to lead to loss of data. The test of such a system is if it can return to the original description if required. The finished system produced good quality graphical borehole logs in a three stage, time consuming process.

Finn and Eldred (1987) produced a microcomputer system that allowed the production of borehole logs. The system used a data structure that was actually a part of the C program in which the whole system was written. Data entry was by menus and the final borehole log produced was of a very high standard. Systems such as these highlight the advantages for computerised data management within the geotechnical

industry. However, custom systems such as this kept the data as part as an integral part of the package, thereby preventing any form of data exchange (Perry, 1991).

Greenshaw et al (1987) produced the Strata 3 package that was designed as a geotechnical data management facility. Utilising an ORACLE database, the system was implemented on a VAX mainframe computer. The data was input manually and the data structure could be modified by skilled personnel to store a wide range of geotechnical and related data. Once resident in the database this data could be linked across to the GINOSURF surface modelling package that allowed complex stratigraphic images to be produced.' In addition multi-surface isometeric projections could be produced. Strata 3 became available as a commercial product in later years and is still available today, however all the database systems have been replaced. Instead of an ORACLE relational database the system now utilise its own simple ASCII data-file and the product has become an analysis package rather than a geotechnical data management system.

Greenwood (1988) produced a geotechnical database with a view to defining a geotecnical data management system. Using the Revelation database system implemented on an IBM PC, the system allowed laboratory results to be added in the laboratory, field results and drillers logs in the field. This data was then combined into the master database, which was used for producing preliminary borehole logs, sample lists, piezometer and water level details as well as the final borehole log itself. Greenwood identified the requirement for a structured ASCII (American Standard Code for Information Interchange) file to enable inter computer transfer. With the advent of the AGS data exchange standard, this level of inter computer data transfer is now achieveable.

Commercial software packages came available that allowed the user to input data from a site investigation and produce the graphical, tabulated and report output

generally associated with a great deal of manual effort for example *gINT* (Staten and Coronna, 1992), SID/GDMS (MZ Associates, 1994), *TechBASE* (MINEsoft Ltd, Denver, Colorado, USA). gINT is a good example of such a product. Based on a Betrieve data file structure this system runs on a Personal Computer (PC) and has a hierarchical structure to store borehole information. The main aim of the system is the production of borehole logs and related reports, allowing geotechnical data to be presented in a professional manner. It is almost as a byproduct of this aim that data can be stored centrally enabling basic data management tasks to be carried out. Data input is a manual process and requires the filling in of different forms, that is screens. It is assumed that each form corresponds to a data file within the overall data structure, however the actual data structure remains hidden from the user.

The more sophisticated packages provide reporting facilities for presentation of borehole logs, laboratory test reports etc. as well as graphical displays of test results against depth (for example), cross-sections, contouring or fence diagrams. These systems have proved invaluable for the geotechnical profession, allowing professional data output, both tabulated and graphical, to be readily available. The majority of the commercially available systems use a Personal Computer (PC) based flat file data storage system, such as DBase or Clipper. These systems offer the flexibility to meet the demands of a data intensive environment, albeit on a local scale.

Many 'in-house' geotechnical databases exist that utilise existing database products. These systems are designed for the express requirements of one sphere of interest, that is company, government department etc, and are designed to meet precisely their needs. Products such as DBase, FoxPro, Access and SuperBase which are all PC based software, can be used to store detailed geotechnical data for a specific purpose. MacKenzie (1994) has produced a hydrogeological database to data specific to the groundwater investigation requirements of an area in Honduras. The system has been implemented in Superbase for Windows for its ease of use and whilst MacKenzie

admits to the system's unpolished design, the system is functional and in use daily. Systems such as these provide the service that is required, however restrict the transfer of data between other systems due to their own specialisations (Threadgold, 1992).

There have been a number of attempts to set up a national database for the UK. The work on *Geoshare* at Queen Mary College, University of London (Day *et al*, 1983; Rapier and Wainwright, 1987) had this as its aim. The British Geological Survey (BGS) has also made a start on such a system (Forster and Culshaw, 1990) and a national borehole index has been developed for borehole information lodged with BGS. Howland (1991) strongly advocates the transfer of information from industry to the BGS, highlighting the commercial, economic and practical advantages of such a scheme. Whilst a national borehole database would be beneficial to all, data security and the commercial implications of such a system have been highlighted as possible areas of complication (Rodger, 1992).

A more realistic approach to that of setting up a national database has been the task of developing a standard format for interchange of geotechnical information. A standard format has been put forward by the Association of Geotechnical Specialists (AGS, 1992). There have also been other attempts to develop standard data structures such as the joint International Society for Rock Mechanics/Society of Petroleum Engineers initiative on rock properties (ISRM-SPE, 1990). These attempts on standardisation of data structures for Geotechnical engineering must also be seen within wider attempts to develop standard data models such as the International Standards Organisation (ISO) Standard for the Exchange of Product (STEP) model. (Moran, 1990; Watson, 1990).

Only within the framework of a national, or international, standard should the new generation of databases be designed, allowing the benefits of mass information storage and transfer to become apparent - both economic and technical (Brink *et al*,

1988; Mott MacDonald *et al*, 1994). Whilst at first there may be a tendency to horde 'in-house' site investigation data, due to the initial capital outlay involved, in the long term the economic advantage of sharing the data must become apparent. If site investigation data has already been carried out on a particular location, it is both practically and economically foolish to duplicate the effort. Also, with the collection of large quantities of data, the potential for large scale analysis are increased, providing the industry as a whole with meaningful data.

Whilst there are practical reasons why a national borehole database may present difficulties, for example who would regulate the costing of the dissemination of the data, the author believes that the overall effect would be beneficial. If an S.I. contractor could supply data directly into a client's geotechnical database - data integrity and processing could be markedly increased. This data could in turn be passed on to a national database.

3.4 Conclusions

Geotechnical engineering is a very suitable area for the use of knowledge based systems techniques, due to the nature and type of the data produced and the reliance upon expert knowledge in the field, rather than 'codified' procedures. There are several prototype systems in the field, using a variety of software platforms and programming techniques. It is interesting to note in the published reviews of available KBS in civil engineering (Moula *et al*, 1994; Miles and Moore, 1994) that the most favoured methodologies for the implementation of KBS are changing. Hybrid systems utilising aspects of rule-based, frame-based and logic programming are becoming more noticeable as the hardware/software platforms become capable of supporting more complex systems. The flexibility of these hybrid systems allows the developer to bring together the advantages of all aspects of KBS technologies.

Many of the systems have led to a common understanding of both the importance of dealing with the data produced in a meaningful manner and approaching the problem in a structured manner. It is within this framework, especially with a view to improved data interpretation, that the KBS described in this thesis has been developed as a contribution to the field.

Geotechnical databases have progressed from their early punch-card days through to sophisticated database systems to assist the user in the production of borehole logs and customised reports. Historically geotechnical databases have been restricted to the requirements of particular user, be they companies, govermental agencies or individuals. The requirement for a national database in the United Kingdom is growing as the quantity of data and the cost of requiring that data increases. It is hoped that the case for a national database will be strengthened with the successful introduction of the AGS Data Exchange Standard.

Chapter 4

Development Tools

4.1 Introduction

The tools utilised to develop a Knowledge Based System (KBS) such as SIGMA are an important component in the overall process. Once the decision has been made it is very difficult to overturn and therefore due consideration must be made in the early stages of the project to ensure the most effective solution.

The criteria for the selection of a hardware platform for the duration of the project are discussed along with the available choices. An important facet in the decision making process is the incompatibility of certain software packages to operate on some hardware platforms. This discussion is followed by an outline of the system finally chosen.

The various types of software environment suitable for the production of the KBS are then discussed with their respective advantages and drawbacks. A brief discussion of the implications of the domain to be investigated and the impact on the final software choice is followed by a summary of the software selection process. The software environment that was chosen is discussed in detail.

Finally the Relational Database Management System (RDBMS) used for the development of the GeoTec database is discussed along with an outline of the operation of such RDBMSs.

4.2 Hardware Requirements

The requirements for a hardware platform to assist in the development of a large knowledge based system for site investigations could be stated as follows:

1) To be able to support a large relational database.
2) To be able to support several environments, that is knowledge engineering, database, operating system, window manager etc.
3) To have multi-user capability with adequate response times.
4) To have ample storage facilities.
5) Purchase and maintenance requirements to lie within financial constraints.

The first four of these criteria suggest that the platform would be either a large personal computer (PC) or a workstation. At the time of choosing the hardware, PC's were evolving at a tremendous rate, yet their processing, multi-user and multi-tasking capabilities were inferior to those offered by workstations.

The interface between hardware and software platforms introduces a new range of selection criteria, for only certain hardware platforms are supported by specific software platforms. The two main factors which seem to affect this interdependency are the length of time the software product has been available and the customer demand for a particular hardware platform. With General Purpose Programming Languages, GPPL's, and to an extent General Purpose Representational Languages, GPRL's, they are delivered on most major hardware platforms due to their longevity and general use in computing environments. With expert system shells and development environments the hardware requirements tend to be more specific, due to their relatively recent emergence.

It was decided that to realise the potential of the project, a powerful workstation was required and a price per performance comparison was carried out on competing systems. The workstation market, indeed the computing market as a whole, changes very rapidly so not only must performance and price be considered but also the reliability of the company, after sales service, warranty agreements and availability of software.

A Sun Sparc Station 2, manufactured by Sun Microsystems US, was eventually chosen for its very competitive pricing, 30 MIPS (Million Instructions per Second) performance and there was an existing maintenance policy with Sun Microsystems and the University of Durham. In addition, the Sparc2 had been available for two years in the United States and most software houses supported the Sparc2 computing architecture, based on the tried and tested Sun4 which it replaced. An additional 1 Gigabyte external hard disk was also purchased to ensure adequate disk storage, along with the relevant networking facilities. The Sun was delivered with 16 Mega Bytes of Random Access Memory, RAM, but this was increased to 32 Megabytes to ensure adequate performance.

4.3 Choice of Software Environment

In the early stages of the project, it was clear that fundamental decisions had to be made as to the software environment to be chosen for the production of the system. This choice would be constrained by the requirements of:

1) The need to link directly to a relational database.
2) The need to store domain dependant knowledge clearly and simply.

3) The flexibility of knowledge representation schemes and inference mechanisms.

4) The hardware platform.

Referring back to Mullarky (1987) the software options lay in the three pre-defined categories, a General Purpose Programming Language (GPPL), a General Purpose Representational Language (GPRL) and an Expert System Shell - including development environments.

Earlier work by the Geotechnical Systems Group at the University of Durham had been developed in a GPRL, Prolog (Moula, 1993; Vaptismas, 1993). Whilst being a powerful tool for producing logic based systems, it was felt that the limitations of Prolog had been reached. This was most noticeable in its ability to interface with users, linking through to large databases and the large memory overhead required to run the systems on the personal computers. A move either up or down in the software hierarchy was therefore required, either 'down' to a GPPL or 'up' to an expert system shell and/or development environment.

A GPPL approach could have been adopted, whereby the systems would be written in base level procedural language. This would entail utilising system and third-party functions for routines such as database access, interface handling and such like and custom writing those not available and assembling the system with a 'bottom up' approach. This methodology has the advantage of being able to produce exactly what is required to achieve the given aim, a flexibility of approach unachievable with an expert system shell and/or development environment. However, there are major drawbacks with the approach, namely:

4.3.1 Knowledge representation

Domain specific knowledge has to be represented. Whilst there are GPPL's available which can handle classes, C++ for example, the ability to separate the knowledge contained in the system from the control thereof becomes difficult. With the advent of environments like Borland C++, which significantly simplify the process, transparency of knowledge is achievable yet complex. However, due to the fundamental nature of the language being used, most things are possible and routines may be produced to separate the areas of the system as required. This brings us to the second area of concern.

4.3.2 Development redundancy

In producing a system in a GPPL, the flexibility is sometimes at the expense of 're-inventing the wheel'. Why should academic researchers be writing programs that commercial developers have already written? It is a point of much debate, the interface between academic research and commercial development, but when it comes to duplication of effort then serious questions must be asked.

The production of, for example, a backward chaining facility would have to be carried out, in order to process the 'knowledge base', but most expert system shells and development environments have built-in backtracking mechanisms. These generic methodologies that will have to be written have already been produced by commercial developers. The mistakes which have been made and the experienced gained have hopefully been used in making an efficient solution. Could, and more importantly, should academic researchers be duplicating these processes? Also, whilst they may produce a more suitable solution to suit their particular domain, is the solution as generic as possible and thereby re-usable?

4.3.3 Developmental resource

The learning curve for understanding, appreciating and beginning to build systems in a GPPL like C++ is lengthy, indeed some training literature refers to periods up to 9 months. This curve is significantly shortened if the developer has a good previous knowledge of such or similar languages, but there still remains a lengthy period of unproductive research. There is, of course, a learning curve with Shells and/or development environments but not as steep and severe.

When these disadvantages are considered, and the same criteria applied to expert system shells or environments, moving up the software hierarchy becomes a more obvious course of action. Also, when the particular domain of site investigation is considered there are other factors that strengthen the expert shell case.

Data Volume

The quantity of data to be processed at a particular time has the potential to be large. From the outset it was understood that the problem would be approached at a level where strata constituents would be broken down to their individual component and descriptors. Dealing with large numbers of boreholes would therefore involve large volumes of data. Shells and environments have the ability to handle large quantities of data.

Inference strategy

The ability of the system to provided a hybrid, not pure, inference mechanism is required due to the differing types of data to be processed. Geotechnical and related data can be of numeric, symbolic, descriptive, multi-valued or functional data types and so requires a flexible inference approach. Most shells offer this variety, or the ability to produce them simply.

Development strategy

As the area of Site Investigation is large and far reaching, the ability to add to the system in both knowledge and functionality is a high priority. With a GPPL, as previously mentioned, it is complex to separate the knowledge from control, making system updates difficult and the result non-transparent. Also, utilising a shell with its knowledge engineering capabilities allows for fast prototyping and therefore demonstration of systems.

Within the financial constraints of the project both expert system shells and development environments were considered, with an emphasis on the later due to the scope of the project.

4.4 Software Selection

Several expert system shells were tested and simple prototype systems were produced with them to try and identify their relative strengths and weaknesses. In addition, several development environments were assessed by a process of searching available literature for critical reviews, visiting academic sites currently using the systems, viewing promotional literature and assessing their technical requirements. A brief summary of the assessment process follows.

The expert system shells provide a good 'entry level' point to try out ideas, begin to develop methodologies for the knowledge representation process and generally reduce the learning process. They are however limited in scope, with exceptions, and generally provide only a single strand of reasoning. Their ability to produce usable and graphically based systems that operate with reasonable response times is poor and, especially with the lower level PC shells, data links are restricted to the flat file format.

It must be stressed however that the majority of the packages that were assessed were very effective at their core functionality. For example Crystal (Intelligent Environments, 1987) is a simple yet very effective rule based shell that allows complex systems to be produced with a minimum of effort. However any attempts to incorporate database access or complex numeric analysis are difficult to develop and result in systems that are very slow to use. KnowledgePro (Knowledge Garden, 1985) is a Windows based package that uses logic programming and simple rule-based reasoning to develop Windows KBSs. The results can be impressive but window and resource management are poor and result in systems that are very clumsy to use and poor in performance terms.

Expert system development environments provide the facility to develop, prototype and if necessary commercialise operational knowledge based systems. They are sold with good upgrade and technical support schemes, have proven track records and histories and finance permitting, provide a good means of approaching large knowledge domains such as site investigation. Some development environments such as Nexpert (Neuron Data, 1989) are highly successful environments that are well established within the commercial arena. They offer the full gamut of knowledge engineering facilities as well as superb backup facilities that ensure manufacturer written custom routines can be provided. This quality of product brings with it a price premium that requires significant investment, which is normally only repayable with the production of commercial systems.

After a review procedure of several months duration, ProKappa developed by IntelliCorp US was chosen as the development environment for SIGMA. ProKappa has developed over many years and evolved through other solid products such as KEE and KLUE. It offers the ability to apply object oriented programming in conjunction

with rule based reasoning and its own logic based language. ProKappa is described fully in the following section.

4.4.1 The PROKAPPA Development Environment

ProKappa is a high level object oriented expert system development environment (Intellicorp, 1991; Johnson, 1991), running through the X Windows user interface. It offers a wide range of functionality - including an object manager, rule based reasoning, inference mechanisms and real time monitoring facilities. The object manager allows complex hierarchies to be constructed, either graphically or programmatically, offering inheritance systems and dynamic structures. ProKappa also has system data links, known as the Data Access System, which allow user defined ProKappa applications to interface very efficiently with a range of relational database management systems. This is achieved through system defined object hierarchies, which ensure rapid data transfer and ease of construction and maintenance.

The PROKAPPA system provides an environment for developing and delivering multiplatform software applications. Recent upgrades of the PROKAPPA system have made it possible to produce Windows 3.1 executables from a workstation based development system. It is a C-based software development system that integrates object-oriented programming, rule-based reasoning and SQL database access in an easy to use graphical environment. Some of the main features of the PROKAPPA system that were used in building the SIGMA application are discussed in some depth below, whilst the others are just introduced briefly.

4.4.2 Object System.

In PROKAPPA the basic structure for representing data is called an *object*. Objects can hold descriptive data about the entity, thing, item, concept, category or template being represented and can contain special functions which define behaviour for the thing being represented.

The PROKAPPA system has two kinds of objects: *classes* and *instances*. Classes are templates for sets of entities with common characteristics, and instances represent individual objects in the *application domain*. The *application domain* is that area which PROKAPPA has assigned to a particular *application* or project. Each *application* is described in full by its application definition, or *.app* file, which informs PROKAPPA what C files, ProTalk files and other system resources are required to load and run the particular application.

The PROKAPPA object system supports arbitrarily complex hierarchies of objects. Object hierarchies are stored in collections called *object bases or knowledge bases*. Objects and object hierarchies may be static models. They may also be dynamic as they can be created, modified and deleted at runtime. The data in an object can be accessed and/or changed by functions, rules and methods, supported by an extensive library of functions for creating and manipulating objects.

Classes and instances are organised hierarchically. The terms *subclass* and *superclass* are used to describe relationships between objects of a hierarchy; *subclass* denotes a class further down the hierarchy from a specified class and *superclass* denotes a class further up the hierarchy from the specified class. Within an object hierarchy, the first object, that which precedes all others is known as the *topclass* object whilst all other classes below that are known as subclasses. At the bottom of the hierarchy are instances which may have no other class below them. This is shown diagramatically in Figure 4.1.

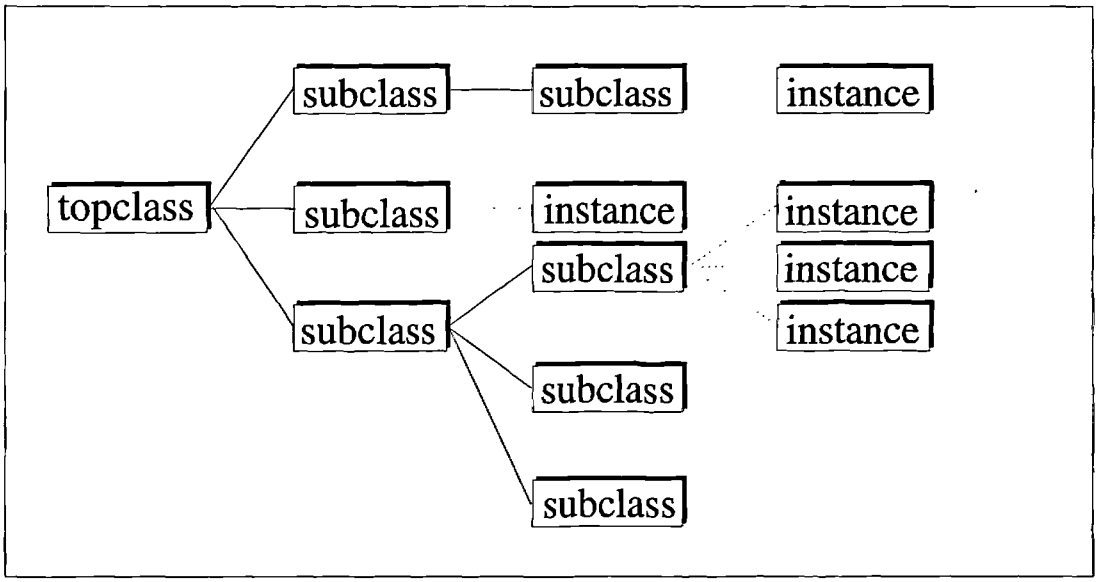


Figure 4.1 - Diagrammatic representation of ProKappa object hierarchy

Both classes and instances have *slots* which represent characteristics or attributes of objects. Slots represent three type of information: i) Attributes or descriptive information about an object, ii) Actions, called *methods*, that the object can perform, iii) Relationships to other objects in a system. There are three kinds of slots: i) *Single-value* slots, which are used to store values as symbols, strings or numbers, ii) *Multi-value* slots, which can hold an arbitrary number of values of any type represented as a list of values and iii) *Method* slots which contain procedures that define the behaviour of an object.

The object system supports *inheritance*. There are two types of inheritance in PROKAPPA: a) slot inheritance which is the inheritance of the existence of slots down the object hierarchy to lower level objects and b) value inheritance which is the inheritance of slot values down the object hierarchy to lower level objects that have inherited the slot. Slot inheritance, or value inheritance only, may be blocked at any level in the object hierarchy preventing the slot or the slot value from being inherited further down.

Slots can be further described by the use of *facets*. Facets are descriptors attached to slots which allow additional information about slots or slot values to be expressed. Like slots, facets have structures and values (a single value or multiple values) and can be inherited.

4.4.3 ProTalk Language

In the PROKAPPA system two languages can be used to implement applications, the C language as extended by PROKAPPA and the ProTalk language.

The PROKAPPA environment supports a version of the C programming language, Saber C, modelled on the ANSI standard plus several libraries of C functions for use specifically within PROKAPPA. However, this version of C varied significantly from those predominantly used at Durham and whilst being flexible and powerful did not have the sophistication of the ProTalk language, which was used extensively for the development of SIGMA.

The ProTalk language is a language developed specifically for use in the PROKAPPA system and can be used as an alternative to, or in combination, with C. It is particularly useful for writing code that expresses relationships between objects and facts and performs searches over object bases. The ProTalk language incorporates a set of pre-defined functions for interacting with object bases and manipulating objects and provides syntax for referring to information in an object base that can be used for manipulating or retrieving information about objects, slots and facets. The ProTalk language also offers several programming constructs such as assignment of values to variables, basic arithmetic operations, comparison operators, conditional statements and iteration constructs. It has the ability to call C functions and incorporate C code. In addition to all this, ProTalk is a non-deterministic language which supports backtracking.

ProTalk is a hybrid language combining aspects of both procedural and rule-based languages. It can be used for writing functions and rules. A ProTalk function is made up of one or more ProTalk statements. Each simple statement ends in a semi-colon. A compound statement is a sequence of zero or more statements wrapped in a pair of curly brackets ({}). Each statement consists of some combination of ProTalk operators, expressions, programming constructs, function calls and variables. In ProTalk there is no need to declare variables before using them, as is required when writing code in C. A function is defined by placing the keyword *function* in front of the function name, which is followed by a pair of parenthesis enclosing its arguments separated by commas. ProTalk code is interpreted at runtime, that is each function is compiled line by line as it is being run within the development environment. Individual ProTalk files may be complied before running, still within the development environment, giving much faster performance due to not requiring run-time processing.

Rules can only be written in the ProTalk language. These are a combination of ProTalk statements grouped together in *rulesets* and can be either forward chaining or backward chaining as well as mixed forward /backward chaining rules.

4.4.4 User interface tools

The Prokappa system allows for building customised end-user interfaces for applications to be built and provides two tools for their development:

- The *Active Images* system
- The *dialog box* system

The Active Images system is a tool for building business and instrumentation images to represent slot values graphically. This tool has not been utilised in developing the user interface for SIGMA, and therefore will not be discussed in any more detail.

The dialog box system is used for obtaining arguments or options required by a command or process which a program is about to execute. It is also used to display information, for instance, on the progress of a processing action. A PROKAPPA dialog box is a window that displays information or provides the facility to input information. A dialog box allows the user to input information in a variety of formats, using the keyboard or the mouse.

The components of a dialog box used to display information, accept information, or initiate action are called *controls*. In effect, a dialog box gets its functionality from the dialog box controls. The dialog boxes and each of its controls are implemented as instances of appropriate classes incorporated in a system object base called DialogBoxApp. These classes represent the types of dialog boxes and dialog box controls supported by the PROKAPPA system. Each non-display control in a dialog box has an associated React! method which defines what happens when the user interacts with that control, e.g. depressing a push button. *It is by the writing of these* React! methods that PROKAPPA applications can gain behaviour through interfaces.

All the dialog boxes in version 2.1 of the PROKAPPA system conform to the Motif standard, an industry common standard for X Windows management systems.

The PROKAPPA development environment supports an interactive *Developer's User Interface* for the rapid prototyping and development of applications. The PROKAPPA Developer's User Interface consists of the *Application Browser*, that manages the creation, editing, loading and compiling of the different components of an application, the *Object Browser* which is a graphical environment for the creation,

modification, viewing and saving of objects, slots and facets, the *C Workbench* which is a code interpreter as well as a source code C debugger, the *ProTalk Workbench* which is a tool for debugging ProTalk code and the *Interface Workbench* that gives the ability to the developer to graphically create dialog boxes for end-user interfaces.

Within the PROKAPPA environment any of the above types could have been used within the framework of the DialogBoxApp, which has the form of a windows based system.

Having being designed with a modular methodology SIGMA has a naturally occurring structure that suits the application of a menu based system, allowing system functions to be easily executed and allowing for the repeatability that 'what-if' situations require. All the data manipulation routines utilise an automated form filling method, whereby the forms are dynamically created from the specification of the data table. The form filling method also allows for data verification on entry to be implemented before the data is entered into the database. This has proved difficult to implement in the PROKAPPA system due to the limitations of PROKAPPA version 2.1 dialog box system, however the basic functionality has been incorporated. This area is discussed further in Chapter 7.

4.4.5 Database Access

The PROKAPPA Data Access System supports links to either flat files or SQL relational databases through database mapping. It was through these database mapping facilities that the GeoTec database was linked through into the PROKAPPA environment. This facility is covered in greater detail in Chapter 6.

4.5 The INGRES Relational Database Management System

The GeoTec database has been designed and implemented using the INGRES Relational Database Management System, RDBMS (INGRES, 1990a). INGRES provides multi-user access to a centralised data structure and is accessible via the industry standard Structured Query Language (SQL), as well as various INGRES variant query languages. INGRES is available to the academic community through the Combined Higher Education Software Trust (CHEST) agreement and as such is fully supported.

The INGRES RDBMS consists of three main components; the data manager, the user interface and the query language, see Figure 4.2

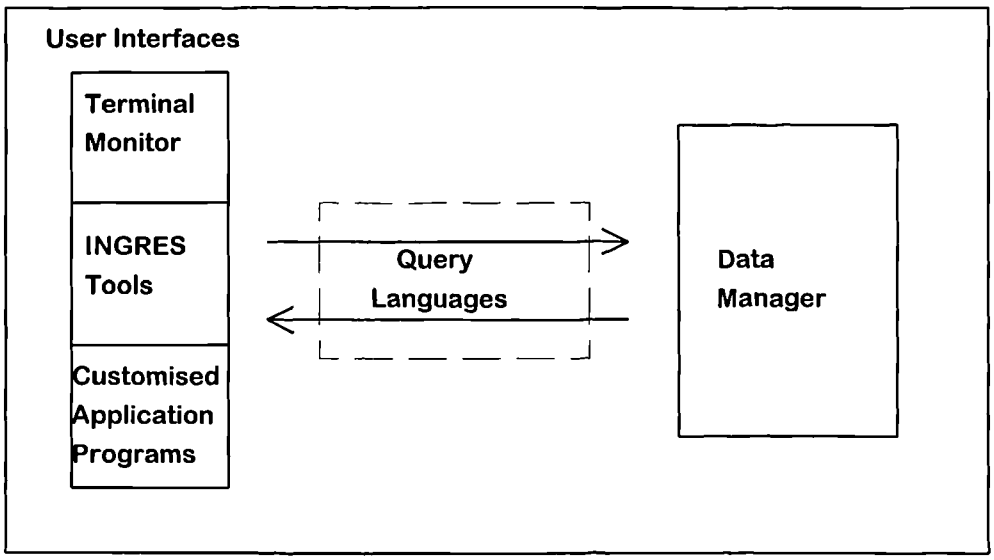


Figure 4.2- INGRES Architecture

4.5.1 The Data Manager.

The data manager accepts the query language instructions and performs specified operations on the database. All basic INGRES tasks, such as data updates and retrievals, are performed by the data manager. However, the user never interacts

directly with the data manager. Instead, the user must give instructions to INGRES through a User Interface

4.5.2 The Query Language

The query language passes instructions to the data manager from a user interface. There are many user interfaces that request different database tasks. The terminal monitor for example allows the user to enter data and direct instructions for the data manager by entering query language statements. Other interface allow data manipulation by INGRES forms and menus. These form based subsystems then send the appropriate query language statements to the data manager. Third party software can also communicate directly with the data manager using an embedded query language in the host software. Equally the host software can produce its own queries and pass them directly to the data manager.

The standard set for a query language is the Structured Query Language (SQL), subsets of which can be translated across database systems. SQL allows data to be selected, inserted, modified and deleted within an existing database. The technique employed by SQL is that of "automatic navigation" through the database, so that SQL is produced to describe *what* is to be carried out and not *how* (INGRES, 1990b).

A simple example of SQL code is given in Figure 4.3, where data from the proj table is requested subject to criteria.

```
select *
from proj
where proj.proj_id = "112343/c"
```

Figure 4.3 - Example of SQL code

The 'select *' syntax instructs the database to extract all the fields in the named data table whilst the 'from' identifies the table to be used. The where clause identifies the

selection criteria to be used for the extraction. This simple example gives an indication of the functionality of SQL, more complex statements can produce powerful selection algorithms.

4.5.3 The user interface

A user interface enables the user to give instructions to the data manager. The user interface accepts instructions from the end user and forwards them to the data manager via a query language. A form based application subsystem frees the user from having to memorise the specific query syntax and provides a working environment that is ideally suited to developing a database structure. Using third party software essentially bypasses the INGRES provided user interfaces when accessing an existing database.

INGRES also provides network support via the INGRES STAR and INGRES NET sub systems. INGRES NET allows multiple INGRES sites to be connected regardless of the hardware platform, so enabling a PC in one location to access a workstation based INGRES database in another location. INGRES STAR allows for databases at several locations to be combined together as one database across a network, allowing data to be available for anyone who requires it and making data duplication unnecessary.

4.6 Conclusions

Much work was carried out to try and ensure that the correct combination of hardware and software was chosen for the production of SIGMA. The increase in performance of PCs meant that there was little difference between them and workstations in the choice of a hardware platform. However when considered in conjunction with the software to be utilised, the data volume that could be involved and the requirements

for multi-user capability the Sun UNIX workstation was deemed the most appropriate choice.

The software environment chosen, ProKappa, offers a flexibility and professional product unattainable with the equivalent shell systems. The object oriented functionality, inherent non-determinism with the ProTalk language and its ability to link with external data sources fulfilled the envisaged project requirements.

The INGRES RDBMS provided an ideal environment for the development of the GeoTec database. Fully supporting the SQL syntax ensured that the GeoTec database would be compatible across a wide range of hardware and software platforms. INGRES's sophisticated record locking and networking facilities would also ensure that secure database access could be allowed over a global range.

Chapter 5

SIGMA - A System for the Interpretation of Geotechnical Information

5.1 Introduction

SIGMA (System for the Interpretation of Geotechnical Information) is a knowledge-based system which has been developed at the University of Durham, School of Engineering and Computer Science over the last 4 years. SIGMA has been developed using the expert system development environment Prokappa, (IntelliCorp, 1991) and is currently implemented on a Sun Sparc2 workstation.

The objective of SIGMA is to aid the geotechnical engineer in arriving at an informed judgement, based on the data available. This is achieved by assisting in the data management of a site investigation and providing interpretation routines to assist the geotechnical specialist in making informed decisions. The core of SIGMA is the GeoTec database which stores ground investigation and related data. SIGMA also contains a number of knowledge bases, each of which contains knowledge about the ground, geotechnical tests and correlations between geotechnical parameters. Modules within SIGMA allow specific tasks to be carried out i.e. parameter assessment, borehole interpolation, data checking and parsing of soils descriptions.

SIGMA has been designed in such a manner that it can organically grow as knowledge concerning a particular domain becomes available. Additional databases

could be added allowing access to other data structures and more processing and analysis sub-systems can be added at a later date.

5.2 The Role of SIGMA

5.2.1 Data Management

As information technology becomes increasingly commonplace in the area of site investigation, the importance of ensuring continuity and integrity of data rises. Electronic storage, manipulation and retrieval of data is only meaningful if the process is as least as productive and efficient, and hopefully significantly more so, than the respective manual transaction. As the use of electronic site investigation data gathering in the field and laboratory becomes more widespread, the requirements to store and process this data become more important (Naylor, 1992).

The quantity of data generated from a medium sized site investigation, say 30 boreholes, is large and can involve a noticeable administrational overhead; the larger the investigation, the larger the overhead. Electronic data management can be seen to be more effective as the size of the investigation increases (Institute of Civil Engineers, 1991) and this need has been met by several software companies who have developed systems to mimic the manual process involved with the production and storage of borehole logs, with great success, for example *gINT* (Staten & Caronna, 1992) and *SID-GDMS* (Mott MacDonald Ltd, UK).

These data management systems allow greater flexibility for the management of a project, more direct and immediate access to the data thereby provided and high level of data security. This security issue is not confined to the protection of sensitive or confidential data, but electronic data can be archived regularly by means of incremental backup systems, so preventing accidental loss or damage.

However, as with most competing software products, there are differences in the manner in which these systems operate, but more importantly, the manner in which they are able to transfer data. Most of the systems currently available have the ability to read and write data to a file in the ASCII format (flat file transfer), which enables most software packages to talk to each other. However, this approach may require pre or post processing to ensure that the data is entirely in the correct form for entry into the appropriate system. This additional processing may be intensive and the degree to which it is required may vary from one software package to another. This variety may lead to the introduction of errors. With the advent of the AGS Standard for Electronic Transfer of Geotechnical Data from Ground Investigations (AGS, 1992), the industry has a standard to adhere to for electronic transfer and hopefully this will lead to a more uniform approach, not only to data transfer but also data management.

The GeoTec database not only conforms to the AGS standards but attempts to take the standard further by defining data types and lengths for the specific fields, an essential step in producing a relational database. The GeoTec database really is the core of SIGMA. It is not only an external data store but an area of the system which is available for the transferring and receiving data. The manner in which these transfers take place is via the same methodology as the knowledge representation scheme in the knowledge bases (that is frame based). This enables data imported from GeoTec to be quickly and easily assimilated.

SIGMA has been designed using a modular methodology which has several advantages as will become clear in this thesis. One of the most important is the ability for the system to grow organically. With GeoTec as the central core there are hosts of additions that could be made in order to make the system more useful to the geotechnical specialist. Additional knowledge bases, case history databases, storage of 3D ground models and additional processing modules can all be added to the existing system. The production of borehole logs, as mentioned previously, is a task

that is suitable for computerisation and benefits from the direct access to the data store. With a large central store the production of logs as required could significantly reduce the paperwork systems within a large geotechnical company, also allowing for custom reports to be easily produced.

Being mounted on a workstation platform also allows for networking to peripheral machines, from within the same building to across the globe. All employees of an organisation can then have access to the same data source and as that source is continually updateable, everyone has access to the same level of data at the same time. Data verification need only occur once, reducing the duplication of effort and enforcing an organisation wide standard. In short, the potential offered for centralised data management to significantly improve the operational efficiency of geotechnical companies, coupled with the availability of interpolation and assessment routines, is large.

5.2.2 Data Interpretation

As mentioned in Chapter 3, there are many computing systems in use in the civil engineering world that utilise Knowledge Based Systems (KBS) technology, but there are also widespread feelings on their applicability, suitability and reliability. Many modern day knowledge based systems could be more suitably termed Decision Support Systems (DSS) that is systems that assist the user in their decision making processes. The manner in which this may be achieved is varied, from supplying the user with the relevant reference documentation and on-line guidance, for example hypertext systems, to checklist type systems which ensure that the user has covered all eventualities.

SIGMA has been designed as a Decision Support System (DSS). SIGMA allows the user to reach decisions in the most informed manner, by allowing the user access to many different types of data and by assisting in the carrying out of mundane cross-

checking. However the user is not excluded from the decision making. SIGMA also offers features over and above this, for example the data management mentioned earlier, borehole interpolation and parameter assessment.

SIGMA's parameter assessment facilities allow a geotechnical specialist to assess parameters for use in the design process at specific locations of a site and if direct measured data is not available correlate the required parameter from other data. Borehole interpretation routines allow the user to perform borehole to borehole correlations in order to assess the sub-surface conditions. It should be stressed that all the modules that offer predictive results allow the user the opportunity to consider differing possibilities in order to gain a clearer picture of the outcome.

Computing systems that offer, or purport to offer 'expert' solutions to any engineering based problem will always be open to scepticism, for expertise is not something that can be gained merely by the purchase of software, it is gained through years of experience. KBS's or DSS's offer the geotechnical specialist another tool with which to assist them in carrying out their profession, a non-exclusive addition to their existing knowledge and experience. Moreover if the system can be seen to be organic, that is as the knowledge of a particular domain grows this can be simply added to the system, this can only assist the acceptance of such technology.

To conclude, SIGMA is a decision support system, not a pure 'expert' system - although the knowledge bases do contain the aggregation of experts' knowledge. Its role is to guide the user through the more complex geotechnical aspects of a site investigation, that is assessment of design parameters, the interpretation of borehole data and the management of the vast quantities of data.

5.3 The functionality of SIGMA

5.3.1 An Overview of SIGMA

In order to clearly illustrate the functionality of SIGMA, Fig 5.1 shows how the system can be visualised as 'levels' radiating out from a central core. It can be seen to consist of 4 levels centred on the core database (Toll *et al,* 1992). The initial level comprises the ground investigation database, GeoTec, containing the geotechnical and related data which requires interpretation. GeoTec is discussed in detail in Chapter 6. Additional core databases can be added at a later date, such as a case history database to take advantage of historical precedents.

