An Intelligent Design Support Environment

The application of Intelligent Knowledge-Based Systems and Advanced HCI Techniques to Building Design

James H. Rutherford, B.A.(Hons), MSc.

A Thesis Submitted
for the degree of Doctor of Philosophy

Department of Architecture and Building Science
University of Strathclyde, Glasgow.
July 1990

ABSTRACT

Building design is becoming an increasingly complex process. Technological advances in building materials and construction methods have necessitated the specification of more rigourous regulatory constraints to which the designer must adhere. Although a diverse range of sophisticated computer based design tools, addressing the formal functional requirements of building design, exist to assist the designer in the decision making process, as a result of their sophistication, such tools often require considerable specialist knowledge of the methodologies employed before they can realistically be utilized on a routine basis.

As a result a growing interest has developed in intelligent user-interfaces in an attempt to make complex application software more accessible, maintainable and extendible. However, owing to inconsistencies between front-ends, the current trend in user interface management systems tends to propagate the encapsulation of application functionality within a static, esoteric style of dialogue; restricting interaction to the lowest common user level and therefore denying the designer unrestricted access to the embedded methodologies required for creative solution synthesis.

By adopting a communications view of the user-interface, this thesis illustrates how a dynamically adaptable user-interface, coupled to a multi-level knowledge based system consisting of surface level models derived from human laws, with deep models of reasoning, employing non-procedural, opportunistic knowledge acquisition mechanisms, may be utilised to accommodate the dynamically varying nature of the design process. The resulting object oriented framework is an intelligent design support system which isolates the user from the low level aspects of CAD tool management; enabling experts from different sub-disciplines to access the functionality of a comprehensive range of design tools in manner suited to their individual conceptual vocabulary, level of expertise, and idiosyncratic design procedures.

Although the framework described within this thesis is generally applicable across a range of domains, specific examples of user stereotypes and dialogue templates used to illustrate the principles behind the system are derived from building performance assessment and prediction.

PREFACE

PREFACE

Much of the work presented within this thesis has been carried out over a two and a half year period of which; two years where spent working on a collaborative, SERC funded, research program between ABACUS and the Informatics Division of the Rutherford Appleton Laboratory, to develop an Intelligent Front-end for energy simulation, and more recently a preliminary investigation has been undertaken to explore the theme of intelligent design assistance which forms the basis of this thesis.

Owing to the collaborative nature of the research it is impossible to describe the contributions made without referring to the work of other individuals. Therefore this work is described in as much detail as is necessary to establish contextual reference points and is acknowledged accordingly.

Research in the area of user interface design, artificial intelligence, and human computer interaction in general is extensive and much has been documented. It is impossible to reference or even be aware of all the existing, related literature within each of these domains. Inspiration for the work documented within this thesis has been drawn from front-line research and existing literature. All attempts have been made to identify original sources of ideas which have directly or indirectly influenced this research. For this purpose a bibliography is provided.

The progression of ideas generated by this research is difficult to represent sequentially. In the interest of the reader and the production of this thesis it should be noted that this thesis does not represent the chronological development of ideas but attempts to present them in a logical coherent manner.

ACKNOWLEDGEMENTS

ACKNOWLEDGEMENTS

The author is indebted to a number of people who have given invaluable guidance throughout the term of research and therefore contributed to the production of this thesis.

Acknowledgements are given to my supervisor, Professor Tom Maver, who, amidst his busy international schedule and commitments as director of ABACUS and chairman of the department of architecture, has provided perceptive guidance and encouragement throughout the various stages of research and structuring of this thesis. In addition to making this research both possible and enjoyable, he has also enabled me to complete this work within a three year period by providing unlimited access to resources and allowing me time to complete the production of this thesis.

I am especially grateful to Dr. Alan Bridges who has been a constant source of information and inspiration, providing informal supervision and direction. His incisive comments, after reading this thesis in draft form, have enabled me to refine the final copy before submission.

The author wishes to express his thanks and gratitude to Professor Joe Clarke, Director of the Energy Simulations Research Unit, University of Strathclyde, and to Mr Damian Mac Randal, of the Informatics Division at the Rutherford Appleton Laboratory, who has been a constant source of inspiration and guidance, providing invaluable feedback and assistance in resolving technical problems during the implementation of the forms package and resource handler. I am especially grateful for the patient and informal tuition in C, object oriented programming and interface design methodologies during the initial stages of the IFe project, enabling me to progress so quickly.

I also wish to thank Mr Andy Brown and Mr Frank Horton of Liverpool University for their continued encouragement and for allowing me to abuse their valuable resources and facilities during the Easter vacation.

Finally I wish express my thanks to my colleagues at ABACUS for tolerating my erratic working hours and to my parents who have supported me all the way.

FIGURES AND TABLES

List of Figures and Tables

CONTENTS

Table of Contents

APPENDICES[1]

---

1 Source code is not included owing to size; the *forms* package alone runs into 20,000 lines of C code.

1. INTRODUCTION

1. INTRODUCTION

Building design has become an increasingly complex process. Technological advances in building materials and construction methods has necessitated the specification of more rigourous regulatory constraints to which the designer must adhere; focussing upon a divers range of issues; from the visual impact of buildings in their surroundings, through to the efficient utilisation of energy resources.

In response to the increasing demands placed upon the designer a new generation of design tool has emerged to assist in design decision making; ranging from the realistic contextual visualisation of design solutions, to the detailed appraisal of the operational performance of environmental control systems, such as heating and lighting.

Despite increasing interest and investment in Computer Aided Building Design (CABD) within the last two decades, there is still, however, an apparent lack of useful of CABD packages. Currently substantial investment is directed towards the development of tools such as computer aided drafting packages which facilitate the output of production information but which contributes nothing to the activity of design decision making [MAVER 90].

Although a diverse range of sophisticated computer based design tools, addressing the formal functional requirements of building design, exist to assist the designer in the decision making process, as a result of their sophistication such tools often require more data input before they can realistically be utilized on a routine basis. Sophisticated design tools are often the product of academic institutions. While such developers are expert in their own field, they tend to have only a rudimentary knowledge of computer science techniques and methodologies relating to the issues of human computer interaction. The type of data required is usually of a specialist nature requiring expert knowledge of the methodologies employed. The production of good (friendly) user-interfaces accounts for more than sixty percent of the development cost of any application program. Funding for research often only covers the cost of implementing the most basic of interaction mechanisms. As a result systems tend to be:

- discrete and domain specific,
- monolithic, functionally bound, and un-maintainable,
- machine dependant,
- unfriendly.

Not surprisingly such design tools are often confined to use as consultation aids, never reaching design practice.

The solution to complex real world problems requires the coordinated and incremental efforts of many designers and experts. In order to support the functional requirements of such a coordinated computer based design environment, a broad spectrum of computing facilities ranging from simple PC's, through workstations to supercomputers is required together with an appropriate project data management system. Idiosyncrasies in the control protocols and a general incompatibility between the data abstractions of one design tool and another tends to impair computer aided design practice of this kind.

While standards are being developed to alleviate the transfer of product information between systems general acceceptance may take some time. With the specification and introduction of standards for the exchange of production information historical evidence indicates continued rivalry between vendors; no one seems to be able to agree on any one particular solution. Consequentially utility tools have to be provided to filter data between different representations resulting in project management problems. The designer, rather than being able to concentrate on the task at hand, is faced with the distracting and often hostile issues of system programming and administration.

The procedural nature of the dialogue employed by many design tools requires that a complete description of a solution is provided before any benefits can be gained. It is often impossible to utilise such design tools with incomplete, partial solutions. Application programs therefore tend to be used in isolated instances often out of context of the overall design solution and usually in a post hoc checking mode.

While many attempts at providing integrated design environments in other problem domains (such as MICON for electrical engineering) have been successful such solutions often tend to be domain specific and therefore difficult to extend to other problem domains.

A fundamental dilemma also exists between making an application easy for inexperienced computer users to employ and ensuring that the design tools are sufficiently powerful and comprehensive for the more proficient designer.

As a result a growing interest has developed in intelligent user-interfaces, in an attempt to make complex application software more accessible, maintainable and extendible. However, owing to the lack of integration and poor front-ends, the current trend in user interface management systems tends to propagate the encapsulation of application functionality within a static style of dialogue; restricting interaction to the

lowest common user level and therefore denying the designer unrestricted access to the embedded methodologies required for creative solution synthesis.

By addressing these fundamental problems this thesis aims to identify a number of basic generic methodologies for user-interface development based upon the work undertaken during the IFe project and extend the principles into a generic framework capable of accommodating and managing existing knowledge resources and design tools [RUTHERFORD 89] within a coherent user environment

Although this thesis does not aim to solve all of the problems associated with CABD it aims to illustrate how intelligent knowledge based systems (IKBS) and advanced human-computer interaction (HCI) methodologies may be utilised to provide an integrated environment and so support intelligent design assistance. Such a system should:

- enable experts from different sub-disciplines to access the functionality of a comprehensive range of design tools in manner suited to their individual conceptual vocabulary and idiosyncratic design procedures, and enable designers of different levels of experience to make useful contributions to emerging design solutions by providing a dynamically adaptable user interface,
- isolate the user from the low level aspects of CAD tool management by providing a design procedures and domain knowledge layer between the interface and CAD tool environment,
- provide mechanisms to enable designers to access knowledge and tools from other domains,
- provide intelligent, contextually relevant assistance and defaults and hence accommodate partial design solutions and incomplete knowledge thus supporting exploratory design,
- facilitate the rapid integration of new and existing design methods to accommodate shifts in design standards and methodologies in a consistent and totally transparent manner to the end user and knowledge engineer,
- automate the acquisition of knowledge and data from incidental design task tools in order to isolate the designer from the operational inconsistencies arising as a result of this response to changing design procedures,

- allow the customization of existing design solution plans facilitating rapid prototyping,
- offer distributed/parallel processing support to speed solution evaluation and synthesis,
- provide an open system architecture to accommodate the integration of the different types of knowledge required to support such a design environment and identify a generic communications protocol; enabling knowledge modules and tools to be tested in isolation and hence allowing contributions from a wide range of sources to be incorporated without disturbing the overall system.

In order to achieve these aims and construct a generic design environment a suitably powerful and flexible computing platform is required. A distributed multi-tasking UNIX environment is therefore assumed.

A pilot study, presented to the SERC IT initiative group, was undertaken to investigate the issues involved in Intelligent Design Assistance [RUTHERFORD 89]. A strategy has been formulated based upon the nature of the design process in terms of inter-disciplinary communication which, it is hoped, yields a sufficiently flexible framework.

Chapter 2 of this thesis identifies existing IKBS and HCI techniques and highlights the deficiencies of existing design principles and methodologies for software development, proposing that the generic infrastructure of the Intelligent Front-end (IFe) [Clarke, MacRandal, Rutherford] is sufficiently flexible to provide a solution to these inadequacies by accommodating the requirements of the brief.

Chapter 3 discusses, in broad terms, the issues involved in human computer communication placing particular emphasis on the issues involved in providing a generic adaptable user interface an instance of which (forming an integral part of the IFe), with the necessary control mechanisms for implementing and orchestrating a dialogue with the user, is described in chapter 4.

The subsequent section documents the collaborative contributions of Damian MacRandal and Professor Joe Clarke placing the interface in the context of the IFe. The application of the IFe to energy simulation is illustrated in chapter 6 while chapter 7 extends the principles described in the previous sections and suggests ways in which this generic infrastructure may be used to support intelligent design assistance. A number of generic interaction modules are also illustrated in accompanying appendices

2. EXISTING SOFTWARE DESIGN METHODOLOGIES AND TECHNIQUES.

2.1. COMPUTER APPLICATIONS AS DESIGN RESOURCES / ASSISTANTS

The role of the designer, is to produce a complete physical specification, *product model*, the purpose of which is to provide enough detailed information for the production of the artifact described therein.

In an attempt to assist the designer in his/her decision making processes, by identifying sequences of logical operations and phases within the design process that could easily be mechanised, once generalised, systematic design methods [CROSS 77], such as the structural analysis of building components, have evolved in the form of computational software systems resulting in a new class of design resource; computer aided design (CAD) packages.

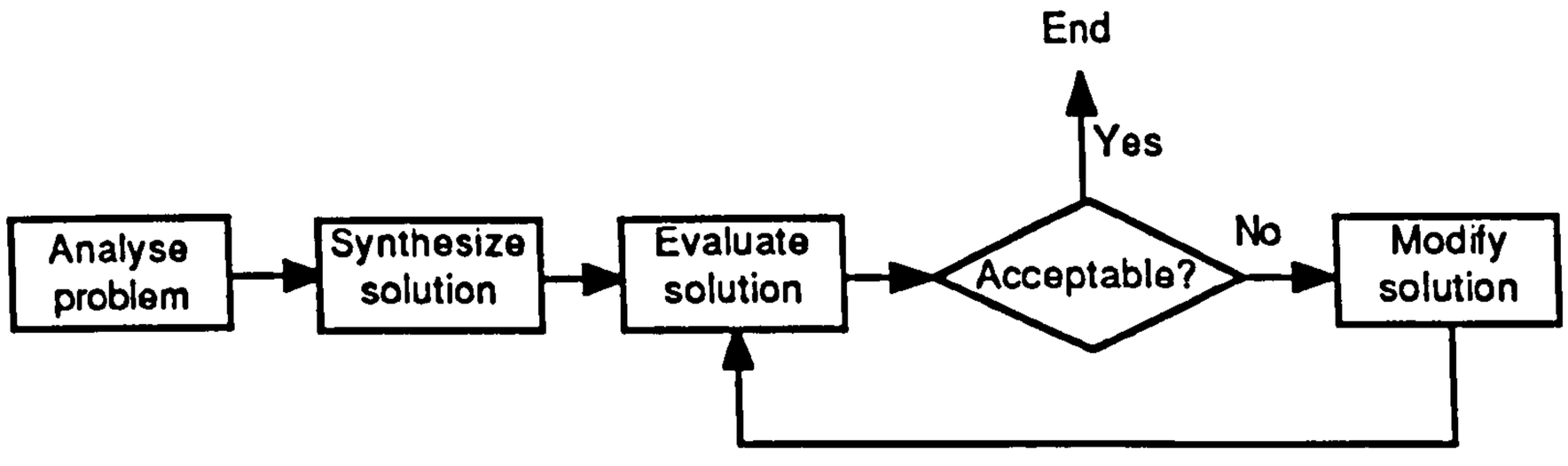


Figure 2.1.1. A classical, procedural model of the design process [MARKUS, MAVER 72].

Models of design such as the procedural Markus, Maver model, figure 2.1.1, and the far less formal approaches of the product semanticists [KRIPPENDORF 89] are mirrored in contemporary software design methodologies. How a software system is designed and implemented therefore influences the usefulness of that system as a design tool.

2.2. SOFTWARE DESIGN METHODOLOGIES AND DESIGN PRACTICE

A software system is a set of mechanisms for performing certain actions on certain data [MEYER 88] and may be categorised by one of two software design methodologies:

i) function oriented, and

ii) data/object oriented.

2.2.1. FUNCTION ORIENTED SOFTWARE

A function oriented software system is one whereby sequences of operations are performed on a given set of data in a pre-defined and systematic way. Such systems are procedural quantitative data processors and are therefore more suited to analytical applications where a design solution has already been formulated. The style of dialogue employed is also often procedural in nature (question answer) and static. Functionally oriented CABD systems often do not allow solutions to be modified interactively and applications usually have to be re-initialised from scratch and re-run.

Current CABD systems of this form are therefore difficult to use in creative exploratory situations where the problem definition may often be changing or incomplete. Function oriented systems are therefore restrictive in their application to creative design procedures. Any design framework developed must be capable of accommodating incomplete descriptions (partial knowledge) of the emerging design solution.

2.2.2. DATA ORIENTED SOFTWARE

The basic premise behind object oriented methodologies is that a software system throughout its diverse forms will almost certainly manipulate the same kind of data [MEYER 88] at least if viewed from a sufficiently high level of abstraction. An object is the encapsulation of two basic entities, normally kept apart in traditional software languages, namely [HOPKINS 89]:

- *data* - the state of the object is maintained within that object,
- *code* - the functional/behavioural mechanisms for modifying and enquiring about the current state (data) of the object are also kept within the object.

An object may therefore be seen as a totally self-contained and self-sufficient data cell; each class of object containing a set of methods for manipulating the data contained within it may be accessed simply by passing messages to object instances. Objects may be defined hierarchically,with new object classes inheriting both functional and behavioural characteristics, from existing object classes.

Data or object oriented software systems are therefore much more open and flexible. In terms of interaction they are far less procedural than function oriented software systems and perhaps more appropriate to creative design processes requiring direct and unrestricted manipulation of ideas and concepts held as data.

The methodology employed can therefore affect the usefulness of a software system as a mechanism to the creatively manipulate objects and ideas in an unrestricted manner.

2.2.3. THE TRADITIONAL ROLE OF COMPUTERS IN DESIGN

Despite the increasing interest and investment in computer aided building design (CABD), within the last two decades, there is still an apparent lack of useful CABD packages for design practice. This is a result of the fact that, spurred by the conviction that it is possible to isolate, generalise and finally translate design processes and operations into programs, "architectural CAD researchers have adopted the classical approaches of industrialized automation" [CARRARA 88], exhibited in Systems Theory and Operations Research [HASHIMSHONI 78, SHAVIV and GALI 79], resulting in procedural, function oriented mechanisms. Current investment by software developers, influenced by academic research, is therefore directed towards systems such as computer aided drafting packages which facilitate the output of production information only and contribute nothing to the creative and intuitive activity of design decision making [MAVER 90].

Packages that do address the formal functional requirements of building design, although useful in their own right, tend to be discrete and domain specific. Traditionally developers of CAD packages have concentrated largely on limited (specialized) aspects of design (eg structural design, energy efficiency and other quantitative aspects) creating discrete design tools.

Regardless of how design is defined "solutions to complex real-world problems result from the combined and incremental efforts of many experts" [WILLIAMS 88] and designers.