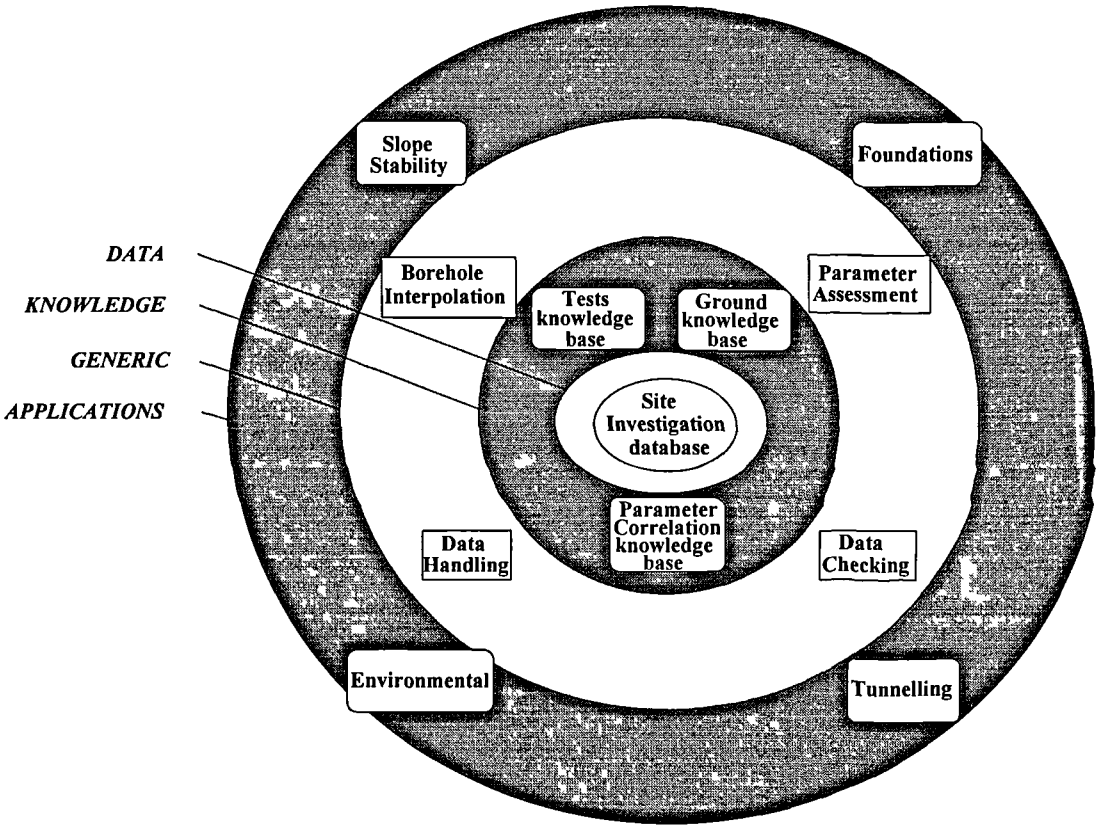


Fig. 5.1 Schematic Representation of SIGMA

5.3.2 Knowledge Bases

The second level of SIGMA currently comprises three knowledge bases. These contain 'general knowledge' about geotechnical engineering, represented in the hierarchical manner previously described in Section 4.4.2. The knowledge bases all utilise the functionality of the PROKAPPA object system, with the data being stored in both multi and single value slots and, where appropriate, facets.

5.3.2.1 Ground Knowledge Base

A Ground knowledge base contains knowledge about soils and rocks. This knowledge base is the combination of the work of Toll (1992), Moula (1993) and Giolas (1994) and takes the form of typical ranges of parameters (e.g. compressibility, strength, permeability etc.). Starting from the topclass object ground, the subclasses progress through generic ground types through to specific soil types. This knowledge base is shown graphically in Appendix 5. Moula (1993) developed routines capable of searching through the ground knowledge base in to either identify the ancestry of a particular ground type and also value ranges of a given parameter for that soil type. Also routines were produced which could allow a user to interrogate the knowledge base to locate which soil type had a particular value of a given parameter (Moula and Toll, 1993). This allowed the inexperienced user to both learn about and identify soil types by their properties. Giolas (1994) has produced a modified Ground knowledge base supported by an interface that allows default ranges for specific ground types to be updated and also provides applicability data for correlations.

5.3.2.2 Test Knowledge Base

A Tests knowledge base contains knowledge about different geotechnical tests, this knowledge base is shown graphically in the Appendix 6. Objects become more specific as the hierarchy is traversed until specific tests are encountered as instance objects. Each test instance has a *test_code* slot that contains the same test code as that

used to identify the tests in the GeoTec database. This methodology allows efficient matching of test objects with the relevant data contained in the database. The knowledge currently stored as slots on each test object can be subdivided into the following two groups:

Reliability and Applicability

The reliability of a test is an indication of how reliable a particular test is at measuring a given parameter. The reliabilities are sub-divided into high, low, medium and none and this is achieved by assigning each reliability as a facet on the slot **Reliability**, as shown in Table 5.1.

Reliability	**Parameter**
High	param1, param2
Medium	param3
Low	
None	

Table 5.1 - Storage of reliability knowledge in Tests knowledge base

This allows one slot on an object to store a range of knowledge about a particular test. Similarly with applicability which is an indication of how applicable a particular test is for a given soil type. Again the applicabilities are subdivided into high, medium, low and none utilising facets on the **Applicability** slot of the appropriate object, as shown in Table 5.2.

Applicability	**Soil Type**
High	soil1, soil2, soil3
Medium	
Low	soil4
None	

Table 5.2 - Storage of applicability knowledge in Tests knowledge base

The data for both reliability and applicability were taken from Moula (1993) and were based upon the results of a knowledge acquisition exercise carried out using a

questionnaire distributed to a group of experts in the field. Reliability and applicability are used by SIGMA to give user guidance during the operation of the parameter assessment module.

Frequency, Test Objective and Test Cost

This additional knowledge is included as a generic source that may be consulted at any time as additional information. Frequency gives an indication of how often a specific test is used (classified as Routine, Less Common, Specialist Test), test objective stipulates the main objective of the test and test cost gives an indication of the test cost, that is high, medium or low. Presently no SIGMA modules access this data.

5.3.2.3 A Parameter Correlation Knowledge Base

A Parameter Correlation knowledge base contains knowledge of the different empirical correlations which exist for relating geotechnical parameters. This work has been carried out extensively by Giolas (1994). The knowledge base can be updated as new correlations are discovered and may act either as a stand alone system or as a module of SIGMA. The parameter assessment module utilises this knowledge base in its final stages and as such is discussed again in Chapter 8.

5.3.3 SIGMA Modules

The third level of SIGMA consists of generic modules i.e. those that are required whatever type of geotechnical application is being investigated. These modules, discussed in greater detail in later chapters, currently comprise:

Data handling - data import, export, soil description parsing and database interfaces

Data checking - cross checking of values and parameters to ensure data integrity

Parameter assessment - assessment of parameters utilised in the design process including parameter correlation

Borehole interpolation - the assessment of sub-surface conditions.

The fourth level contains the modules which are specific to a particular application (e.g. foundations, slopes, tunnels) and can be considered to be the specific user interface to the system for particular tasks. These fourth level modules are beyond the scope of this thesis.

As can be seen, SIGMA has been implemented using a modular approach, allowing a prototype system to be available for use and demonstration at an early stage. As more modules become available, they can be independently tested and then incorporated into the existing prototype, gradually building up the final version. This modular approach allows the system to be continually updated, allowing for new areas of interest to be investigated and new techniques to be applied.

The various knowledge bases and the core database of SIGMA are used either in conjunction or independently to supply the geotechnical specialist with information to assist in the decision making process. The GeoTec database is linked directly to the system via Structured Query Language (SQL) commands and can be accessed either as a part of a SIGMA module or independently to provide additional data.

The inference mechanism of the system is provided in the main by the ProTalk language, a General Purpose Representational Language (GPRL) provided within the ProKappa development environment and described in detail in section 4.4.3. The ability of ProTalk to allow non-deterministic programming to be written alongside procedural functions, and the direct access it allows to the objects contained in the knowledge bases enabled the inference mechanism to be written in a flexible and practical manner.

ProTalk code is structured somewhat differently from procedural code in that the flow of the program is dictated by the object base and the interface with the user. To merely browse through reams of computer print in an attempt to understand the functionality of a computer program is never an easy task but with ProTalk it is significantly harder. The code that makes up SIGMA is included with this thesis Appendix 8.

5.4 Conclusions

SIGMA has been developed as a Decision Support System to assist the geotechnical specialist in two specific areas; data management and interpretation. By providing a centralised data store, and allowing several levels of access to that data, SIGMA provides an important data management role. As the quantity of electronic site investigation data increases data management will become an increasingly important aspect of geotechnical companies.

Interpretation routines assist the geotechnical specialist in their decision making, hopefully leading to more informed decisions. Assessment of design parameters is an important task and SIGMA provides a straightforward interface to allow the user to examine selected locations and correlate parameters if required. Borehole interpolation routines assist in the determining of sub-surface ground conditions utilising borehole to borehole correlations.

The modularity of the SIGMA design allows for generic growth of the system as new process are added and new knowledge sources obtained.

Chapter 6

The Geotec Database

6.1 Introduction

The relational data model and the increasing importance of a data management to a site investigation have been detailed in sections 2.7 and 3.3. As stated in section 5.1, the central core of SIGMA is Geotec, a ground investigation database. In this chapter the design history of the database is discussed followed by a detailed discussion of the relational structure of the Geotec database. This structure identifies what could be a standard for geotechnical data structures, including a methodology for the storage of parsed layer descriptions and detailed test data storage.

The mechanism by which the database and the PROKAPPA development environment combine is discussed in detail, along with the operation of the Unique Identifier (UID) methodology. This is followed by a discussion on the manner in which the two systems communicate through the Structured Query Language (SQL).

6.2 Design History of Geotec

The evolutionary process which has led to the final implementation of Geotec has involved the production of several interim databases and standard database design methodologies.

On examining the various data models, see section 2.7, it was decided to implement the relational model, due to the broad range of data to be stored, redundancy

minimisation and to accommodate the widest possible querying potential for the user (Giles, 1992). The importance of multi-user capability and the ability to store potentially vast quantities of data, coupled with the dynamic integrity checks and associated stability that come with large scale RDBMS were important factors in the decision. In addition, the software and hardware requirements to implement such a RDBMS lay within the constraints of the project.

Due to the nature of the hardware platform and the available database and development software, the connection between the two systems could be seen to be of the enhanced expert system type (see section 4.3.2). This mode of connection allows the finalised knowledge based system to retain control over the flow of data, i.e. dictates when data is to be transferred. This enables SIGMA to retain the desired control over the data transfers throughout the consultation process, whilst still allowing the database to be accessed as a separate entity by non-SIGMA users. A tight coupling approach was deemed most suitable, as the software permitted such a connection and it provides the system with the greater flexibility and control.

Identifying the data to be stored in the database was carried out by establishing a data dictionary. This dictionary not only collects the data in one central reference location but allow the formation of field types and lengths to be carried out at an early stage in the design process. The sources used to collate this data were the appropriate British Standards (BS 1377, 1990; BS 5930, 1981), that available from the AGS standard (AGS, 1992), several projects being undertaken at the University of Durham (Moula, 1993; Sylvester, 1991; Mavroidi, 1993) and through collaboration with Scott Wilson Kirkpatrick and Partners, the civil engineering consultancy group. This process identified the entities to be stored in the final data structure, an entity being a conceptual model representation of an object in the real world, for example a borehole.

Once the relevant entities had been identified, their inter-relationships, attributes and primary keys could be identified. Attributes are properties that are possessed by an entity. For example, the entity borehole has amongst its attributes borehole identifier, final depth etc. The primary keys ensure that the data held within a given row of a data table can be uniquely identified (section 2.6). An Entity-Relationship, ER, diagram was produced, enabling the relationships between entities, e.g. one-to-one and one-to-many to be clearly defined and a 3rd order normalisation analysis subsequently performed. Normalisation is the process whereby conceptual data models are transferred into a form acceptable to relational database (Codd, 1970). The result of the normalisation process is a data model that has a minimum level of duplication and redundancy, the relationships between the attributes (fields), are clearly shown and a more flexible data model is produced (Bamford and Curran, 1987; Date, 1983). On the completion of the normalisation process, individual tables could be translated directly from the entities, their columns identified along with the referential keys required to link these tables in their associated relationships, giving rise to the structure shown in Fig 6.1.

The database was actually produced by writing SQL script files in an INGRES interface subsystem known as Interactive Structured Query Language, ISQL. This enabled the lengthy commands required to produce the database to be saved to an ASCII file and executed as a separate process. If changes were then required to the database, the original version was be removed, changes made to the SQL script files and the database remade. And example of an SQL script file is included in Appendix 3.

6.3 Data Structure of Geotec

An outline schema for the database is shown in Figure 6.1, where the boxes represent tables in a relational database structure.

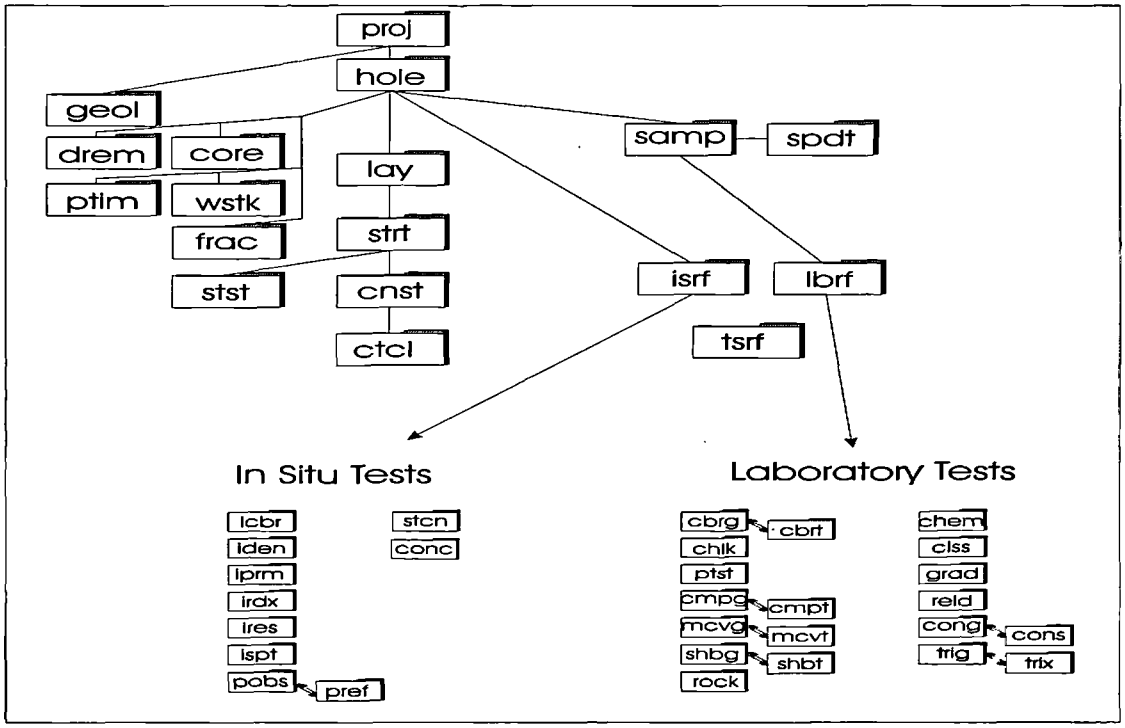


Fig. 6.1 - Schema for the GeoTec Database structure (see Fig 6.2 for legend)

Each table stores data which represents a data group, the data group being a function, property or parameter of the site investigation. This structure produces an efficient structure for data retrieval and handling, necessary for the potential volume of data to be stored. A full listing of all the tables in the database, their data fields, data types and a description of the field's purpose are shown in Appendix 1.

LEGEND

proj	Project		drem	Depth Related Remarks
hole	Borehole / Trial Pit		core	Core
geol	Geology		frac	Fracture Detail
lay	Layer		ptim	Progress with Time
strt	Strata		wstk	Water Strike
stst	Stratum Structure		samp	Sample
cnst	Constituent		spdt	Specimin Detail
ctcl	Constituent Colour		isrf	In Situ Test Index
tsrf	Test Reference		lbrf	Laboratory Test Index

LEGEND - Laboratory Tests

cbrg	California Bearing Ratio - General		cbrt	California Bearing Ratio - Detail
cmpg	Compaction Tests - General		cmpt	Compaction Tests - Detail
mcvg	Moisture Condition Value - General		mcvt	Moisture Condition Value - Detail
shbg	Shear Box Testing - General		shbt	Shear Box Testing - Detail
cong	Consolidation Tests - General		cons	Consolidation Tests - Detail
trig	Triaxial Tests - General		trix	Triaxial Tests - Detail
clss	Classification Tests		grad	Particle Size Distribution
reld	Relative Density		chem	Chemical Properties
chlk	Chalk Crushing Value		ptst	Laboratory Permeability Tests
rock	Rock Properties			

LEGEND - In Situ Tests

icbr	In Situ California Bearing Ratio		iden	In Situ Density
iprm	In Situ Permeability		irdx	In Situ Redox
ires	In Situ Resistivity		ispt	Standard Penetration Test
stcn	Static Cone Penetration		conc	Cone Calibration
pobs	Piezometer		pref	Piezometer - Detail

Figure 6.2 - Legend for GeoTec database structure

As far as possible the table names have been adopted to be compatible with the AGS headings (AGS, 1992). The top level table is the *proj* table which contains information on the location and date of the project and the parties involved. A departure from AGS is the inclusion of a geology table *geol* which has been linked to the *proj* table through the *proj_id* key. The *geol* table allows storage of identified geological horizons which could exist at the site. This stratigraphic information will generally be obtained from a desk study at the feasibility stage of the project before any ground investigation has been started (boreholes or trial pits). Therefore the information need not be related to specific holes but is attached at the project level. The information can be linked to specific layers identified at a later stage (during or after the ground investigation) using the *horz_no* field. Other tables for storing desk

study (sources) information could be attached at *proj* level. This is an area that has not been addressed by AGS who have concentrated only on ground investigation data (i.e. borehole and trial pits investigations). The other departure from the AGS standard has been the inclusion of data structures to store parsed ssoils descriptions, as noted in the following sections.

The term Hole has been adopted as the generic name for boreholes, trial pits or shafts (as per AGS). The *hole* table contains details of boreholes or trial pits such as location, date and method of boring. The details of the ground conditions observed at the hole are stored in the *lay* table. This contains depth and thickness information about the layers observed, and these can be linked to the appropriate geological horizon in the *geol* table. Also present is a text field containing the soil or rock description. The reasons for maintaining this text field are described below.

Minor comments on the ground conditions which are identified by a particular depth, and which do not correspond with layer depths, are not stored in a structured form but are held as text fields in the *drem* table, attached at *hole* level. Fracture spacing data is also stored separately in the *frac* table. This is also attached at the *hole* level, rather than being identified with a particular layer (Fig. 6.1). This is because zones of similar fracture spacing identified will not necessarily coincide with layer boundaries.

6.3.1 Soil and Rock Descriptions

Current database systems only store soil and rock descriptions as text fields. These descriptions can be long and complex, for example *Moist reddish brown stiff thinly bedded closely fissured silty sandy CLAY with a little dark greenish grey sub-rounded fine gravel and frequent inclusions of sand.* If information needs to be abstracted from the description, say the consistency of the soil *(stiff)*, the text description must be parsed. To parse a description each time a piece of information is

needed from it would be very time consuming. A more efficient practice would be for the description to be parsed once, and once only, at the time of data entry. The information contained in the description would then be stored as separate fields within the database, and would be easily and efficiently accessible.

To develop data structures for storing the information contained in a soil description is not straightforward. A large amount of varied information is contained in the description and the vocabulary used is often complex. Therefore the task of developing a representation scheme which can handle the full range of information is difficult. However, having said that, much of the very detailed information contained in the description plays only a minor role in engineering design. A representation scheme which can handle the majority of soil descriptions was put forward by Toll *et al* (1991), and this scheme has been adopted in the implementation of the database.

Since it is possible that the structured representation will not be able to handle some of the more esoteric descriptions which can be found on borehole logs, provision has been made in the *lay* table to store the description in its full form as a text field. Therefore, the full description is always available to the engineer processing the data if required, although interpretation of the data by the KBS will use the structured representation. It will be seen that the structured representation is still very detailed; it might even be argued that it is too detailed for the purposes of most investigations. However, it is felt that a reasonable compromise has been reached on the ability to represent very complex descriptions without carrying too much redundant information. Example tables covering those tables used in the storage of the parsed soils description information are shown in Table 6.1.

This format can also deal with the case where a soil or rock changes significantly from the top to the base of a layer eg *Silty SAND becoming Clayey SAND.* The first stratum *Silty Sand* would be identified in the *stst* table as *Top* and *Clayey SAND* as a

separate stratum identified as *Base*. Other information pertaining to the structure can be recorded such as bedding dip, orientation and spacing, or frequency of inclusions.

Layer Table

Project ID	Hole ID	Layer No.	Depth to top	Thickness	Description	Legend Code	Horizon No.
7702	B5	3	3.00	9.00	Thinly spaced layers of moist stiff CLAY interbedded with thickly spaced moist firm SILT. Bedding dip 167/05°		2
7702	B5	4	12.0	8.15	Red and brown mottled stiff silty CLAY becoming brown very stiff clayey SILT		2

Strata Table

Project ID	Hole ID	Layer No.	Stratum No.	Main Constituent	Moisture Condition	Consistency	Weathering
7702	B5	3	1	CLAY	MOIST	STIFF	
7702	B5	3	2	SILT	MOIST	FIRM	
7702	B5	4	1	CLAY		STIFF	
7702	B5	4	2	SILT		VERY STIFF	

Stratum Structure Table

Project ID	Hole ID	Layer No.	Stratum No.	Structure No	Structure	Spacing	Dip	Orient.	Surface
7702	B5	3	1	1	INTERBEDDED	THIN	5	167	
7702	B5	3	2	1	INTERBEDDED	THICK	5	167	
7702	B5	4			TOP				
7702	B5	4			BASE				

Constituent Table

Project ID	Hole ID	Layer No.	Stratum No.	Constituent No	Constituent	Amount	Grading	Shape	Texture
7702	B5	3	1	1	CLAY	MAIN			
7702	B5	3	2	1	SILT	MAIN			
7702	B5	4	1	1	CLAY	MAIN			
7702	B5	4	1	2	SILT	SECONDARY			
7702	B5	4	2	1	SILT	MAIN			
7702	B5	4	2	2	CLAY	SECONDARY			

Colour Table

Project ID	Hole ID	Layer No.	Stratum No.	Constituent No	Colour No	Main Colour	Second Colour	Colour Modifier	Colour Structure
7702	B5	4	1	1	1	RED			MOTTLED
7702	B5	4	1	1	2	BROWN			MOTTLED
7702	B5	4	2	1	1	BROWN			

Table 6.1 - Example Data Tables for storing parsed description information

The structured representation of soil and rock descriptions consists of four tables: *strt* - stratum, *stst* - stratum structure, *cnst* - constituent and *ctcl* - colour. This is because descriptions of layers may contain more than one stratum (soil or rock), for example *SANDSTONE interbedded with SILTSTONE* or *CLAY with pockets of SAND.* In these examples two distinct strata are present within the layer, yet they cannot be distinguished as separate layers (a layer being defined by depth and thickness). Therefore the representation scheme allows for the possibility of multiple strata within a layer (identified by strata number, *strt_no),* with the relationships between the strata being stored in the Stratum structure, *stst* table. In the first example given above, the term *interbedded* would describe both the strata *SANDSTONE* and *SILTSTONE.* In the second example the term *pockets* would describe the *SAND* while *CLAY* would be described as *dominant.*

Stratum represents the whole stratum eg *Silty sandy CLAY,* whereas *Constituent* indicates the constituents combining to make up the stratum eg *silt, sand, clay* etc. In the *Stratum* table, information which relates to the whole stratum is stored such as *Moisture condition, Consistency* or *Weathering,* and the dominant constituent is also included. In the constituent table (*cnst*) information which relates to individual constituents is stored. Using the scheme identified by Toll *et al*, 1991, the amount of each constituent is as identified as *Main* if the constituent is dominant (*SILT* etc.), or as *Minor* (*Slightly silty* etc.), *Secondary* (*Silty* etc.), or *Major* (*Very silty* etc.) for the lesser constituents. This scheme can also be used to represent descriptions which are not those recommended by BS 5930, but which are still in use, such as *with some ...*, *with a little ...* etc.

Since it is sometimes possible for detailed information on *Grading, Shape* and *Texture* to relate to a particular constituent, rather than the stratum as a whole (eg *CLAY with a little black subrounded coarse gravel* where subrounded and coarse refer to the lesser constituent, gravel) this information is stored in the *Constituent*

table, rather than at the *Stratum* level. Colour can also relate to the individual constituents, rather than to the stratum as a whole (as when the lesser constituent gravel is described as black in the example above). Colour is represented as Main colour, Secondary Colour and Modifier. Since there can be multiple colour descriptors for a constituent a separate *Colour* table is used to handle this level of complexity, attached at *Constituent* level.

6.3.2 Geotechnical Test Data

Insitu or laboratory tests are linked to the *Hole* table as shown in Figure 6.1. Insitu tests can be identified directly from the depth at which they were carried out. Laboratory tests are carried out on samples. Information on samples is stored in the Sample (*samp)*, and Sample detail (*spdt*) tables. Test results are identified by *proj_id* and *hole_id* but are not attached to a particular layer. This is because the test information will often be used in determining the layer boundaries.

Between the sample table and the test tables there are two additional tables, *isrf* - insitu test reference, and *lbrf*, laboratory test reference which are shown in Tables 6.2 and 6.3. Due to the methodology of the PROKAPPA Data Access link, these tables are required to ensure a smooth data transfer. They record which tests have been carried out, keyed as shown in the Tables 6.2 and 6.3. They act as an additional indexing mechanism to extract the specific test data and are automatically generated using simple SQL routines. They contain the test code of whatever tests have been carried out on a particular layer - they are in effect a summary table. When data needs to be extracted, they ensure that only two tables need to be accessed to ensure that all the relevant data can be retrieved. These tables are additional to the AGS standard and are a result of the combination of the GeoTec database with SIGMA. The table *tsrf*, lightly shaded in Figure 6.1, contains reference data for all the different tests stored in the database. This table is shown without links to any other,

as its contents may be accessed directly or by several of the tables in the data structure.

isrf			
Field Name	**Field Type**	**Field Description**	**Remarks**
proj_id	varchar(10)	Project / Site Investigation Code	K
hole_id	varchar(10)	Borehole / Trial Pit Code	K
test_top	f4	Depth to top of test	K
test_code	varchar(20)	Code of test used	

Table 6.2 - In situ test reference table

lbrf			
Field Name	**Field Type**	**Field Description**	**Remarks**
proj_id	varchar(10)	Project / Site Investigation Code	K
hole_id	varchar(10)	Borehole / Trial Pit Code	K
samp_ref	i2	Sample reference number	K
spec_ref	i2	Specimin reference number	K
test_code	varchar(20)	Test code	

Table 6.3 - Laboratory test reference table

Laboratory test information can relate to different degrees of sub-division of samples. A sample is usually sub-divided in order to carry out different tests. For compatibility with AGS, this first sub-division of a sample is called a specimen. In some cases, the location of a specimen within a sample may be significant, for example, if the sample crosses a stratum or layer boundary. For this reason, a *Sample Detail* table, *spdt*, is provided for identifying the location relative to the sample top. A comments field is also provided, for recording any peculiarities which are not true for the sample as a whole.

Triaxial (trig) table: Level 1

Project ID	Hole ID	Sample Ref.	Specimen Ref.	Specimen Condition	Cohesion
7702	B5	12c	2		35
7702	B5	12c	2		4

Triaxial (trig) table: Level 1(cont)

Angle of friction	Undrained Shear Strength	Property code	Remarks (_rem)	Test code
		UPI		TRIQU
27		EPI		TRIID

Triaxial (trix) table: Level 2

Project ID	Hole ID	Sample Ref.	Specimen Ref.	Test Ref.	Cell pressure	Deviator stress
7702	B5	12c	2	12/22	100	275
7702	B5	12c	2	12/23	100	277
7702	B5	12c	2	12/24	100	279

Triaxial (trix) table: Level 2 (cont)

Strain	Property code	Moisture content	Remarks (_rem)
10	UPI	25.3	B512c0-34
10.5	UPI	25.3	B512c0-34
11	UPI	25.3	B512c0-34

Table 6.4 Example Test Tables showing multi level structure

For some tests (eg triaxial) the specimen is further divided into sub-specimens, each of which is tested in order to produce an interpreted result for the specimen as a whole (for the triaxial test the interpreted result would be c and ϕ). However, it is also important that the results obtained for the sub-specimens are also stored. This allows the individual results to be re-examined by the geotechnical specialist, if there is some doubt about the interpretation. In the case of the triaxial test the individual results would be shear strengths measured at different cell pressures. Table 6.4 shows how this multi-level test storage operates - not all the fields in the data table are shown to aid clear illustration.

There is also a further level of test data, which are the 'raw data' from which the results on specimens, or sub-specimens, were obtained. The triaxial test version of this would be the data points defining the stress-strain curve; for the moisture content test, it would consist of the weight of the sample, wet and dry and the tin weight.

While it would be possible to define data structures for storing this level of detail, it is questionable whether this is worth-while. Different companies will have differing views on what needs to be stored at this level; some might even question the need for this raw data to be included at all in a structured database. However, with the development of integrated data management systems, it would make sense for the data management system to have access to this raw data, particularly since the widespread use of laboratory data acquisition systems will mean that a large part of the data will already be stored in computer files. To provide this facility, the database system developed can store pointers to files containing the raw data. These files do not need to be structured; they can be simple ASCII files, formatted Document files or Picture files. The *_rem* field in the appropriate test table has been allocated for this usage. If no raw data exists for ASCII storage, this field merely contains comment and remarks on the test, up to 250 characters. However, if raw data is available, this field stores the pointer and full path name of the relevant file or files.

The results from laboratory tests can therefore be stored at different levels. The top level is a result which applies to a specimen and is identified by Specimen Number, *spdt_ref.* For the simple tests, such as moisture content, this is all that is required. The next level is a result which applies to a sub-specimen, and is identified by a test or stage number, *_tesn* (as per AGS). Pointers to files containing the raw data can be provided at either level. This provides the possibility of storing Level 3 data in an unstructured form. Doing this would mean that the Level 3 data would be available for reading by the geotechnical specialist, but would not be suitable for access by the KBS.

6.4 The SIGMA - Geotec Connection

The designation of the SIGMA/Geotec connection as tightly coupled, see section 2.8.2.2, allowed for the independent design of both systems. However, the structure of the actual connection between the two systems has been dictated by the design of the PROKAPPA Data Access System (DAS), a PROKAPPA provided package. The DAS, retrieves data from external data sources and places it into unique named PROKAPPA objects; it can also place data from these objects back into the external data sources. To do this, the DAS needs to know how to map the external data into and out of PROKAPPA objects. There are four components in this mapping process, as shown in Figure 6.3.

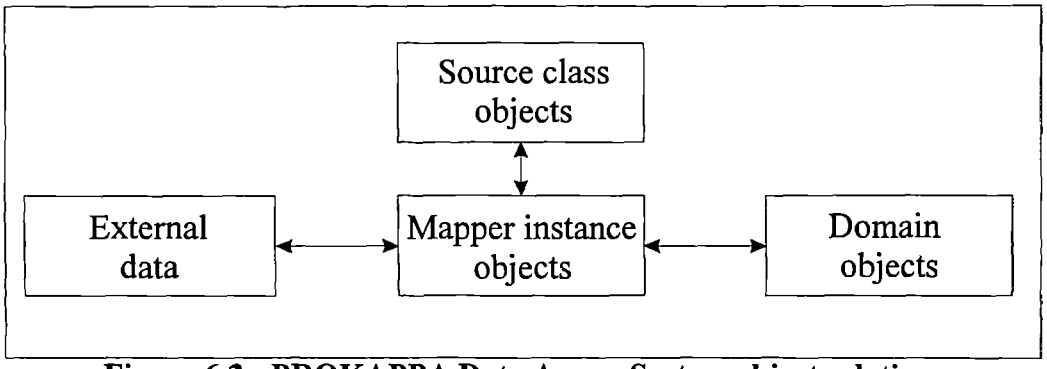


Figure 6.3 - PROKAPPA Data Access System object relations

1. The external data is the data stored in the relational database or flat file.

2. The domain objects are the objects that contain copies of portions of the external data.

3. The mapper instance objects, contained in the mapper application, are objects that describe the mapping between the external data source and the domain objects.

4. The Sources application contains the class objects for the instances in the mapper application.

6.4.1 UID Nomenclature

PROKAPPA allows many applications to be active at one time, so if the DAS application is live, then the objects can be interrogated and data transfer initiated. To ensure data integrity, a concept known as Unique Identifier, UID is employed. This is a PROKAPPA methodology that ensure that whenever a data instance is imported into, or being prepared for export from, a database, each instance has a unique name. All PROKAPPA objects must have a unique name and the UID concept utilises the primary keys in the relational data structure to ensure this singularity. This can lead to lengthy nomenclature of instances where tables have lengthy keys, take for example the sample constituent table shown in Table 6.5.

Field	Values	Keys
proj_id	7702	K
hole_id	B5	K
lay_no	3	K
strt_no	1	K
cons_no	1	K
cons_cons	CLAY	
cons_amnt	MAIN	
cons_grad		
cons_shpe		
cons_tex		

Table 6.5 - Sample constituent table

The UID nomenclature for this particular instance, which is in fact the first row from the previous Table 6.1, is **cnst("7702","B5",3,1,1)**. However, with the non-deterministic nature of ProTalk complex names of this nature do not present a problem. If the constituent instances need to be processed in any way, the command *all instanceof* **cnst** gathers the names of all the current instances of the object **cnst** into a list. This list may then be processed one instance at a time using the structure

for **?inst** *inlist* **<list of constituent instances>**

whereby the variable **?inst** is bound to each member of the list in turn, so removing the requirement for the system to know the name specifically.

6.4.2 Mapping the GeoTec database

In designing the particular mapping for the Geotec database, each individual table of the database is defined for PROKAPPA in terms of its field formats and contents, keys and overall structure. This is accomplished with the aid of a specific set of developers tools which form part of the PROKAPPA DAS. This windows based interface may be augmented if required by the direct use of the objects and methods in the Sources application. There are several data conversion constructs that the DAS is familiar with. However others, specifically date mapping, have to be coded by the developer. Once this data is known, PROKAPPA will attempt to construct both the domain and mapping applications, enabling the mapping to be tested.

The objects and instances created in the mapping application take the form of one object for each table in the database and a group of instances containing row and column data for each table, as shown in Figure 6.4.

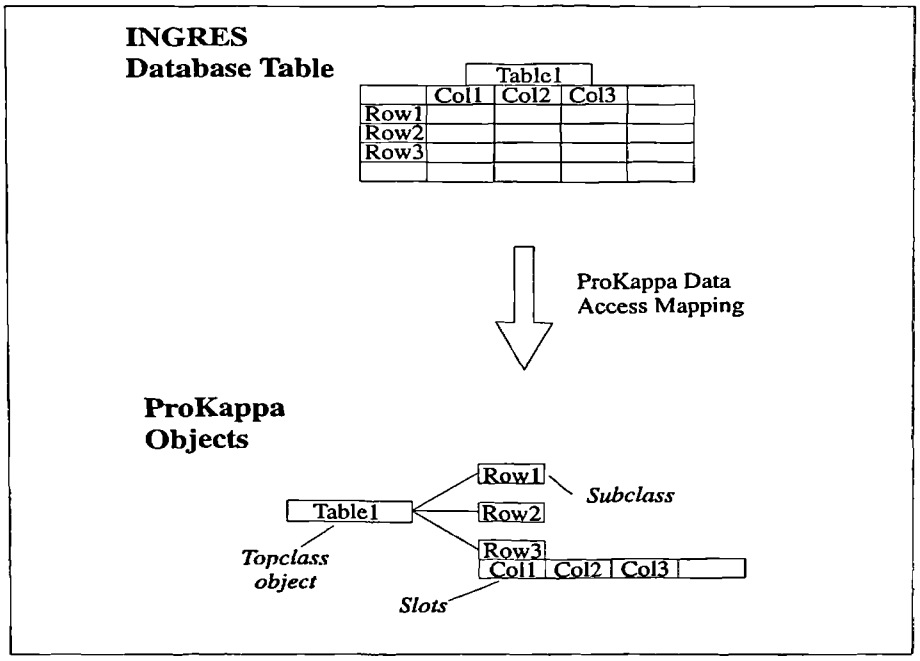


Figure 6.4 - Illustration of INGRES to ProKappa representation

The mapping and domain applications are shown diagramatically in Figure 6.5. In addition, instances are created whenever a Geotec to SIGMA transaction is carried out. The system defined slots on the mapping objects allow for a full and thorough examination and processing of the external data. It is with this functionality that the DAS provides a tightly coupled interface.

The mapping instances contain the full description of the fields contained within the specific tables, along with their format and other key data.

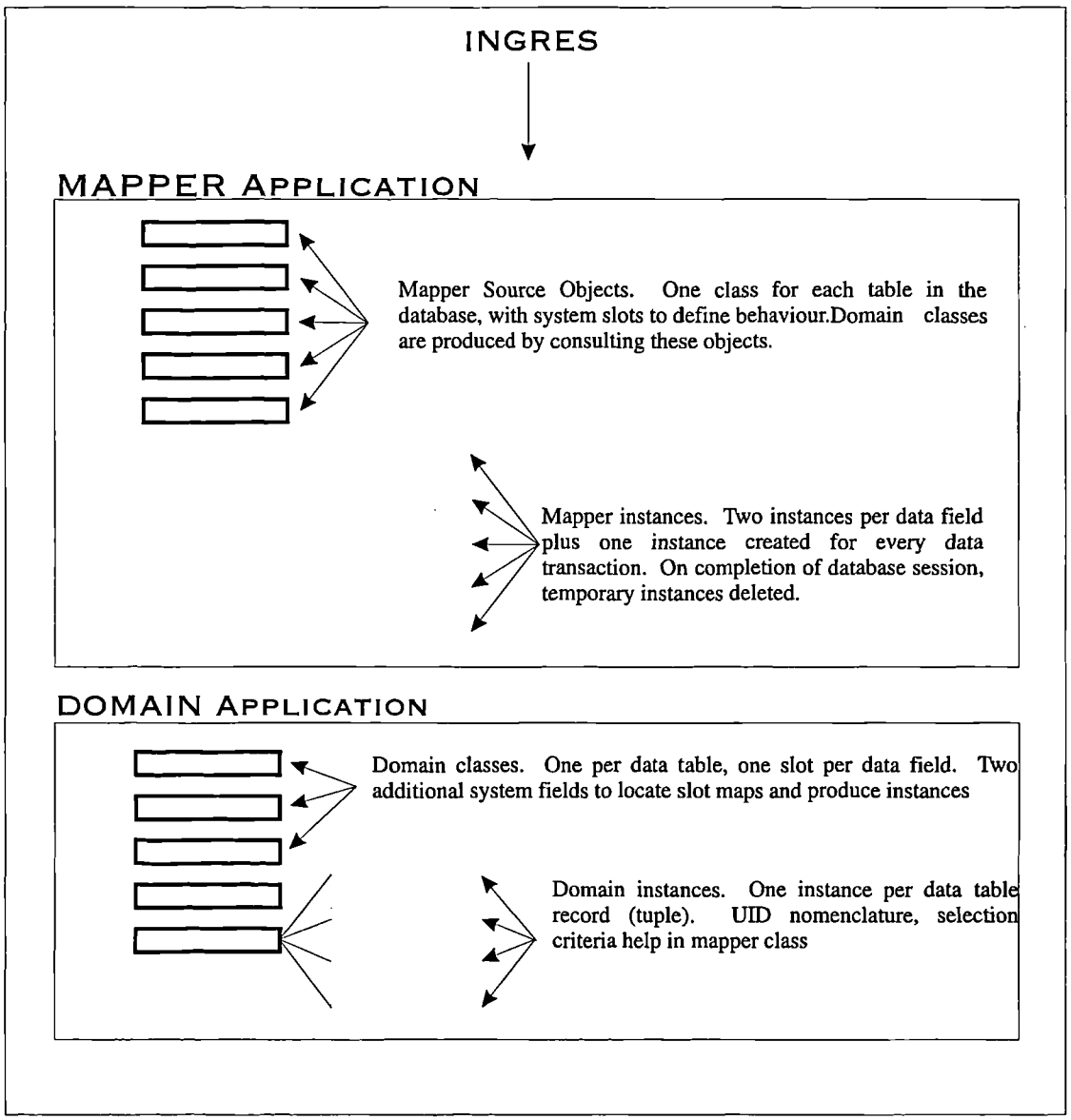


Figure 6.5 Diagrammatic representation of mapping and domain applications

The objects contained in the domain application are in essence replicas of the tables in the Geotec database. They are constructed from data held in the mapping application and on data retrieval, instances are created of these objects which represent specific rows from the relevant tables. The data may be held within the domain application or moved/copied elsewhere to be processed. However, if data is to be placed into the Geotec database, this can only be achieved from the domain application.