2.3. DESIGN ASSISTANCE AS AN ACTIVITY - A COMMUNICATIONS VIEW

Design assistance is the utilisation of specialist, expert consultants, *design resources*[1], to supplement a designers own design experience and expertise, figure 2.3.1.

Figure 2.3.1. Multiple Design resources

An important decision, often made in the early stages of the design process, directly effecting the outcome of the overall design solution, is one of employing the correct design resource to solve particular problems.

The need to employ and the choice of specific design consultants is obviously determined by the deficiencies in the designers own experience. The role of the designer or design coordinator is to establish what expertise is required to produce a complete design solution and, based upon the self critical evaluation of the designer's own capabilities, identify a number of specific tasks that must be dealt with by other design resources. In order to identify problem areas and delegate specific tasks to individual resources the design coordinator must posses knowledge of the capabilities of a number of domain related resources.

Owing to the prototypical nature of design, it is impossible for a design coordinator to know whether a design resource is capable of handling a specific task. The knowledge held by the designer about individual resources therefore consists only of a general categorisation of the functional capabilities of each known resource in order to aid the identification of potential design collaborators. Each is then invited to analyse the problem and provide a more detailed description of the their capabilities in

---

1 A design resource may be an individual design consultant or a complete design organisation.

relationship to the required task, which is influential in the final decision to employ a particular resource.

This is typical of the traditional tendering process that exists in design practice. The final decision to employ a particular resource is governed by many factors other than functional capabilities; cost often features significantly in the final evaluation.

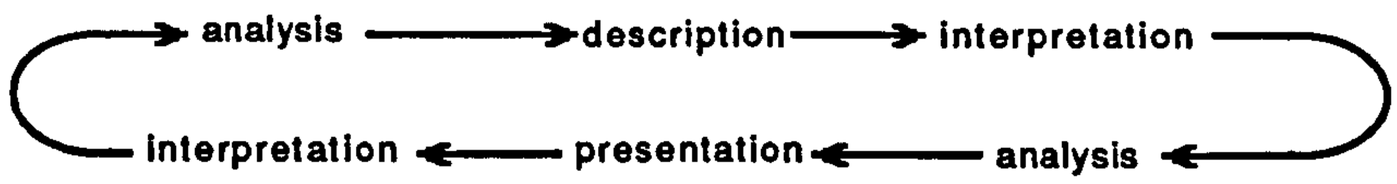


Figure 2.3.2. Communication cycle in problem solving

Whether a designer employs his or her own design experience or that of another designer a significant proportion of the design activity is concerned with communication, figure 2.3.2., between resources. Any CABD support system must account for this distribution of design tasks.

When appropriate resources have been selected the designer must provide adequate detailed descriptions of the product together with additional, specific detailed information for each consultant employed. The product model is therefore decomposed into elemental parts (concepts) or layers (meta-concepts) for each sub-discipline.

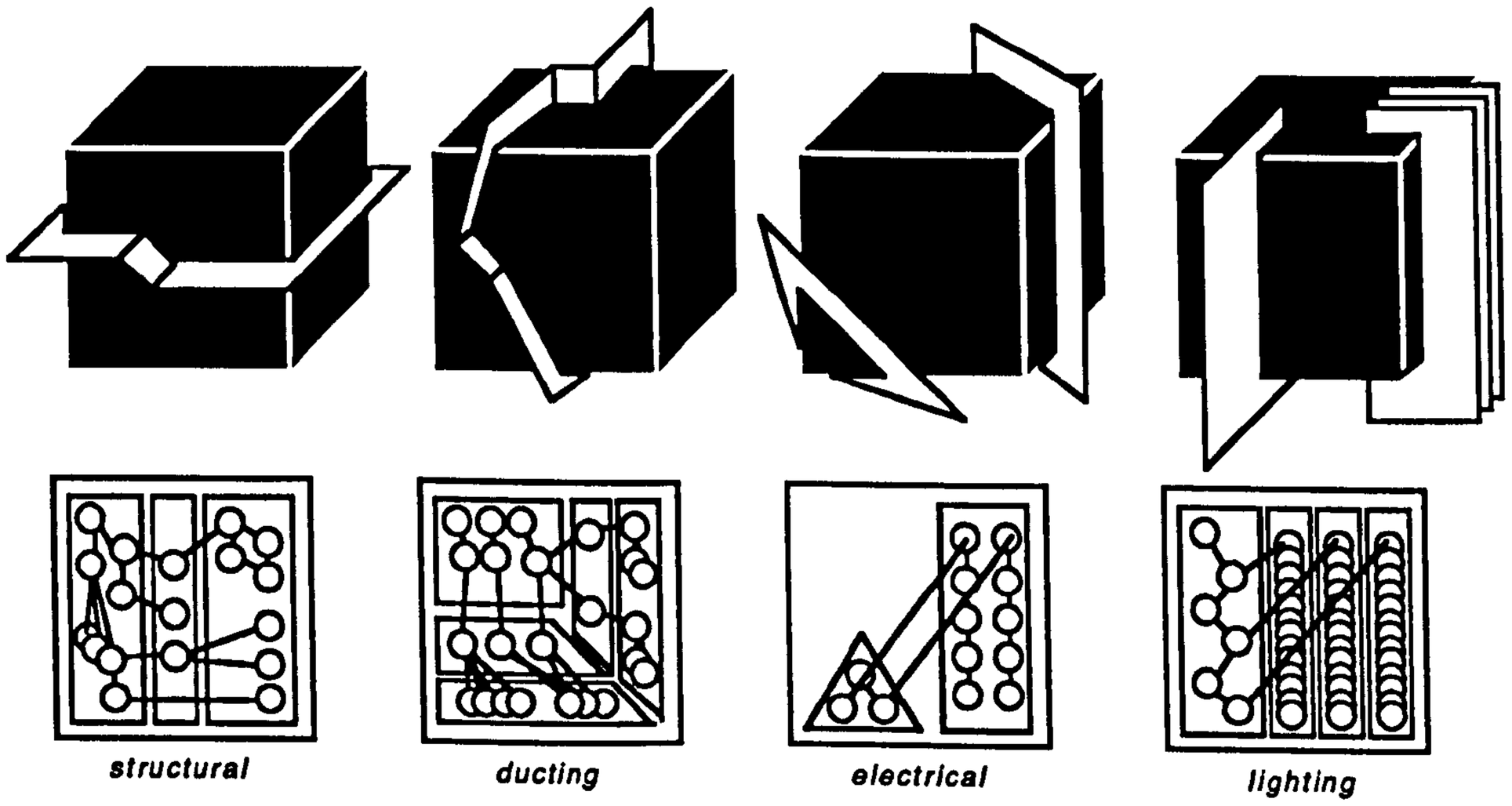


Figure 2.3.3. Four individuals' view of the same design space data [from Vidovic 1990].

Each consultant or resource may express an interest in only one aspect of the overall product in relation to a view of the entire product or a view one or several

abstraction levels above the area of interest. The design resources, therefore, have their own view (or schema) of the product model, figure 2.3.3. Vidovic suggests that "each view is not just a subset, but contains semantic information via the relations set up among the data objects by the individual".

For efficient and effective use of a design consultant, both the designer and the consultant must be able to communicate to each other in a common language; they must employ the same conceptual vocabulary and symbolic representation and also be able to communicate their intentions clearly enough to enable information together with both explicit and implicit relations between objects to be extracted.

2.4. THE QUEST FOR AN INTEGRATED COMPUTER BASED DESIGN ENVIRONMENT

A fundamental problem encountered by a designer, faced with this new environment, is one of how well computational methods and traditional manual design procedures can be integrated together [CARRARA 88]. Many solutions (originating in the 1970s) to this problem where to create a total design environment within the computer. Early attempts at providing such an environment where thwarted by limitations in both software design methodologies and hardware platforms. Integrated design tools where either implemented as monolithic functionally oriented systems which soon reached the limits of processing power, or as discrete data processors. When two or more application programs are utilised, even when they exist within the same computing environment, a problem of integration between individual systems is usually introduced often resulting from incompatible data formats.

Despite advances in hardware platforms and therefore a corresponding increase in the power and sophistication of software, CABD systems continue to be developed as quantitative data processors, and often as packages become more sophisticated the amount of data input required before such systems can be usefully utilised, increases significantly.

The type of data required to initialise the application program and the style of dialogue employed to communicate to the designer or end-user tends to be of a specialist nature, literally reflecting the design processes and operation the program attempts to mechanise. The user is therefore required to posses expert knowledge of the methodologies encoded within the application program and therefore this approach to the development of design tools defeats the fundamental aims of computer aided design. ESP [CLARKE 86] is a typical example of a powerful energy simulation package requiring a substantial input of energy related information. The dialogue

employed is terse and directed towards specialist end-users. As a result ESP, although powerful, is often rejected by the design profession.

The time consuming procedure of communicating design information between collaborating consultants is true of computer applications (many computer applications don't read drawings). For each application utilised by the designer the information contained within the product model has to be re-formatted (usually by hand) to suite the conceptual representation of each application program.

Utility programs for data bridging, the process of converting one data format to another, are often employed to overcome the immediate isolation between systems but introduce even more difficulties for the designer:

- *data management*: several copies of the original data set in different formats may exist. The designer has the problem of knowing which particular set is the most current solution.
- *storage*: each copy of the original data set increases the demand upon the storage device.
- *resolution*: detail may be lost between filtering processes due to rounding errors or a mismatch between data abstractions.
- *computer literacy*: data filtering is often a low level process and therefore requires a high degree of operating system knowledge.

Inconsistencies in the user interface, resulting from idiosyncratic interface design methodologies, and the conceptual vocabulary employed by different application programs can introduce serious problems; a command used to invoke a process in one system may invoke a completely different, perhaps destructive process in another.

Such mechanised design tools, therefore, tend to be used in isolated situations, often out of context of a complete design solution, and usually in a "post-hoc" checking mode, quite often in a consultation role performed by the original design tool developer.

Packages that do cater for a more diverse range of design topics are either bound in to an unwieldy monolithic system that is difficult to extend or are too fragmented resulting in inconsistencies both in data abstractions and in the user interface, making them difficult to use.

While attempting to solve an number of design related issues, current CABD software therefore introduces more fundamental problems relating to communication and integration and generally tend to be:

- discrete and domain specific,
- monolithic, functionally bound and un-maintainable,
- machine dependant,
- unfriendly.

These fundamental problems have affected the general acceptance of CABD systems in design practice.

For a system to be used successfully as a design tool it must enable a designer to develop and manipulate design solutions at both conceptual and detail levels and in a manner suited to the designers own conceptual vocabulary. While current software systems facilitate detail design, Lansdown suggests a number of relevant characteristics of the modelling process which takes place during concept design - of which any design aid must take account [MAVER 88]:

- *creativity*: concept design requires imagination and inventiveness
- *multiplicity*: to any design problem many feasible solutions exist
- *empiricism*: reliable theoretical foundations may be lacking
- *approximation*: fuzziness and uncertainty surrounds the knowledge base
- *expertise*: some people seem to be good at it.

Crucial to these issues is how the user of such a system perceives and manipulates system modules together with problem specific data. This in turn is determined by the overall system architecture and software design methodologies.

What is required, in order to achieve a flexible computer based design environment, is a software design strategy that yields an infinitely extendible architecture capable of supporting, in parallel, all the resources required for the definition of a diverse range of products with a consistent user-interface capable of responding to the differing and idiosyncratic (sub-disciplinary) conceptual vocabularies of participating members of the design team.

The implications of such inadequacies in current CABD stress the need for highly familiar command formats and dialogue structures in order to reduce the time

required to learn and remember the operation sequence of application packages. Communication must be fluid enough to enable the user to concentrate on the matter at hand and not focus the user's attention on the inadequacies of the style of dialogue employed

Building design is heavily governed by regulatory guideline's and restrictions. As opinions change (at the dictates of fashion) so do the standards that represent and enforce these beliefs. Any CAD framework must therefore be sufficiently flexible and responsive enough to reflect shifts in design standards without changing the user's perception of that framework.

Traditional software design methodologies have been overly based upon the functional requirements of the system, resulting in a tightly bound and rigid system architectures that are difficult to modify and extend, and therefore not suitable for such an environment.

The formulation of a practical symbiosis between designers and computer systems is therefore necessary in order to increase accessibility and fluid dynamic communication. Many attempts have been made to solve one or more of these problems. However all have been tailored solutions and as yet no generalised software design methodologies have been formalised for CABD systems.

The user of sophisticated application programs often requires a particular familiarity with application specific concepts which is not often possessed by casual users. Application programmes often have a diverse range of interfaces increasing the problems of the user and an equally varied number of data formats resulting in data management problems.

The problems associated with conventional CABD design tools are therefore two fold:

1) the lack of natural dialogues between designer and computer application and

2) incompatibility between the output of applications.

These problems may be resolved by:

- the provision of a consistent and *friendly* user-environment which must address issues such as the:
  - intelligent interpretation of utterances from the user
  - presentation and interpretation of results; tailored to the user
  - provision of multiple levels of assistance; again directed at a particular class of user
  - mechanisms available for error handling and recovery
  - customisation of the interface and definition of task macros
- the seamless integration of numerous CAD resources, achieved by accommodating the transparent interchange of information and knowledge between application programs.

The aim is to create an intelligent "responsive" user-interface. Additional benefits, such as improved systems management, will also result.

As the end-user possess a significant body of domain related knowledge the idea of a CAD resource is extended to include the designer in the overall view of the design framework. By adopting this view, the issues related to the integration of design tools with manual techniques, should be accommodated.

Regardless of the domain, there is a need to provide a communications buffer (user-interface) between the end-user and target application program.

As individual designers and those from other disciplines each have idiosyncratic vocabularies, this approach also highlights the need to accommodate several, potentially different types of end-user. The actual mechanisms and design implications for implementing a suitable adaptable user-interface are discussed in chapter 3. Here the discussion will focus upon the general, currently available methods for developing user-interfaces and will look at current methodologies and methods for bringing together several different types of knowledge source.

Current trends in interface design methodologies have focused upon the formal issues of consistency, both in the visual appearance of the interface and the interaction mechanisms available for inputing data, together with semantic consistency of command languages.

2.5. EXISTING SOLUTIONS TO INTERFACE CONSISTENCY AND SOFTWARE INTEGRATION.

The application developer has to meet both the requirements of the user and satisfy the application brief. It is often uneconomical or unfeasible to provide anything other than the simplest of user interfaces; these often being highly inappropriate. A need has therefore developed for a range of tools to assist in the design, development, and management of the user-interface.

The solution to the problem of integration, from the point of view of designer, has been tackled by addressing the issues of communication (chapter 3). In particular by providing similar interfaces to a number of applications. By addressing the various aspects of consistency a number, providing partial solutions, of well established methods to aid the development of user-friendly interfaces have evolved. These include:

- Standardised graphics toolkits or libraries (GL).
- User Interface Management system (UIMS)
- Expert Systems (natural language interfaces) (ES)
- Intelligent User Interface Management Systems (IUIMS = UIMS+ES) or intelligent front-ends (IFEs)

From the point of view of ensuring compatibility between application data, numerous standards have been developed usually for the interchange of production information; DXF, IGES, etc. Recent developments such as STeP and PDES focus on the more relevant issues of the interchange of product models or complete descriptions of artifacts and is discussed further in chapter 7.

2.5.1. THE USE OF HIGH-LEVEL GRAPHICS TOOLKITS

The traditional approach to developing a user interface is to acquire data as and when it is required by the application program using various terminal or graphical based prompting.

There are many high level (and some not so) function libraries and toolkits each providing functions to handle specific types of representation, interaction and graphical output (GKS, PHIGS, DORE to name just a few). Most workstations come with vendors graphics toolkits (Sunview and Core on Sun workstation, Angel with Whitechapels).

By utilising a single graphics toolkit, providing a high level of functionality, the application developer is able to achieve both visual and interaction consistency. While this approach provides adequate software, the main disadvantage is one of bound functionality; interface (I/O) and application specific routines and procedures are intermingled, figure 2.5.1, resulting in an unwieldy monolithic piece of code which is difficult to maintain and extend.

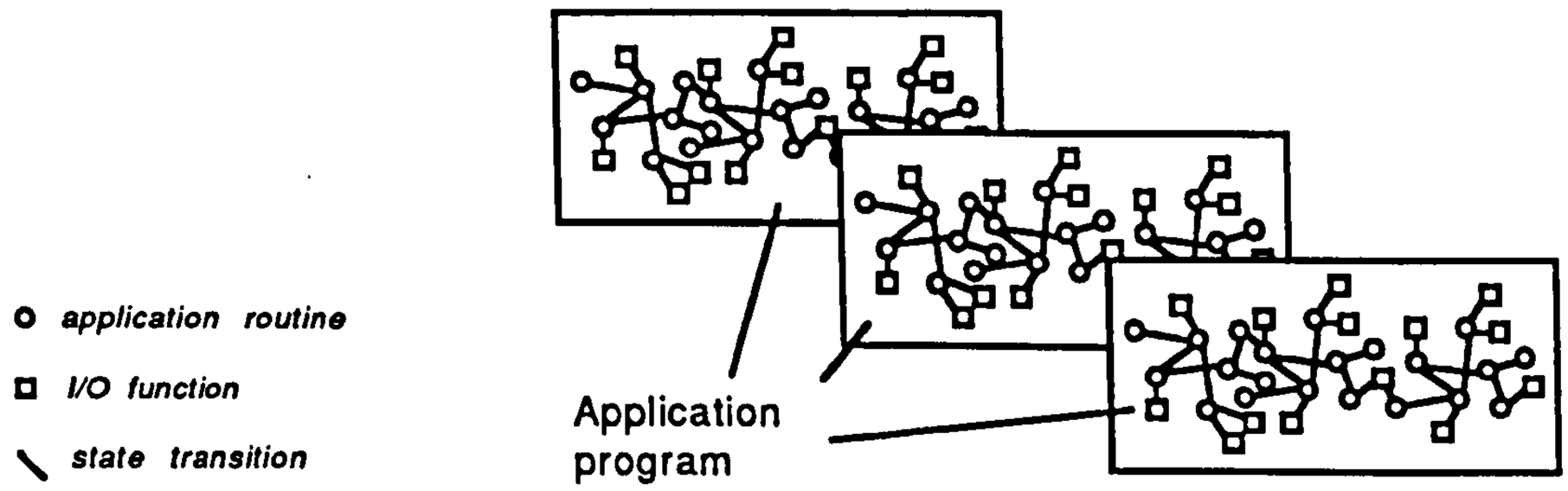


Figure 2.5.1. Intermingled I/O and application specific routines.