6.4.3 Creating SQL communications with ProKappa

Once developed, communication with the Geotec database is via standard SQL commands. The DAS has the ability to create its own SQL syntax from commands issued through the various constructs within the system, or to allow enhancement with additional user defined syntax. SIGMA creates this additional SQL which is utilised to specify exactly which records are required to be imported, updated or retrieved.

As mentioned in section 6.4.2, the mapping domain contains source objects for each of the data tables to be mapped across into ProKappa. There are 38 system (that is the Data Access System) defined slots on each of these class objects which give the DAS its functionality, shown in Appendix 7. Of these slots the majority have system defined values that are altered either on initialisation of the system or during its operation. The user, or indeed the system developer, has little control over these slot values other than those slots which are required to be directly manipulated during routine operation of the system. The slot **AdditionalWhereString** is one such case in point. The DAS checks this slot value whenever a database access is to be performed. If a value is present, the value is incorporated into the DAS formulated SQL syntax in the form of an addition to the **where** clause of the SQL statement (Section 4.5.2).

This methodology is widely used within SIGMA. The user selects the appropriate action/selection from a menu and, on a database access being required, the system will construct the relevant SQL. The syntax for the AdditionalWhereSlot is of the form

field = **value** or *field* in **(list of values)**

so the only knowledge required by SIGMA is the name of the appropriate data field that corresponds to the user selection. Several fields may be required to be incorporated into one **AdditionalWhereString** due to the complex nomenclature of the UID methodology. Once this SQL has been formulated it is assigned to the value of the **AdditionalWhereString** of the respective source object. This removes the user having to construct any SQL of their own and allows the system to handle all the data management with the DBMS.

6.5 Conclusions

Data structures have been developed for storing all aspects of a ground investigation and have been implemented as a geotechnical database, GeoTec, which forms the core of the Knowledge Based System SIGMA. The database design has been conducted within the framework of the AGS standard, complemented by the addition of structures to allow for storage of parsed soil/rock description data, multi-level test storage, which could include pictures or document files. The database has been designed using the relational data model and implemented using the INGRES Relational Database Management System.

The connection between the GeoTec database and SIGMA utilises an object based mapping system. This system means that the data transfer operates in the same object oriented manner as the majority of SIGMA subsystems ensuring a continuity across the system.

Chapter 7

Data Handling in SIGMA

7.1 Introduction

To fully utilise the potential of the centralised store of geotechnical information, data handling subroutines have been incorporated into SIGMA. These routines allow SIGMA to parse soil descriptions, interface with the GeoTec database, import data from AGS standard files and cross check data.

A soil description parser has been implemented, based on BS 5930 (1981) and the soil representation scheme suggested by Toll *et al* (1991). The parser processes a text sting containing the full soil description and breaks it down into individual components. It is capable of transferring data to the GeoTec database, incorporates exception handling (i.e. it has a method for handling words it does not recognise) and can be simply enhanced. It operates on a sequential two stage basis and has been designed in a modular manner using the object oriented functionality of the PROKAPPA environment.

Interfaces based on the PROKAPPA dialog box system have been implemented to provide access to the GeoTec database from the SIGMA environment. Data may be viewed, updated or deleted through these interfaces. Data may be imported into the GeoTec database via AGS standard files, although a limited amount of pre-processing is necessary to allow this. The facilities provided with PROKAPPA to read data directly from file were not sufficient for this task.

Data checking routines have been implemented which allow data from the GeoTec database to be imported into an analysis area and values can be cross-checked to ensure integrity within the system. In addition, parsed soil description data may also be used to further cross-check the data.

7.2 Necessity of a Parser in a Site Investigation KBS

As mentioned in Chapter 4, to fully utilise all the available information gained in a ground investigation it was found to be necessary to parse the soil description. A soil description is carried out by an experienced geotechnical specialist for the purpose of identifying layers within a borehole in order to facilitate a better understanding of the sub surface conditions. Individual layers can be recognised within the profile, each defined by depth and thickness. Engineering descriptions of soils are complex expressions containing no verbs, only adjectives and nouns. The vocabulary used is theoretically limited by the appropriate British Standard, BS5930, however in practise these limits cannot be rigorously enforced. This leads to terms being used in the description that, whilst commonly in use in the geotechnical field, are not defined by any standard.

The more detailed data the engineer/geologist can record on the appearance, feel and texture of the layer, the greater the aid to understanding. However, this can lead to a significant variation in the content of a soil description, for each engineer/geologist has their own subjective approach to the task. A comparison of borehole logs illustrates this point, some logs having layer descriptions within two lines whilst other logs have paragraphs to describe a single layer.

Obviously, the context of the site investigation is the most important factor in this. Some layers will necessitate a much lengthier description due to the complex structure of the strata. However, a degree of subjectivity has to be borne in mind when approaching the problem of parsing the soil description; the parser must be flexible enough to allow for this variability in descriptions yet robust enough to withstand day to day usage.

One use of the detailed qualitative information stored within a soil description, is for borehole interpolation, that is the interpretation between discrete points of sub surface ground conditions. Layers in neighbouring boreholes have to be compared in order to know if they comprise a continuation of the same layer. The base level of data that must be extracted from the description to fulfil this task are the dominant constituent types. However other detailed data can also be extracted and used to enhance both the interpolation process and general data integrity. The interpolation process can be improved by comparing the colour, consistency and structure of the layers (Vaptismas and Toll, 1993) and so this data needs to be identified within the description. In addition, qualitative data given within the description can be used to check against laboratory data to ensure the integrity of the data to be stored in the GeoTec database. For example, if within the description a layer has been described as stiff and yet the laboratory test results give the undrained shear strength as 50kPa, then further examination of this sample would be recommended. It is for these reasons that the decision was made to incorporate a soil description parser into SIGMA.

The decision to produce a custom built parser was taken for practical and compatibility reasons. Commercially available parsers which are mainly involved with the parsing of correctly structured sentences were not very useful as they would have needed major modifications to handle the problem of parsing a soil description. At the same time their full functionality would not have been utilised. Also, finding a commercially available

parser that was capable of being incorporated into the SIGMA environment would have been difficult. The parser that has been developed for use in the SIGMA project was based on previous work carried out by Vaptismas (1993) who produced a PC-based parser for soils descriptions, written in Prolog.

7.3 Design Methodology for Soil Description Parser

In the initial design stages of the parser, several fundamental principles were used to outline the requirements of the final system: The parser must :

1) Be based upon the relevant British Standard, and be able to be updated along with that standard.
2) Be able to produce suitable output for transfer to the GeoTec database using the tightly coupled linkage of the SIGMA environment.
3) Allow simple enhancement for improvement or addition as new geotechnical terms come into use.
4) Be able to handle terms with which it is unfamiliar, that is exception handling.
5) Be usable in both batch and individual case mode.

In order to achieve these aims, the parser was designed in a modular manner, using the object-oriented facilities of the PROKAPPA system. The parser's design follows the fundamentals of a top down recursive parser utilising single token lookahead (Bennett, 1990; Rayward-Smith, 1983), whereby the description is examined one term at a time and this term is passed to all available comparison functions or clauses. If a term is found to have matched a particular clause element to a suitable conclusion, the sequence is re-initiated with the next term in the description.

However, there are some occasions when a term is a part of a phrase, in which case the structure of the parser is such that the terms will be extracted until a phrase is fully identified and any required actions initiated. In the example *CLAY **with a little** sand*, the parser will extract the full phrase ***with a little*** and take the appropriate action for determining a minor constituent. This ability has required a large suite of support routines for the parser, some of them separate from the comparison clauses themselves. These support routines allow not only for the differing phraseology within the description but manage the interface of the parser to both the user and the remainder of the SIGMA environment.

7.4 Operation of the Soil Description Parser

7.4.1 Output requirements

By making use of the object-oriented facilities of PROKAPPA in both the parsing of the descriptions and the database transfer, the production of the results of the parsing process into a suitable form for transfer into the GeoTec database is simplified significantly. As discussed in Section 6.4.2, the mapping of data to and from the GeoTec database utilises the Sig93D domain, where class objects of the data tables are present and Sig93M domain, where control and source classes reside.

To place data into the database, instances have to be created of the relevant classes. These instances have to be named according to the UID nomenclature (section 6.4.1) and the appropriate data values assigned. The data values, that is those values that have been produced as a result of the parsing process, are much easier to assign if the final structure

of the parsing process is compatible with that of the Sig93D class. Using this criteria as one of the design principles for the parser (Section 7.3) shifts the emphasis of data transfer into the main parsing algorithm rather than post processing a solution.

As previously stated in Section 4.5.2.1, the structure of a parsed soils description can be represented in five data tables, namely: *lay* - layer, *strt* - stratum, *cnst* - constituent, *stst* - stratum structure and *ctcl* - constituent colour, as shown in Figure 7.1.

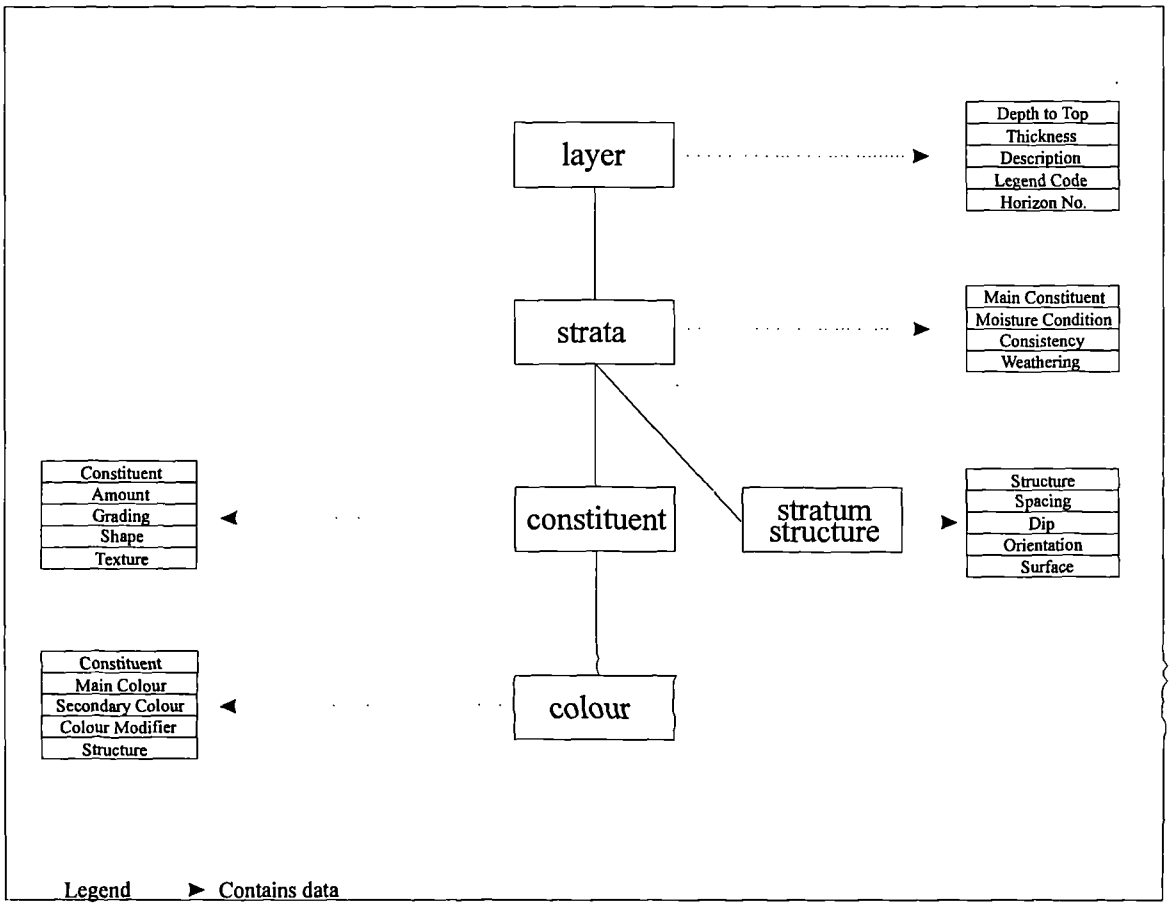


Figure 7.1 - Full detail of storage of parsed soil description

To facilitate the transfer process, the parser has been designed so that it has the same structure as that of the data tables that will be receiving the parsed soils data. As will be discussed within the following sections, this has required a very specific methodology to

be enforced within the parser itself in terms of internal functionality, yet the final outcome is one that is fully compatible with the GeoTec data structure.

7.4.2 Data Import for parser

The parser operates on the principle that the data to be parsed resides within the GeoTec database. Data may be imported into the parser either on an individual or batch basis, however due to the nature of the exception handling routines, see Section 7.4.5, the actual descriptions are parsed individually.

To import an individual layer into the parser, the investigation id, the borehole number and the layer number must be entered into the appropriate entry box on the parser dialog box, Figure 7.2. On pressing the import push button SIGMA produces the relevant text string which is then passed to the **AdditionalWhereString** slot of the lay_Source class object, as previously detailed in Section 6.4. A initiation signal is then sent to the **Get!** slot of the same source object, lay_Source, which initiates the data import procedure. On completion of this procedure the appropriate instance is created in the Sig93D domain. This is the manner in which all ProKappa data import transactions are carried out. Once the instance has been created in the Sig93D domain its name is placed in the 'Available for parsing' list box in the parsing dialog box.

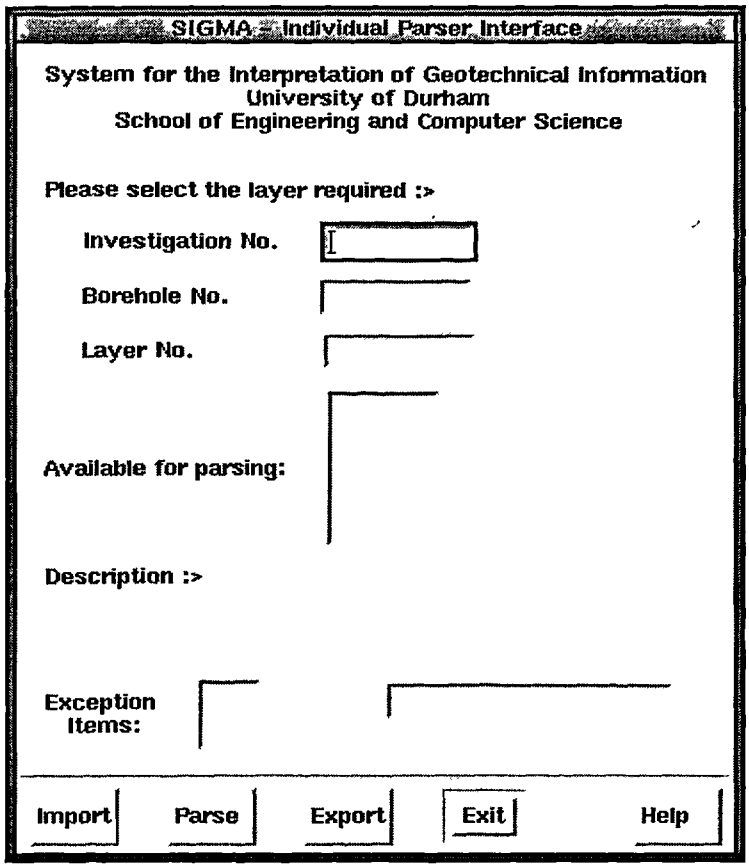


Figure 7.2 - Parser Interface

If the user wishes to import more than one layer description, that is a batch of descriptions, this may be accomplished using the same interface. If the user wished to parse all the descriptions of a particular borehole, then the layer entry box is simply left blank and the system will import all the layers. Selecting certain layers in a particular borehole may be accomplished by listing those layers required by number in the layer entry box separated by commas. The same method can be used for boreholes by listing those boreholes required in a list separated by commas in the borehole entry box. This is illustrated in Figure 7.3. When the appropriate layer instances have been imported into the Sig93D domain their names are placed in the 'Available for parsing' list box.

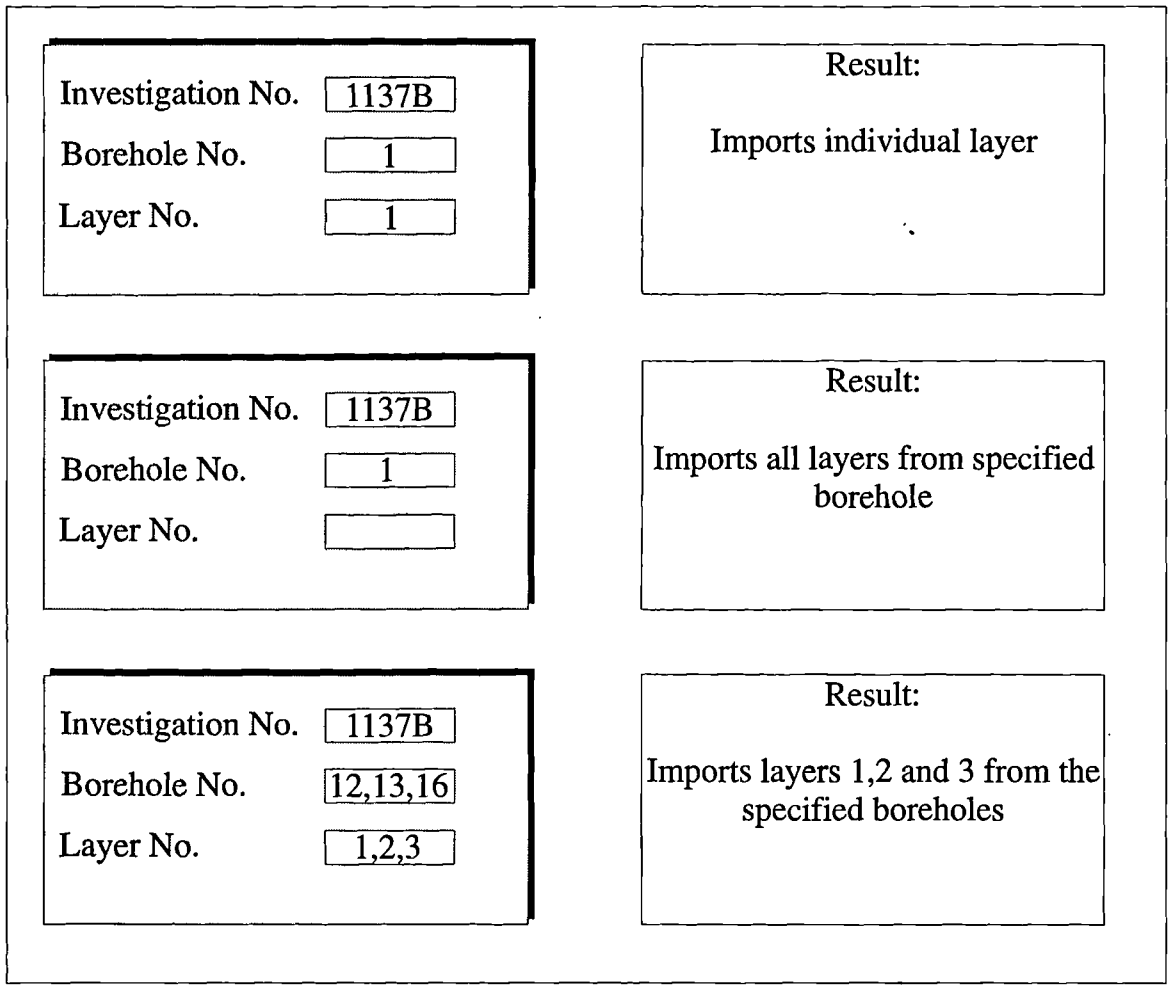


Figure 7.3 - Illustration of batch layer selection for parser module

Once all the names have been placed in the 'Available for parsing' list box, the user may highlight the instance to be parsed. The instances are listed in order of the UID nomenclature, allowing the user to identify which layer the particular instances corresponds to.

7.4.3 Parser Operation

In order to simply describe the operation of the parser, its basic facets have been represented in a flow diagram, Figure 7.4.

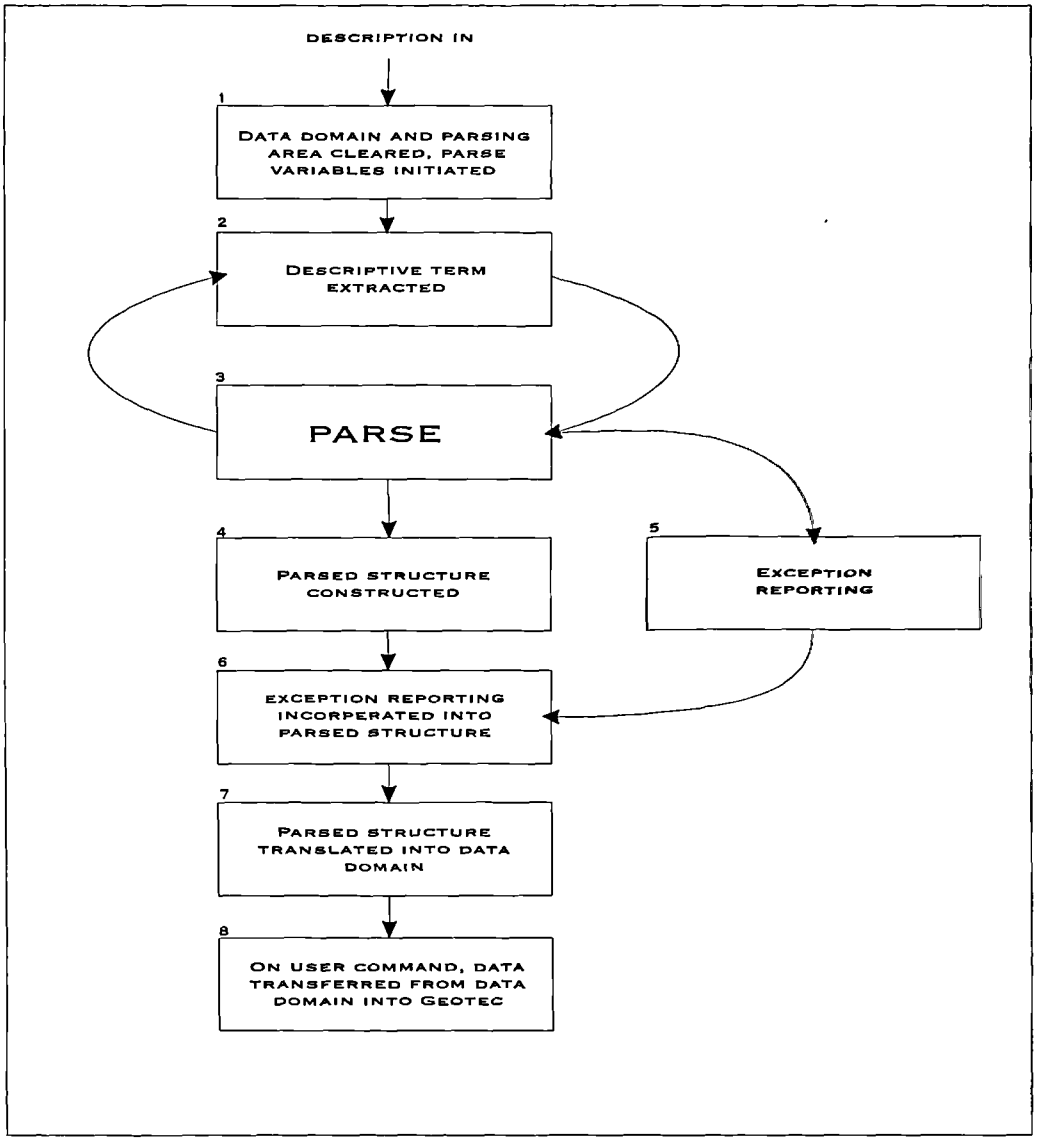


Figure 7.4 - Flow chart of the soil description parser

On initiation of the parsing process by the user, the system ensures that both data transfer application domains are cleared of previous data. SIG93D, the database mapping domain area, has all the instances of any previous data import cleared from those objects used in the parsing routines. Other objects do not have their instances removed as they may be required by other ongoing SIGMA routines. Sig93M, the database mapping domain, has all the instances from a connection to database session automatically deleted when that session is closed by the user. All objects that are present in PSD, the parser's working

area or domain (Section 3.5.3.1) are deleted in order to commence a new parsing batch, except for the top-level objects. This deletion is carried out automatically to ensure that starting point for the parsing routines is always consistent.

The final stage of the initialisation process is the setting of various global variables to their initial state and the placing of a copy of the description into the appropriate slot on the top level object **SoilDes**. In the context of this object-oriented environment, the phrase global variables refers to slots on a calculation object, **Kalker**, that are addressable by all objects, external sources, user and system defined routines. **Kalker** contains counter slots, flags, temporary variables and lists that are utilised in the parsing algorithm.

The cycle of parsing involves simple sequential extraction, using the clause *extract*, and the passing of the appropriate term to all the clauses in the parsing system. The methodology of the *extract* clause is to move through the description locating the next blank space. The manner in which the clauses are acquired by the parser is covered in Section 7.4.4.1, that is by a non-deterministic call to the subclasses of the object **parser**. If a term is encountered that is understood by the parser yet is incomplete by itself, for example the modifier *moderately,* this is stored on the **Kalker** and the next term in the description is extracted.

As the parsing progresses, an intermediate parsed structure (Section 7.4.6) is built up until the extraction routines arrive at the end of the description. Exception routines will then allow the user to manually incorporate any data unknown to the parser (Section 7.4.5). When this task is completed, the parser then converts the intermediate objects into those that are suitable for transfer in to the database domain, SIG93D, whilst deleting the intermediate object hierarchy. As stated in section 7.4.1, this final stage

object hierarchy has the same structure as the data tables within GeoTec so aiding the transfer process.

7.4.4 Parser Clauses

7.4.4.1 Object Oriented functionality of the Parser Clauses

Each comparison clause, or function, is stored in the slot *parser_clause* of an object of the same name, all objects being subclasses of the object **parser**, see Figure 7.5. This allows parsing to be initiated using a non-deterministic call to **parser**, enabling all the available sub-classes to be used without their names having to be known.

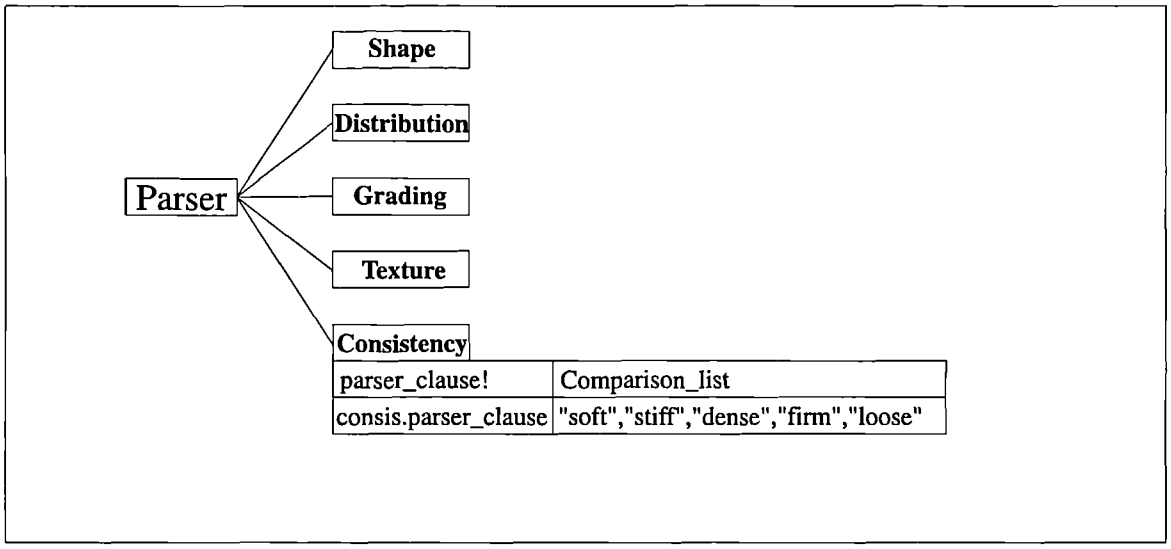


Figure 7.5 - Example of the object hierarchy for the parser

The non-deterministic call gathers into a symbolic list all **parsers'** subclasses, thereby allowing this list to be processed by extracting one symbol at a time. This allows for generic growth of the parser to be managed without the addition of complex ProTalk programme code, a new subclass object is simply added to **parser** and the new comparison clause will then be consulted on each new parse. Also this structure allows

for simple addition of vocabulary to an existing parser clause (Section 7.4.4.2). As mentioned earlier in this section, PROKAPPA's object hierarchies are intensively used in the actual representation of the description during the parsing processes and the transfer to the GeoTec database.

The syntax of a soil description presents an interesting grammatical problem, in that the structure possesses certain rules yet the order of individual terms tends to the unsystematic. The sequential parsing structure described in section 7.3 in conjunction with the object oriented approach allows this variability to be easily managed. The variation in which the terms can occur is shown in Figure 7.6.

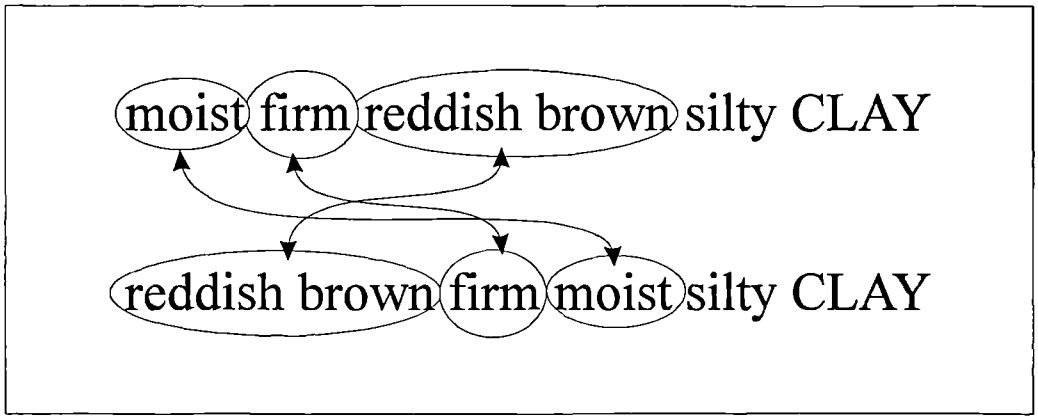


Figure 7.6 - Variation in placement of terms in soil description

The actual positioning of the term moist is of no consequence to the parser as the term will be passed to the appropriate clause wherever in the syntax it is met. The soil type itself is broken down into the individual components (Toll *et al*, 1991), each identified as an amount as shown in Table 7.1. In the example shown in Figure 7.6 the soil type *silty CLAY* would be broken down into *CLAY* - **main**: *silt* - **secondary**.

Soil Type, descriptive term	Amount	Example
	Main	CLAY
very	Major	very clayey
y	Secondary	clayey
slightly	Minor	slightly clayey

Table 7.1 - Soil amount representation scheme

PROKAPPA's object oriented functionality also proves useful in the building up of the final structure of the parsed description. As data values are inherited down an object hierarchy, data pertaining to classes lower in the structure, for example at the constituent colour level, is automatically assigned during the parsing process, see section 7.4.4.4. This ensures both data continuity and the correct naming of the transfer instances.

7.4.4.2 Clause Structure

Where possible a standard form has been used for the clauses of the parser, as illustrated in Figure 7.7, and an example clause is shown in Figure 7.18.

This standard form can be seen to consist of five distinct phases. When a new term is passed to a clause it must be ascertained whether or not a match has already been made. This is accomplished using the Kalker.Hit notation (Section 7.4.5), whereby if a previous clause has matched a term, this variable will be set as a flag. This reduces the processing time of one parsing sweep, as the remainder of the clauses can be ignored. If this initial test proves negative then flow through the clause continues with the generation of the list of comparison items. This data is held in the slot *comparison_list* on the appropriate **parser** subclass object, in the form of a list of allowable terms for a particular clause, as shown previously in Figure 7.5.

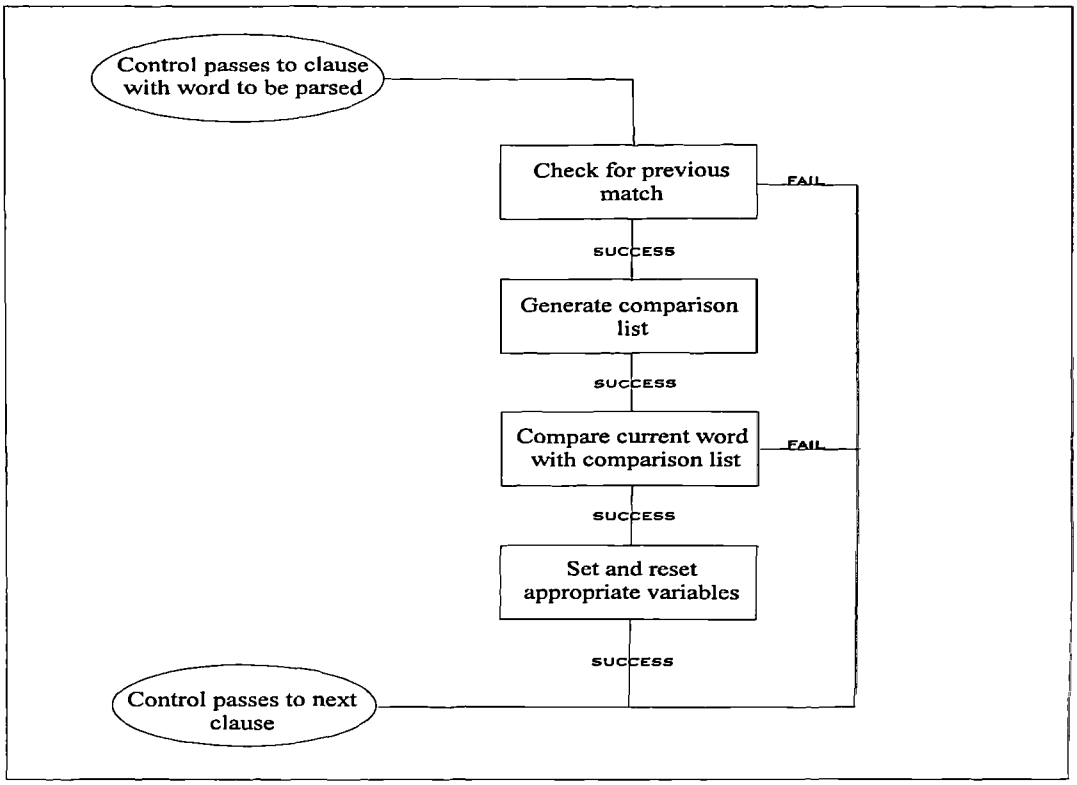


Figure 7.7 - Program flow through standard parser clause

This allows the data for the comparison to be easily obtainable by the parsing module and also allows for simple updating of the vocabulary in the list. If another term needs to be added to the list the user selects the appropriate slot and simply adds the new term. In all subsequent parsing sweeps, this new term will be included.

One word at a time is extracted from the comparison list and compared with the term that has been passed to the parser clause. If a match is found then the appropriate temporary object (Section 7.4.6) is set to this value, in addition to the value of Kalker.ModKalk. This ModKalk variable is initialised by the call to the function modify() which ascertains if the term being processed is a modifier, for example *very*. The **modifier** object, a subclass of **parser**, contains slots for each term that may have a modifier present. Modify() checks the relevant modifier slot to check if the term is a valid modifier, and if it is sets the value of Kalker.ModKalk to that modifier value. The constituent functions

ensure that no confusion arises with the same terms when used to describe a constituent amount, as in Table 7.1. The final action of the function is to set the Kalker.Hit variable to ensure that no more processing is carried out on this particular term.

```
method shape.parser_clause!(?word, ?ls)
{
   if Kalker.hit == "hit";
   then fail;
   ?word2 = ConvertToString(modify(?self, ?word, ?ls));
   if ListLength(FindListElmt(all shape.comparison_list, ?word2)) > 0;
   {
         Kalker.Shape = AppendStrings(Kalker.ModKalk, ?word2);
         Kalker.ModKalk = "";
         Kalker.hit = "hit";
   }
}
```

Figure 7.8 Example of parser clause

7.4.4.3 Rules / ProTalk

The rules that govern the syntax of the soil description can generally be expressed in the form of *if... then* clauses, as shown in Figure 7.9.

if last letter of word is **y**
AND
(word - last letter) *inlist* (silt, clay, sand, gravel, cobble, boulder)
AND
modifier **very** not encountered
THEN
constituent amount *equals* secondary

Figure 7.9 Example of general rule implemented in the parser

In addition, this rule can provide a good example of the flexibility required from the parser, in that two of the secondary soil types do not conform with this rule, namely *clayey* and *gravelly*. It is in situations such as this that the versatility of the ProTalk language allows for constructs to be implemented to enhance this simple rule to allow all the terms to be compared within one clause (Figure 7.10). The inclusion of the additional check for the modifier *very* is to ensure that there is no confusion with the soil amount *major*.

```
SELECT
{
        CASE : last letter of word is y
        CASE : last 2 letters of word is ey
        CASE : last 2 letters of word is ly
}
AND
        or {(word - last letter) inlist (silt, clay, sand, gravel, cobble,
boulder);
        (word - 2 last letters) inlist (silt, clay, sand, gravel, cobble,
boulder);}
AND
modifier very not encountered
THEN
        constituent amount equals secondary
```

Figure 7.10 - Example of extended general parser clause.

7.4.4.4 Dominant Constituent Type

As discussed in the previous section, the parser is based on the descriptive terms laid out in BS5930 in conjunction with the representation scheme put forward by Toll *et al*, 1991, by which each constituent type is associated with an amount, see Table 7.1. This methodology for identifying the correct constituent type is very important to the operation of the parser as it allows the dominant constituent type to be separated from

the other constituents. It is this main or dominant constituent to which the preceding qualitative terms will apply and therefore non-standard clauses have been written to ensure their correct handling.

The identification of soil types and their amounts is carried out in a three stage process. Firstly, the word being parsed is passed to the clause *soil_type* which tests whether the term is a modifier for soil types, for example *slightly*. If this test is positive, the next word is extracted and immediately passed to the function *sec_st*, whose purpose is to identify the actual soil type, that is silty, clayey etc. If the test for a modifier is negative, the initial term itself is tested by *sec_st*, and failing that *main_st*, to determine if it is actually a dominant soil type.

sec_st has a similar basic structure to the example shown in Figure 7.10. Obviously the example is significantly simplified to aid the illustration. Both *sec_st* and *main_st* include routines to create new objects within the temporary object hierarchy as a positive result proves the existence of either a new stratum (*main_st*) or a new constituent (*sec_st*) within the description. There are other stratum related structural terms that will also necessitate the creation of new objects, *interbedded* for example (Section 7.4.6). The new objects are created as subclasses of their superclass, inheriting those slots already defined and being given new slots as required. This allows for the passing down of the common data, for example investigation, borehole and layer number at constituent level and main soil type or amount at lower levels.

7.4.5 Exception Handling

Exception handling, or the ability to deal with unknown terms, is an important facet of the parser as it allows it to operate within a very flexible environment. The exception handling currently implemented within the parser can accommodate two eventualities,

that is a mis-spelt word or one that lies outside of the current vocabulary of the parser. Within SIGMA both of these occurrences can be managed utilising the same methodology.

As can be seen from Figure 7.8, the first action of every clause is to check a global variable **Kalker.Hit**, which in reality is the slot *Hit* on the object **Kalker**. If Kalker.Hit is set to True, then the clause is immediately exited as the particular term has been identified. Utilising this principle, if at completion of one parsing sweep Kalker.Hit is still set to False, or in ProTalk terms Null, then this term is unknown to the parser and requires exception handling. This term is placed in the Exception Items list box of the Parser dialog box, Figure 7.2, and the parser continues with the next term.

On completion of the parsing of the entire soil description the user is prompted to deal with those terms in the exception list. On selecting a term from the exception list box it is automatically placed in the adjacent entry box, allowing the term to be corrected or removed. If the term is corrected, due to spelling mistake, the newly corrected term is replaced into the original layer description and the mis-spelt term removed from the exception list. If the term is deleted from the entry box it is also deleted from the original description and the exception list. The user may manually add the term to the comments slot of the appropriate instance in the final parsed structure or choose to ignore the term completely.

In either event on choosing the export option, whereby the parsed description is entered into the database, if the original description has been altered the user is reminded of this by a warning dialog box. This warning reminds the user that the original description has been changed and gives the option of updating the original description or leaving it unchanged. This warning is important in that if data residing in the database is to be

changed, then it must always be with the consent of the user. SIGMA, as a Decision Support System, should also act as an assistant to the user and never attempt to take control.

7.4.6 Construction of the Parsed Structure

Due to the variability in the syntax of the soil description (see sections 3.2 and 7.2) the parsing takes place in a two stage operation to enable complex descriptions to be broken down. Firstly, the parsed description is built up into temporary objects, adhering to the final structure but having a modified slot pattern and nomenclature. This allows for any variability in the description to be accommodated. As main soil types, constituents and structures are discovered, the temporary model is translated into a static final structure with the same hierarchy. In theory, the temporary model could be transposed directly across to the Sig93D domain, the only area from which data may be transferred to the database, but this would leave no room for error in the transferring of the data. It is vital that the data is consistent before transfer, so the final parsed data model resides in the PSD domain and the transfer to GeoTec takes place as a separate process.

This may at first appear to be over complication, but the constraints of the Data Access System with the UID methodology (see Section 4.7) require a unique name for each instance to be imported into the GeoTec database. This unique identifier, UID, is constructed from the keys of the particular table to which the record is to be added and due to the complexity of the data structure (Figure 7.1), counters are employed to identify the specific layer, constituent, colour and structure. To conform with the UID methodology these counters must be employed as keys to name each instance uniquely.

For example, in site investigation PK/1234, a borehole BH907 contains several layers, layer 3 having the description *brownish red becoming red silty CLAY.* The colour

instances of the parsed soil descriptions are named according to the key of *cscl*, the constituent colour table. The keys for this table are shown in Table 7.2

Field	Value colour 1	Value colour 2
proj_id	PK/1234	PK/1234
hole_id	BH907	BH907
lay_no	3	3
strt_no	1	1
cons_no	2	2
ctcl_clno	1	2
ctcl_mcl	red	red
ctcl_scl	brown	
ctcl_strc	top	base

Table 7.2 - Make up of key for constituent colour table

and the associated UID's for this example would be **Colour 1 instance UID** = cscl("PK/1234", "BH907", 3,1,2,1), **Colour 2 instance UID** = cscl("PK/1234", "BH907", 3,1,2,2). As can be seen, if several constituent classes and sub classes are present in a particular description, the direct naming of instances becomes complex. The use of an intermediate or temporary object hierarchy to store the developing parsed description allows the completed description to be easily converted into a form suitable for transfer. It enables the model to have a dynamic quality, so that if several strata are present within the one layer, each with several constituents and associated colours and structures, the model is able to store this data in temporary objects until the dominant constituent is located. The only other method of adapting to the complex nature of the description would be to store all the associated data on the one calculation object which could become unfeasibly large with complex descriptions. This methodology allows for both simple and complex descriptions to be accommodated within the same algorithm.

Take for example the layer description *red silty CLAY interbedded with sand.* Figure 7.11 shows the temporary structure directly before the parsing run.

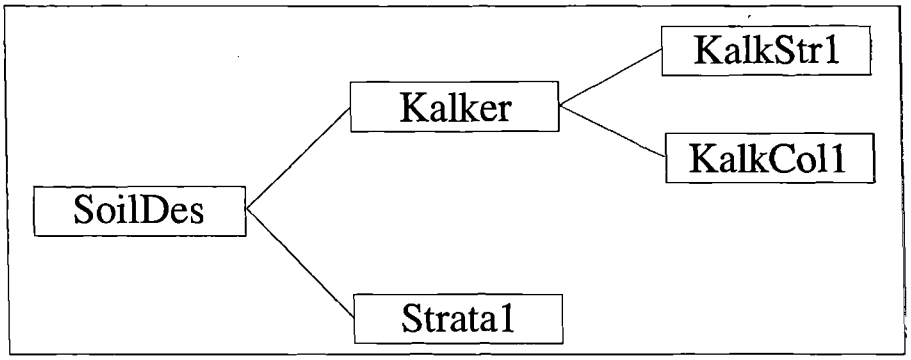


Figure 7.11 Initial state of parser object hierarchy

Kalker, **KalkStr1** and **KalkCol1** are the general, structure and colour temporary variables and **SoilDes** is the top level object which contains the description to be parsed, as well as investigation, borehole and layer numbers. The object **Strata1** exists at this stage as every description by definition will contain at least one main soil type. The parser examines the term *red*, allocates this as a main colour on the **KalkCol1** object and progresses to the next term *silty*. *Silty* is identified as a secondary constituent and as such is allocated a new class as a subclass of **Strata1** and named **Strata1Ct1** (Figure 7.12).

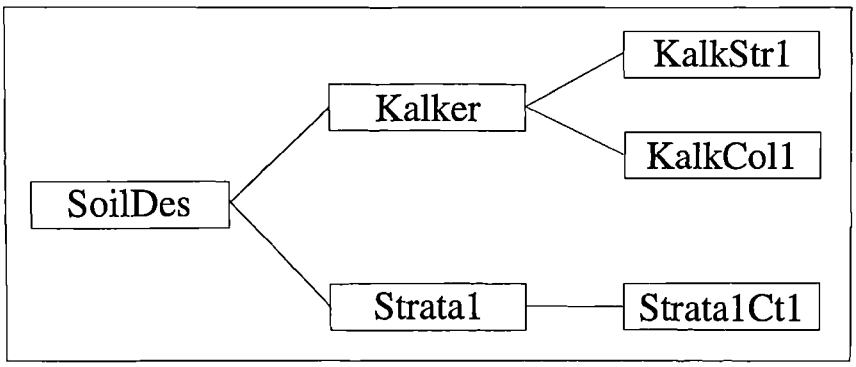


Figure 7.12 - Parser object hierarchy after identifying one soil constituent

The parser now examines the next term which is identified as the main soil type *CLAY*. A new subclass of **Strata1**, **Strata1Ct2**, is created and allocated the soil amount to be *main* and the soil type to be *clay*, Figure 7.13. A subclass of this object is also created, **Strata1Ct2C1**, to contain the colour details for the main soil type, that is the main colour *red,* which has been held up till now in the temporary variable **KalkCol1**.

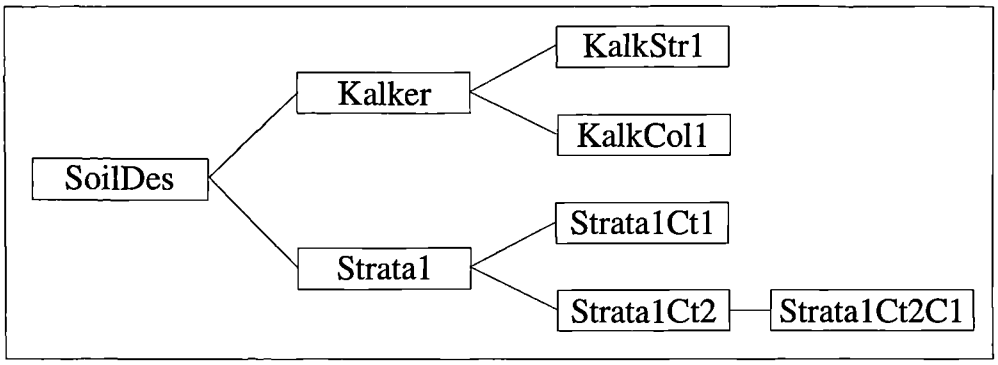


Figure 7.13 - Parser object hierarchy after identifying main soil type

The parser then extracts the next term and identifies it as *interbedded* which signals that **Strata1** is now complete and that a new class **Strata2** is to be created as a subclass of **SoilDes,** Figure 7.14.

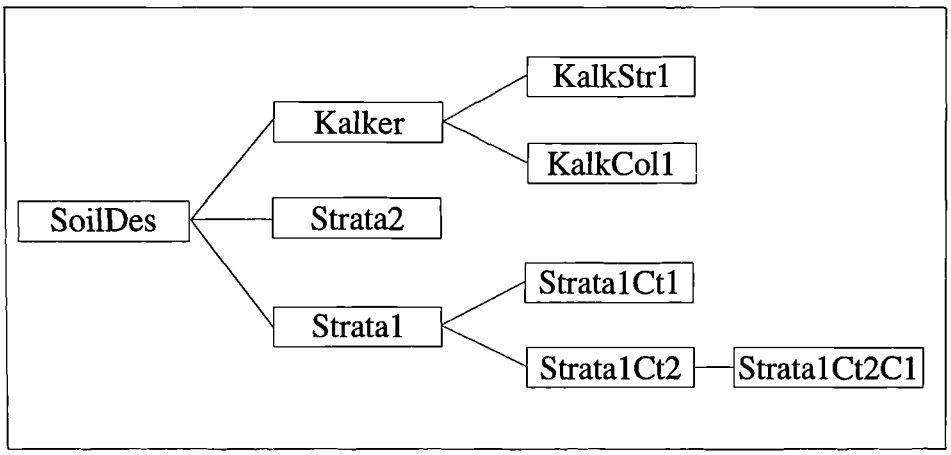


Figure 7.14 - Parser object hierarchy after identifying main soil type

The **Strata1** hierarchy is now complete and all the **Kalker** variables are reset, apart from the soil counter which is incremented. The parser, still within the *interbedded* clause, extracts the next term which is identified as *with*, which is within the limited syntax expected to follow in the phraseology of interbedded. Accordingly the parser extracts the next term and identifies this as a main soil type, *SAND*, and creates the appropriate subclass of **Strata2, Strata2Ct1** and sets the appropriate values of soil amount and soil type. On attempting to extract the next term the parser is presented with an empty string signifying the end of the description. The calculation objects are then discarded and a new subclass of **SoilDes** created, **Str1/2,** which contains the *interbedded* term and is in effect the mapping for the data table *stst*, stratum structure. The final structure ready for any exception processing and transfer to the database via Sig93D is shown in Figure 7.15

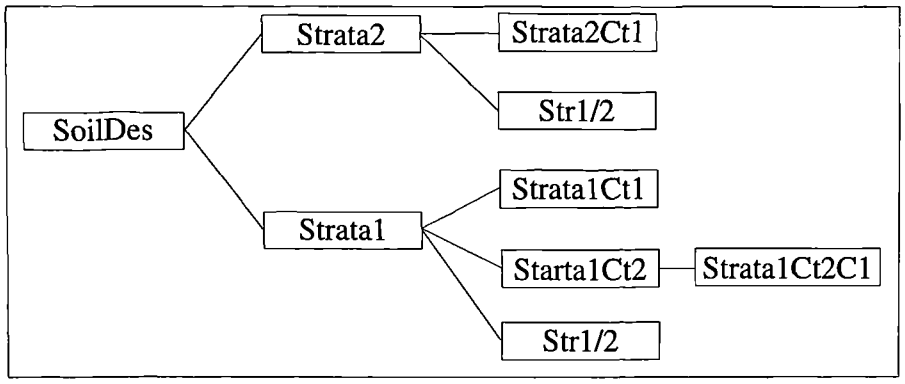


Figure 7.15 - Final parser object hierarchy

The naming of the objects in this structure, for example **Strata1Ct2C1**, are purely for ease of recognition and construction. The non-deterministic nature of ProTalk means that the system does not need to know the names of the particular subclasses and the UID methodology necessitates renaming of the instances when they are transposed into Sig93D, section 6.4.1.

7.5 Interfacing to the GeoTec database with SIGMA

The ability to treat the GeoTec database as a standalone database within the confines of the tightly coupled enhanced expert system (section 4.5.1), is important to the concept of SIGMA. For this reason there are direct interfaces that allow manipulation of GeoTec entities via SIGMA. These interfaces allow for the manual entry of data into any of GeoTec's tables as well as the ability to update and delete. These interfaces utilise the PROKAPPA Dialog Box functions that are provided with the development environment. It would be possible for a user to use these routines to fully populate the database but the process would be very time consuming and inefficient. The ability to enter the data from AGS format files is described later would be used for entering the majority of the data. Therefore these direct entry routines would be used for the addition of supplementary details such as test results, the updating of data and manual browsing of records by the user. An experienced user may wish to have direct access to the data, rather than relying entirely on SIGMA for interpretation.

Each table has an interface through SIGMA, examples of which are shown in Appendix 4. As mentioned in Section 3.3.3.3, the dialog boxes and their controls supplied through DialogBoxApp provide the full functionality required to build and operate the interfaces. It utilises the concept that a dialog box owns the controls that are its constituent elements. The dialog box itself and all the controls are instances of the blueprint classes that exist in DialogBoxApp. The interfaces can either be constructed programatically when required or permanently using the Interface Workbench, a production tool provided in the development environment. Initially the programmatic approach was used. However this method whilst providing the flexibility for SIGMA itself to generate its interfaces as necessary, provided cumbersome in use. The programming code required to produce the dialog boxes and their controls, to initiate the various settings

and provide the behaviour was lengthy and proved very slow in operation. A discussion of interface techniques is included in Chapter 2.

Using the interface workbench provided the more suitable solution, enabling rapid production of the interfaces and due to their permanent nature the speed of the system was enhanced. In addition, general behavioural routines were able to be written which further increased their efficiency. This approach does however lead to the production of many instances which are an additional load to the overall performance of SIGMA. As previously mentioned, the PROKAPPA object modeller is its most efficient asset and theoretically unaffected by the number of objects associated with an application. However, in a system the size of SIGMA this can be seen to be not wholly the case, with performance degradation appearing as the object bases become massive. This subject will be discussed in further detail in Chapter 9.

7.6 AGS Data Entry

One of the main reasons for the utilisation of the AGS Electronic Data Transfer standard (AGS, 1992) was to enable the inclusion of that data into the GeoTec database. This inclusion enables data from many sites to be easily and simply inserted into the database and thereby accessible to SIGMA. The larger the quantity of data available and the ease with which this can be incorporated into the system for analysis, the more meaningful and useful results can be drawn, see Section 3.

An example of the AGS format is given in Figure 7.16. The use of headers and format specifiers allows not only for the electronic transfer between information systems but also for ease of reading by an engineer. This is important as it does not restrict this format

purely to computer systems, an often encountered side-effect of implementing information technology in new areas.

```
"**PROJ"
"*PROJ_ID","*PROJ_NAME","*PROJ_CLNT"
"123/abc","Towy Valley Cyder Company","A.Client and Partners"

"**HOLE"
"*HOLE_ID","*HOLE_TYPE","*HOLE_HOLE_NATE","*HOLE_NATN",
"*HOLE_GL","*HOLE_FDEP","*HOLE_STAR","*HOLE_LOG"
"501","","554293","221884","91.90","30.6","","A.O."


"**GEOL"
"*HOLE_ID","*GEOL_TOP","*GEOL_BASE","*GEOL_DESC"
"501","0.0","14.1","Very stiff brown slightly sandy CLAY with extremely closely
"<CONT>","","","spaced fissures"
```

Figure 7.16 - Example of AGS Data File

There are several points to note with regards to the AGS format, namely:

1) Each data group, or table, has the format specifier ** and is followed by an ordered list of data fields, for which the format specifier is *, and their associated data values.

2) If no data is provided for a particular item, a blank set of quotation marks is substituted. This reduces the potential for incorrect data reading as all data values have an associated value, even if that value is Null.

3) If a data value is longer than the specified length of the ASCII file line, a continuation symbol "<CONT>" is used to indicate that the data on the next line is merely an addition to the preceding. Again, these format specifiers allow the extraction routines to be easily automated.

4) The field types of all the data values, or fields, are given as character strings, being enclosed in double quotation marks. This will necessarily entail data conversion routines to be either written by pre-processing software or routines inherent within the RDBMS. Data conversion is an issue which can present problems in the transfer of data between database systems, and is traditionally an area where standards tend to have subtle differences. With data transfer between PC based systems this is not particularly a problem as data types are dealt with in a simpler manner and in some cases are purely text based. However with larger PC and workstation based RDBMSs, very careful attention must be paid to the types of data being converted otherwise anomalies may occur. The use of text strings is the most flexible way of approaching the problem. However the onus is placed on the receiver of the data to process the incoming data accordingly. This does require some pre-processing on behalf of the receiver.

Whilst the structure of the GeoTec database has been based on the AGS format, there are differences, for example in table names and the inclusion of parsed soils data. These differences, along with basic data processing requirements, lead to a requirement for pre-processing of the data before entry into the GeoTec database. Due to the nature of these pre-processing routines, a procedural language like C would be used to ensure the most efficient result.

PROKAPPA's Data Access System, DAS, has a flat file transfer option, but in practise this was found to be only suitable for writing data to a file. On data import, or reading a

flat file, the software is not flexible enough to allow the AGS file to be read even with extensive pre-processing. The DAS requires that the same data resides in the same column throughout the file; the only manner in which the data transfer could be utilised throughout the DAS is by sub-dividing each AGS file into several table files, a clumsy and impractical approach. Prototype programs have been written which have read data direct into the GeoTec database via pre-processed AGS format files. These routines utilise the ability of INGRES to read data into specified tables from ASCII files.

7.7 Data checking within SIGMA

Site investigations by their very nature produce a wealth of information. It is the task of the geotechnical specialist to organise and analyse this data in order to make meaningful and reasoned decisions. The advantages of a centralised data store have been mentioned previously (Section 4.2) as a method of increasing the efficiency and minimising the possibility of inaccuracies of data management. When this data is to be used for decision making it is important that the data is consistent, that there is no conflicting data, for this may lead to erroneous or ambiguous results.

To this end SIGMA has a data checking module specifically designed for this purpose, allowing the user to select specific borehole records, import them into an assessment area and perform consistency checks. The consistency checks currently take the form of simple rule based comparisons whereby values from a test are compared with values gained from other appropriate tests. Also it is proposed that the results of the parsed soil descriptions, qualitative data, can be used to check against quantitative data from tests. Although this type of comparison has less significance than direct test to test comparison they are still valid as an aid to the geotechnical specialist. The checks that can currently

be carried out are of a limited nature, merely demonstrating the functionality of such a system

When the user chooses the data checking option from the data handling menu, a dialog box requests the identification of the borehole and layer to be checked. Presently layers are checked individually, but it would be only a small modification to implement batch processing for boreholes. Once the process has been initiated, the data checking module extracts all the available details from the GeoTec database that correspond to the borehole layer selected. This is enabled by the use of the two test reference tables, *lbrf* and *isrf,* following the procedure outlined in section 6.3.2. The data checking module consults these tables to determine all the tests that have been carried out on the particular borehole layer and the respective table names.

Once the relevant data has been imported, rulesets are initiated that carry out the relevant comparisons. Whist PROTALK has an inbuilt rule system, using the *fcrule* and *bcrule* nomenclature, this was ignored in favour of custom written routines. The complication of the task in hand and the flexibility offered by custom written rulesets meant that they prove more effective.

Once the relevant test data for the requested layer has been imported, the tests are grouped and sorted according to their test code. Once the data has been collated into the appropriate groups, those tests and the number of tests carried out are reported to the user, as shown in Figure 7.17.

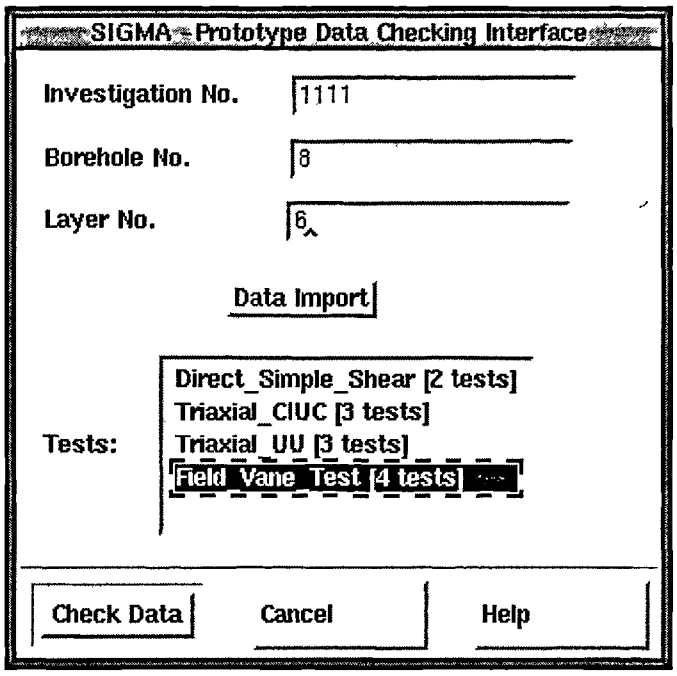


Figure 7.17 - Data Checking interface showing grouped tests for selected layer

SIGMA contains the knowledge of which tests produce comparable data, in the form of the TESTS knowledge base as previously described in section 5.3.2, and applies this knowledge to the given situation. Tests are assessed for their applicability in measuring a given parameter and those with similar applicability ratings have their associated measured parameter value compared. Presently, those tests having an applicability rating of high and medium (section 5.3.2.1) are compared with one another. User selection of applicability rating could be implemented allowing the user to decide the level of checking required for a specific comparison.

Those tests that are deemed suitable are then compared. Where individual tests have been taken in the specific layer, this value is used for comparison. Where several tests have been conducted, the values are averaged. The results are then displayed on the screen in a separate interface, see Figure 7.18. If the user selects one item from the main list box, all the appropriate test instances and their associated values are displayed in the

lower list box. The data reported in this lower list box are the UID of the actual instance, the result returned from the test and the percentage variance from the mean. In practicality. It is the identification of those results that lay significantly away from the mean that are like to interest the user. These 'rogue' results may indicate an important variation in a geotechnical property or merely an error. To assist in identifying any results that may be inconsistent or ambiguous, the system highlights those results that lie a specified distance from the mean, for that particular test.

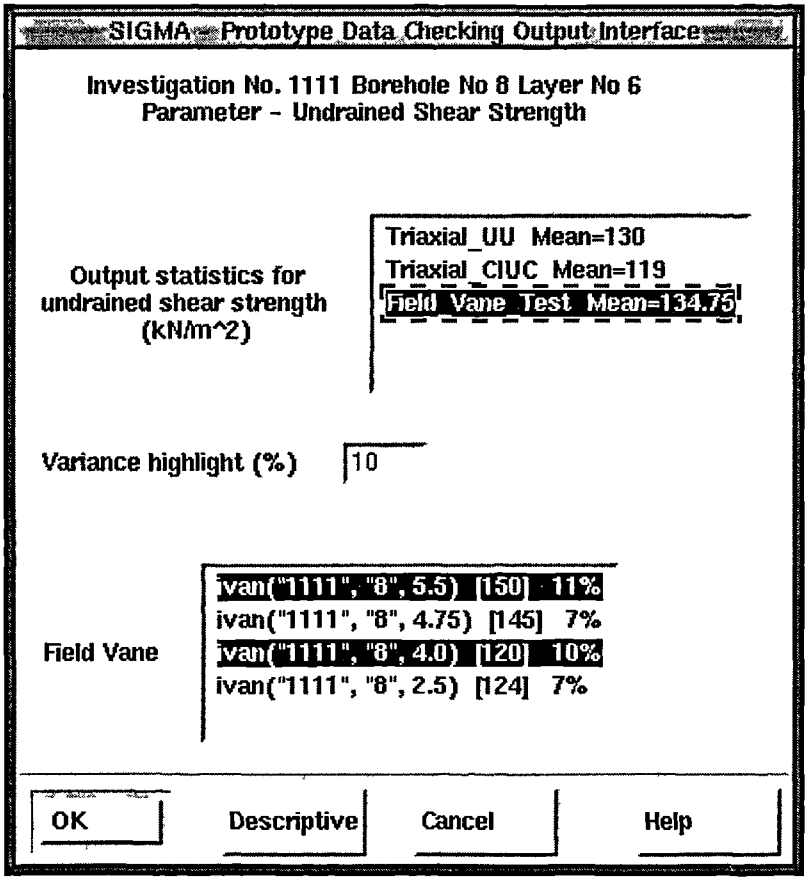


Figure 7.18 - Data Checking results interface

By this process any inconsistencies found during the data checking process are brought to the attention of the user. The user may select the variance to be highlighted in the lower list box by replacing the default 10% in the appropriate entry box. It is then the

responsibility of the user to investigate the reasons for this inconsistency and rectify the situation. The user may decide to accept the inconsistency as being within acceptable bounds of variability and leave the data unammended. Geotechnical data is by its very nature variable (Attewell and Farmer, 1976) and it is left to the geotechnical specialist to judge whether or not a particular parameter variability is acceptable.

Another course of action would be to mark the data as being doubtful and not to be used in further analysis. It is proposed that the mechanism for this is to use the comments field on the appropriate data table and associated data instances. A code could be placed within this slot which could act as a flag for other SIGMA and non-SIGMA routines, the presence of this flag indicating that the data may be of questionable accuracy. Other facets of the SIGMA environment, for example the database interfaces, can be used to investigate these inconsistencies or consult references external to SIGMA. A third possible course, altering the data, would be a dangerous practise unless the data stored in the database can be proven to be wrong, for instance if it were the result of a spelling mistake or some other error in data entry. It is in the identification of these types of errors that the data checking module is useful, in the role of a secondary checking facility.

SIGMA will never attempt to change any data in the GeoTec database, a fundamental principle in its role as a Decision Support System. Any changes that are made are initiated by the user, SIGMA's role being to bring to the attention of the user any possible data inconsistencies prior to analysis being carried out utilising this data.

The Ground knowledge base (Giolas, 1994) could also be utilised to cross-check qualitative data against the parameter being checked. For example, if a layer has an undrained shear strength of 280 kPa and yet has been described as soft within the

description of the layer there is an obvious inconsistency. Work is ongoing to incorporate this level of data checking within the overall environment of SIGMA.

7.8 Conclusions

With the implementation of the parser, SIGMA provides a level of functionality previously unavailable from other systems in the field, that is the ability to not only extract the constituent and dominant soil/rock types from a layer description but also other qualitative data. This additional information may be stored in the GeoTec database, complementing but not replacing the original text description. Utilising PROKAPPA's object oriented facilities also allows for simple addition and improvement of the parser, so adhering to the principle of transparency mentioned in Chapter 2.

The data handling routines mean that the database interfacing can be managed from within SIGMA's environment, although external access to GeoTec is always available. These interfaces allow detailed browsing of the data residing in the database. AGS standard files may be directly entered into GeoTec with some pre-processing.

On systems such as this where the integrity of the data is of such importance, the data checking routines described give the user the ability to ensure consistency and continuity of data. The more confidence the user has in the data being used within SIGMA, the more likely is the acceptance of SIGMA and such tools in the geotechnical workplace.

Chapter 8

Data Interpretation

8.1 Introduction

Previous chapters have covered the design behind SIGMA and the methodologies employed to manage site investigation data. This chapter deals with how SIGMA can provide tools to assist the geotechnical specialist in interpreting site investigation data. Utilising the wealth of data that is available from a central database and combining this with knowledge stored within the Knowledge Based System (KBS) are important facets in systems such as SIGMA.

The parameter assessment module is an example of how a central database facility can be put to a practical use for the geotechnical engineer. When operating at the design stage of a project it is important for the geotechnical specialist to be able to identify the value of particular parameters at given locations throughout the site. Using the parameter assessment module the user may focus in on a particular location and depth and extract the required parameter data. This data is displayed to the user along with the associated test code of the tests used to measure the parameter, as well as indications of the reliability and applicability of these tests. Correlation routines may be employed to augment the assessment exercise. The section concludes with a description of the operation of a session with the parameter assessment module.

Borehole interpolation routines have been included to assist the engineer in understanding the sub surface ground conditions. Previous work in this area is

described leading to a discussion of the methodology employed by Vaptismas and Toll (1993). This is followed by an outline of the object oriented approach used to implement these routines within SIGMA. Examples are given of some of the dynamic object hierarchies that are used in the analysis of the data followed by a detailed description of one of the main functions used. The proposed object oriented methodology for the implementation of the interpretation routines is then discussed followed with an illustration of a typical session.

8.2 Parameter Assessment with SIGMA

The parameter assessment module in SIGMA provides data and knowledge to the geotechnical specialist in order to assist in the choice of a value for a particular parameter to be used for design purposes. On some occasions the parameter required may have been measured with a variety of different tests. On others it may be that no direct measurements have been made. In this situation an indication of the parameter value may be assessed from other information. SIGMA provides three levels of parameter assessment (Toll and Oliver, 1993):

1) From direct measurements of the parameter
2) From correlations with other test results
3) From the engineering description of the ground

If the parameter has been directly measured, the system extracts these data from the relevant data-tables. The results are presented separately for different test types, and knowledge about the reliability of the test to measure the parameter is also provided (from the Tests knowledge base). The applicability of the test to measure the parameter in the type of ground of the chosen layer is also reported at this time. This

direct measurement stage demonstrates how efficient data management can assist the geotechnical specialist in fulfilling their task.

When the directly measured data are retrieved from the database, any other measured data that have been obtained are also extracted. The Parameter Correlation knowledge base can then be accessed to see if the parameter required can be obtained from correlations with these other data (Giolas, 1994). A wider scope of results are therefore given to the user from which to formulate a judgement. The system responds with a selection menu of correlations that are applicable. Each correlation may be executed in turn as required by the user. Results from the correlations can be compared with the directly measured results, providing a means of checking the validity of the measurements. In some cases the user might use the results from the correlations as well as, or even in preference to, the direct measurements in coming to a decision as to which value to choose for design.

As a final check, or in the event of there being no measurements which can be used, the parameter required can be assessed from the field description by accessing the Ground knowledge base. The parsed layer description can be assessed to obtain a broad range of typical values (Giolas, 1994). The more detailed the field description, the narrower and more precise will be the range of values.

8.3 Operation of Parameter Assessment Module

Geotechnical specialists need to know answers to specific questions before they can begin to make informed decisions and it is the role of Decision Support Systems to assist in this process. The parameter assessment module of SIGMA allows the engineer to select a specific location on a known site investigation and extract data on

a variety of parameters for design purposes and if necessary correlate other parameters. The work carried out for the correlation and parameter estimation modules have been undertaken by Giolas (1994) and as such are outside the scope of this thesis. Work within this section will concentrate on the development of the first stage of the parameter assessment module, but will discuss how the parameter assessment module as a whole will operate which will include the correlation interface.

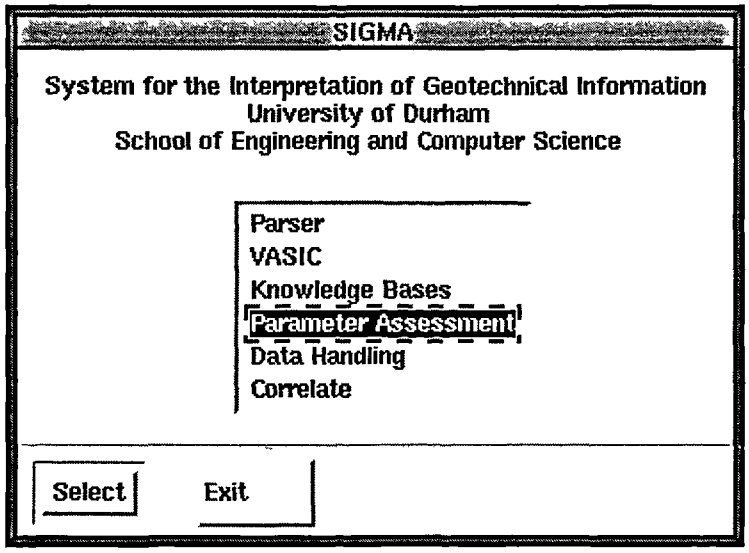


Figure 8.1 - Initial SIGMA screen

Figure 8.1 shows the initial screen of SIGMA, displaying the currently available modules. On choosing the parameter assessment option, the system automatically links through to the database and extracts all the data held in the project (investigation) table. This data extraction process is described in detail later in this section. These data are then displayed in a selection menu, Figure 8.2, with its identification number, start date and location. Several other data fields are available from the project table, e.g. project client, contractor, and any of these could also be shown on screen. Also, a data field can have a search condition attached, so as to reduce the amount of data to extract and search. The user selects a project by use of the mouse, and confirms this action by depressing the OK button.

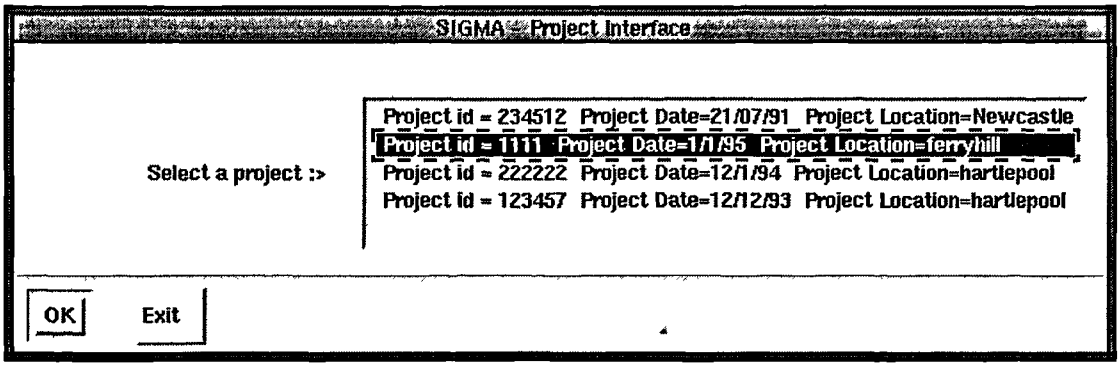


Figure 8.2 - Parameter Assessment project screen

The project identification number (proj_id) for the selected item is then used as the search criteria for the hole table. All the boreholes/trial pits for that specific project id are extracted from the data-table and formatted for display in another selection menu, Figure 8.3. As with the previous screen, the data shown (borehole identification number, grid reference and final hole depth) could be modified to suit the user.

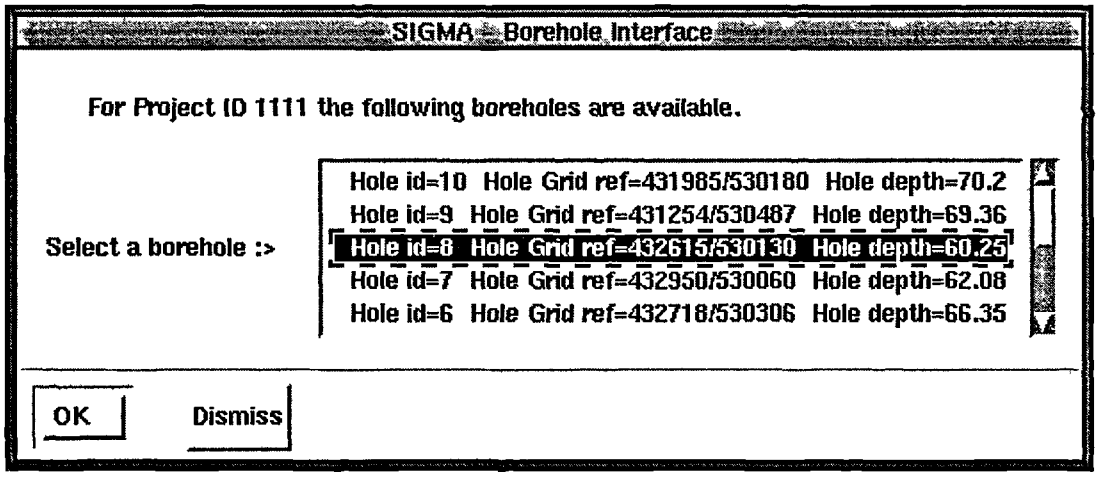


Figure 8.3 - Parameter Assessment Borehole screen

The user then selects a borehole to investigate, again with the use of the mouse, and the layer data for that particular borehole are extracted. Due to the structured format of the SIGMA database, data from several tables are combined to produce the output seen in Figure 8.4. The selection conditions that allow specific data to be extracted from several tables are manipulated into text strings and these text strings are then

passed to the appropriate **AdditionalWhereString** slots on the respective source object (Section 6.4.3). This methodology allows precise access to the data through a simple selection menu interface.