Software development has largely been prototypical. A number of solutions have evolved but tend to focus on providing tools to aid development and management of software. There is a distinct lack of standards, although some are now emerging such as OPEN LOOK and Andrew for X and NeWS.

Although toolkits standardise the appearance and style of interaction, inconsistencies in command vocabularies are inevitable owing to the idiosyncratic and often proto-typical nature of software development.

The standardisation of high level graphics toolkits has lead to the emergence of User Interface Management Systems which attempt to address the more pertinent issues of consistency within human computer interaction.

2.5.2. USER INTERFACE MANAGEMENT SYSTEMS (UIMS) AND FRONT ENDS

UIMS are the result of a number of software development criteria. From the developers point of view UIMS attempt to:

- free the application developer from the low-level details of the user interface, enabling the software designer to concentrate on aspects specific to the user interface and application development
- reduce the cost of software development for the user interface and hence the overall cost of application development
- separate the user interface code from that of the application. The separation should ideally allow different interface to the same application.

UIMS must also address the needs of the user in the form of dialogue sequencing and control:

- the dialogue may be *flat* allowing commands to be accessed at any time, alternatively a *hierarchical* command structure may be employed
- the user may be able to carry out tasks using direct manipulation of graphical objects or by a formal command language or may be lead by a procedural question and answer dialogue.
- multi-threaded dialogues may be possible, allowing one operation to be suspended while another is executed. This facility may be extended into a concurrent environment allowing several, simultaneous operations to be controlled.

UIMS are also seen as valuable aids in providing a seamless integration across a range of application software within a coherent and consistent user interface provided that a single system gains global acceptance.

User interface management systems essentially consist of a library of building blocks that may be used to assemble a user interface for a particular program [TANNER 83] and may be thought of as a set of tools to support the design, implementation, maintenance, and evaluation of the user interface [BAECKER 87]. A recent report [PRIME 88][2] has been used as a reference for this section on UIMS.

---

2 *User interface Management Systems (UIMS) - a current product review*, Martin Prime, Informatics Division, Rutherford Appleton Laboratory, 1988.

UIMS attempt to alleviate the problems of traditional (wirewool) software development by conceptually providing a clear distinction between the application and the user interface. This is achieved by breaking the interface into three layers:

[presentation] [control] [application]

first suggested by Sehiem [SEHEIM]. Tanner and Buxton provide a more illustrative generic description of a UIMS, shown in figure 2.5.2.1.

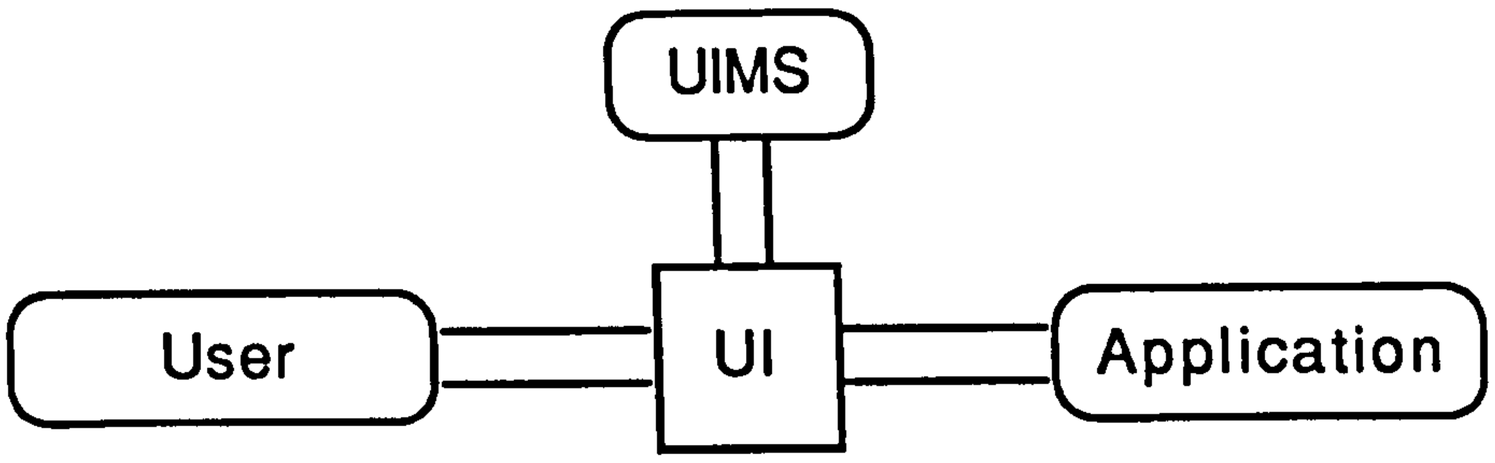


Figure 2.5.2.1. Typical UIMS in Context (from [BAECKER 88]).

A typical UIMS consists of two distinct modules:

- *a pre-processor* used at the dialogue design stage to design and implement the user interface
- *interaction handler* which manages the interaction between the user and the application at run time.

A critical feature determining the effectiveness of any UIMS is how these two modules interact with each other. Typically a user interface, UI, definition is used to communicate between the pre-processor and interaction handler. The UI (encompassing the user's task model and the functional capabilities of the target application) may be implemented as a shared data structure or a knowledge base. The Sassafras system [HILL 86], figure 2.5.2.2, provides a more complete overview of a typical UIMS, highlighting the separation between layers and indicating the relationship between the various components and stages that comprise the development graphical user interfaces.

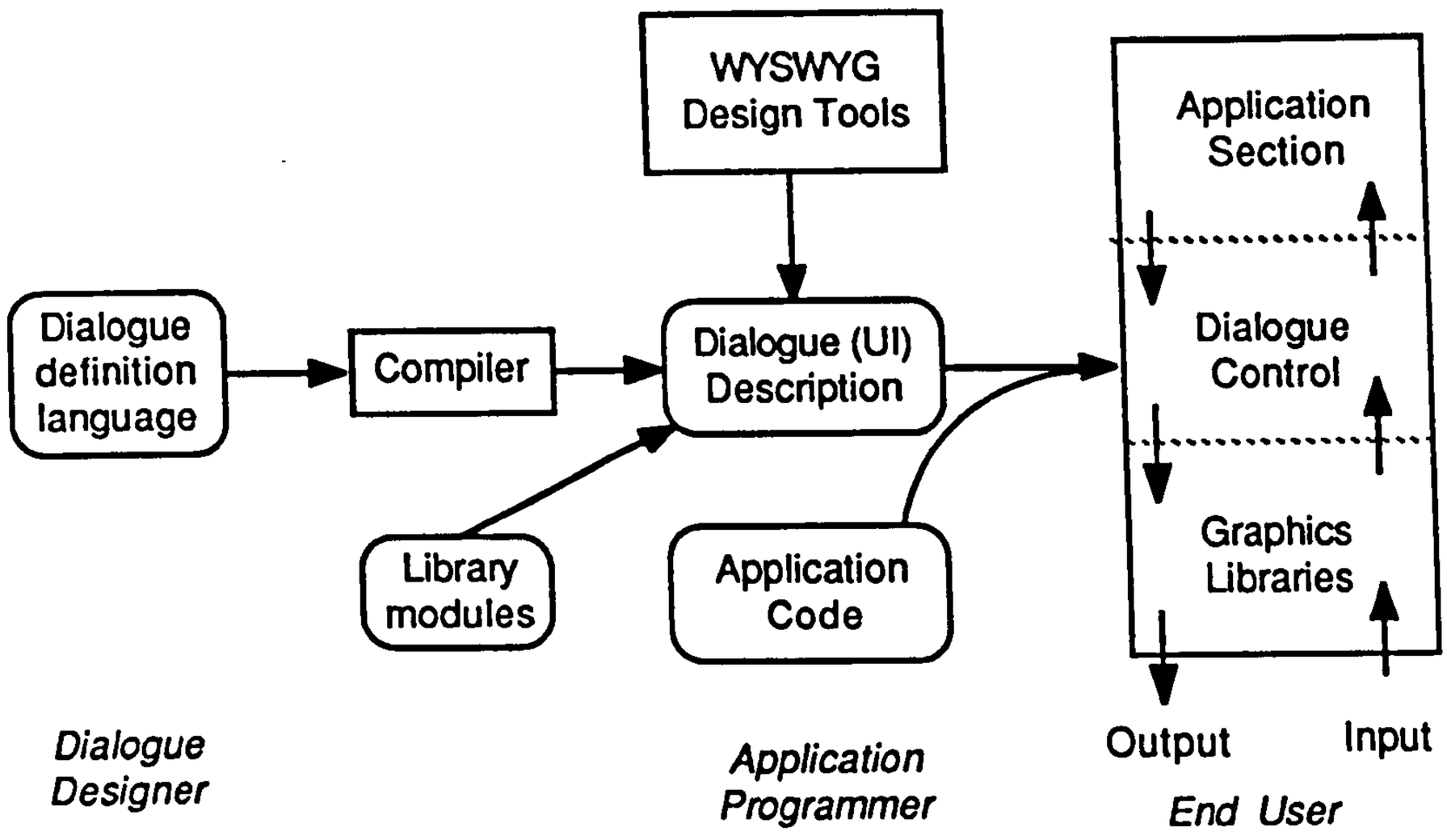


Figure 2.5.2.2. Sassafras reference model (from [PRIME 88]).