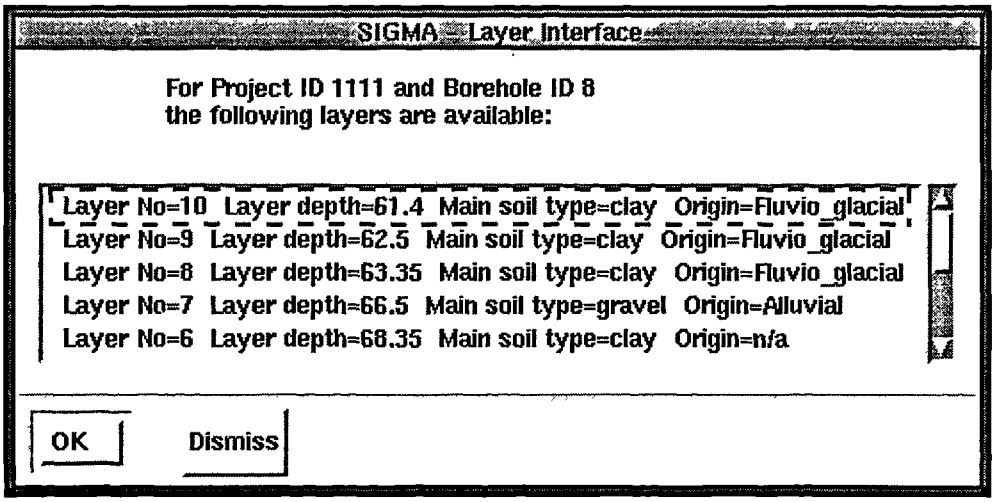


Figure 8.4 - Parameter Assessment layer screen

The user selects a layer, i.e. a depth, that is of interest and the system responds with a selection menu of the parameters available for assessment, Figure 8.5. On selecting a parameter the system identifies which tests have been carried out on the layer in question.

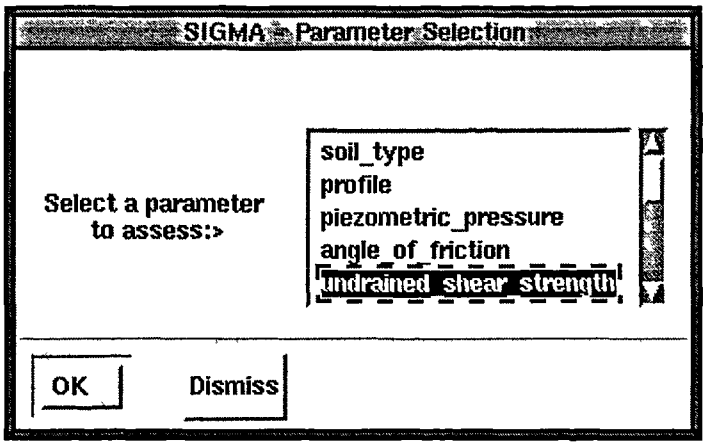


Figure 8.5 - Parameter Assessment parameter selection screen

This is accomplished by consulting the *lbrf* and *isrf* tables in conjunction with the Tests knowledge base. All the relevant data for the layer in question is imported into the Sig93D domain using the **AdditionalWhereString** methodology described in section 6.4.3. The test codes of all those tests that have been carried out on a particular layer are contained within the *lbrf* and *isrf* tables (section 6.3.2). These tables are consulted and the test data imported from the appropriate test tables. Now the Tests knowledge base must be consulted to ascertain the applicability and reliability for these tests in the soil type of the particular layer in question. For this purpose the main soil type is used - no account is taken of other constituent types.

As previously mentioned in section 5.3.2 each test (whether laboratory or in-situ) is identifiable by a test code. The same test code is used in the GeoTec database as in the Tests knowledge base, enabling the required tests to be easily located. Facets on the Reliability and Applicability slots of the appropriate test objects are then consulted to identify which are capable of measuring the required parameter and how applicable they are for the soil type in question. This is carried out by using a non-deterministic call to the facets in question and placing the successful results into the appropriate slot of the calculation object **SIG_K** in the parameter assessment domain. This calculation object provides similar functionality to the **Kalker** object in the parser domain.

Once this data has been established it can be reported to the user. This is achieved in the form of a list box on the Direct Measurement Dialog Box, with each row representing one test type, see Figure 8.6. Each row comprises the test code of the test in question followed by a brief statistical summary in the form of an average, maximum, minimum values and a sample size. The appropriate applicability and reliability data for each test is also shown.

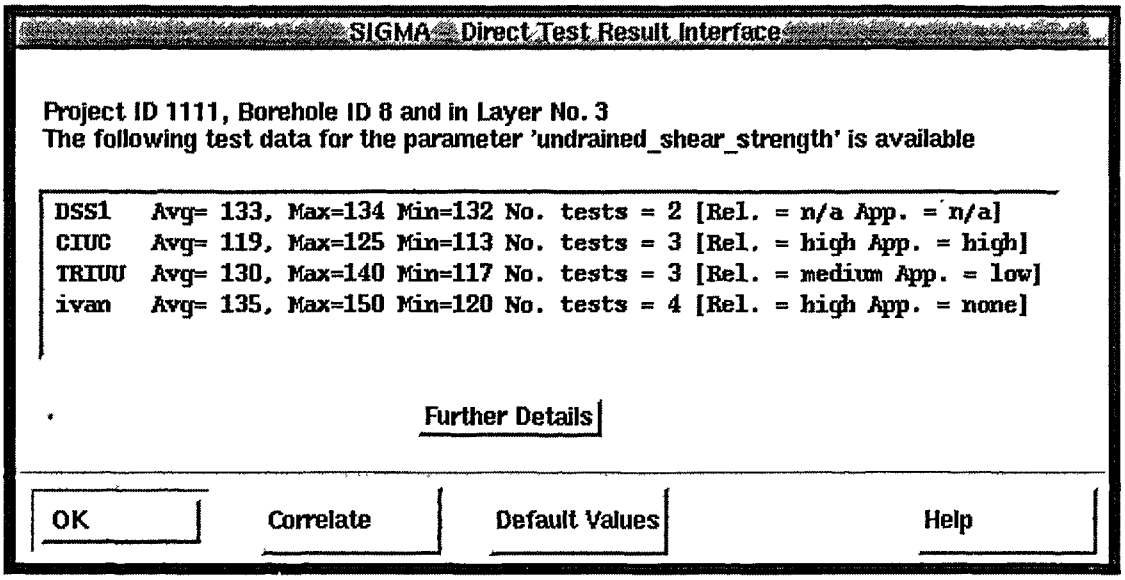


Figure 8.6 - Parameter Assessment direct measurements screen

If the user selects any row of the Direct Measurement Dialog Box and then clicks on the Further Details button, the individual data table records are displayed. This is accomplished using the database interfaces previously described in Section 7.5. This gives the user the ability to browse through the actual data to gain a fuller understanding of the overall picture.

When the directly measured data are retrieved from the database, any other measured data are also extracted. The Parameter Correlation knowledge base can then be accessed to see if the parameter required can be obtained from correlations with these other data. This gives the user a wider scope of results from which to formulate their judgement.

The relevant information is passed across to the correlation module, i.e. the selected parameter, the ground type and the other measured data. The system responds with a selection menu of correlations that are applicable. The user then selects those of interest and each correlation is displayed with its own individual interface, as shown in Figure 8.7.

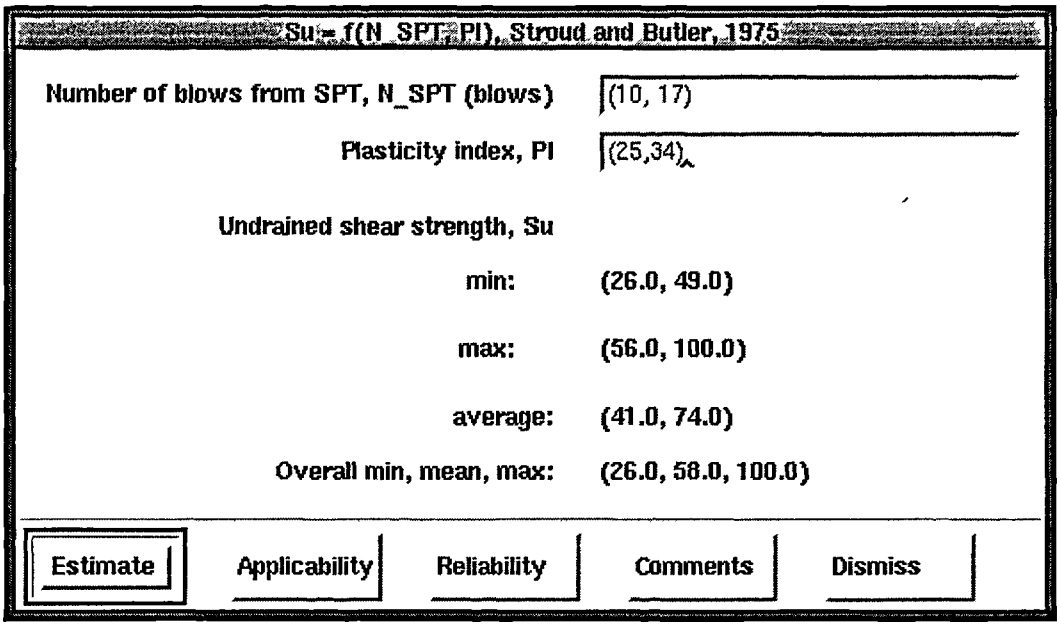


Figure 8.7 - Parameter Assessment correlation screen

The user may then execute each correlation in turn. The results from the correlations can be compared with the directly measured results, providing a means of checking the validity of the measurements. Each correlation has comments and reference material associated with it that can be viewed separately by the user, Figure 8.8. In some cases the user might use the results from the correlations as well as, or even in preference to, the direct measurements in coming to a decision as to which value to use for design.

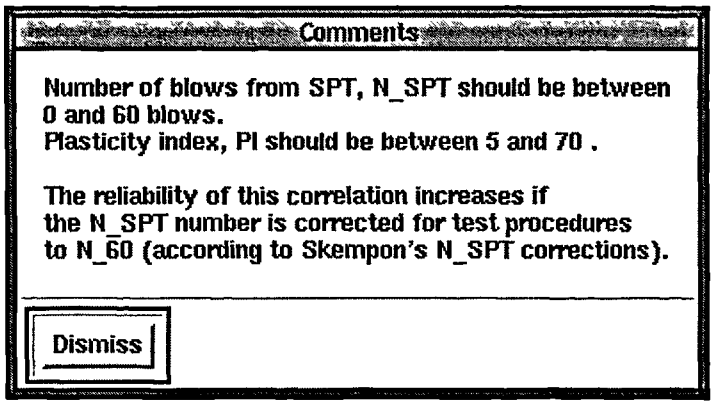


Figure 8.8 - Sample correlations additional reference material

As a final check, or in the event of there being no measurements which can be used, the parameter required can be assessed from the field description by using the Ground Knowledge base (Giolas, 1994). It is proposed that this will provide a broad range of typical values to act as a rough guide to the user. The more detailed the field description, the narrower and more precise will be the range of values. This work is still in the development stage and has not been fully implemented within SIGMA although an operational prototype is available

8.4 Borehole Interpolation

One of the primary aims of a site investigation is to identify the sub surface ground conditions throughout the site, based on observations at discrete borehole locations. This interpolation is usually carried out by geotechnical specialists who can apply experience gained from previous investigations, a knowledge of the site, published literature and fellow experts to arrive at hypothesises of the sub surface conditions. These hypothesises underlie the decision making in the ensuing design process.

The application of computer based techniques to this process has been attempted on many previous occasions and has proved a difficult task due to the inherent variability of geological and geotechnical systems. This variability makes it an area that presents an ideal situation for the application of knowledge based techniques, as stated in section 3.2. The task of computerising the process of interpolation is also particularly suited to the structure of SIGMA with the availability of a centralised data store and the modular object oriented functionality. Indeed, it has been the aim of the implementation of the borehole interpolation to discover if an object oriented approach to the problem can add to the already discovered knowledge of the domain.

It is undoubtedly true that the object oriented methodology with its dynamic hierarchies of objects, each object having variable storage capacity and behaviour combined with the principle of knowledge separation have been used to good effect in many engineering and research projects. Many real time applications, that is the monitoring and control of live or active systems as have large areas of the Artificial Intelligence community (Stefik and Bobrow1986), have benefited from its introduction, indeed some of PROKAPPA's main applications are in this area. The author has highlighted those areas where the new approach could improve the method and concentrated the implementation on these areas.

8.4.1 Previous Work

Carrying out this borehole interpolation, or parts thereof, with the use of a computer systems has been attempted on several previous occasions (Neidell, 1969; Rudman *et al*, 1973; Day *et al*, 1983, Rehak *et al*, 1985; Lok, 1987; Vaptismas, 1993; Ibrahim, 1994) with varying degrees of success. Two approaches have been developed, a cross-checking mechanism to identify patterns within the adjoining strata and rule based systems.

The approach of Rudman *et al* and, at a more sophisticated level, by Neidell has been to implement a cross-checking methodology to match similar patterns in adjoining strata. This methodology, designed for use in the oil industry, suffers from the assumption that all strata will participate in all boreholes and that the strata will be geometrically similar in separate logs. However the reality of the situation is that strata die out and appear and so simple pattern matching will break down. Neidell introduced an ambiguity function into the pattern matching process whereby a shrinking and stretching algorithm was applied to the strata in an effort to effect a suitable match, producing a more effective methodology.

However, cross-checking approaches to borehole correlations have proved essentially inconclusive, possibly due to their inherent simplicity. It is this simplicity, attached to experienced human perception, that makes cross-checking such a useful manual tool. Humans can match patterns with a degree of variability very successfully, allow for small variations in scaling, that is stretching or shrinking, and produce a meaningful result. In addition the geotechnical processes by which the strata have been formed is an important input into the interpolation process. If a non continuous time span is present in one of the boreholes then the cross checking method is not comparing like with like.

Norkin (1985) and Rehak *et al* (1985) implemented a rule based methodology to determine the correlation between borehole layers. General rules were applied to description data known from borehole samples to ascertain a similarity each with an assigned certainty percentage. This method works well theoretically on simple descriptions, however the complexity of real soil descriptions (see Section 3.2) may limit this approach in practise. The breadth and depth of the rule base, incorporated with the ability to update this rule base as required will also necessitate a powerful environment in which to operate.

The approach of Vaptismas with the Value Assignment and Similarity Calculation (VASIC) methodology, utilised borehole correlation based upon parsed soils description data. This data was subject to analysis over a range of parameters in an attempt to gain a more detailed comparison. Similarity Numbers are produced which allow the strata to be compared on a numerical basis (Toll *et al*, 1993). Automated processes were established that allowed complex strata with several soil types to be successfully compared. A fuller description of the VASIC methodology is given in the following section.

Ibrahim took the rule based idea put forward by Norkin and Rehak *et al* much further forward by implementing a set of rules based on several attributes of the particular strata. This level of detail allows for complex cross analysis to be achieved on the basis of multi-variate correlations. Each attribute is primarily assigned a qualitative descriptor from the rule base and then all the descriptions available are converted into a numerical value, in a similar manner to that used by Vaptismas. These numerical values, or Numerical_Index (NI) are then weighted and bounded to produce a ranking of possible solutions. The system lacks the ability to deal with complex strata descriptions, for example **silty CLAY** is deemed to be dissimilar to **clayey SILT**. In addition the data is stored and manipulated in such a manner as to be only accessible from the system itself. Implemented in Leanardo, a PC based expert system environment, the system only has the ability to link to its own flat file structures.

The decision was taken to implement the VASIC method for correlation of borehole strata within the SIGMA environment as it was seen as the only method which can handle the complexities of real soil descriptions. The data required was available in the correct structure from the GeoTec database (the output from the parser). An object oriented implementation should offer scope for enhancement not only of VASIC but of the computational approach to the interpolation problem. What follows is an implementation of aspects of Vaptismas' work and as such the methodology behind it has already been covered in great depth (Vaptismas, 1993). The work presented in the thesis concentrates on those areas where the implementation differs from the original and where the object oriented approach has been shown to provide an improvement to the previous methodology.

8.4.2 VASIC Methodology for Comparing Borehole Layers

The fundamental principle behind the methodology is that a nominal particle size distribution can be generated by processing the soil types (constituents and their

associated amounts) stored in the GeoTec database. In the original methodology the overall correlation takes into account not only the soil type but also the consistency, structure and colour. The module that has been implemented within SIGMA currently relies solely on the soil type attribute, this being the most important aspect within the soil description (Vaptismas, 1993) in addition to being the most difficult facet to compute. The additional attributes and associated weighting factors could be added at a later stage.

Once a stratum has been broken down to the soil type level, the strata can be represented as a combination of Main, Major, Secondary and Minor constituent amounts. These combinations can be converted into the relative percentages of each constituent type present. Vaptismas defined percent lists, different for fine and coarse grained soils for combining the constituent percentages (Appendix 2). These percent lists were constructed using the British Soil Classification System (BS 5930, 1981) which presents the data in the form of ranges for a particular descriptive modifier.

In order to construct a notional particle size distribution it was necessary to represent the percentage passing by a single value instead of a range of values. The values could not be uniquely defined for a given descriptive term but depended on the number (and amounts) of other soil types given in the description. Therefore a matrix of percentage values was defined for different combinations of amounts present in the description. This is illustrated in Figure 8.9.

Soil description: *Slightly silty, slightly gravelly sandy CLAY.*		
Main constituent type:	Clay	[1]
Major constituent type:	N/A	[0]
Secondary constituent type:	Sand	[1]
Minor constituent type:	Silt, Gravel	[2]

Figure 8.9 - Example of constituent combination in layer description

Therefore the description has one Main constituent, no Major constituent, one Secondary constituent and two Minor constituents. Vaptismas expressed this as an amount list, presented in the order [Main, Major, Secondary, Minor]. Therefore the Amount-list for the above example would be [1,0,1,2] and since the soil is fine grained (main constituent type: CLAY), the appropriate Percent-list would be [40, 0, 30, 15] (see Appendix 2).

The particle size distribution so generated can then be compared numerically with another distribution to give a Similarity Number. The comparison between the two distributions was made by observing the difference in percentage at a number of particle sizes. The similarity is given as 100 minus the average absolute difference. For n points on the particle size distribution

$$\text{Similarity Number} = 100 - \frac{1}{n} \sum_{1}^{n} |\text{Percentage Difference}|$$

This number is calculated using n=6, the six points representing the limits between the six different inorganic soil types (particle diameters of 0.002, 0.06, 2, 60, 200 and >200 mm). The Similarity Number has a value between 0 and 100, a higher number implying increased similarity.

An example of a comparison together with the calculated Similarity Numbers are given in Table 8.1. In the example a "very silty clayey SAND", is compared to a "silty SAND", indicating the following percentage differences at the six points identified above:

Description	Particle Diameter (mm)	0.002	0.06	2	60	200	> 200
very silty clayey SAND	**Percent Passing of Soil 1**	10	35	100	100	100	100
silty SAND	**Percent Passing of Soil 2**	0	10	100	100	100	100
	Percentage Difference	10	25	0	0	0	0

The Similarity Number is : 100 - 1/6 (10 + 25+0+0+0+0), so Similarity Number = 94.

Table 8.1 - Similarity Number Calculation

The notional particle size distribution for the two soils being compared is shown in Figure 8.10

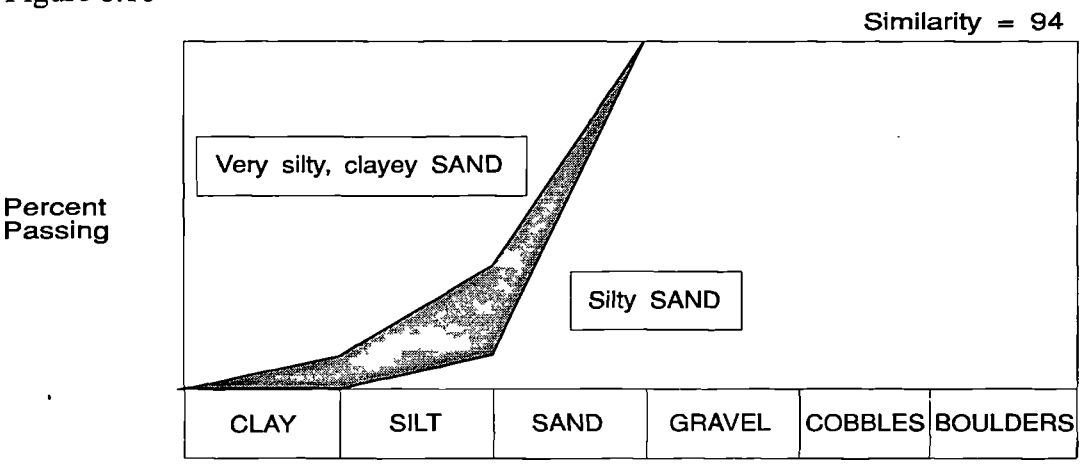


Figure 8.10 Examples of comparisons between soils in terms of soil type.

Vaptismas extended this methodology by introducing the concept of an Area Identifier Number, AIN, which is a numerical representation for a particular layer, as opposed to the Similarity Number which represents a comparison between two layers. Vaptismas widened the overall VASIC method to cover the interpretation of a site by adopting a seven point approach.

- Identification of Possible Marker layers
- Configuration of Triangles
- Connection between Pairs of Marker layers

- Assessment of Planar Marker Beds
- Calculation of Dip Angle and Dip Orientation
- Continuity of Planar Marker Beds Across the Site
- Borehole-to-Borehole Correlation

The site is configured into triangles in order to assist processing, thereby dividing the site into manageable portions, the nodes of the triangles being the borehole locations. The algorithm to produce this triangulation was proposed as an automatic mesh generation scheme (Frederick *et al*, 1970; Lindholm, 1983). Work has been carried out at the University of Durham (Wade, 1994) to automate this procedure for any given site.

The site wide implementation is shown in Figure 8.11. The process first identifies a site-wide model of the ground conditions using marker layers; these are layers which 'stand out' from the general ground conditions (in terms of either soil type, colour or consistency) and can therefore be more easily traced across the site.

For instance, if a layer of gravel is present among what are otherwise clayey layers, this would be highlighted. This is achieved by comparing layers within the borehole using the Similarity Number. Trigger levels of Similarity Number are specified for identifying significantly different layers, so as to highlight them as potential marker layers.

The continuity of marker layers is examined at a number of levels. Again, the methodology of Toll *et al* (1993) is used in which a Similarity Number is calculated from a comparison of the qualitative terms used to describe two layers in adjacent boreholes. Links are established between the most similar marker layers, providing a threshold value of similarity number is achieved.

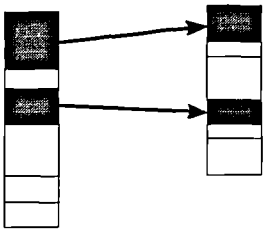
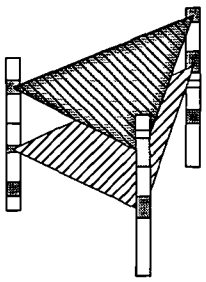
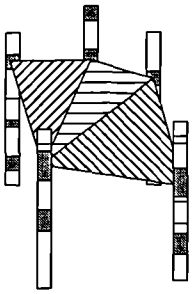
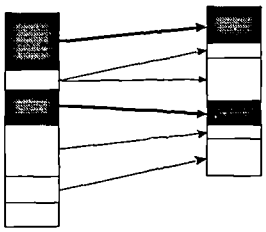
1) **SITE WIDE** Identify Marker layers	
2) Construct Links between adjacent boreholes	
3) Construct Planar Marker Beds from Links	
4) Construct Trends from Planar Marker Beds	
5) **BOREHOLE TO BOREHOLE** Construct Hypotheses	

Figure 8.11 - Illustration of site-wide borehole interpolation (Vaptismas and Toll, 1993)

Initially this is done between borehole pairs and then within groups of three boreholes (triangles). Layers which are found to be continuous within these groups (triangles) are then checked for compatibility with adjacent triangles. Where continuity exists between adjacent triangles, site-wide 'trends' are established. Each trend represents a hypothesis which could explain the ground conditions observed. The trends are presented to the user for a final decision as to which will be selected as the basis of the model of the ground conditions to be generated.

8.5 Object Oriented Implementation of VASIC Methodology

The initial implementation of the borehole correlation method was in PROLOG, a General Purpose Representational Languages (GPRL). Two of PROLOG's main advantages over conventional programming languages are its ability to create and apply simple logic within the structure of the program and its ability to quickly and efficiently manipulate lists. It was this list manipulation that enabled the prototyping to be carried out for the borehole correlation at a very early stage. However, the end result was a complex standalone PROLOG program that was manipulating lists of up to 20 to 30 elements. However for site wide comparisons this figure could increase markedly. Therefore it was decided to include the borehole interpolation routines within the SIGMA environment, thus providing integration of another tool to aid the geotechnical specialist.

Using the ability of PROKAPPA's model engine it was thought that the processing problem could be approached from a different aspect whilst retaining the original

methodology. The seven point approach used by Vaptismas has been reduced within SIGMA, the site wide interpolation process being reduced to a proposed five steps, namely:

- Configuration of Triangles
- Identification of Marker layers
- Assessment of Planar Marker Beds
- Continuity of Planar Marker Beds Across the Site
- Borehole-to-Borehole Correlation

This reduction in phases of the method could be achieved due to the structure of the new system. As marker layers are being identified within one borehole they are also compared with the other boreholes participating within the triangle, allowing for quick identification of planar marker beds. These triangular planar beds are then compared within adjoining triangles to identify if they belong to the same site wide trend.

For the implementation of the Vaptismas method, the hierarchical structure has proven very effective. Using a HOLE hierarchy each participating borehole and layer is represented in a dynamic model that is constructed for each run of the system. The triangulation methodology that was utilised by Vaptismas has also been transposed to operate in the object oriented environment, and has incorporated the links and bedding planes subsystems as well.

8.5.1. Object Hierarchies within the VASIC Method

As with many of the SIGMA modules, the heart of the borehole interpolation module are the dynamic object hierarchies that are utilised during processing. In procedural

languages, active memory is utilised and managed throughout the execution of a program, either manually or automatically, through the medium of variables. These variables can be local, public or global and take many forms, for example single value, arrays or pointers. Within an object oriented environment this active memory management is carried out via the objects, subclasses and instances that constitute the object hierarchy, in conjunction with the environment's own memory subsystems. This allows data to be passed down the hierarchy in the form of inheritance, so providing a continuity throughout the dynamic model. The flexibility this offers to the system's developer is the ability to view the system's exact potential and value at any stage in the proceedings. The flexibility it offers the final system is the ability to model a given dynamic situation or domain, provide any inference or maintenance as required and then apply the model to a series of problems.

The VASIC module utilises three discrete object hierarchies, namely HOLE, TRIANGLE and LAYER. The HOLE model is the representation of the boreholes and their respective layers within the area being considered for VASIC processing, the detail being imported from the GeoTec database via the Sig93D domain. The LAYER model is the structure that is used to calculate the AIN for marker layers and Similarity Number in borehole to borehole comparisons. The TRIANGLE model holds details of the triangles into which the site must be subdivided in order for the site wide comparisons to be carried out.

8.5.2 The HOLE hierarchy

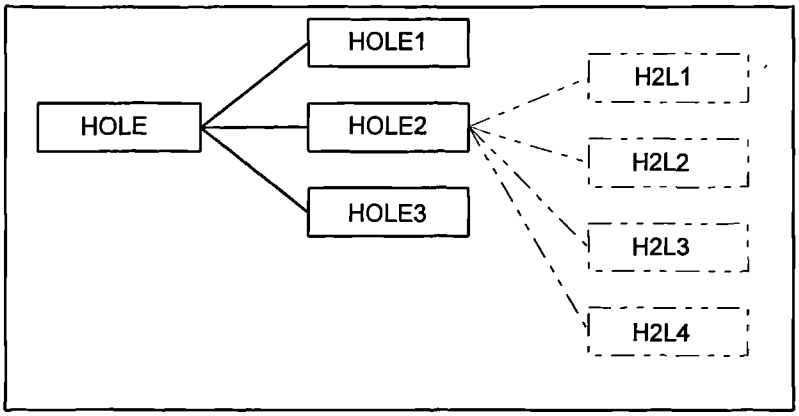


Figure 8.12 HOLE object hierarchy

The HOLE model in its initial state consists of just a single top class object, HOLE. After the VASIC interface has imported the requested data into the Sig93D domain, the preliminary VASIC routines identify the number of boreholes and creates a subclass of hole for each. The imported data is interrogated to identify the number of layers within each of the boreholes. One instance per layer is then created as an instance of each of the hole subclasses and named accordingly, so arriving at the structure shown in Figure 8.12. As can be seen, the final structure is that of one instance per layer of each participating borehole. An example slot table for one of the layer instances is shown in Table 8.2, detailing the information capable of being stored in each instance.

Slot Name	Description	Typical Value
AIN	Area Identifier Number	234
dtop	Depth to top of layer (m)	76.43
hole_no	Borehole number (id)	1
marker_lay	Used in the identification of marker layers	"TRUE"
nate	Natural Easting	431162
natn	Natural Northing	530687
constituents	Pointers to instances that make up the layer	sldt("1111","1",1,1,1), sldt("1111","1",1,1,2)
thck	Layer thickness (m)	3.6

Table 8.2 Slot Table for layer instance with typical values

There are several points to note on the contents of the slot table, namely:

AIN - The Area Identifier Number, which is calculated independently as a precursor to the identification of bedding planes.

marker_lay - Used as a marker to identify those layers that appear to participate in a marker layer.

constituents - Contained in this multi-valued slot are the names of those instances within the Sig93D domain that participate in the particular layer. This allows the data stored in these instances to be available to the interpolation routines without actually being present within the VASIC domain. The nomenclature used is the UID methodology previously discussed in sections 6.4.1, 7.4.6 and illustrated in Table 8.3.

sldt("1111","1",1,1,1)	sldt("1111","1",1,1,2)
project_id = "1111"	project_id = "1111"
hole_id="1"	hole_id="1"
layer_no=1	layer_no=1
stratum_no=1	stratum_no=1
constituent_no=1	constituent_no=2

Table 8.3 - Illustration of UID nomenclature for sldt objects

In effect this method links across PROKAPPA applications and creates a linked object hierarchy without having a duplication of the data records themselves. Some duplication of the actual data is required, but only to the extent that the same data may be present within several domains simultaneously.

8.5.3 The Triangle Object Hierarchy

The VASIC method utilises the sub-division of the site into triangles in order to process the borehole data. Each triangle has three participating boreholes and an associated quality index, q. This is a measure of the geometry of the triangle, where for an equilateral triangle q=1 (i.e. good quality), whereas a triangle which tends to a straight line has q=0 (i.e. poor quality). Correlation of boreholes inside triangles of

poor geometry may cause misinterpretation of the ground conditions, so q is used as a guide as to which triangles are appropriate for use.

The object hierarchy for processing the triangles consists of a top level class TRIANGLE which has subclass objects for each triangle in the site being investigated. The slots contained on the triangle subclass objects are listed in Table 8.4.

Slot Name	Description	Typical Values
BH	Contains the numbers of the participating boreholes	3, 4, 8
Links	Contains the individual links	H3L1-H4L1, H4L1-H8L1, ...
mar_layer	Contains pointers to the HOLE instances that make up the bedding layer	(H3L1, H4L1, H8L1), ...
q	Quality Index	0.807

Table 8.4 - Slot table for TRIANGLE subclass object

The *Links* multi-value slot contains the names of the instances that have been identified by the comparison routines as being similar. These links are then analysed to ascertain if they constitute a marker layer, that is that three links together form a discrete triangle of layers within the main triangle itself. Once this has been established these marker layers are stored in the multi-value slot *mar_layer*.

As possible bedding layers are identified they are added to the multi-valued *mar_layer* slot on the appropriate triangle subclass. Using a sorting routine to ensure a similar order, the planes are added utilising the ProTalk += construct, so avoiding duplication of existing bedding planes. The += construct adds a value to a multi-value slot or facet, however if the value is already present, no addition is made.

```
{
8.    bound inputs;

9.  /*** Initialise program variables
10.   ?main = 0;   ?sec = 0;   ?maj = 0;   ?min = 0;   ?accum = 0;   ?total = 0;

11. /*** Deleting objects ready for processing
12.   clearing();

13. /*** Main loop for forming the reference string
14.   for ?inst inlist ?inst_list;                          /*** Backtracks through all the
15.   do {                                                  /*** participants in the layer
16.      select      {
17.       case:?inst.sldt_amnt == "main";
18.         {?main = ?main+1;                    /*** Increment soil amount counter
19.         main.soil_type += ?inst.sldt_type;}/*** Add soil type to SOIL subclass
          case:?inst.sldt_amnt == "major";
20.         {?maj = ?maj+1;
21.         maj.soil_type += ?inst.sldt_type;}
22.       case:?inst.sldt_amnt == "secondary";
23.         {?sec = ?sec+1;
24.         sec.soil_type += ?inst.sldt_type;}
25.       case:?inst.sldt_amnt == "minor";
26.         {?min = ?min+1;
27.         min.soil_type += ?inst.sldt_type;}
28.     }
29.   }

30. /*** Classify the main constituent as either fine or coarse grained
31.   ?class = classify();

32. /*** Construct reference string for percent look up
33.   ?ref_string = AppendStrings("C",
34.                       ConvertToString(?main),
35.                       ConvertToString(?maj),
36.                       ConvertToString(?sec),
37.                       ConvertToString(?min));


38. /*** Looks up percent list
39.   ?perc_list = GetFacetValues(Ref, ?class, ?ref_string);

40. /*** Allocate percent passing
41.   for ?x from 0 to 3;
42.   do {
43.      ?soil_cat = ConvertToSymbol(ListNth(`(main, maj, sec, min), ?x));
44.      ?soil_cat.perc_pass = ListNth(?perc_list, ?x);
45.   }
```

```
46. /*** Set CONS percentages for individual constituent AIN
47.   for ?inst inlist all subclassof Layer;
48.   do {
49.     find1 ?inst.soil_type == ?;
50.     for ?ind_soil inlist GetValues(?inst, soil_type);
51.     do  CONS.?ind_soil = ?inst.perc_pass;
52.   }

53. /*** Calculate and allocate AIN
54.   for ?soil_cat inlist `(clay, silt, sand, gravel);
55.   do  {
56.     ?accum = CONS.?soil_cat + ?accum;
57.     ?total = ?total + ?accum;
58.     CONS.?soil_cat = 0;
59.   }
60.  ?lay_name.AIN = ?total
61.}
```

The function ain() is called by a function that is cycling through the participating boreholes. When a borehole has been identified, its layer instances are then ascertained and each layer is passed to ain(), bound to the variable ?lay_name. Each layer instance holds the names of the constituents that make up that layer (see Table 8.3) and this detail is also passed to the function, bound to the variable ?inst_list. After setting various temporary variables and counters to zero and returning the object base to its initial state, control passes to the main reference loop.

Within the reference loop each soil constituent instance has its *soil_amnt* slot value (which contains the amount - Main, Major, Secondary, Minor) examined individually using a select/case statement (lines 16 - 28), which has the syntax of a multiple if statement. Four variables are used as counters for the four soil amounts and the control of the function passes to the next statement group when all the instances have been examined. The variable *?class* is then bound to either the value Fine or Coarse, depending on the classification of the soil. This classification is carried out by the function classify() (line 31) which uses the British Standard Soil Classification to determine the generic group to which the main constituent type or types belong.

The soil amount counters are then concatenated into a reference string to produce a five digit code *?ref_string*. The percent passing data (the percent-lists described earlier) is stored as facet values on two slots, Coarse and Fine, of the reference object, Ref as shown in Figure 8.13: which slot is chosen depends upon the value of the *?class* variable. *?ref_string* is then compared with the facet names on the appropriate slot and on a suitable match the variable *?perc_list* is bound to the appropriate percentage passing values.

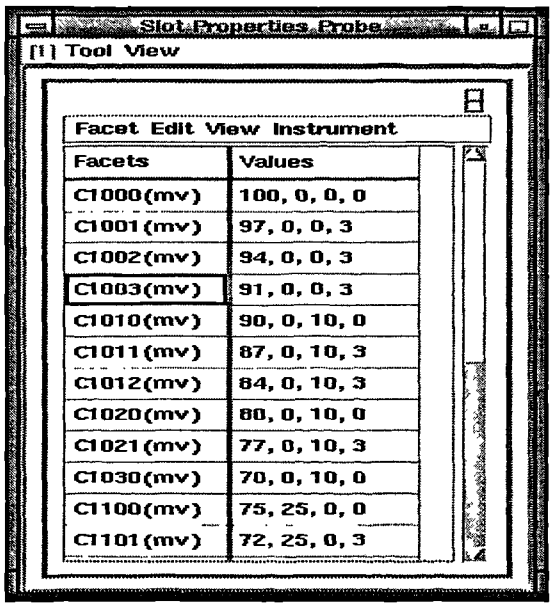


Figure 8.13 - Facet storage of Coarse grained percent list data

Due to the difference in the nature of the addition of the AIN variable, constituent type and the calculation of the percentage passing list, soil amount, the calculation of the actual AIN becomes a two stage process. Initially ?perc_list is broken down and allocated to the subclasses of LAYER (line 47), for example Main.perc_pass = 65%. These percentages are then collected into the six main soil groups for addition in the object CONS (line 54). On completion of this addition the AIN value is set as a slot value on the ?lay_name instance (line 60).

A similar structure to ain() is employed for the calculation of the Similarity Number, however there are two objects involved due to two layers being required to calculate the Similarity Number. The body of the function is the same, the main difference being that the Similarity Number is calculated by subtraction rather than the addition of relative percentages.

8.7 Operation of the Borehole Interpolation Module

To access the VASIC system, the VASIC option is chosen from the main menu of SIGMA, which brings up the initial dialog box, Figure 8.14. This screen allows the user to choose the data that is to be imported into SIG93D domain and this choice may be effected in three different ways.

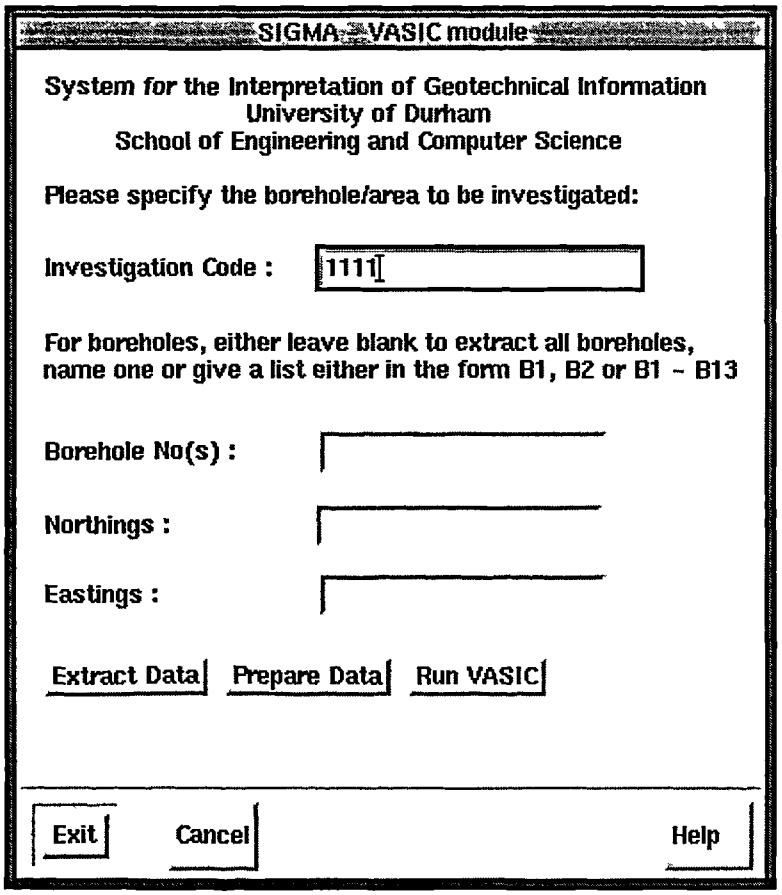


Figure 8.14 - Initial VASIC Dialog Box

Firstly, the user can input only the investigation code which will then cause all the borehole data for that specific investigation to be imported. Secondly the user may select specific boreholes to be imported, either by name in the form of a list or by stipulating a range. The range input will only be effective in those situations where the boreholes are sequentially numbered. This option may be used in conjunction with specifying the investigation code, or independently depending on the user's requirements. Thirdly the user may specify Eastings and Northings for the selection of the data. This is done by specifying an upper and lower bound for both Eastings and Northings separated by commas, so as to describe four vertices of a rectangle as shown in Figure 8.15. Again, this option can be used in conjunction with the investigation code if required.

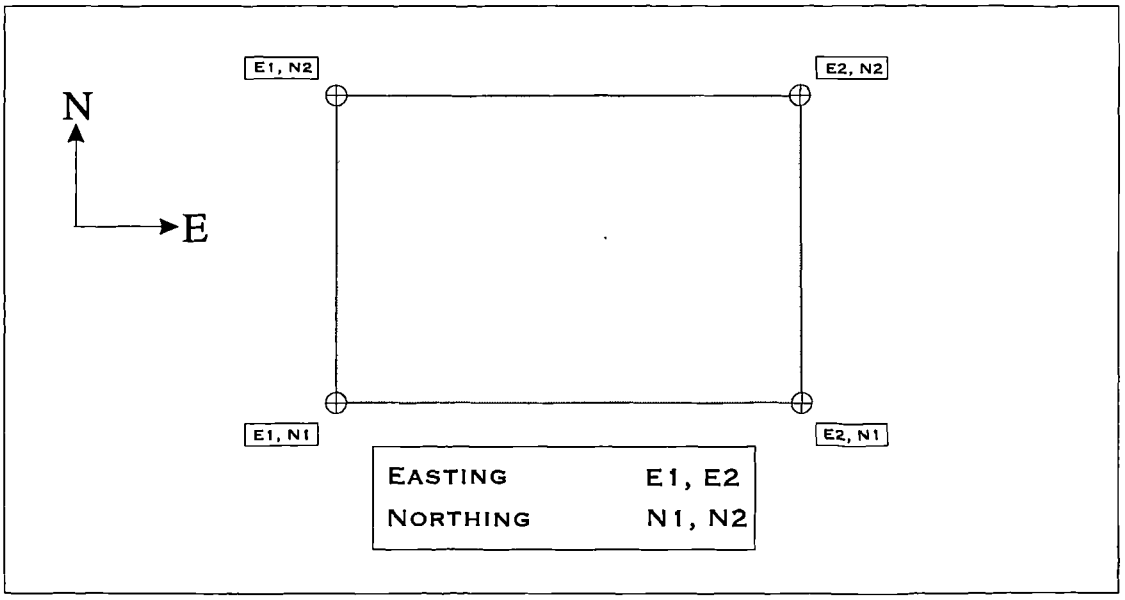


Figure 8.15 VASIC data import using Eastings and Northings

On completing any of the procedures described above the user presses the extract button and the system will formulate the correct SQL from the data supplied and import the data into the SIG93D domain, reporting as it does so the number of boreholes imported.

In order to carry out the VASIC process the working area has to be prepared, that is the HOLE hierarchy constructed and the LAYER, CONS, CONS1, CONS2 and TRIANGLE hierarchies made ready. This process, initiated by the pressing of the Prepare Data button, is a lengthy process and for this reason is separated as a distinct processing phase. Once carried out this process does not have to be repeated over the entire VASIC session.

On completion of data preparation the user may initiate the VASIC routines by pressing the appropriately marked button. The system will then attempt to identify marker layers by following the methodology described in section 8.4.2. Once this has been completed the statistics are reported back to the user. These are of the form of the mean and standard deviation of AIN for all of the boreholes that have been imported, Figure 8.16.

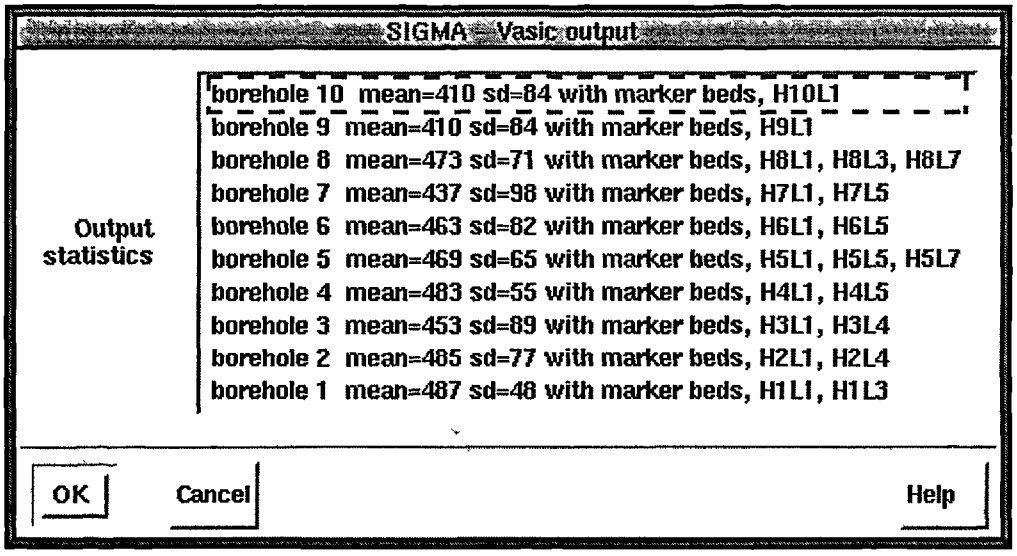


Figure 8.16 Output statistics from VASIC marker bed routines

As discussed in section 8.5, as marker layers within one borehole are identified they are immediately assessed within the participating triangle to see if the layer is present within all the boreholes in the triangle. If it is found to be present in all layers, the three layers that form this marker bed are stored separately in the form (H1L1, H2L2,

H3L2), a simple string concatenation of the relevant borehole and layer. This enables stage 3 in Figure 8.11 of the site wide methodology to be arrived at early in the processing stage. The next stage is the linking of the marker beds contained within the triangles to assess their continuity over the site as a whole. This process has not currently been implemented within SIGMA

On completion of the identification of the marker layers across the site a TREND interface is then displayed to the user on which are displayed any hypotheses that have been generated. These hypothesis, or trends, are generated by matching marker beds across triangles and are rated in order based on how many triangles participate within a given hypothesis.

The TREND interface initially contains a trend list which displays those trends that have been identified. On selection of a particular trend, all the triangles participating and their associated marker layers are displayed in a list box. The user may then select any triangle and those boreholes that make up the triangle are displayed in the form of push buttons. The user may select which boreholes to correlate by depressing the appropriate buttons and the system then carries out the calculation of Similarity Numbers for these boreholes. The correlation takes into account the boundaries of the marker layers, that is no correlation will cross over a planar marker bed. The correlations are based on a similarity thresh-hold, that is only those correlations surpassing this threshold will be reported in the appropriate list box. This similarity thresh-hold can be set by the user, allowing full control of the decision making process. Once discovered, the links are reported as concatenated strings of the form borehole followed by layer, as in the marker layer. An interface showing a completed session is shown in Figure 8.17.

This trend output informs the user of the subsurface borehole to borehole correlations in conjunction with planar marker beds that have been identified across the site. This

data could also be displayed graphically, either in the form of fence diagrams or 2D planar sections across the site. This would enhance the understanding of the data whilst displaying the data to the geotechnical specialist in a format with which they were familiar. To accommodate this graphical output, an interpolated model of the site would need to be produced and stored - work is currently ongoing at the University of Durham in this area (Chen, 1994).

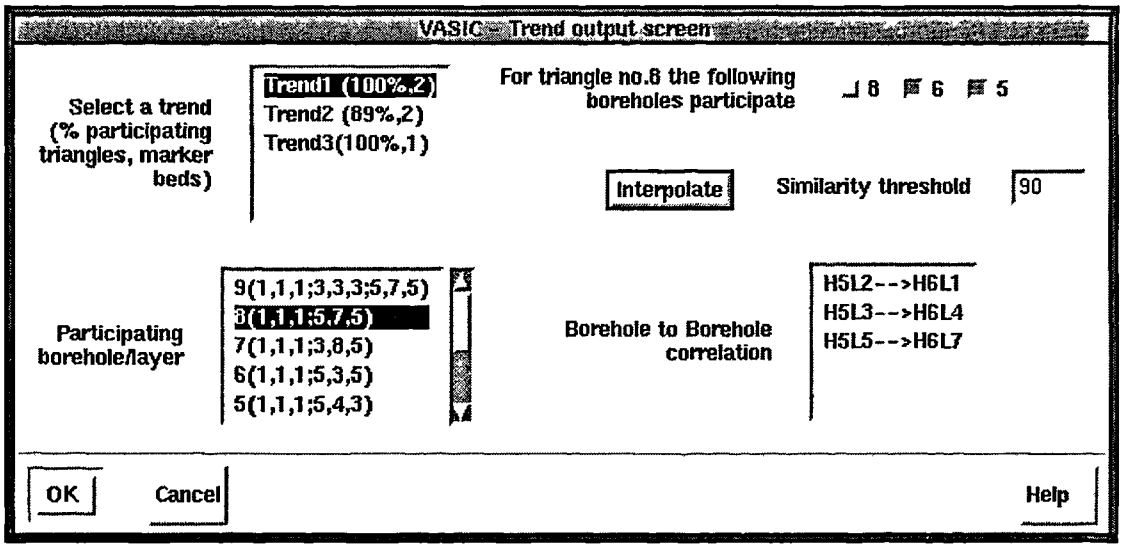


Figure 8.17 Trend output from VASIC session

8.8 Conclusions

One practical aspect of having a knowledge based system with a site investigation database as its core is that it allows the geotechnical specialist rapid access to this data in differing forms. With the inclusion of the test data, this access allows interrogation of the data in order to ascertain design criteria for a project. With the parameter assessment module the user can not only access directly measured data for the parameter being sought, but can also access all the other data measured at this location. Correlations can then be applied to this data in order to obtain the required

parameter. This therefore gives as wide a range of values as is possible and the user can make use of all this information in deciding in a value to use in design.

The ability to compare, contrast and utilise this data in conjunction with those facts stored in knowledge bases gives the user of SIGMA access to a large breadth of knowledge. As the knowledge of a particular domain increases, the generic nature of the structure of both the database and the knowledge bases allows for organic growth of the system.

Tools such as the borehole interpolation module allow the user to manipulate the data stored and carry out preliminary borehole and site wide interpretation to gain a feel for the nature of the sub surface conditions. The ability to run through these routines allows the user to make more informed decisions about potential hypothesises which could explain the ground conditions.

The implementation of aspects of the VASIC methodology for correlation of borehole layers in the object oriented ProKappa environment has proved a significant contribution. The dynamics of the HOLE hierarchy provides an excellent framework for the interpretation of ground conditions, a model that can be seen to be truely generic. Whilst each site being investigated is unique, the model can be applied to assist the geotechnical specialist in arriving at their understanding of the sub-surface ground conditions. Furthermore, the linking to the GeoTec database allows this data model to be as current as possible and due to the similarity in structure, the data transfer and incorporation into the VASIC domain is smooth and efficient.

To fully appreciate the adoption of the VASIC methodology a graphical output is required, to visualise the effect of the correlations. Work is currently being carried out at the University of Durham to this end (Chen, 1994).

Chapter 9

Discussion and Conclusions

9.1 Discussions

9.1.1 Role of knowledge based systems

The development of SIGMA has involved bringing together database and knowledge base technology. The application of KBS in geotechnical engineering is still in its infancy and SIGMA represents a significant contribution to this area.

Knowledge Base Systems (KBS) technology, as a component of the Artificial Intelligence field, has been viewed by the public and industry alike with a degree of scepticism. Terms such as 'expert system' and overstated claims for the advantages and capabilities of such systems have certainly not helped their acceptance. The reality of the situation is that KBS can provide an environment in which it is significantly simpler to produce computing systems that have access to domain specific knowledge.

Much of KBS technology has now developed to the stage where it can be usefully used in geotechnical engineering. Knowledge can be stored in structured knowledge bases, allowing access with simple inference programs or in the form of rules which, dependant upon a given set of data, can control the flow of a program. Symbolic processing, the ability to treat data, functions or concepts in the same manner, is a software methodology that has been in existence for many years. Object oriented methodologies are now well established in both the academic and commercial arenas.

KBS can be viewed as the bringing these different facets of computing technology in a structured manner within the framework of a suitable environment. They offer the ability to produce systems that can access large quantities of data and knowledge, communicating these results to an end user. As we enter an era where the quantity and quality of information increases daily, systems to manage and analyse this information will become of increasing importance. It is the author's view that KBS technology can assist in that process, possibly with advantages over other conventional procedural computing systems.

Geotechnical engineering is an industry where the potential for integrated information systems is great but as yet their implementation has been only on a piecemeal basis. Whilst this situation exists there is a lack of continuity in the information flow through a particular project, company or organisation. This makes the task of managing the information more cumbersome and possibly inefficient. Any tool that can assist in the managing of this information is of a positive benefit and systems such as SIGMA can contribute in two specific areas.

Firstly the GeoTec database provides a conduit for the information flow. Based on the AGS Data Transfer Standard, information from a ground investigation may be entered directly, including all the test data. Many geotechnical companies have data logging systems monitoring the laboratory tests and it would be feasible to have this data transferred directly into GeoTec. Borehole reports could be produced, management could monitor the progress of a particular investigation, engineers could utilise the data for design calculations - all using the same database facility. This continuity not only of data storage but also data flow enhances both effective data management and data integrity.

Secondly, the combination of SIGMA's knowledge bases, database and analysis modules can act as a Decision Support System, assisting the geotechnical specialist

and providing an information management resource. Tools such as SIGMA offer an addition to the often limited skilled resource within an organisation.

9.1.2 Comments on development software

As stated in Chapter 4, the decision was made very early in the project as to the type of hardware to be purchased and the software environment to be chosen for the development of the system. A brief discussion of both of these choices is included as a guide to others embarking upon similar projects.

The Sun Sparc2 has proved to be a very reliable workstation. The system as purchased has required no maintenance at all over the period of the project. The additional disk capacity that was purchased did need replacement after 14 months due to hardware failure. The replacement disk is still functioning satisfactorily. Software for the Sun never presented any problems, the only point of note being the generally high price of packages.

Whilst the PROKAPPA environment has been used to produce a working prototype of SIGMA and demonstrated the appropriate methodologies, there are several points to note with regard to its use in a research environment.

The PROKAPPA Data Access System, DAS, is a generic post sales module supplied by IntelliCorp, the US manufacturers, capable of supporting many database types. As a consequence it has been developed on a general rather than a specific scale. In hindsight it might have been preferable for IntelliCorp to have produced a more database specific product as the DAS did not prove to be as robust as other PROKAPPA systems. For example to produce a data link the developer must map each table individually and specify the data tables keys. If any later modification is then made to the data table outside of the PROKAPPA environment, that is via INGRES, the map within the DAS has to be modified accordingly. This proved to be

a very difficult process as PROKAPPA seemed to maintain a link to the original map rather than any amended version. A way for circumventing this was developed but the situation remained far from satisfactory.

Additionally, the DAS suffered from a compiler error, manifesting itself in the inability to make complied versions of any ProTalk file containing Data Access references. This error is recognised by IntelliCorp and has been the subject of corrective measures on their behalf. In practice this leads to slower performance times for any ProTalk code that requires database access.

The problem of object congestion also affected the performance of the system. This is particularly noticeable in applications that contained many objects, such as the interface control area, database records and their associated mapping data and the VASIC module. The manner in which the interface tools are handled (that is a dialog box, itself an object, contains several other dialog box controls, each one of which is an object) leads to the creation of hundreds of control instances. Database access requires many class objects plus the creation of runtime instances (Chapter 6) and the VASIC module builds a dynamic map of the boreholes to be analysed. Whilst the model engine of PROKAPPA is excellent, large numbers of objects in the development environment significantly reduced both system performance and stability. This resulted in more memory being fitted in the Sun, expanding the RAM up to 32 megabytes, significantly improving the systems response. Whilst the model engine of the ProKappa system is one of its strongest points, the can become a significant drain on system resources if used extensively. The Sun is capable of RAM expansion up to 64 megabytes and only on reaching this limit would limitations of the system be reached.

With regard to stability, SIGMA was developed using version 2.1 of the PROKAPPA system which since it is not the latest version would suggest that the environment

should have been out of the period where software crashes are commonplace. However, PROKAPPA was found to be occasionally unstable, more noticeable after long periods of use. This would manifest itself by the system reporting internal errors which would lead to automatic shutdown, yet the errors occurred after carrying out simple tasks which had been undertaken successfully several times previously in the same session.

Software will always have its failings and the overall impression of PROKAPPA was very favourable. It is a powerful and easy to use environment with which to design KBS. The above points have been raised as an indication of the type of problems that may be encountered when undertaking a project such as SIGMA, regardless of the software environment utilised.

The use of such software to develop prototype KBSs also highlights another important issue within engineering research, namely who should be carrying out the research on largely computer based projects. To construct prototype systems that are aimed at commercial end products then a large developmental multi-disciplined resource is required. This allows the engineers to have their input and the computer programmers to develop the systems at a good pace utilising the current computing methodologies. On an academic level, the resources simply are not there to allow this and so a compromise has to be chosen. Engineers do have a good understanding of data and its vagaries and if a level of computing expertise can be combined with this, it is hoped that practicable systems can be developed, albeit at a slower pace.

9.2 Further Work

SIGMA has been developed to investigate the application of KBS technology to the area of Geotechnical Engineering.

SIGMA has been produced as a prototype system, in order to demonstrate the effectiveness of the methodologies and the approach rather than producing a finished system. Accordingly there are several areas within the existing system that could be enhanced to bring the system to a final state, especially in the borehole interpolation module. The aim of producing this module was to see if an object oriented approach could improve the previous work in the area; the aim was not a full implementation of existing work. Accordingly some aspects of the implementation could be enhanced, namely the triangulation of the site and the site wide trend acquisition.

On the database side, a fully integrated suite of file reading programs could be included in the system to pre-process and import an AGS file directly into INGRES. The routines that are currently used are limited in nature and separate from SIGMA.

The scope for further work is massive with such a large scale project such as SIGMA. One of the main considerations however before any further work is undertaken would be whether or not to continue using the PROKAPPA environment. Whilst powerful and capable of producing cross platform systems, there are a number of similar products in the marketplace, both for workstations and PC's. Some of the competitors are as powerful and capable as PROKAPPA whilst running on a PC based platform, indeed the new version of PROKAPPA, KAPPA, is aimed more at PC platforms. A possible approach in the future may be to have a workstation as the central hub of a large scale KBS. This hub could act as a central data store, file and network server where the power and multi-user potential of the workstation could be best served.

PC's could logon to this main hub and either download data and software as required or remotely access these facilities over a local or wide scale network.

If the system was to be taken further towards commercialisation then a bottom up approach could also be considered, writing the established routines in a powerful low level language which has system supplied database and object-oriented tool kits. This approach could produce a polished and efficient end product.

The adoption of a national borehole database is a process which will be enhanced by the establishment of a standard format for interchange of geotechnical information. As stated in Chapter 6 there are advantages of scale and economics to be gained from the establishment of a national borehole database and the wide scale implementation of the AGS data exchange standard can only assist in this process. The administration of such a national database would present a major challenge, as well as the considerations of data confidentiality, collection and ownership. However these problems are not insurmountable and if the geotechnical and civil engineering communities could be shown the advantages to be gained from a national centralised electronic data store then its case would be greatly enhanced. Hopefully work such as SIGMA will go some way to illustrate the advantages to be gained.

The combination of Geographical Information Systems (GIS) with data structures such as those used for the GeoTec database may prove an interesting area for further development. Existing GIS systems have as an integral component a Relation Database Management System (RDBMS) to store the data integral to their operation, for example ARC/INFO has INFO, ARGIS has ORACLE (Scholten *et al*, 1990). The application of a graphical interface to complement GeoTec would allow more efficient data access albeit perhaps only when applied to a national database level.

Additions to the data structure of the GeoTec database are another area where further work could be carried out. The establishment of data structures to store the spatial results of the borehole interpretation process with the framework of GeoTec would be a significant advantage. Graphical routines could then use these structures to produce 2D fence and 3D contour diagrams of the area in question. Desk study data could also be incorporated into the GeoTec structure and initial work on these data structures is ongoing.

On completion of these further works, a full test of SIGMA utilising data from an actual site investigation should be carried out using geotechnical specialists as users. This would endeavour to prove the worth of such systems and SIGMA in particular.

9.3 Conclusions

A Knowledge Based System (KBS) called SIGMA has been developed to assist the geotechnical specialist. Its role is to provide a source of knowledge and data that can be consulted by the engineer to aid in the decision making process.

Geotechnical engineering is a suitable area for the application of KBS techniques due to the nature and type of the data produced and the reliance on expert knowledge in the field. As hardware and software platforms become capable of supporting more complex systems, hybrid KBSs that utilise aspects of rule-based, frame-based and logic programming become possible. The flexibility these hybrid systems offers allow the developer to bring together the advantages of all aspects of KBS technologies within a common environment. The ProKappa development environment running on a Sun Sparc2 has been used in this work and has proved capable of the task.

SIGMA represents a contribution to the field of Decision Support Systems, a subset of the KBS field, within the geotechnical engineering environment. SIGMA's role is twofold - it provides the structures for geotechnical data management and tools to assist the geotechnical specialist with the task of data interpretation.

SIGMA provides a methodology for the storage and analysis of ground investigation data in an organised and controlled manner. Data structures have been developed for storing all aspects of a ground investigation and have been implemented as a geotechnical database, Geotec, which forms the core of SIGMA. The database design has been conducted within the framework of the AGS standard, complemented by the addition of structures to allow for storage of parsed soil/rock description data and multi-level test storage, which could include pictures or document files. The database has been designed using the relational data model and implemented using the INGRES Relational Database Management System. The integration of the Geotec database and the PROKAPPA KBS is that of a tightly coupled externally enhanced expert system, giving the flexibility of dynamic data transfer when required whilst allowing both systems to act as independent units.

SIGMA has been designed in a modular manner so that generic routines can manipulate and analyse data to provide solutions to geotechnical problems. One of the fundamental modules of SIGMA is the parser which is capable of extracting the constituent and dominant soil/rock types and any other qualitative data from a layer description. This additional information may be stored in the Geotec database, complementing but not replacing the original text description.

With the parameter assessment module the geotechnical specialist has access to not only the measured parameter data being sought, but also to all the other data measured at this location. This data can then be correlated back to the original parameter in order to give as wide a range of values as is possible.

Tools such as the borehole interpolation module allow the user to manipulate the data stored and carry out preliminary borehole and site wide correlations to gain a feel for the nature of the sub surface conditions. It is hoped that with the ability to run through these correlation routines the user is capable of making more meaningful decisions. That is the main aim of SIGMA, to allow the geotechnical specialist to make better informed decisions.

The continuity that SIGMA represents should be able to increase the efficiency of the data management of a site investigation, important in these days of increased information flow. The GeoTec database can centrally store geotechnical data and make this information accessible to a global community, either via AGS standard data transfer or direct database communication.

On systems such as this where the integrity of the data is of such importance, the data checking incorporated within SIGMA give the user the ability to ensure consistency and continuity of data. The more confidence the user has in the data being used within SIGMA, the more likely is the acceptance of such tools in the geotechnical workplace.

Knowledge bases can be continually updated as the knowledge of a particular domain grows and the GeoTec database can accept data in a variety of formats, enabling the system to remain current. Due to the modular structure of the system, other sub systems may be added at later stages as new techniques and concepts become available to aid the geotechnical specialist.

Due to time constraints on the project a full test of SIGMA on a real site investigation was not carried out. This would have given the opportunity to evaluate

the usefulness of such systems and to gain valuable feedback from prospective end users.

The project has demonstrated the applicability of KBS technology to Geotechnical Engineering. The way forward is the development of interpretation systems such as SIGMA which can take advantage of the growth in computer based data storage. Such systems can assist geotechnical specialists in processing and interpreting geotechnical data.

References

Adams T.M., Christiano P. and Hendrickson C. (1989), *Some expert system applications in geotechnical engineering*, in Foundation Engineering: Current principles and practices, ASCE, New York, pp 885-902.

AGS (1992) *Electronic transfer of geotechnical data from ground investigations,* Publ. AGS/1/92, Association of Geotechnical Specialists, Wokingham.

Adeli H. (1987), *Knowledge-Based Systems in Structural Engineering*, in The Application of Artificial Intelligence Techniques to Civil and Structural Engineering, pp 71-78.

Adeli H. (ed.) (1988) *Expert Systems in Construction and Structural Engineering*, Chapman and Hall, London, England.

Alim S. and Munro J. (1987), *PROLOG-Based Expert Systems in Civil Engineering*, Proc. Instn. Civ. Engrs., Part 2, Volume 83, pp 1-14.

Allwood R.J., Stewart D.J and Trimble E.G. (1987), *Some Experiences from Evaluating Expert System Shell Programs and Some Potential Applications*, in Application of A.I. Techniques to Civil and Structural, (ed. Topping B.H.V.), CIVIL-COMP Press, Edinburgh, pp 1-6.

Al-Zobaidie A. and Grimson M.J. (1984) *Expert Systems and database systems: how can they help each other?* Expert Systems, Volume 4, Number 1, 30-37.

Attewell P.B. and Farmer I.W. (1976), *Principles of Engineering Geology*, Chapman and Hall Ltd, London.

Bamford C. and Curran P. (1987) *Data structures, files and databases.* MacMillan Education, Basingstoke, Hamps.

Benchimol G., Levine P. and Pomerol J.C. (1987), *Developing Expert Systems for Business*, North Oxford Academic Publishers Ltd, Oxford, England.

Bennett J.P. (1990) *Introduction to compiling techniques: a first course using ANSI C, LEX and YACC.* pp117-129.McGraw-Hill, Berks, England.

Beynon-Davis P. (1991) *Expert System and Databases: A gentle introduction* McGraw-Hill, Berks, UK.

Borland (1993) *Borland C++ Version 4: User Guide* Borland, Scotts Valley, US.

Brink A.B.A, Boniface A. and Oliver H.J. (1988) *The use of data banks in tunnelling,* The Civil Engineer in South Africa, Volume 30, Number 6, pp 291-297.

British Standard 5930 (1981), *Code of Practice for Site Investigations*, British Standards Institution, London.

British Standard 1377 (1990). *Methods of test for soils for civil engineering purposes,* British Standards Institution, London.

Buller J.V. (1964) *A computer-oriented system for the storage and retrieval of well information,* Bulletin of Canadian Petroleum Geology, Volume 12, Number 4 pp847-891.

Bundy A. (1990) A Catalogue of Artificial Intelligence Techniques Deptartment of Artificial Intelligence, Edinburgh University, Springer-Verlag, pp 23-24

Card S.K., Moran T.D. and Newell A.A. (1983) *The Psychology of Human Computer Interaction* Lawrence Erlbaun Associates, Hillsdale, NJ, US.

Carpaneto R. and Cremomini M.G. (1991), *Evaluation of Geotechnical Design Parameters by Expert System Techniques*, Proceedings of the Geotechnical Engineering Congress, in Geotechnical Special Publication, Volume 1, Number 27, (eds. McLean F.G., Campbell DW. A. and Harris D.W.), ASCE, Boulder, Colorado, pp 422-433.

Chaplow R. (1986) *Production of borehole logs using a microcomputer* Quarterly Journal of Engineering Geology, Volume 19, No. 3, pp 291-300.

Chahine J.R. and Janson B.N. (1987), *Interfacing Databases with Expert Systems: a Retaining Wall Management Application,* Microcomputers in Civil Engineering, Volume 2, No. 1, pp 19-38.

Chen H.M. (1994) *The graphical representation of ground conditions from borehole information* Final Year Project Report, School of Engineering and Computer Science, University of Durham.

Clayton C. R. I., Simons N. E. and Matthews M. C. (1982) *Site Investigation,* Granada Publishing Ltd, London.

Clocksin W.F. and Mellish C.S. (1981) *Programming in PROLOG*, Springer-Verlag, Berlin, Germany.

Codd E.F. (1970) *A relational model for large shared data banks,* Communications of ACM, Volume 13, Number 6, pp 377-387

Council for Science and Society (1989) *Benefits and Risks of Knowledge Based Systems*. Report of the working party. Oxford University Press, Oxford, UK.

Cripps J.C. (1978) *Computer storage of geotechnical data for use during urban development,* Bull. Int. Ass. Eng. Geol., 19, pp 290-299.

Date C.J. (1983) *Databases: A Primer* Addison-Wesley, Reading, Mass, US.

Date C.J. (1986) *An Introduction to Database Systems* Volume 1, 4th Edition, Addison-Wesley, Reading, Mass, US

Davey-Wilson I.E.G. (1991), *Geotechnical Laboratory Test Simulation using AI Techniques*, in Artificial Intelligence and Civil Engineering (ed. Topping B.H.V.), CIVIL-COMP Press: Edinburgh, pp 119-124.

Day R.B., Tucker E.V. and Wood L.A. (1983) *The computer as an interactive geotechnical data bank and analysis tool*, Proc. Geol. Assoc., Volume 94, Number 2, pp 123-132.

Easden K.D. (1977) *Human Relationships and User Involvement in Systems Design* Computer Management Volume 19, Number 10-12.

Finn P.S. and Eldred P.J.L. (1987) *Data Management with microcomputers in geotechnical engineering practice* Quarterly Journal of Engineering Geology, Volume 20, Number 2, pp 131-137.

Frederick C. O., Wong Y. C. and Edge F. W. (1970) *Two Dimensional Automatic Mesh Generation for Structural Analysis*, International Journal of Numerical Methods in Engineering, Volume 2, pp. 133-144.

Forster A. and Culshaw M.G. (1990) *The use of site investigation data for the preparation of engineering geological maps and reports by planners and civil engineers,* Engineering Geology, Volume 29, pp 347-354.

Gallaire H and Minkler J. (1978) *Logic and Databases*, Plenum Press, New York US.

Gaschnig J. (1982) *Prospector: an expert system for mineral exploration* Machine Intelligence, Infotech State of the Art. Report 9 (ed. Bond A.), MIT Press, Camb, Mass, US.

Giles D. (1992) *The geotechnical computer workstation: the link between the geotechnical database and the geographical information systems* International Conference on Geotechnics and Computers, Paris, France, pp 685-690.

Gillette D.R. (1991), *An Expert System for Estimating Soil Strength Parameters*, Proc. Geotechnical Engineering Congress, in Geotechnical Special Publication, vol. 1, n. 27, (eds. McLean F.G., Campbell DW. A. and Harris D.W.), ASCE: Boulder, Colorado, pp 276-287.

Giolas A. (1994) *A knowledge based system for the estimation of ground properties* PhD thesis, University of Durham, *in preparation*

Greenshaw L.M., Bentley S.P. and Rice S.M.M. (1987) *The management of Site Investigation Data using 'Strata 3'*, Civil-Comp 87, Civil-Comp Press, Edinburgh, pp 223-227.

Greenwood J.R. (1988) *Developments in computerised ground investigation data,* Ground Engineering, Volume 21, Number 6, pp 36-41.

Halim I.S., Tang W.H. and Garrett J.H.Jr. (1991), *Knowledge-Assisted Interactive Probabilistic Site Characterization*, Proceedings of the Geotechnical Engineering Congress, in Geotechnical Special Publication, Volume 1, Number 27, (eds. McLean F.G., Campbell DW. A. and Harris D.W.), ASCE, Boulder, Colorado, pp 264-275.

Howland and Polanski (1985) *A microcompter based system to provide report quality borehole records* Quarterly Journal of Engineering Geology, Volume 18, Number 4, pp 357-361.

Howland A.F. (1991) *'New boreholes for old' The exchange of borehole information with BGS,* Geoscientist, Volume 1, Number 5, pp 20-21.

Ibrahim J. (1994) *The Application of Knowledge Based Technology to Geotechnical Site Investigation* PhD Thesis, Herriot-Watt University.

INGRES (1990a), *An Introduction to INGRES*, HECRC, ISG 245, Relational Technology Inc, California, USA.

INGRES (1990b), *SQL Reference Guide*, HECRC, ISG 219, Relational Technology Inc, California, USA.

Institution of Civil Engineers (1991) *Inadequate Site Investigation*, (ed Littlejohn G.S.) Thomas Telford, London.

Intellicorp (1991) *PROKAPPA Users Guide*, Intellicorp PK2.0-UG-2, Intellicorp Inc, California, USA.

Intelligent Environments (1987) *Crystal User's Guide* Intelligent Environments, Richmond, Surrey, UK

Ishikawa H., Izumida Y. and Kawato N. (1991) *An Object-Oriented Database: System and Application.* IEEE Pacific Rim Conference on Communications, Computers and Signal Processing, May 1991, pp 288-291.

ISRM-SPE (1990) *Rock Properties for Petroleum Engineers,* International Society for Rock Mechanics and Society of Petroleum Engineers joint commission, U.K. Working Group - Circular Number 1, August 1990.

Jarke M. and Vassiliou Y. (1984) *Coupling expert systems with database management systems* Computer Surveys 16(2), pp112-152.

Johnson K.R.(1991) *An Evaluation of ProKappa,* AIAI Tech. report AIAI-TR-96, Artificial Intelligence Applications Institute, Edinburgh.

Kim W. (1990) *Architecture of the ORION Next-generation Database System.* IEEE Transactions on Knowledge and Engineering. Volume 2, pp109-124.

Knowledge Garden (1985) *Knowledge Pro for Windows: User's guide* Knowledge Garden, New York, US.

Konigsberger H.K. and De Bruyn F.W.G.M. (1990), *Prolog from the Beginning,* McGraw-Hill Book Company (UK) Limited, London, England.

Kowalski R. (1985) *Logic Programming* Byte Volume10, Number 8, pp161-176.

Lindholm D. A. (1983) *Automatic Triangular Mesh Generation on Surfaces of Polyhedra,* IEEE Transactions on Magnetics, Volume MAG-19, Number 6, pp. 2539-2542.

Lindsay P.H. and Norman D.A. (1977) *Human Information Processing* Academic Press, London, UK.

Lok M.H. (1987) *LOGS: a prototype expert system to determine stratification of subsurface profiles from multplie borings,* MSc. Thesis, Carnegie-Mellon University, US.

MacKenzie A. (1994) *A Hydrogeological database for Honduras* Accepted for Geol. Soc. Special Publication on Geological Data Management.

Maher M.L. (ed.) (1987) *Expert Systems for Civil Engineers,* American Society of Civil Engineers, New York, U.S.A.

Maher M.L. and Allen R. (1987), *Expert Systems Components,* in Expert Systems for Civil Engineers, (ed. Maher M.L.), American Society of Civil Engineers, New York, U.S.A., pp 3-14.

Magnan J.P. (1992) *CESSOL: Bilan du Développement d'un Système-expert,* Proc. Int. Conf. on Geotechnics and Computers, Presses de l'École Nationale de Ponts et Chaussées: Paris, pp 579-585.

Marcellus D.H. (1989), *Expert Systems Programming in Turbo Prolog,* Prentice-Hall Inc, New Jersey.

Martin J. (1973) *Design of Man Computer Dialogues* Prentice Hall, Englewood Cliffs, New Jersey, US.

Mavroidi A. (1991) *Rock Properties Database,* MSc Dissertation, School of Engineering and Computer Science, University of Durham.

Miles J. and Moore C. (1994) *Practical knowledge based systems for conceptual design*, pp 35-53, Springer-Verlag, Berlin, Germany.

Missikoff M. and Wiederhold G. (1986) *Towards a unified approach for expert and database systems* Expert Database Systems: Proc. first International Conference (ed. Kerschberg L.) Benjamin Cummings, Redwood City, California, US.

Moran J.A. (1990) *Computerised Geotechnical Information -Trends,* SERC Seminar on Information Technology in Geotechnical Engineering, University of Durham, March 1990.

Mott MacDonald and Soil Mechanics Ltd (1994) *Study of the efficiency of site investigation practices* Project Report 60 E063A/HG, Department of Transport, London, UK.

Moula M (1993) *A knowledge based system to assist in the selection of appropriate geotechnical field tests*, PhD Thesis, School of Engineering and Computer Science, University of Durham.

Moula M. & Toll D.G. (1993*) Representing Geotechnical Knowledge: Soils and Field Tests*, in Knowledge Based Systems for Civil and Structural Engineering (ed. Topping B.H.V.), Civil-Comp Press: Edinburgh, pp 171-182.

Moula M., Toll D.G. and Vaptismas N. (1994) Knowledge-based systems in geotechnical engineering, Geotechnique (in press).

Mullarkey P.W. (1986), *A Geotechnical KBS Using Fuzzy Logic*, in Applications of A.I. in Engineering Problems, 1st Int. Conf., Southampton University (eds. Sriram D. and Adey R.), Springer-Verlag, Volume 2, pp 847-859.

Mullarkey P.W. (1987), *Languages and Tools for Building Expert Systems*, in Expert Systems for Civil Engineers, (ed. Maher M.L.), American Society of Civil Engineers, New York, U.S.A., pp 15-34.

Mullarkey P.W. and Fenves S.J. (1986), *Fuzzy logic in a geotechnical knowledge based system : CONE*, Civil Engineering Systems, Volume 3, Number 2, pp 58-81.

Myopoulos J (1989) *On knowledge based management systems* Readings in Artificial Intelligence and Databases (ed. Myopoulos J. and Brodie M.) Morgan Kaugmann, New York, US

MZ Associates (1994) *SID: A Geotechnical Data Management System* Information Pack, MZ Associates, Llwynhaf, Camarthen, Wales, UK

Navanthe S.B. (1989) *Fundamentals of database systems* Addison-Wesley, Reading, Mass, US.

Naylor J.A. (1992) *Geotechnical Data Management* 2nd BGS Young Engineers Symposium, University of Glasgow.

Neuron Data (1989) *Nexpert: Integrated Expert Systems in Real World Applications* Software Sciences, Neuron Data, US.

Neidell N.S. (1969) *Ambiguity functions and the concept of geological correlations Symposium on Computer Applications in Petroleum Exploration*, (ed. Merrian D.F.), Computer Contribution Number 40, Kansas Geological Survey, pp 19-33.

Norkin D.D. (1985), *Expert System for Geotechnical Site Characterisation*, MSc dissertation, Carnegie-Mellon University, Dept. of Civil Engineering.

Perry J.(1991) *The transfer of geotechnical data on highway schemes* Report for AGS meeting on Data Standards.

Rapier J. and Wainwright D. (1987) *Use of the geotechnical database GEOSHARE for site investigation data management,* Quarterly Journal of Engineering Geology, Volume 20, Number 3, pp221-230.

Rayward-Smith V.J. (1983) *A first course in formal language theory.* pp 85-102 Balckwell Scientific Publications, Oxford.