2.5.2.1. DIALOGUE ORCHESTRATION IN UIMSS

The fundamental difference between traditional application development and that utilising a UIMS is that operational sequences of the application program are defined by the interface designer.

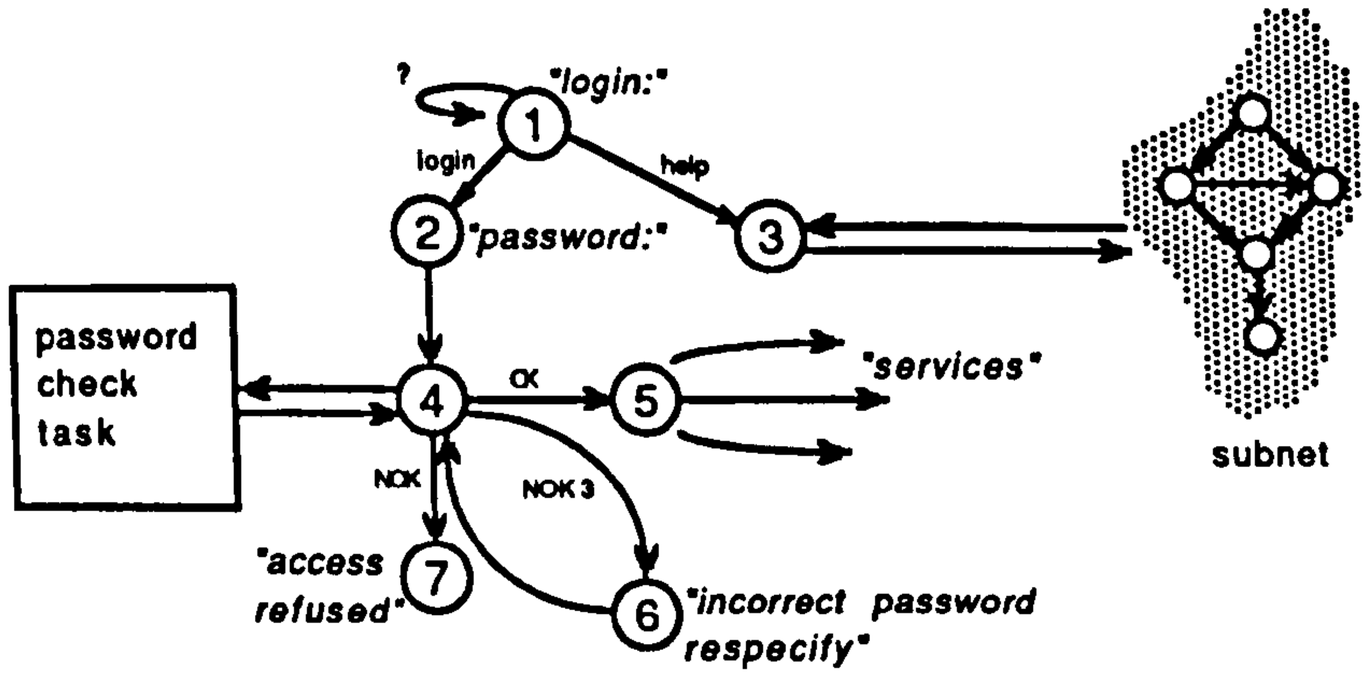
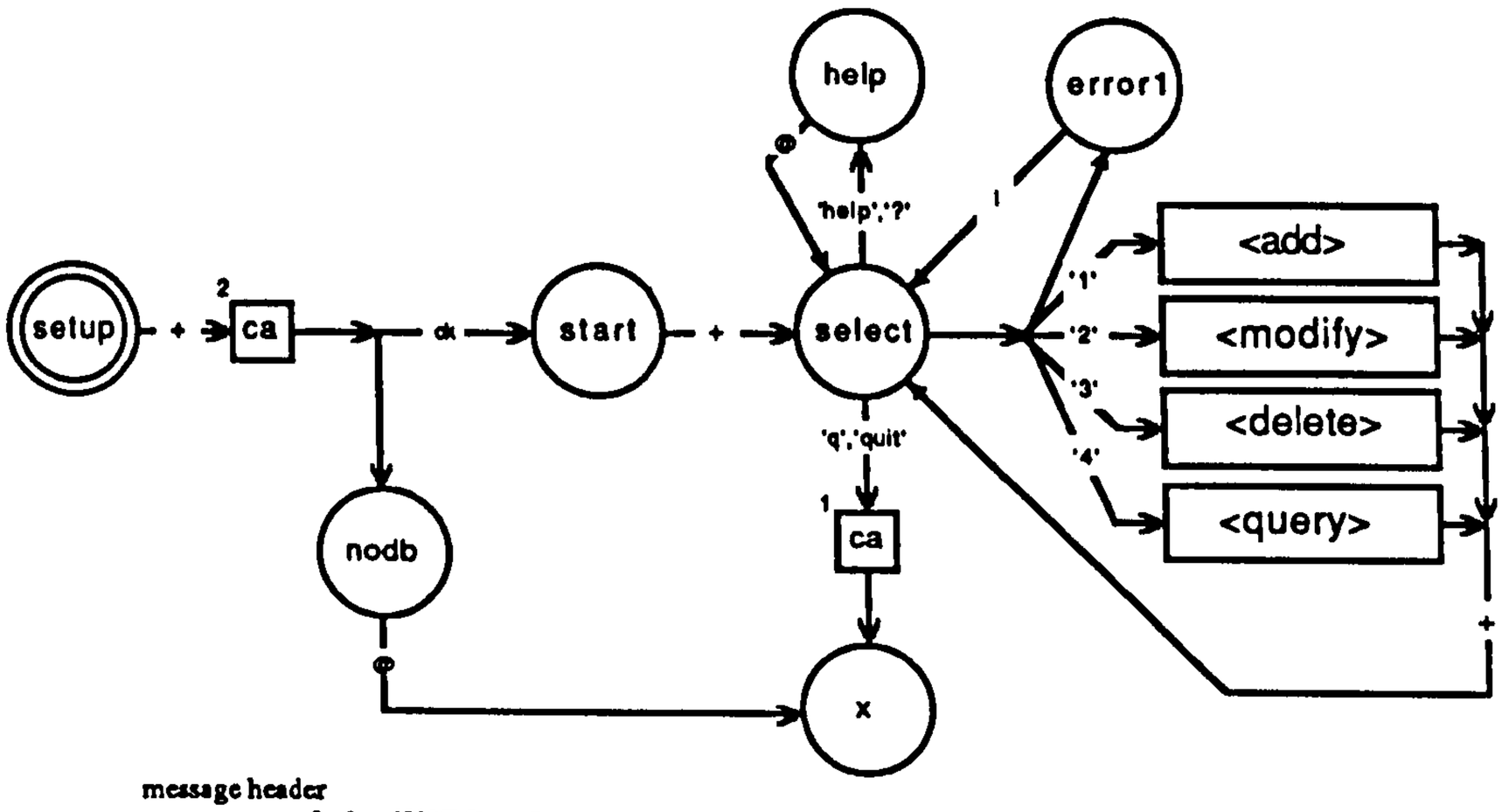


Figure 2.5.2.1.1. Typical state transition network (from the CONNECT system) [ALTY 83].

Typically a finite state automaton is employed for the definition of the UI. usually in the form of a table of state transition information, figure 2.5.2.1.1, (such as that used in SET, described in [PRIME 88], and RAPID/USE by Shewmake and Pircher, described in [WASSERMAN 84]) This network is then used to orchestrate the dialogue and control where the application should pause for input (states), represented by circles. The paths between states are known as transitions, shown as arcs with arrows pointing to the next state. State transition diagrams are popular for

the specification of interactive dialogues systems as they can be easily visualized graphically.