Rehak D.R., Christiano P.P. and Norkin D.D. (1985), *SITECHAR: An Expert System Component of a Geotechnical Site Characterization Workbench*, in Applications of knowledge-based systems to engineering analysis and design (ed. Dym C.L.), Am. Soc. Mech. Eng., New York.

Rhind D.W. and Sissons J.B. (1971) *Data banking of Drift borehole records for the Edinburgh area* Report 71/75 Institue of Geological Science, London.

Righetti G.A. and Cremonini M.G. (1988), *The DAISY Environment and the Expert System GUESS*, in Artificial Intelligence in Engineering: Diagnosis and Learning, (ed. Gero J.S.), Elsevier, Amsterdam.

Rodger L. (1992) *Borehole Exchange Support* Geoscientist, Volume 1, Number 6, pp 32-33.

Rudman A.I. and Lankston D.G. (1973) *Stratigraphic Correlation of Subsurface Geology Data by Computer,* Journal of the Institue of Mathematical Geology, Volume 4, Number 4, pp 577-588.

Santamarina J.C. and Chameau J.L. (1987), *Expert Systems for Geotechnical Engineers*, Computing in Civil Engineering, Volume 1, Number 4, pp 241-252.

Siller J.T. (1987), *Expert Systems in Geotechnical Engineering*, in Expert Systems for Civil Engineers: Technology and Applications, (ed. Maher M.L.), ASCE, NY, pp 77-84.

Smith I.G.N. and Oliphant J. (1991), *The Use of a Knowledge-Based System for Civil Engineering Site Investigations*, in Artificial Intelligence and Civil Engineering (ed. Topping B.H.V.), CIVIL-COMP Press, Edinburgh, pp 105-112.

Stefik M. and Bobrow D.G. (1986) *Object-oriented programming: Themes and variations* AI Magazine, Volume 6 Number 4, pp 40-62.

Staten P.M. and Caronna S. (1992) *Geotechnical Data Management of a Major Highway Scheme in the UK*, Proc. International Conference on Geotechnics and Computers, Paris, pp 741-748, Presses de l'École Nationale de Ponts et Chaussées: Paris.

Stonebraker M. (1986) *The INGRES papers* Addison-Wesley, Reading Mass, US.

Sukaviriya P (1993) *From user interface design to the support of intelligent and adaptive interfaces: an overhaul of user interface software infrastructure*, Knowledge-Based Systems, Butterworth-Heinemann Ltd, Volume 6 Number 4.

Sutcliffe A. (1988) *Human Computer Interface Design.* pp 6-49 MacMillan Education, Basingstoke, Hamps.

Sylvester C (1991) *A Geotechnical Database,* Final Year Project Report, School of Engineering and Computer Science, University of Durham.

Thomas P.R., Kor F.H. and Matheson G.D. (1992) *A Knowledge-based System for Ground Investigation in Rock*, Proc. EUROCK'92 Conf., Chester, 1992.

Threadgold L. (1992) *Borehole Database* Geoscientist April/May 1992, Volume 2, Number 2, pp 30-31

Toll D.G.(1989) *Representing the Ground*, Proc. NATO Advanced Study Institute: Optimisation and Decision Support Systems in Civil Engineering, Heriot-Watt University, Vol II.

Toll D.G., Moula M. and Vaptismas N. (1991) *Representing the engineering description of soils in knowledge based systems*, in Applications of artificial intelligence in engineering VI (eds Rzevski G & Adey R.A.), pp 847-856, Computational Mechanics Publications, Southampton.

Toll D.G., Moula M., Oliver A. and Vaptismas N. (1992), *A Knowledge Based System for Interpreting Site Investigation Information,* Proc. International Conference on Geotechnics and Computers, Paris, pp 607-614, Presses de l'École Nationale de Ponts et Chaussées: Paris.

Toll, D.G. and Oliver A.J. (1993), *SIGMA: A System for Interpreting Geotechnical Information,* Proc. SERC Conf. on Informing Technologies for Construction, Civil

Engineering and Transport, (eds. Powell J.A. & Day R.), Brunel University with SERC: London, pp 245-254.

Toll D.G., Vaptismas N. and Moula M. (1993) *A Methodology for Comparing Soils for use in Knowledge Based Systems,* Computer Systems in Engineering (in press).

Tomlinson M.J. (1986), *Foundation Design and Construction,* Fifth Edition, Longman Scientific and Technical, Essex, England.

Ullman J.D. (1989) *Principles of Database and Knowledge-based Systems (Vol2),* Computer Science Press, Rockville, Md

Vaptismas N. (1992), *A Methodology for the Interpretation of Ground Conditions from Borehole Information,* PhD thesis, University of Durham.

Vaptismas N. and Toll D.G. (1993) *Interpreting Borehole Information,* in Knowledge Based Systems for Civil and Structural Engineering (ed. Topping B.H.V.), Civil-Comp Press: Edinburgh, pp 153-159.

Vergobbi P, Puech A. and Meimon Y. (1992) *SAGITAIRE: Système d'Assistance Géotechnique à l'Engéniere de Reconnaissance,* Proc. Int. Conf. on Geotechnics and Computers, Presses de l'École Nationale de Ponts et Chaussées: Paris, pp 615-622.

Wade B.R. (1994) *The triangulation of boreholes for interpretation of site investigation* Final Year Project Report, School of Engineering and Computer Science, University of Durham.

Walker A. (1984) *Databases, Expert Systems and PROLOG* Artificial Intelligence: Applications for Business (ed. Reitman W.) Aldex Publishing, New York, US.

Watson A.S. (1990) *CAD data exchange in construction,* Proc. Instn. Civ. Engrs, Part 1, 88 pp 955-969.

Weltman A.J. and Head J.M. (1983), *Site Investigation Manual,* CIRIA Special Publication 25, Construction Industry Research and Information Association, London.

Winter M.G. & Matheson G.D. (1992) *Development of a Knowledge-based System for Ground Investigation,* Proc. Int. Conf. on Geotechnics and Computers, Presses de l'École Nationale de Ponts et Chaussées: Paris, pp 623-630.

Whitby B. (1988) *AI: a handbook of professionalism.* Ellis Horwood, Chicester.

Wong K.C., Poulos H.G. and Thorne C.P. (1989), *Site Classification by Expert Systems,* Computers and Geotechnics, Volume 8, pp 133-156.

Appendices

List of appendices

Appendix 1
Structure of Data Tables in the GeoTec Database

Legend:

K - Key Field
A - Comments field which could contain a pointer to an ASCII file

proj

Field Name	Field Type	Field Description	Remarks
proj_id	varchar(10)	Project / Site Investigation Code	K
proj_name	varchar(100)	Name of Project / Site Investigation	
proj_loc	varchar(100)	Location of Project	
proj_date	varchar(15)	Commencement date	
proj_clnt	varchar(50)	Project Client	
proj_cont	varchar(50)	Main Project Contractor	
proj_eng	varchar(40)	Principal Project Engineer	
proj_memo	varchar(250)	Project Comments / Details	A

hole

Field Name	Field Type	Field Description	Remarks
proj_id	varchar(10)	Project / Site Investigation Code	K
hole_id	varchar(10)	Borehole / Trial Pit Code	K
hole_type	varchar(10)	Type of Exploratory Hole	
hole_nate	i4	Natural Easting	
hole_natn	i4	Natural Northing	
hole_gl	f4	Ground Level (m)	
hole_fdep	f4	Final Depth (m)	
hole_star	date	Start Date of Exploratory Hole	
hole_log	varchar(40)	Definitive Person Responsable for logging the hole	
hole_rem	varchar(250)	Comments / Details	A
hole_lett	varchar(10)	O.S. letter Grid Reference	
hole_locx	i2	Local Grid X coordinate (m)	
hole_locy	i2	Local Grid Y coordinate (m)	
hole_diam	varchar(250)	Hole diameter details	
hole_casg	varchar(250)	Casing details	
hole_endd	date	Hole end date	
hole_bacd	date	Backfill date	
hole_crew	varchar(100)	Drillers Name	
hole_ornt	i2	Orientation of hole, from north (deg)	
hole_incl	i2	Inclination of hole, from horizontal (deg)	
hole_exc	varchar(100)	Equipment used for excavation	
hole_shor	varchar(100)	Shoring / Support used	
hole_stab	varchar(100)	Stability comments	
hole_dimw	f4	Trial pit width (m)	
hole_diml	f4	Trial pit depth (m)	

geol

Field Name	Field Type	Field Description	Remarks
proj_id	varchar(10)	Project / Site Investigation Code	K
horz_no	i4	Horizon number	K
geol_desc	varchar(300)	Description of geological horizon	
geol_leg	varchar(5)	Legend for horizon	

lay

Field Name	Field Type	Field Description	Remarks
proj_id	varchar(10)	Project / Site Investigation Code	K
hole_id	varchar(10)	Borehole / Trial Pit Code	K
lay_no	i4	Layer Number	K
lay_dtop	f4	Depth to top of layer (m)	
lay_thck	f4	Layer thickness (m)	
lay_des	varchar(300)	Original description of layer	
horz_no	i4	Horizon number	

strt

Field Name	Field Type	Field Description	Remarks
proj_id	varchar(10)	Project / Site Investigation Code	K
hole_id	varchar(10)	Borehole / Trial Pit Code	K
lay_no	i2	Layer Number	K
strt_no	i2	Strata number	
strt_mstp	varchar(30)	Main soil type	
strt_mscd	varchar(30)	Moisture condition	
strt_cons	varchar(30)	Consistency	
strt_psty	varchar(30)	Plasticity	
strt_wthg	varchar(30)	Weathering	

stst

Field Name	Field Type	Field Description	Remarks
proj_id	varchar(10)	Project / Site Investigation Code	K
hole_id	varchar(10)	Borehole / Trial Pit Code	K
lay_no	i2	Layer Number	K
strt_no	i2	Strata number	K
stst_stno	i2	Structure number	
stst_stct	varchar(30)	Structure feature	
stst_spc	varchar(30)	Structure spacing	
stst_dip	i2	Dip	
stst_ornt	f4	Orientation (deg)	
stst_srf	varchar(30)	Surface	
stst_dcon	varchar(30)	Discontinuity modifier	

cnst

Field Name	Field Type	Field Description	Remarks
proj_id	varchar(10)	Project / Site Investigation Code	K
hole_id	varchar(10)	Borehole / Trial Pit Code	K
lay_no	i2	Layer Number	K
strt_no	i2	Stratum number	K
cons_no	i2	Constituent number	
cnst_type	varchar(20)	Constituent Type	
cnst_amnt	varchar(20)	Constituent Amount	
cnst_grdg	varchar(30)	Grading	
cnst_shp	varchar(30)	Shape	
cnst_txt	varchar(30)	Texture	
cnst_dstb	varchar(30)	Distribution	

ctcl

Field Name	Field Type	Field Description	Remarks
proj_id	varchar(10)	Project / Site Investigation Code	K
hole_id	varchar(10)	Borehole / Trial Pit Code	K
lay_no	i2	Layer Number	K
strt_no	i2	Strata number	K
cons_no	i2	Constituent number	K
ctcl_clno	i2	Colour number	
ctcl_mcl1	varchar(30)	Main colour 1	
ctcl_scl	varchar(30)	Secondary colour	
ctcl_mod	varchar(30)	Colour Modifier	
ctcl_strc	varchar(30)	Colour Structure	

wstk

Field Name	Field Type	Field Description	Remarks
proj_id	varchar(10)	Project / Site Investigation Code	K
hole_id	varchar(10)	Borehole / Trial Pit Code	K
wstk_dep	f4	Depth of water strike (m)	
wstk_cas	f4	Casing depth at water strike (m)	
wstk_date	date	Date of water strike	
wstk_time	varchar(15)	Time of water strike	
wstk_post	f4	Post strike depth after wstk_nmin minutes	
wstk_nmin	i4	Minutes after strike	
wstk_flow	varchar(100)	Flow rate remarks	
wstk_seal	f4	Depth at which water strike sealed by casing	

ptim

Field Name	Field Type	Field Description	Remarks
proj_id	varchar(10)	Project / Site Investigation Code	K
hole_id	varchar(10)	Borehole / Trial Pit Code	K
ptim_date	date	Date of progress reading	
ptim_time	varchar(15)	Time of progress reading	
ptim_dep	f4	Depth at ptim_time (m)	
ptim_cas	f4	Casing depth at ptim_time (m)	
ptim_wat	f4	Depth to water at ptim_time (m)	
ptim_rem	varchar(250)	Comments / Details	A

samp

Field Name	Field Type	Field Description	Remarks
proj_id	varchar(10)	Project / Site Investigation Code	K
hole_id	varchar(10)	Borehole / Trial Pit Code	K
samp_top	f4	Depth to top of sample (m)	K
samp_type	varchar(30)	Sample Type	
samp_ref	i4	Sample reference number	
samp_dia	f4	Sample diameter (m)	
samp_base	f4	Depth to base of sample (m)	
samp_desc	varchar(250)	Sample description	
samp_ublo	i4	Number of blows required to drive sampler	
samp_rem	varchar(250)	Comments / Details	A
geol_stat	varchar(10)	Stratum code (for use with trial pits)	

spdt

Field Name	Field Type	Field Description	Remarks
proj_id	varchar(10)	Project / Site Investigation Code	K
hole_id	varchar(10)	Borehole / Trial Pit Code	K
samp_ref	i4	Sample reference number	K
spec_ref	i4	Specimen reference number	K
spdt_dtop	f4	Depth to top of specimen (m)	
spdt_dbot	f4	Depth to bottom of specimen (m)	
spdt_spty	varchar(50)	Specimen type	
spdt_rem	varchar(250)	Comments / Details	A

core

Field Name	Field Type	Field Description	Remarks
proj_id	varchar(10)	Project / Site Investigation Code	K
hole_id	varchar(10)	Borehole / Trial Pit Code	K
core_top	f4	Depth to top of core run (m)	K
core_bot	f4	Depth to bottom of core run (m)	
core_prec	f4	Percentage of core recovered	
core_srec	f4	Percentage of solid core recovered	
core_rqd	f4	R.Q.D. for core run	
core_rem	varchar(250)	Comments / Details	A

frac

Field Name	Field Type	Field Description	Remarks
proj_id	varchar(10)	Project / Site Investigation Code	K
hole_id	varchar(10)	Borehole / Trial Pit Code	K
frac_top	f4	Depth to top of Fracture Index zone (m)	K
frac_base	f4	Depth to base of Fracture Index zone (m)	
frac_fi	varchar(5)	Fracture Index over zone	
frac_imin	varchar(5)	Minimum Fracture Index over zone	
frac_iave	varchar(5)	Average Fracture Index over zone	
frac_imax	varchar(5)	Maximum Fracture Index over zone	

drem			
Field Name	**Field Type**	**Field Description**	**Remarks**
proj_id	varchar(10)	Project / Site Investigation Code	K
hole_id	varchar(10)	Borehole / Trial Pit Code	K
drem_dpth	f4	Depth of drem_drem (m)	K
drem_rem	varchar(250)	Depth related remark	A

ispt			
Field Name	**Field Type**	**Field Description**	**Remarks**
proj_id	varchar(10)	Project / Site Investigation Code	K
hole_id	varchar(10)	Borehole / Trial Pit Code	K
ispt_top	f4	Depth to top of test (m)	K
ispt_seat	i4	Number of blows for seating drive	
ispt_main	i4	Number of blows for main test drive	
ispt_npen	f4	Total penetration for test (m)	
ispt_nval	i4	SPT N Value	
ispt_cas	f4	Casing depth at time of test (m)	
ispt_wat	f4	Water depth at time of test (m)	
ispt_type	varchar(10)	Type of SPT test	
ispt_rem	varchar(250)	Comments / Details	A
ispt_inc1	i4	Number of blows for 1st 75mm	
ispt_pen1	i4	Penetration (mm)	
ispt_inc2	i4	Number of blows for 2nd 75mm	
ispt_pen2	i4	Penetration (mm)	
ispt_inc3	i4	Number of blows for 3rd 75mm	
ispt_pen3	i4	Penetration (mm)	
ispt_inc4	i4	Number of blows for 4th 75mm	
ispt_pen4	i4	Penetration (mm)	
ispt_inc5	i4	Number of blows for 5th 75mm	
ispt_pen5	i4	Penetration (mm)	
ispt_inc6	i4	Number of blows for 6th 75mm	
ispt_pen6	i4	Penetration (mm)	

pref			
Field Name	**Field Type**	**Field Description**	**Remarks**
proj_id	varchar(10)	Project / Site Investigation Code	K
hole_id	varchar(10)	Borehole / Trial Pit Code	K
pref_tdep	f4	Depth to bottom of piezometer tip (m)	K
pref_date	date	Installation date	
pref_type	varchar(10)	Type of Piezometer	
pref_trps	f4	Depth to top of response zone (m)	
pref_brps	f4	Depth to base of response zone (m)	
pref_rem	varchar(250)	Comments / Details	A

pobs

Field Name	Field Type	Field Description	Remarks
proj_id	varchar(10)	Project / Site Investigation Code	K
hole_id	varchar(10)	Borehole / Trial Pit Code	K
pobs_tdep	f4	Depth to bottom of piezometer tip (m)	K
pobs_date	date	Date of piezometer reading	
pobs_time	varchar(15)	Time of piezometer reading	
pobs_dep	f4	Depth to water below ground surface (m)	
pobs_head	f4	Head of water above piezometer tip (m)	
pobs_rem	varchar(250)	Comments / Details	A

clss

Field Name	Field Type	Field Description	Remarks
proj_id	varchar(10)	Project / Site Investigation Code	K
hole_id	varchar(10)	Borehole / Trial Pit Code	K
samp_ref	i4	Sample reference number	K
spec_ref	i4	Specimen reference number	K
clss_nmc	f4	Natural moisture content	
clss_ll	f4	Liquid limit	
clss_pl	varchar(5)	Plastic limit	
clss_pi	varchar(5)	Plasticity index	
clss_dden	f4	Dry Density (Mgm^{-3})	
clss_bden	f4	Bulk Density (Mgm^{-3})	
clss_pd	f4	Particle Density (Mgm^{-3})	
clss_425	f4	Percentage passing 425 um sieve	
clss_prep	varchar(100)	Method of preparation	
clss_slim	f4	Shrinkage limit	
clss_ls	f4	Linear shrinkage	
clss_hvp	i4	Hand vane undrained shear strength, peak (kNm^{-2})	
clss_hvr	i4	Hand vane undrained shear strength, remoulded (kNm^{-2})	
clss_ppen	i4	Pocket penetrometer undrained shear strength (kNm^{-2})	
clss_rem	varchar(250)	Comments/Details	A

grad

Field Name	Field Type	Field Description	Remarks
proj_id	varchar(10)	Project / Site Investigation Code	K
hole_id	varchar(10)	Borehole / Trial Pit Code	K
samp_ref	i4	Sample reference number	K
spec_ref	i4	Specimen reference number	K
grad_size	f4	Sieve size	
grad_perp	f4	Percentage passing	
grad_type	varchar(10)	Grading analysis test type	
gread_rem	varchar(250)	Comments/Details	A

trig			
Field Name	**Field Type**	**Field Description**	**Remarks**
proj_id	varchar(10)	Project / Site Investigation Code	K
hole_id	varchar(10)	Borehole / Trial Pit Code	K
samp_ref	i4	Sample reference number	
spec_ref	i4	Specimen reference number	
trig_type	varchar(10)	Test type	
trig_cond	varchar(50)	Sample condition	
trig_rem	varchar(250)	Comments / Details	A
trig_cu	i4	Value of undrained shear strength (kNm^{-2})	
trig_coh	i4	Cohesion intercept associated with trig_phi (kNm^{-2})	
trig_phi	i4	Angle of friction for effective shear strength triaxial test (deg)	

trix			
Field Name	**Field Type**	**Field Description**	**Remarks**
proj_id	varchar(10)	Project / Site Investigation Code	K
hole_id	varchar(10)	Borehole / Trial Pit Code	K
samp_ref	i4	Sample reference number	K
spec_ref	i4	Specimen reference number	K
trix_tesn	i4	Triaxial test/stage number	
trix_sdia	i4	Specimen diameter (m)	
trix_mc	f4	Specimen initail moisture content	
trix_cell	i4	Total cell pressure (kNm^{-1})	
trix_devf	i4	Deviator stress at failure (kNm^{-1})	
trix_slen	i4	Sample length (m)	
trix_bden	f4	Initial bulk Density (Mgm^{-3})	
trix_dden	f4	Initial dry Density (Mgm^{-3})	
trix_pwpi	i4	Porewater pressure at start of shear test (kNm^{-1})	
trix_pwpf	i4	Porewater pressure at failure (kNm^{-1})	
trix_strn	f4	Strain at failure	
trix_mode	varchar(40)	Mode of failure	

cbrg			
Field Name	**Field Type**	**Field Description**	**Remarks**
proj_id	varchar(10)	Project / Site Investigation Code	K
hole_id	varchar(10)	Borehole / Trial Pit Code	K
samp_ref	i4	Sample reference number	K
spec_ref	i4	Specimen reference number	K
cbrg_cond	varchar(50)	Sample condition	
cbrg_meth	varchar(100)	Method of remoulding	
cbrg_rem	varchar(250)	Comments / Details	A
cbrg_nmc	f4	Natural moisture content	
cbrg_200	i4	Weight percent retained on 20mm sieve	
cbrg_swel	f4	Amount of swell recorded	

cbrt

Field Name	Field Type	Field Description	Remarks
proj_id	varchar(10)	Project / Site Investigation Code	K
hole_id	varchar(10)	Borehole / Trial Pit Code	K
samp_ref	i4	Sample reference number	K
spec_ref	i4	Specimen reference number	K
cbrt_tesn	i4	CBR test number	
cbrt_top	f4	CBR at top	
cbrt_bot	f4	CBR at bottom	
cbrt_mct	f4	Moisture content at top	
cbrt_mcbt	f4	Moisture content at bottom	
cbrt_bden	f4	Bulk Density (Mgm^{-3})	
cbrt_dden	f4	Dry Density (Mgm^{-3})	

chem

Field Name	Field Type	Field Description	Remarks
proj_id	varchar(10)	Project / Site Investigation Code	K
hole_id	varchar(10)	Borehole / Trial Pit Code	K
samp_ref	i4	Sample reference number	
spec_ref	i4	Specimen reference number	
chem_tsul	f4	Total sulphate content (%)	
chem_asul	f4	Sulphate aqueous extract 2:1 soil/water (gl^{-1})	
chem_wsul	f4	Water sulphate content (gl^{-1})	
chem_ph	f4	Soil/water pH value	
chem_rem	varchar(250)	Comments / Details	A
chem_orgm	varchar(250)	Method of organic test	
chem_org	f4	Organic matter content (%)	
chem_020	f4	Percentage passing 20 um sieve (%)	
chem_loi	f4	Mass loss on ignition (%)	
chem_co2m	varchar(250)	Method of carbonate test	
chem_co2	f4	Carbonate content as CO_2 (%)	
chem_acl	f4	Percentage of acid soluble chloride ions (%)	
chem_wcl	f4	Percentage of water soluble chloride ions (%)	
chem_dcl	i4	Dissolved chloride ions (mgl^{-1})	
chem_cln	varchar(250)	Notes on chloride test	
chem_tdsm	varchar(250)	Total dissolved solids. Test methods and notes	
chem_tds	f4	Total solids disolved in water (%)	
chem_resm	varchar(250)	Resistivity test method	
chem_res	i4	Resistivity of soil sample corrected to 20 C (ohm)	
chem_remc	f4	Moisture content of sample for resistivity	
chem_rebd	f4	Bulk Density of sample for resistivity (Mgm^{-3})	
chem_rdxm	varchar(250)	Redox test information	
chem_rdx	i4	Redox potential	
chem_rdph	f4	pH of redox sample	

cong			
Field Name	**Field Type**	**Field Description**	**Remarks**
proj_id	varchar(10)	Project / Site Investigation Code	K
hole_id	varchar(10)	Borehole / Trial Pit Code	K
samp_ref	i4	Sample reference number	K
spec_ref	i4	Specimen reference number	K
cong_type	varchar(10)	Test type	
cong_cond	varchar(50)	Sample Condition	
cong_rem	varchar(250)	Comments / Details	A
cong_incm	f4	Coefficient of volume compressibility over cong_incd (m^2MN^{-2})	
cong_incd	varchar(40)	Defined stress range (kNm^2)	
cong_dia	i4	Test specimen diameter (mm)	
cong_higt	i4	Test specimen height (mm)	
cong_mci	f4	Initial moisture content (%)	
cong_mcf	f4	Final moisture content (%)	
cong_bden	f4	Initial bulk Density (Mgm^{-3})	
cong_dden	f4	Initial dry Density (Mgm^{-3})	
cong_pden	f4	Particle Density	
cong_satr	f4	Initial degree of saturation (%)	
cong_sprs	f4	Swelling pressure (kNm^2)	
cong_sath	f4	Height change of specimen on saturation as % of original height	

cons			
Field Name	**Field Type**	**Field Description**	**Remarks**
proj_id	varchar(10)	Project / Site Investigation Code	K
hole_id	varchar(10)	Borehole / Trial Pit Code	K
samp_ref	i4	Sample reference number	K
spec_ref	i4	Specimen reference number	K
cons_incn	i4	Oedometer stress increment number	
cons_ivr	f4	Initial voids ratio	
cons_incf	f4	Stress at end of stress increment/decrement (kNm^2)	
cons_ince	f4	Voids ratio at end of stress increment	
cons_inmv	f4	Coefficient of volume compressability over stress increment	
cons_incv	f4	Coefficient of consolidation over stress increment	
cons_insc	f4	Coefficient of secondary compression over stress increment	

chlk			
Field Name	**Field Type**	**Field Description**	**Remarks**
proj_id	varchar(10)	Project / Site Investigation Code	K
hole_id	varchar(10)	Borehole / Trial Pit Code	K
samp_ref	i4	Sample reference number	K
spec_ref	i4	Specimen reference number	K
chlk_tesn	i4	Chalk crushing test number	
chlk_ccv	f4	Chalk crushing value	
chlk_mc	f4	Chalk natural Moisture content	
chlk_smc	f4	Chalk saturated Moisture content	
chlk_010	f4	Weight percent of material retained on 10mm sieve	
chlk_rem	varchar(250)	Comments / Details	A

rock

Field Name	Field Type	Field Description	Remarks
proj_id	varchar(10)	Project / Site Investigation Code	K
hole_id	varchar(10)	Borehole / Trial Pit Code	K
samp_ref	i4	Sample reference number	K
spec_ref	i4	Specimen reference number	K
rock_pls	f4	Uncorrected point load	
rock_plsi	f4	Size corrected point load index	
rock_pltf	varchar(10)	Point load test type	
rock_ucs	f4	Uniaxial compressive strength (size corrected)	
rock_prem	varchar(250)	Details of point load test	A
rock_urem	varchar(250)	Details on uniaxial compressive test	A
rock_e	i4	Elastic modulus	
rock_mu	f4	Poissons ratio	
rock_braz	i4	Tensile strength by the Brazilian method	
rock_brem	varchar(250)	Comments / Details on Brazilian method	A
rock_sdi	f4	Slake durability	
rock_srem	varchar(250)	Remarks / details of slake durability test	A
rock_poro	f4	Rock porosity	
rock_pore	varchar(250)	Notes on rock porosity test	A
rock_mc	f4	Natural moisture content	
rock_dden	f4	Rock dry Density (Mgm^{-3})	
rock_soun	f4	Soundness test	
rock_mrem	varchar(250)	Notes on rock soundness test	A

cmpg

Field Name	Field Type	Field Description	Remarks
proj_id	varchar(10)	Project / Site Investigation Code	K
hole_id	varchar(10)	Borehole / Trial Pit Code	K
samp_ref	i4	Sample reference number	K
spec_ref	i4	Specimen reference number	K
cmpg_rem	varchar(250)	Comments / Details	A
cmpg_type	varchar(30)	Compaction test type	
cmpg_mold	varchar(50)	Compaction mould type	
cmpg_375	i4	Weight pecent of material retained on 37.5mm sieve	
cmpg_200	i4	Weight pecent of material retained on 20mm sieve	
cmpg_pden	varchar(10)	Particle Density (Mgm^{-3}), measured or assumed	
cmpg_maxd	f4	Maximum Dry Density (Mgm^{-3})	
cmpg_mcop	f4	Moisture content at maximum Dry Density (Mgm^{-3})	

cmpt

Field Name	Field Type	Field Description	Remarks
proj_id	varchar(10)	Project / Site Investigation Code	K
hole_id	varchar(10)	Borehole / Trial Pit Code	K
samp_ref	i4	Sample reference number	K
spec_ref	i4	Specimen reference number	K
cmpt_tssn	i4	Compaction point number	
cmpt_mc	f4	Moisture content	
cmpt_dden	f4	Dry Density (Mgm^{-3}) at cmpt_mc Moisture content	

icbr

Field Name	Field Type	Field Description	Remarks
proj_id	varchar(10)	Project / Site Investigation Code	K
hole_id	varchar(10)	Borehole / Trial Pit Code	K
icbr_dpth	f4	Depth to top of CBR value	K
icbr_rem	varchar(250)	Comments / Details	A
icbr_icbr	f4	CBR value	
icbr_mc	f4	Moisture content relating to test	

iden

Field Name	Field Type	Field Description	Remarks
proj_id	varchar(10)	Project / Site Investigation Code	K
hole_id	varchar(10)	Borehole / Trial Pit Code	K
iden_dpth	f4	Depth of in situ Density (Mgm^{-3}) test	K
iden_rem	varchar(250)	Comments / Details	A
iden_iden	f4	In situ bulk Density (Mgm^{-3})	
iden_mc	f4	Moisture content relating to test	

iprm

Field Name	Field Type	Field Description	Remarks
proj_id	varchar(10)	Project / Site Investigation Code	K
hole_id	varchar(10)	Borehole / Trial Pit Code	K
iprm_base	f4	Depth to base of test zone	K
iprm_top	f4	Depth to top of test zone	
iprm_type	varchar(30)	Test type	
iprm_prwl	f4	Depth to water in borehole or piezometer prior to test	
iprm_swal	f4	Depth to water at start of test	
iprm_tdia	f4	Diameter of test zone	
iprm_sdia	f4	Diameter of standpipe or casing	
iprm_iprm	f8	Permeability	
iprm_rem	varchar(250)	Comments / Details	A

irdx

Field Name	Field Type	Field Description	Remarks
proj_id	varchar(10)	Project / Site Investigation Code	K
hole_id	varchar(10)	Borehole / Trial Pit Code	K
irdx_dpth	f4	Depth of redox test	K
irdx_rem	varchar(250)	Comments / Details	A
irdx_ph	f4	pH	
irdx_irdx	i4	Redox potential	

ires

Field Name	Field Type	Field Description	Remarks
proj_id	varchar(10)	Project / Site Investigation Code	K
hole_id	varchar(10)	Borehole / Trial Pit Code	K
ires_dpth	f4	Depth range to which in situ resitivity test relates	K
ires_type	varchar(30)	Test type	
ires_ires	i4	Resitivity	
ires_rem	varchar(250)	Comments / Details	A

mcvg

Field Name	Field Type	Field Description	Remarks
proj_id	varchar(10)	Project / Site Investigation Code	K
hole_id	varchar(10)	Borehole / Trial Pit Code	K
samp_ref	i4	Sample reference number	K
spec_ref	i4	Specimen reference number	K
mcvg_rem	varchar(250)	Comments / Details	A
mcvg_200	i4	Weight percent of material retained on 20mm sieve	
mcvg_nmc	i4	Natural Moisture content	
mcvg_prcl	varchar(20)	MCV precalibrated value, and indication of higher or lower	

mcvt

Field Name	Field Type	Field Description	Remarks
proj_id	varchar(10)	Project / Site Investigation Code	K
hole_id	varchar(10)	Borehole / Trial Pit Code	K
samp_ref	i4	Sample reference number	K
spec_ref	i4	Specimen reference number	K
mcvt_tesn	i4	MCV test number	
mcvt_mc	f4	MC	
mcvt_relk	f4	MCV value at mcvt_mc Moisture content	
mcvt_bden	f4	Bulk Density (Mgm^{-3}) related to the mcvt_relk MCV	

conc

Field Name	Field Type	Field Description	Remarks
proj_id	varchar(10)	Project / Site Investigation Code	K
hole_id	varchar(10)	Borehole / Trial Pit Code	K
conc_ref	varchar(10)	Cone Identification Reference	K
conc_x	f4	X coordinate on Calibration Curve	
conc_y	f4	Y coordinate on Calibration Curve	
conc_date	varchar(15)	Date of calibration	

stcn

Field Name	Field Type	Field Description	Remarks
proj_id	varchar(10)	Project / Site Investigation Code	K
hole_id	varchar(10)	Borehole / Trial Pit Code	K
stcn_dpth	f4	Depth of result for static cone test	
stcn_forc	f4	Axial force	
stcn_fric	f4	Friction force on sleeve	
stcn_res	f4	Cone resistance	
stcn_fres	f4	Load unit side friction resistance	
stcn_pwp1	f4	Porewater pressure	
stcn_typ	varchar(30)	Cone test type	
stcn_ref	varchar(10)	Cone Identification Reference	
stcn_inc	i4	Cone inclination from vertical	
stcn_con	f4	Conductivity	
stcn_pwp2	f4	Second porewater pressure	
stcn_temp	i4	Temperature	

shbg

Field Name	Field Type	Field Description	Remarks
proj_id	varchar(10)	Project / Site Investigation Code	K
hole_id	varchar(10)	Borehole / Trial Pit Code	K
samp_ref	i4	Sample reference number	K
spec_ref	i4	Specimen reference number	K
shbg_type	varchar(50)	Test type	
shbg_rem	varchar(250)	Comments / Details	A
shbg_pcoh	f4	Peak cohesion intercept	
shbg_phi	f4	Peak angle of friction	
shbg_rcoh	f4	Residual cohesion intercept	
shbg_rphi	f4	Residual angle of friction	

shbt

Field Name	Field Type	Field Description	Remarks
proj_id	varchar(10)	Project / Site Investigation Code	K
hole_id	varchar(10)	Borehole / Trial Pit Code	K
samp_ref	i4	Sample reference number	K
spec_ref	i4	Specimen reference number	K
shbt_tesn	i4	Shear box stage number	
shbt_mc	f4	Specimen initail Moisture content	
shbt_bden	f4	Bulk Density (Mgm^{-3})	
shbt_dden	f4	Dry Density (Mgm^{-3})	
shbt_norm	i4	Shear box normal stress	
shbt_disp	f4	Displacement rate	
shbt_peak	f4	Shear box peak shear stress	
shbt_res	f4	Shear box residual shear stress	
shbt_pdis	f4	Displacement at peak shear stress	
shbt_rdis	f4	Displacement at residual shear stress	
shbt_pden	f4	Particle Density (Mgm^{-3})	
shbt_ivr	f4	Initail voids ratio	
shbt_mci	f4	Initail Moisture content	
shbt_mcf	f4	Final Moisture content	

ivan

Field Name	Field Type	Field Description	Remarks
proj_id	varchar(10)	Project / Site Investigation Code	K
hole_id	varchar(10)	Borehole / Trial Pit Code	K
ivan_dpth	f4	Depth of vane test	K
ivan_rem	varchar(100)	Comments / Details	A
ivan_ivan	f4	Vane test result	
ivan_ivar	f4	Vane test result remoulded	

tsrf

Field Name	Field Type	Field Description	Remarks
test_name	varchar(200)	Full name of test	
test_code	varchar(20)	Test code	K
test_ref	varchar(20)	Reference for test	
test_det	varchar(300)	Details of test	A
test_tble	varchar(4)	Table of test	

reld

Field Name	Field Type	Field Description	Remarks
proj_id	varchar(10)	Project / Site Investigation Code	K
hole_id	varchar(10)	Borehole / Trial Pit Code	K
samp_ref	i4	Sample reference number	K
spec_ref	i4	Specimen reference number	K
reld_rem	varchar(250)	Comments / Details	A
reld_dmax	f4	Maximin Dry Density (Mgm^{-3})	
reld_375	f4	Weight percent of sample retained on 37.5mm sieve	
reld_063	f4	Weight percent of sample retained on 6.3mm sieve	
reld_020	f4	Weight percent of sample retained on 2mm sieve	
reld_dmin	f4	Minimum Dry Density (Mgm^{-3})	

ptst

Field Name	Field Type	Field Description	Remarks
proj_id	varchar(10)	Project / Site Investigation Code	K
hole_id	varchar(10)	Borehole / Trial Pit Code	K
samp_ref	i4	Sample reference number	K
spec_ref	i4	Specimen reference number	K
ptst_tesn	i4	Permiability test number	
ptst_rem	varchar(250)	Comments / Details	A
ptst_cond	varchar(100)	Sample condition	
ptst_szun	f4	Size cut off material too coarse for testing	
ptst_uns	f4	Proportion of material too coarse for testing	
ptst_dia	f4	Diameter of test sample	
ptst_len	f4	Length of test sample	
ptst_mc	f4	Initail Moisture content of test sample	
ptst_bden	f4	Initial bulk Density (Mgm^{-3}) of test sample	
ptst_dden	f4	Dry Density (Mgm^{-3}) of test sample	
ptst_void	f4	Voids ration of test sample	
ptst_k	f4	Coefficient of test sample	
ptst_tstr	f4	Mean effective stress at which permeability measured	
ptst_isat	f4	Inital degree of saturation	
ptst_fsat	f4	Final degree of saturation	
ptst_pden	f4	Particle Density (Mgm^{-3}), measured or assumed	

isrf

Field Name	Field Type	Field Description	Remarks
proj_id	varchar(10)	Project / Site Investigation Code	K
hole_id	varchar(10)	Borehole / Trial Pit Code	K
test_top	f4	Depth to top of test	K
test_code	varchar(20)	Code of test used	

lbrf

Field Name	Field Type	Field Description	Remarks
proj_id	varchar(10)	Project / Site Investigation Code	K
hole_id	varchar(10)	Borehole / Trial Pit Code	K
samp_ref	i2	Sample reference number	K
spec_ref	i2	Specimen reference number	K
test_code	varchar(20)	Test code	

Appendix 2

Percent Lists utilised in the Vasic methodology

Coarse Grained Soils

Amount-list	Percent-list		Amount-list	Percent-list
[1, 0, 0, 0]	[100, 0, 0, 0]		[1, 2, 1, 0]	[65, 15, 5, 0]
[2, 0, 0, 0]	[50, 0, 0, 0]		[1, 1, 2, 0]	[65, 15, 10, 0]
[1, 1, 0, 0]	[75, 25, 0, 0]		[1, 1, 0, 2]	[69, 25, 0, 3]
[1, 0, 1, 0]	[90, 0, 10, 0]		[1, 2, 0, 1]	[66, 16, 0, 2]
[1, 0, 0, 1]	[97, 0, 0, 3]		[1, 0, 1, 2]	[84, 0, 10, 3]
[1, 1, 1, 0]	[65, 25, 10, 0]		[1, 0, 2, 1]	[77, 0, 10, 3]
[1, 1, 0, 1]	[72, 25, 0, 3]		[1, 0, 0, 3]	[91, 0, 0, 3]
[1, 0, 1, 1]	[87, 0, 10, 3]		[1, 0, 3, 0]	[70, 0, 10, 0]
[1, 1, 1, 1]	[65, 23, 10, 2]		[1, 3, 0, 0]	[40, 20, 0, 0]
[1, 0, 0, 2]	[94, 0, 0, 3]		[2, 0, 0, 1]	[48, 0, 0, 4]
[1, 0, 2, 0]	[80, 0, 10, 0]		[2, 0, 1, 0]	[45, 0, 10, 0]
[1, 2, 0, 0]	[66, 17, 0, 0]		[2, 1, 0, 0]	[37, 26, 0, 0]

Fine Grained Soils

Amount-list	Percent-list		Amount-list	Percent-list
[1, 0, 0, 0]	[100, 0, 0, 0]		[1, 0, 0, 2]	[50, 0, 0, 25]
[2, 0, 0, 0]	[50, 0, 0, 0]		[1, 0, 1, 1]	[40, 0, 35, 25]
[1, 1, 0, 0]	[35, 65, 0, 0]		[1, 0, 2, 0]	[40, 0, 30, 0]
[1, 0, 1, 0]	[50, 0, 50, 0]		[1, 2, 0, 0]	[36, 32, 0, 0]
[1, 0, 0, 1]	[65, 0, 0, 35]		[1, 0, 1, 2]	[40, 0, 30, 15]
[1, 1, 1, 0]	[35, 40, 25, 0]		[1, 1, 0, 2]	[35, 35, 0, 15]
[1, 1, 0, 1]	[35, 45, 0, 20]		[1, 0, 2, 1]	[35, 0, 25, 15]
[1, 1, 2, 0]	[35, 25, 20, 0]		[1, 2, 0, 1]	[35, 30, 0, 5]
[2, 0, 0, 1]	[40, 0, 0, 20]		[1, 2, 1, 0]	[35, 25, 15, 0]
[2, 0, 1, 0]	[35, 0, 30, 0]		[1, 1, 1, 1]	[35, 35, 20, 10]
[2, 1, 0, 0]	[35, 30, 0, 0]			

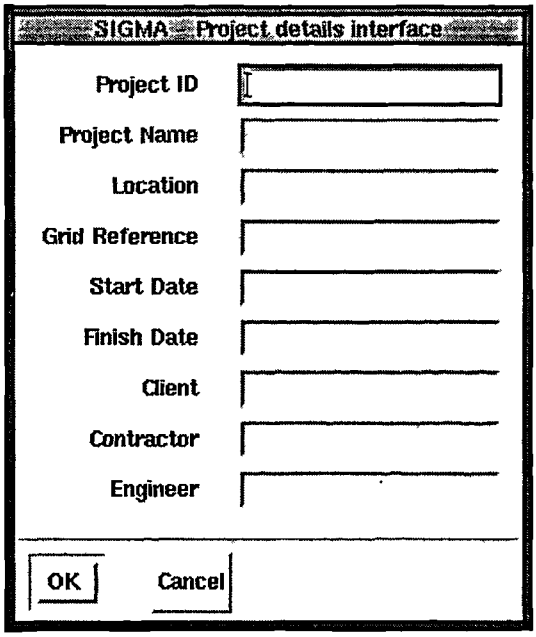
Appendix 3

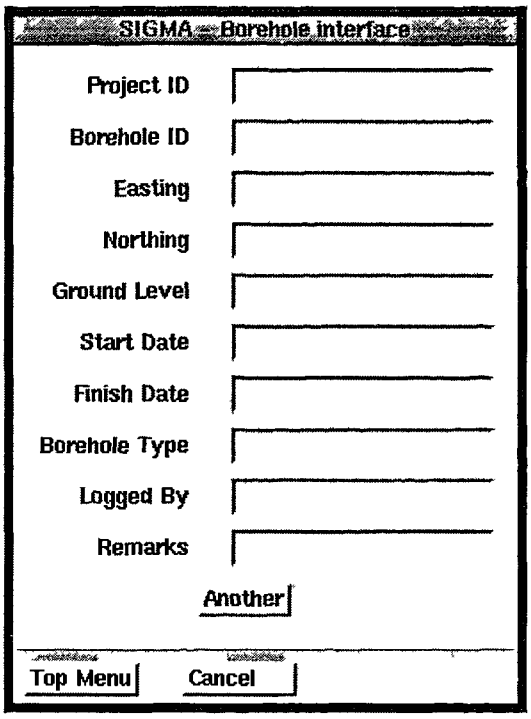
Example of SQL script files used for the generation and alteration of the GeoTec database

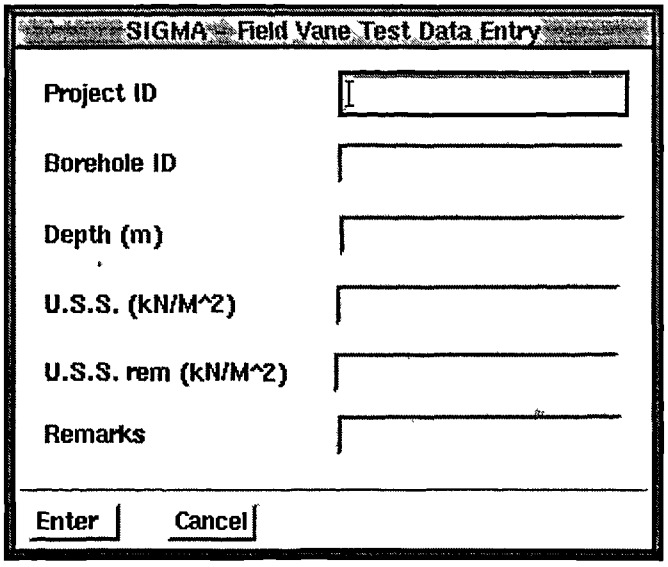
```
create table ctcl
(
proj_id          varchar(10),
hole_id          varchar(10),
lay_no           i2,
strt_no          i2,
cons_no          i2
ctcl_clno        i2
ctcl_mcl1        varchar(30),
ctcl_mcl2        varchar(30),
ctcl_scl         varchar(30),
ctcl_mod         varchar(30),
ctcl_strc        varchar(30)
)
;

create table lbrf
(
proj_id          varchar(10),
hole_id          varchar(10),
samp_ref         i2,
test_code        varchar(20)
) with noduplicates
;

create table isrf
(
proj_id          varchar(10),
hole_id          varchar(10),
samp_top         float4,
test_code        varchar(20)
) with noduplicates
;

create table tstr
(
test_name        varchar(200),
test_code        varchar(20),
test_ref         varchar(20),
test_det         varchar(300),
test_tble        varchar(4)
) with noduplicates
```

Appendix 4

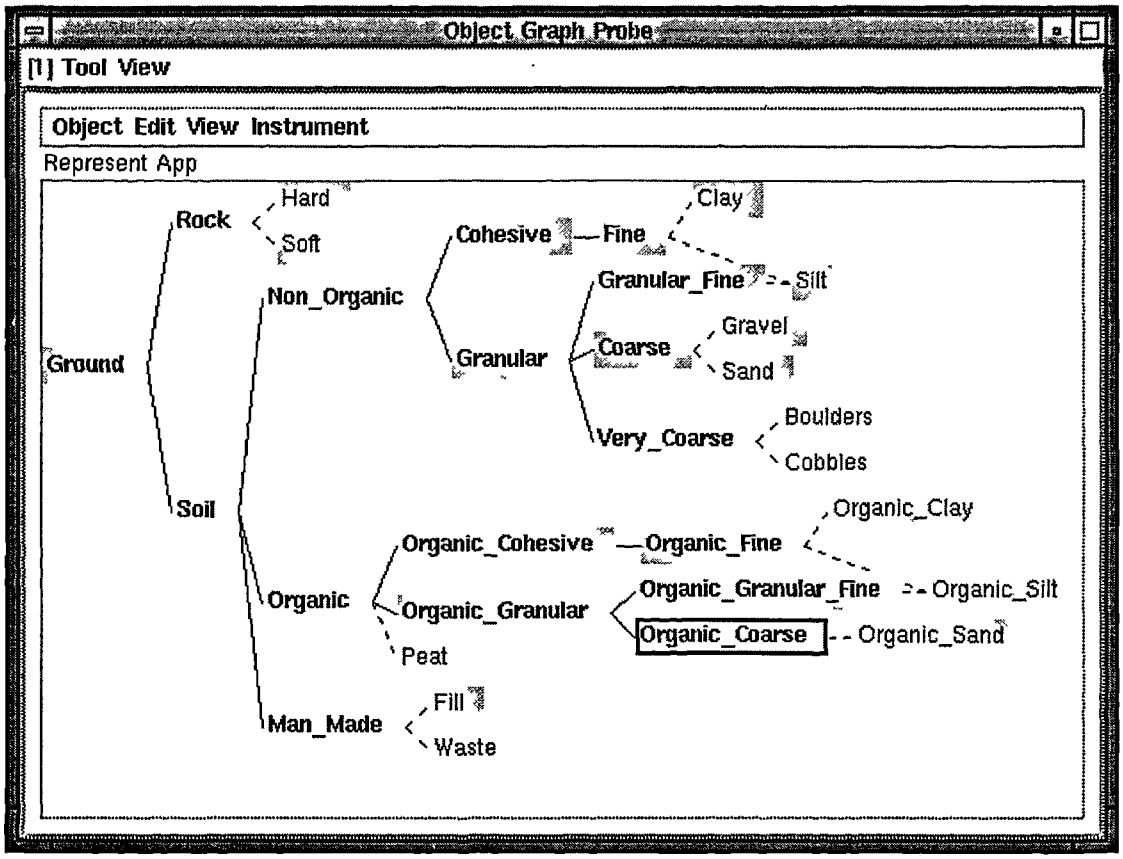
Examples of GeoTec data entry interfaces via SIGMA



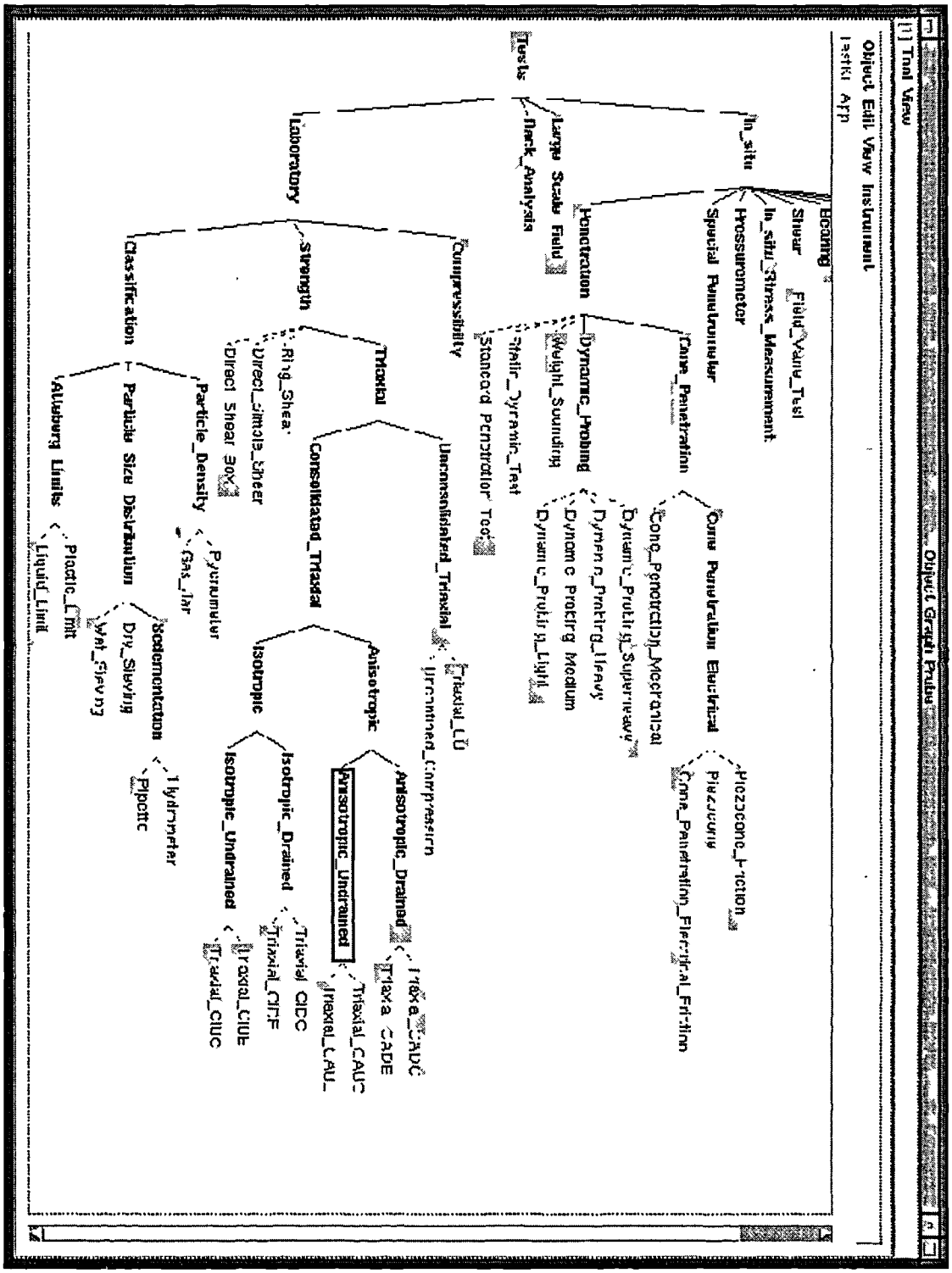




Appendix 5
Ground Knowledge Base (Moula, 1993)



Appendix 6
Test Knowledge Base



Appendix 7

Data Access Slots available on a Source class object in the mapping domain

AdditionalWhereString	Utilised in generating the SQL. Described fully in Section 6.4.3
AfterRowProcessed!	User defined method that allows immediate post processing of data on import
Class	Domain class object associated with the table
ComputeDeleteString!	System method to compute the Delete! method
ComputeInsertString!	System method to compute the Insert! method
ComputeSQLString!	System method to compute the SQL string for the Get! method
ComputeUpdateString!	System method to compute the Update! method
ComputeUID!	System method to compute column values for UID
ComputeUIDName!	System method to compute the UID name
Connection	Name of connection object for current database session
ConnectionType	System database specific connection type object
Delete!	System method for deletion of database table rows
DeleteInstances!	System method to delete all instances of a domain class object
FinishSource!	Generates and caches SQL string
Get!	System data retrieval method
Insert!	System data insert method
InstanceModule	Specifies the module in which the domain instance objects are created
Joins	Lists joins between data tables and how they are joined
NumberOfRowsProcessed	Automatically filled with number of instances processed on either Get!, Insert!, Update! or Delete!
PrimaryTable	Primary table object and alias
SendSQL!	Sends arbitary SQL to the database
SlotMaps	Lists slot maps for source mapping
Tables	Lists tables for source mapping
UIDColumns	Lists columns that make up the UID
Update!	System update method

Appendix 8

ProTalk code written for SIGMA

```
#include<prk/lib.pth>


/* parser_sys.ptk */
/* */
/* Contains the system files to run the soil description parser.  All other*/
/* functions contained in parser_meth, _func and _comps*/

/* SIGMA - System for the Interpretation of site investiGation inforMAtion*/
/* University of Durham, GeoTechnical Systems Group*/

/* Andy Oliver - June 1994*/


function initial(?desc)
{
/*  Main initialiser function.  Places the description into the PartSD slot*/
/*  on SoilDes for begining of run and creates the first Soil class and Kalker*/

   SoilDes.PartSD = check_desc(?desc);
   ?soil_name = ConvertToSymbol("Soil1");
   MakeClass(?soil_name, PSD, SoilDes, `(soil_no, soil_cons, soil_mscd, soil_mstp,
                                          soil_psty, soil_strc, soil_wthg));
   set_slt(?soil_name); /* Set the correct inheritence type for Soil1 */
   MakeClass(Kalker, PSD, SoilDes,
            `(ModKalk, PosiKalk, range, hit ,SoilCnt, SoilTypeCnt,
                SoilStruCnt, ColCnt,
               Shape, Texture, Grading, Structure, Spacing, WithFlag, Distribution));
/* Initialise all Kalker slots */
   Kalker.SoilStruCnt = 1;
   create_Kalker_sub();
   Kalker.WithFlag = "Disabled";   Kalker.range = "";   Kalker.ModKalk = "";
   Kalker.Grading = "";   Kalker.Distribution = "";   Kalker.Structure = "";
   Kalker.Spacing = "";   Kalker.SoilCnt = 1;   Soil1.soil_no=1;
   Kalker.SoilTypeCnt = 0;   Kalker.ColCnt = 1;   Kalker.hit = "";
   exe1.Values = Null;   ex11.Values = Null;
}

function check_desc(?desc)
{
/* Ensures that the last element of description is a string.  This is */
/* how the parser knows it has reached the end of the description */
   bound inputs;
   if Substring(?desc, StringLength(?desc)-1, StringLength(?desc)) != " ";
   then ?desc = AppendStrings(?desc," ");
```

```
   return ?desc;
}

function create_Kalker_sub()
{
 /* Sets up the initial sub classes for Kalker, that is Structure and Colour */

   MakeClass(KalkCol1, PSD, Kalker, `(slcl_clno, slcl_mcl1, slcl_mcl2,
                                      slcl_mod, slcl_scl, slcl_strc));
   ?kst_name = ConvertToSymbol(AppendStrings("KalkStr",
                                              ConvertToString(Kalker.SoilStruCnt)));

   MakeClass(?kst_name, PSD, Kalker, `(slst_stct, slst_spc, slst_dcon,
                                      slst_dip, slst_ornt, slst_srf, slst_stno));

   ?kst_name.slst_stno = Kalker.SoilStruCnt;

 }

 function set_slt(?soil_name)
 {
  /* Sets Soil inheritence types. C: construct is a way of embedding C code in ProTalk */
    ?soil_name = ConvertToSymbol(?soil_name);
  C:PrkSetSlotType(?soil_name, `soil_cons, 3);
  C:PrkSetSlotType(?soil_name, `soil_mscd, 3);
  C:PrkSetSlotType(?soil_name, `soil_psty, 3);
  C:PrkSetSlotType(?soil_name, `soil_wthg, 3);
  C:PrkSetSlotType(?soil_name, `soil_strc, 3);
  C:PrkSetSlotType(?soil_name, `soil_mstp, 3);
    ?soil_name.soil_cons = "";
 }



function set_slots(?str_name)
{
 /* Sets inheritence types. C: construct is a way of embedding C code in ProTalk */
   bound inputs;
  C:PrkSetSlotType(?str_name, `sldt_shp, 3);
  C:PrkSetSlotType(?str_name, `sldt_dstb, 3);
  C:PrkSetSlotType(?str_name, `sldt_amnt, 3);
  C:PrkSetSlotType(?str_name, `sldt_type, 3);
  C:PrkSetSlotType(?str_name, `sldt_grdg, 3);
  C:PrkSetSlotType(?str_name, `sldt_txt, 3);
 }
```

```
function class_init(?st_type)
{
 /* Initialises soil_type slots */
   ?st_type = ConvertToSymbol(?st_type);
   ?st_type.sldt_shp = "";
   ?st_type.sldt_txt = "";
}

function translate()
{
 /* Sets the shape, texture,grading and distribution slots on the newly created */
 /* constituent level */

   select
   {
    case: Kalker.Shape != "";
         {
            ?st_name = AppendStrings("Soil", ConvertToString(Kalker.SoilCnt), "St",
                                     ConvertToString(Kalker.SoilTypeCnt));
            ?st_name = ConvertToSymbol(?st_name);
            ?st_name.sldt_shp = Kalker.Shape;
            Kalker.Shape = "";
         }
    case: Kalker.Texture != "";
         {
            ?st_name = AppendStrings("Soil", ConvertToString(Kalker.SoilCnt), "St",
                                     ConvertToString(Kalker.SoilTypeCnt));
            ?st_name = ConvertToSymbol(?st_name);
            ?st_name.sldt_txt = Kalker.Texture;
            Kalker.Texture = "";
         }

    case: Kalker.Grading != "";
         {
            ?st_name = AppendStrings("Soil", ConvertToString(Kalker.SoilCnt), "St",
                                     ConvertToString(Kalker.SoilTypeCnt));
            ?st_name = ConvertToSymbol(?st_name);
            Kalker.Grading = ConvertToString(Kalker.Grading);
            if Substring(Kalker.Grading, StringLength(Kalker.Grading)-2,
                         StringLength(Kalker.Grading)) == ", ";
            then Kalker.Grading = Substring(Kalker.Grading, 0,
                                  StringLength(Kalker.Grading)-2);
            ?st_name.sldt_grdg = Kalker.Grading;
            Kalker.Grading = "";
```

```
        }

    case: Kalker.Distribution != "";
        {
          ?st_name = AppendStrings("Soil", ConvertToString(Kalker.SoilCnt), "St",
                                   ConvertToString(Kalker.SoilTypeCnt));
          ?st_name = ConvertToSymbol(?st_name);
          Kalker.Distribution = ConvertToString(Kalker.Distribution);
          if Substring(Kalker.Distribution, StringLength(Kalker.Distribution)-2,
                       StringLength(Kalker.Distribution)) == ", ";
          then Kalker.Distribution = Substring(Kalker.Distribution, 0,
                                               StringLength(Kalker.Distribution)-2);
          ?st_name.sldt_dstb = Kalker.Distribution;
          Kalker.Distribution = "";
        }

    }
}


function check_col(?word)
{
/* Checks coolour against the relevant vocabularly list. Seperate function due to */
/* the complexity of colour function. Called several times so seperated as */
/* distinct function to save duplication*/

   ?word = ConvertToString(?word);
   ?chk_flag = 0;

   ?chk = FindListElmt(all colour.comparison_list, ?word);

   if ?chk != `();
   then ?chk_flag = 1;

   return ?chk_flag;
}


function remover(?starter)
{
   bound inputs;
/* Generic function that removes hierarchies prior to starting processing, removes */
/* all subclasses below object passed as ?starter */
   ?name = find direct subclassof ?starter;
   for IsObject(?name);
```

```
    do
    {
        for ?name2 = find direct subclassof ?name;
        do {
          for IsObject(?name2);
          do {
              for ?name3 = find direct subclassof ?name2;
              do {
                    DeleteObject(?name3);
                    fail;
                  }
              DeleteObject(?name2);
              fail;
          }
        }
    }
    DeleteObject(?name);
    remover(?starter);
}


function check_str(?kst_name)
{
 /* Checks to see if new structure subclass of Kalker is required, utilising the */
 /* structure counter on Kalker itself */

    if not IsObject(FindObject(?kst_name));
    MakeClass(?kst_name, PSD, Kalker, `(slst_stct, slst_spc, slst_dcon,
                                        slst_dip, slst_ornt, slst_srf, slst_stno));
    ?kst_name.slst_stno = Kalker.SoilStruCnt;
}


function conv_kalk_sub()
{
 /* Converts the colour and structure subclasses of kalker to their appropriate */
 /* form for transfer to GeoTec.  Called when either a Strata or parse is complete*/

    ?soil_name = ConvertToSymbol(AppendStrings
                        ("Soil", ConvertToString(Kalker.SoilCnt)));
    ?st_name = AppendStrings(ConvertToString(?soil_name),
                            "St", ConvertToString(Kalker.SoilTypeCnt));

    for ?n = find direct subclassof Kalker;
    do {
```

```
        select
        {
         case:Substring(?n, 4, 5) == "S";
           {
                ?str_num = Substring(?n, 7, 8);
                ?str_name = ConvertToSymbol(AppendStrings(ConvertToString
                                                (?soil_name),"Str", ?str_num));
                if ?n.slst_stct != Null;
                then {
                   ?slot_list = `(slst_stct, slst_spc, slst_dcon,
                                  slst_dip, slst_ornt, slst_srf, slst_stno);
                   MakeClass(?str_name, PSD, ?soil_name, ?slot_list);

                   for ?list_mem inlist ?slot_list;
                   do ?str_name.?list_mem = ?n.?list_mem;
                   fail;
                }
           }
         case:Substring(?n, 4, 5) =="C";
                   {
                ?col_num = Substring(?n, 7, 8);
                ?col_name=ConvertToSymbol(AppendStrings
                                                (ConvertToString(?st_name),
                                                "C", ?col_num));
                if ?n.slcl_mcl1 != Null;
                then {
                   ?slot_list = `(slcl_clno, slcl_mcl1, slcl_mcl2,
                                  slcl_mod, slcl_scl, slcl_strc);
                   MakeClass(?col_name, PSD, ?st_name, ?slot_list);
                   for ?list_mem inlist ?slot_list;
                   do ?col_name.?list_mem = ?n.?list_mem;
                   fail;
                }
           }
        }
   }
   remover(Kalker);
   Kalker.ColCnt = 1;
}


function placer(?insert_list)
{
/* Takes the finished parsed model, makes the relevant instances and places them */
/* in the GeoTec database.  Called from export react*/
```

```
bound inputs;
for ?soil inlist `(find direct subclassof SoilDes@);
do {
    if Substring(ConvertToString(?soil),0,4) == "Soil";
    then {
      ?soil_name = soil_Source.ComputeUIDName!(`(soil@), `(?soil.proj_id,
                                                          ?soil.hole_id,
                                                          ?soil.lay_no,
                                                          ?soil.soil_no));
      MakeInstance(?soil_name, Sig93D, soil@);
      fill_inst(?soil_name, ?soil);
      for ?elmt inlist `(find direct subclassof ?soil);
      do {
          if StringLength(ConvertToString(?elmt)) == 8;
          then {
                  ?inst_name = sldt_Source.ComputeUIDName!(`(sldt@),
                                                          `(?elmt.proj_id,
                                                           ?elmt.hole_id,
                                                           ?elmt.lay_no,
                                                           ?elmt.soil_no,
                                                           ?elmt.cons_no));
            MakeInstance(?inst_name, Sig93D, sldt@);
            fill_inst(?inst_name, ?elmt);
          }
          else {
                  ?str_name = slst_Source.ComputeUIDName!(`(slst@),
                                                          `(?elmt.proj_id,
                                                          ?elmt.hole_id,
                                                          ?elmt.lay_no,
                                                          ?elmt.soil_no,
                                                          ?elmt.slst_stno));
            MakeInstance(?str_name, Sig93D, slst@);
            fill_inst(?str_name, ?elmt);
          }
          for ?cs inlist all subclassof ?elmt;
          do {
                  ?col_name = slcl_Source.ComputeUIDName!(`(slcl@),
                                                          `(?cs.proj_id,
                                                          ?cs.hole_id,
                                                          ?cs.lay_no,
                                                          ?cs.soil_no,
                                                          ?cs.cons_no,
                                                          ?cs.slcl_clno));
            MakeInstance(?col_name, Sig93D, slcl@);
```

```
                    fill_inst(?col_name, ?cs);
                }
            }
        }

    }

    for ?ins_obj inlist ?insert_list; /* uses insert_list to determine those to be */
                                      /* placed*/
    do {
        for ?insert inlist all instanceof ?ins_obj;
        do {
            ?source = ConvertToSymbol(AppendStrings(?ins_obj,"_Source"));
            ?source.Insert!(FindObject(?insert));
            ?source.SendSQL!("commit");
            DeleteObject(?insert);
        }
    }
}


function fill_inst(?inst_name, ?sub_name)
{
 /* automatically transfers the data, that is the slot values, into the appropriate */
 /* slots on the SIG93D instances.*/
    bound inputs;
    for ?slt inlist ObjectSlots(?sub_name);
    do ?inst_name.?slt = ?sub_name.?slt;
}
```

```
#include <prk/lib.pth>

/* parser_func.ptk */
/* */
/* Contains the non-standard soil identification and main structure clauses */
/* for the parser, as well as the main loop function soil_parse1*/
/* All other functions contained in parser_meth, _sys and _comps*/

/* SIGMA - System for the Interpretation of site investiGation inforMAtion*/
/* University of Durham, GeoTechnical Systems Group*/

/* Andy Oliver - June 1994*/



function starter(?desc)
{
/*  called by  main React to run parser*/
   remover(SoilDes); /* clears working area to initial state */
   initial(?desc); /* initialises domain and kalker variables */
   soil_parse1(); /* main parsing loop function */

}


function soil_parse1()
{
/* main parsing loop function */
   ?desc = ConvertToString(SoilDes.PartSD);

   extract(?desc, ?fs, ?ls); /* extracts the next term to be parsed */

   SoilDes.PartSD = ?ls; /* sets remainder of description for next pass */

   SetDialogBoxControlValue(parse1, St6, Values, AppendStrings("Currently parsing ",
                                                    ConvertToString(?fs)));
   Print("Parsing", ?fs, "...\n");

   for ?i inlist all subclassof parser; /* non-deterministic call to all subclasses of */
                                        /* parser that passes the term bound to ?fs to */
                                        /* all of the clauses in turn*/
   do ?i.parser_clause!(?fs, ?ls);

   Kalker.ModKalk = "";
   if Kalker.hit == ""; /* if no clauses identified the term, then start exception */
```

```
                        /* handling routines*/
   then {
        exl1.SelectionItems +== ?fs;
        SoilDes.except_flag = 1;
        PrintLine(?fs," is unknown to the parser");
        SetDialogBoxControlValue(parse1, St6, Values,
                                        AppendStrings("Unknown term", ?fs));
   }
   Kalker.hit = "";
   if StringLength(?ls)>0;
   then soil_parse1();
   else { /* description completed */
        conv_kalk_sub();
        DeleteObject(Kalker);
        if ListLength(all exl1.SelectionItems) > 0; /* check for exception handling */
        then SetDialogBoxControlValue(parse1, St6, Values,
                                        "Complete exception terms\nbefore continuing");
        else SetDialogBoxControlValue(parse1, St6, Values, "Data parsing completed.");
   }
}


method soil_type.parser_clause!(?word, ?ls)
{
/* Identifies main, secondary, major and minor constituents*/

   ?word = ConvertToString(?word);
   if ?word == "very";
        then
        {
          ?amount = "major";
          extract(?ls, ?word2, ?rest);
          ?test = sec_st(?word2, ?amount);
          if ?test == "pass";
          then SoilDes.PartSD = ?rest;
        }

   if ?word == "slightly";
        then
        {
          ?amount = "minor";
          extract(?ls, ?word2, ?rest);
          ?test = sec_st(?word2, ?amount);
          if ?test == "pass";
          then SoilDes.PartSD = ?rest;
```

```
        }

    ?test = sec_st(?word, "secondary");          /* tests for secondary */
    main_st(?word); /* tests for main */
}


function main_st(?word)
{
/* Identifies the main soil type. Due to the implemenataion of With, this is */
/* a 2 stage process depending on current whereabouts in the description.  If With */
/* has already been identified the main object exists, if not it requires creating*/

    ?soil_name = AppendStrings("Soil", ConvertToString(Kalker.SoilCnt));
    ?chk = FindListElmt(all soil_type.comparison_list, ?word);

    if ?chk != `();
    then
    {
        select
        {
         case:ConvertToString(`Kalker.WithFlag) == "Enabled";
           {
                Kalker.WithFlag = "Disabled";
                ?st_name = AppendStrings(?soil_name, "St",
                                          ConvertToString(Kalker.SoilTypeCnt));
                ?st_name = ConvertToSymbol(?st_name);
                ?st_name.sldt_type = ?word; check_main(?soil_name, ?word);
                Kalker.hit = "hit";
                conv_kalk_sub(); translate();
                create_Kalker_sub();
           }
         case:ConvertToString(`Kalker.WithFlag) == "Disabled";
           {
                Kalker.SoilTypeCnt = Kalker.SoilTypeCnt+1;
                ?st_name = AppendStrings(?soil_name, "St",
                                          ConvertToString(Kalker.SoilTypeCnt));
                MakeClass(?st_name, PSD, ?soil_name, `(sldt_type, sldt_amnt, soil_no,
                                                  cons_no, sldt_shp, sldt_grdg,
                                                  sldt_txt, sldt_dstb));
                set_slots(?st_name);
                class_init(?st_name);
                ?st_name = ConvertToSymbol(?st_name);
                ?st_name.sldt_type = ?word; check_main(?soil_name, ?word);
                ?st_name.sldt_amnt="main";
```

```
                ?st_name.cons_no = Kalker.SoilTypeCnt;
                Kalker.hit = "hit";
                conv_kalk_sub(); translate();
                create_Kalker_sub();
            }
        }
    }

}


function check_main(?soil_name, ?word)
{
 /* Enables two main soil types to be handled, ie Sand and Gravel [sand / gravel] */

   ?soil_name = ConvertToSymbol(?soil_name);
    if ?soil_name.soil_mstp == Null;
    then ?soil_name.soil_mstp = ?word;
                                          else          ?soil_name.soil_mstp          =
AppendStrings(?soil_name.soil_mstp,"/",ConvertToString(?word));
}


function sec_st(?word, ?amount)
{
 /* identifies secondary constituents and creates the appropriate objects where */
 /* required.  Operates on a test basis as it is called from soil_type.parser_clause*/

    ?ans = "fail";
    ?soil_name = AppendStrings("Soil", ConvertToString(Kalker.SoilCnt));
    ?word_len = StringLength(?word);
    if Substring(?word, ?word_len-1, ?word_len) == "y";
    then
    {
          ?test = FindSubstring(?word, "ey");
          if ?test<0;
          then ?test = FindSubstring(?word, "ly");
          if ?test<0;
          then ?test = ?word_len-1;
          ?word = Substring(?word, 0, ?test);

          ?chk = FindListElmt(all soil_type.comparison_list, ?word);

          if ?chk != `();
          then
```

```
        {

            Kalker.SoilTypeCnt = Kalker.SoilTypeCnt+1;
            ?st_name = AppendStrings(ConvertToString(?soil_name), "St",
                                     ConvertToString(Kalker.SoilTypeCnt));
                MakeClass(?st_name, PSD, ?soil_name, `(sldt_type, sldt_amnt, soil_no,
cons_no,
                                                    sldt_shp, sldt_grdg, sldt_txt, sldt_dstb));
            Kalker.hit ="hit";
            set_slots(?st_name);
            class_init(?st_name);
            ?st_name = ConvertToSymbol(?st_name);
            ?st_name.sldt_type=?word; ?st_name.sldt_amnt=?amount;
            ?st_name.cons_no = Kalker.SoilTypeCnt;
            ?ans = "pass";

        }
    }
    return ?ans;
}


function extract(?desc, ?fs, ?ls)
{
 /* main extract clause, utilsed from soil_parse1 and whichever multi-term clauses */
 /* that require their own extraction */

    ?first = FindSubstring(?desc, " ");
    ?last = StringLength(?desc);
    ?fs = Substring(?desc, 0, ?first);
    ?ls = Substring(?desc, ?first+1, ?last);
}


function modify(?word, ?ls)
{
 /* tests for modifiers for all parser clauses */
    ?word = ConvertToString(?word);
    for ?comp_var inlist `("very", "medium", "slightly", "low",
                           "high", "intermediate", "extremely");
    do {
        if ?word == ?comp_var;
        then {
            extract(?ls, ?word2, ?rest);
            ?word = ConvertToString(?word2);
```

```
            SoilDes.PartSD = ?rest;
            ?ls = ?rest;
            Kalker.ModKalk = AppendStrings(?comp_var, " ");
          }
    }
    return ?word;
}


method with_it.parser_clause!(?word, ?ls)
{
/* Identifies with and its related phraseology.  Requires up to 3 terms to */
/* be extracted to identify the full meaning of the phrase*/

    ?soil_name = AppendStrings("Soil", ConvertToString(Kalker.SoilCnt));
    if ?word == "with";
    {     extract(?ls, ?word2, ?rest);
          ?word2 = ConvertToString(?word2);
          select
          {
             case: ListLength(FindListElmt(all inclusion.comparison_list, ?word2)) > 0;
               {
                    make_new_soil("dominant", ?rest, ?word2);
                    fail;
               }
             case:?word2 == "some";
               {?amount = "secondary";?word =  ?word2;}
             case:?word2 == "few";
               {?amount = "secondary";?word = ?word2; }
             case:?word2 == "occasional";
               {?amount = "minor";?word = ?word2; }
             case:?word2 == "numerous";
               {?amount = "major";?word = ?word2; }
             case:?word2 == "many";
               {?amount = "major";?word = ?word2; }
             case:?word2 == "frequent";
               {?amount = "major";?word = ?word2;}
             case:?word2 == "a";
               {
                    extract(?rest, ?word3, ?rest1);
                    select
                    {
                     case:?word3 =="little";
                       {
                            ?amount = "minor";
```

```
                    ?rest = ?rest1;
                    ?word = ?word3;
                }
              case:?word3 =="trace";
                {
                    extract(?rest, ?word4, ?rest2);
                    ?amount = "minor";
                    ?rest = ?rest2;
                    ?word = ?word4;
                }
            }
        }
       otherwise:{Kalker.hit = "hit";return;}
      }
      Kalker.WithFlag = "Enabled";
      Kalker.SoilTypeCnt = Kalker.SoilTypeCnt+1;
      ?st_name = AppendStrings(ConvertToSymbol(?soil_name), "St",
                          ConvertToString(Kalker.SoilTypeCnt));
      MakeClass(?st_name, PSD, ?soil_name, `(sldt_type, sldt_amnt, soil_no, cons_no,
                                    sldt_shp, sldt_grdg, sldt_txt, sldt_dstb));
      ?st_name = ConvertToSymbol(?st_name);
      ?st_name.cons_no = Kalker.SoilTypeCnt;
      set_slots(?st_name); class_init(?st_name);
      ?st_name = ConvertToSymbol(?st_name);
      ?st_name.sldt_amnt=?amount;
      SoilDes.PartSD = ConvertToString(?rest);
      Kalker.hit = "hit";
  }
}


method and_it.parser_clause!(?word, ?ls)
{
/* Identifies 'and' which signifies two amin soil types */
  if ?word == "and";
   then {
      conv_kalk_sub();
      Kalker.SoilCnt = Kalker.SoilCnt + 1;
      ?soil_name = ConvertToSymbol(AppendStrings("Soil",
                                    ConvertToString(Kalker.SoilCnt)));
      MakeClass(?soil_name, PSD, SoilDes, `(soil_no, soil_cons, soil_mscd, soil_mstp,
                                soil_psty, soil_strc, soil_wthg));
      ?soil_name.soil_no = Kalker.SoilCnt;
      set_slt(?soil_name);
      Kalker.SoilTypeCnt = 0;
```

```
        Kalker.ColCnt = 1;
        Kalker.SoilStruCnt = 1;
        create_Kalker_sub();
        SoilDes.PartSD = ?ls;
        Kalker.hit = "hit";
    }

}


function wrapper(?desc)
{
/* wraps soil description round to look nice, used in parser.  Could be improved */
/* by passing the width to be formatted, set here to max of 43 (?) */

    bound inputs;
    ?loop = -1;
    ?len = StringLength(?desc);
    ?count = ConvertToFixnum(?len/50);

    while ?loop != ?count-1;
    do {
        ?loop = ?loop + 1;
        ?i = 43 + ?loop*50;
        while Substring(?desc, ?i, ?i+1) != " ";
        do ?i = ?i + 1;
        ?desc = AppendStrings(Substring(?desc, 0, ?i), "\n", Substring(?desc,?i+1));
    }
    return ?desc;
}
```