In the majority of cases networks are usually drawn on paper and translated by the interface developer into a textual description. Figure 2.5.2.1.2 illustrates a transition diagram from the USE Data Dictionary system [WASSERMAN 84] and shows the resulting coded description.




```
message header
        cs,r2,c0,c_'USE Data Dictionary'
message lastline
        r$,co,'Hit any character to continue'
node setup
node select
        tomark_A,ce,r+3,t_0,"Please choose',
        r+2,t_1,'1: Add a dictionary enter.'
        r+2,t_1,'2: Modify a dictionary entry.',
        r+2,t_1,'3: Delete a dictionary entry.',
        r+2,t_1,'4: Query data dictionary.',
        r+2,t_1,'help: Information on use of program',
        r+2,t_1,'quit: Exit USE/Data Dictionary.',
        r+2,t_0,'Your choice:'
node help
        cs,r$-3,c0,'For more information about a command, enter',
        r$-2,c0,'the command number, press return and then type "help" or "?"',
        r$,c0,'Hit any key to continue'
node nodb
        cs,r$,c0,Could not open database directory'
node start
        header,mark_A
node x
        ok
node error1
        r$-1,c0,rv,bell,'Please type a number from 1 to 4.',rv,
        lastline
```

Figure 2.5.2.1.2. Top-level state transition diagram from the RAPID/USE system [WASSERMAN 84]. Note characters at the beginning of each line are used to control the position of 'text' on the screen: cs - clear screen, c - column, r - row.

The corresponding dialogue description fragment for the transition diagram, figure 2.5.2.1.2, is illustrated below in figure 2.5.2.1.3.

```
arc setup
        skip call startup
            when ok to start
            when fail to nodb

arc select
        on 'q',         'quit' call shutdown to x
        on '4'          to <query>
        on '2'          to <modify>
        on '3'          to <delete>
        on '1'          to <add>
        on 'help','?'   to help
        else to error1

arc help noecho single_key
        else to select

arc <modify>
        skip to select

arc <delete>
        skip to select

arc nodb noecho single_key
        else to x

arc start
        skip to select

arc <add>
        skip to select

arc <query>
        skip to select

arc error1 single_key
        else to select
```

Figure 2.5.2.1.3. Dialogue description fragment (from [WASSERMAN 84]).

Network descriptions are then compiled by the pre-processor into a skeleton program containing a main program together with subroutine entries into which the application code is inserted to produce the final program [PRIME 88]. This is illustrated conceptually in figure 2.5.2.1.4.

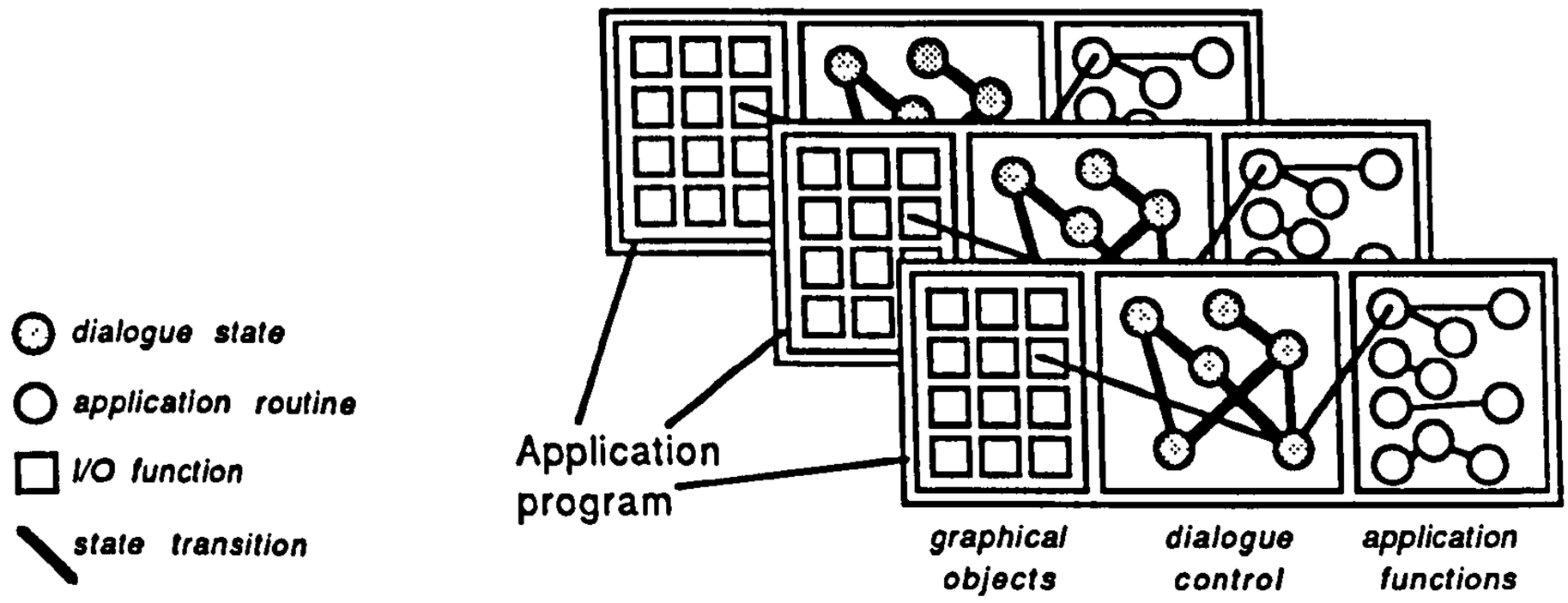


Figure 2.5.2.1.4. I/O and application specific routines separated by a state transition (dialogue control) network.

State transition networks are well understood and have been used for the design of human-computer dialogues for some time [ALTY 84, NEWMAN 68, WASSERMAN 84]. They are however restricted to procedural dialogues and do not support concurrency; necessary for multi-threaded dialogues, which are useful for the simultaneous execution of related tasks. In design it is often advantageous to investigate several solutions to a problem. Haenen, [HAENEN 87], identifies three types of concurrency within UIMSs:

- *concurrent output*: simultaneous output of audible and visual feedback and/or simultaneous updating of display windows.
- *concurrent input*: input of data by multiple input devices making the interaction more natural.
- *concurrent dialogues*: enabling the user to interact with several related interfaces simultaneously.

The latter form of concurency is, according to Haenen, rarely found in UIMSs. Other forms of dialogue model include:context-free grammars and event models. However in order to support concurency a radically different (rule based) approach is adopted which is described in a later section.

The control over the definition of the UI is also important in determining the effectiveness of a UIMS for prototyping. The majority of systems require the compilation of the UI and some post processing before a run time system can be used. Suspended-time UI editing is a feature of interpretive environments such as Smalltalk and LISP. This allows the developer to suspend the execution of the application in order to modify the UI definition and continue execution from the point of interruption with a different UI template. Examples of suspended-time editing systems are: Sassafras [HILL 86] (which Prime [PRIME 88] has used as a reference model), and the Peridot system [MYERS 86]. Strangely, non of the current (state of the art) range of UIMS provide facilities for suspended-time editing.

Some other systems such as DICE, while not a true UIMS, allow dialogue sequences to be defined as a hierarchy of units and compiled into a multi-stream parser and scheduler which controls the flow of the dialogue [PRIME 88].

Examples of currently marketed UIMS may be found in [PRIME 88] and [BAEKER & BUXTON 87] and include:

- *TIGER* [KASIK 82] is an early second generation UIMS (with a powerful glue system) developed by the Boeing Corporation (1984) for the development of large CAD systems. According to Beaker and Buxton

TIGER has a powerful language for defining the low-level details of the user-interface and it is possible to extend the functionality of the system by coding and inserting new primitives.

- *RAPID/USE*. Although this system, described in [WASSERMAN 84], utilises graphical interface techniques it was not included in Prime's evaluation of UIMS possibly owing to the fact that it can only support text-based interaction. This system is an example of a UIMS employing state transition networks for the UI definition.

- The *Trillium* system, described in [HENDERSON 86] and BAEKER 88], was developed to facilitate the rapid prototyping of controls for photocopying machines. It has a powerful glue system that supports suspended-time editing; allowing the developer to interrupt execution while the interface is modified and then step back into interaction mode with the modified dialogue.

- *SET* (PA Management Consultants), defines the UI in terms of an "interaction model" [PRIME 88] which includes command structure, restrictions on command arguments and prompting. Like the RAPID/USE system, it too employs a network description.

- *BLOX* (Template Graphics Software Inc) is according to Prime a UIMS providing dynamic run-time control over the interface manager and provides mechanisms for inquiring about and controlling the user interface.

- *Autocad M/P/E* is used to develop CAD applications and facilitates the development of interactive graphical user interfaces.

- *MacApp* [SCHMUCKER 86] is a software development framework for the Apple Macintosh. The system provides an extensive toolkit of pull-down menus and buttons and enables the developer to rapidly put together application programs by cloning and then customising previously defined (ready made) applications.

- *Peridot* UIMS, reviewed by Baecker,1987, provides semi-intelligent tools for designing the interface by allowing the developer to sketching how the interface should look and indicating how it should behave.

It was not possible for the author to evaluate the UIMS described, rather the evaluation was carried out independently by Martin Prime (RAL). The final report was not available until the second year of the IFe project and owing to the limited resources available it was not feasible to purchase any of the existing UIMS. This provided an opportunity to investigation dialogue control and interface design methodologies in some depth.

While the modularity of interface design processes can improve the quality of user interfaces it is apparent that UIMS are tools which aid the development of application software. The final result being similar in nature to traditionally implemented application software.

A UIMS should provide intelligent assistance to ensure that mistakes in data input are diminished. This is facilitated by the provision of:

- on line, contextually relevant guidance,
- intelligent, contextual, defaulting,
- backtracking - a process where by the user may trace the definition of an entity [PRIME 88] to its origins. None of the reviewed systems facilitated this activity which is important in an unrestricted design environment.
- good error recovery either by undoing or by the play-back of recorded sessions (input logging)

While UIMS facilitate the development of portable application programs (most may be ported to a number of hardware platforms) there are a number of problems with contemporary UIMS, which may be categorised as follows:

- encapsulation
- code structuring
- explicit semantics
- mode-locked operations

2.5.2.2. ENCAPSULATION

UIMS aim to provide access mechanisms to the functional methodologies encoded within an application. The functionality of the applications is often encapsulated with an opaque UI (network) definition template. Although this encapsulation results in hopefully a consistent interface it is at the cost of denying the user direct access and control over individual methods.

2.5.2.3. CODE STRUCTURE

From a development point of view, the way in which UIMS are structured (usually around a template or network) can restrict the software developer to a rigid inflexible methodology which may be inappropriate for a given application or design procedure. Although not strictly a UIMS, in the formal sense of UI definition, Sun's SunView (Sun Visual/Integrated Environment for Workstations) requires that application event handling routines, a reference to which, are inserted into the object definition of the interface primitives. It may be difficult to decompose existing software into a modular form. From experience, any attempts to interfere with existing, working application software, often provides more headaches than re-coding from scratch.

While UIMS and IUIMS attempt to integrate application software traditional approaches have failed owing to a lack of well defined and systematic methods [HAMALAINEN 88]. Although UIMS alleviate the problem of software development existing software would require significant re-coding.

It is the aim of this research to utilise as many existing design aids as possible. A great deal of time and effort has been spent developing software applications in academic institutions much of which never reaches the design profession. In many instances software has to be re-written as it is incompatible with current design philosophy; this is extremely wasteful of development resources [RUTHERFORD 89].

Another problem with network oriented dialogues is that they often result in mode locked interfaces. The style of interaction offered by a graphical user-interface may be categorised into one of two groups:

i) *modeless*: where the user selection is not confined within one set of operations
ii) *mode locked*: selection is restricted to a single set of choices.

2.5.2.4. MODE LOCKED OPERATIONS

Mode locked operations are often employed when the users selection must be restricted. It is often a "feature" of operations requiring confirmation:

- overwriting files
- quiting an application

Mode locked operations must provide at least one alternative non-destructive option usually "cancel" to enable users to abort the operation and return to the initial state. Mode locked operations are often inappropriate as they:

- result in procedural application interfaces
- dictate an operational sequence which may be inappropriate for a particular task and therefore
- increase navigational interaction especially when frequently used resulting in user fatigue and restlessness

An example where mode locked operations may cause frustration is when data stored within memory is to be written to disk. If the data set requires slight amendment before storing, the user must abort the operation, make the changes and repeat the operational sequence.

2.5.2.5. MODELESS OPERATIONS

Modeless operations, in contrast, allow the user to access other functions within an application in addition to the current set and, although they require greater knowledge of commands and operational sequences, are much more flexible.

2.5.2.6. EXPLICIT SEMANTICS

Although a number of UIMS such as the Boeing Corporation's TIGER system allow the range of primitives within the package to be extended, the conceptual vocabulary employed by the interface remains static in all current UIMS. Although it is possible to encode mechanisms for adjusting the dialogue, there are no facilities available to dynamically modify the level of dialogue to suit the level of experience of the user or adjust the dialogue according to differences in the cultural backgrounds of users, for instance:

red = danger

green = safe

Separate applications have to be compiled for each user opening up the possibility of semantic and syntactic inconsistency.

Little attention is paid to subtle issues of communication between cognitive systems. The emphasis is usually on providing tools to aid development.

2.5.5. THE END USER'S TASK

Wilson, 1987, identifies the types of knowledge a user of a design tool draws upon during interaction and categorises them as primary and secondary sources of knowledge, illustrated in figure 2.5.5.1. The first group, which the more proficient user of design tools draws upon, contains knowledge of:

- the problem
- the interface dialogue
- system operations
- the physical interface
- the computer version of the domain

The remaining types of knowledge are those sources a novice user will employ.

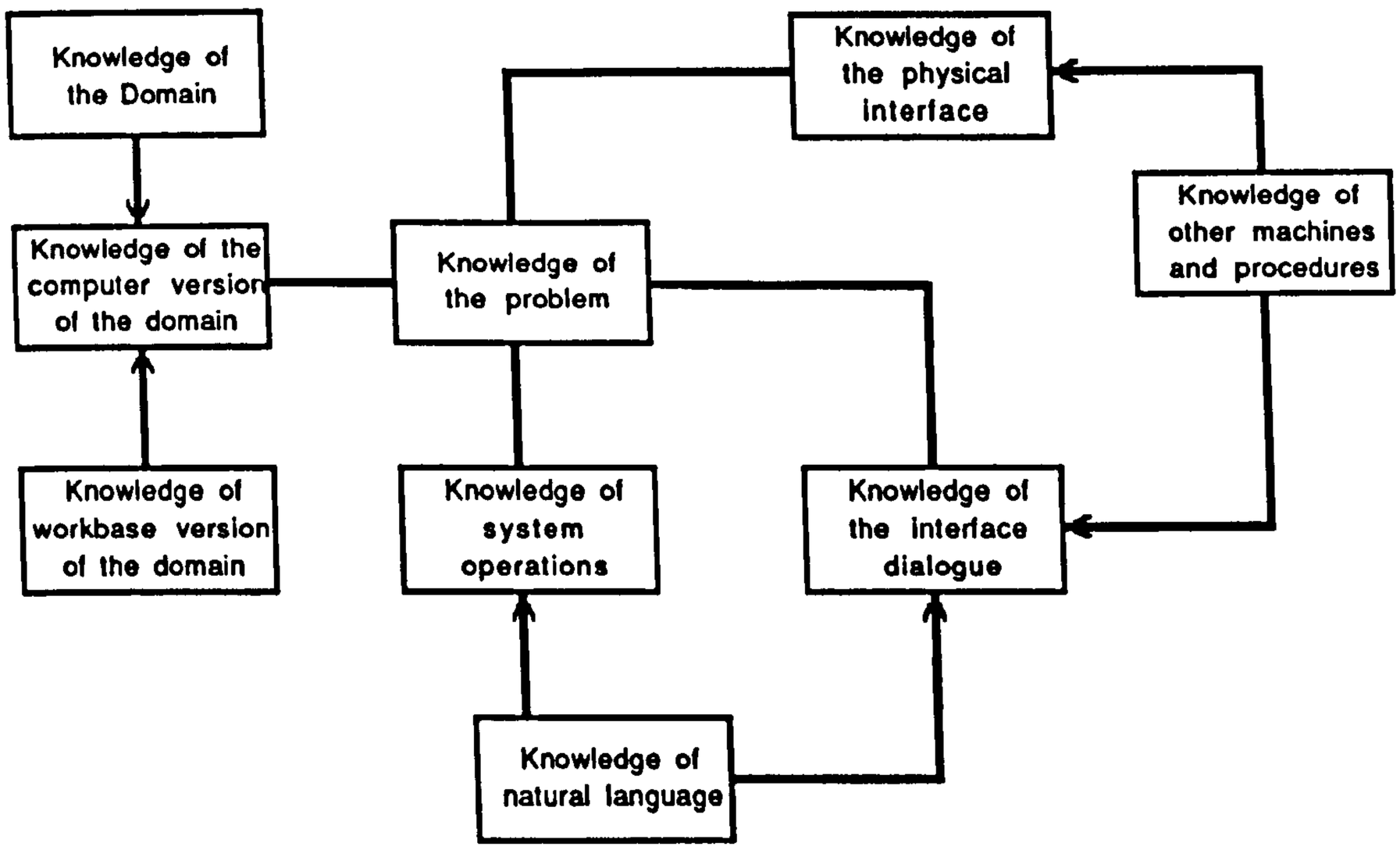


Figure 2.5.5.1. A Block Interaction Model of the knowledge a user brings to using a computer system to solve a problem (after Morton et al, 1979) in Task analysis for knowledge acquisition, from [WILSON 87].

There is a distinct need particularly at the novice end of computer usage to guide the user through an interaction session with a design tool. Intelligent front-ends have evolved to do just that.

2.5.6. INTELLIGENT USER INTERFACE MANAGEMENT SYSTEMS (IUIMS).OR INTELLIGENT FRONT-ENDS (IFE)

IFE's are a special case of "front end" or "user interface" [EDMONDS 82]. "They are distinguished from merely "rational" or well engineered front ends by their inclusion of explicit models of both the user and the task (i.e. the package or system which is being interfaced to)" [BOULAY 82]. This has been designated the general title of user modelling.

2.5.6.1. USER MODELLING

User modelling has been employed in a variety of AI fields, summarised by [WAHLSTET 89] in figure 2.5.6.1, in an attempt to predict user actions by representing their beliefs and goals within a model of formal rules. This kind of stereotypical model of the user is often utilised in a predictive explanatory role which is reflected in its current range of applications indicated below.

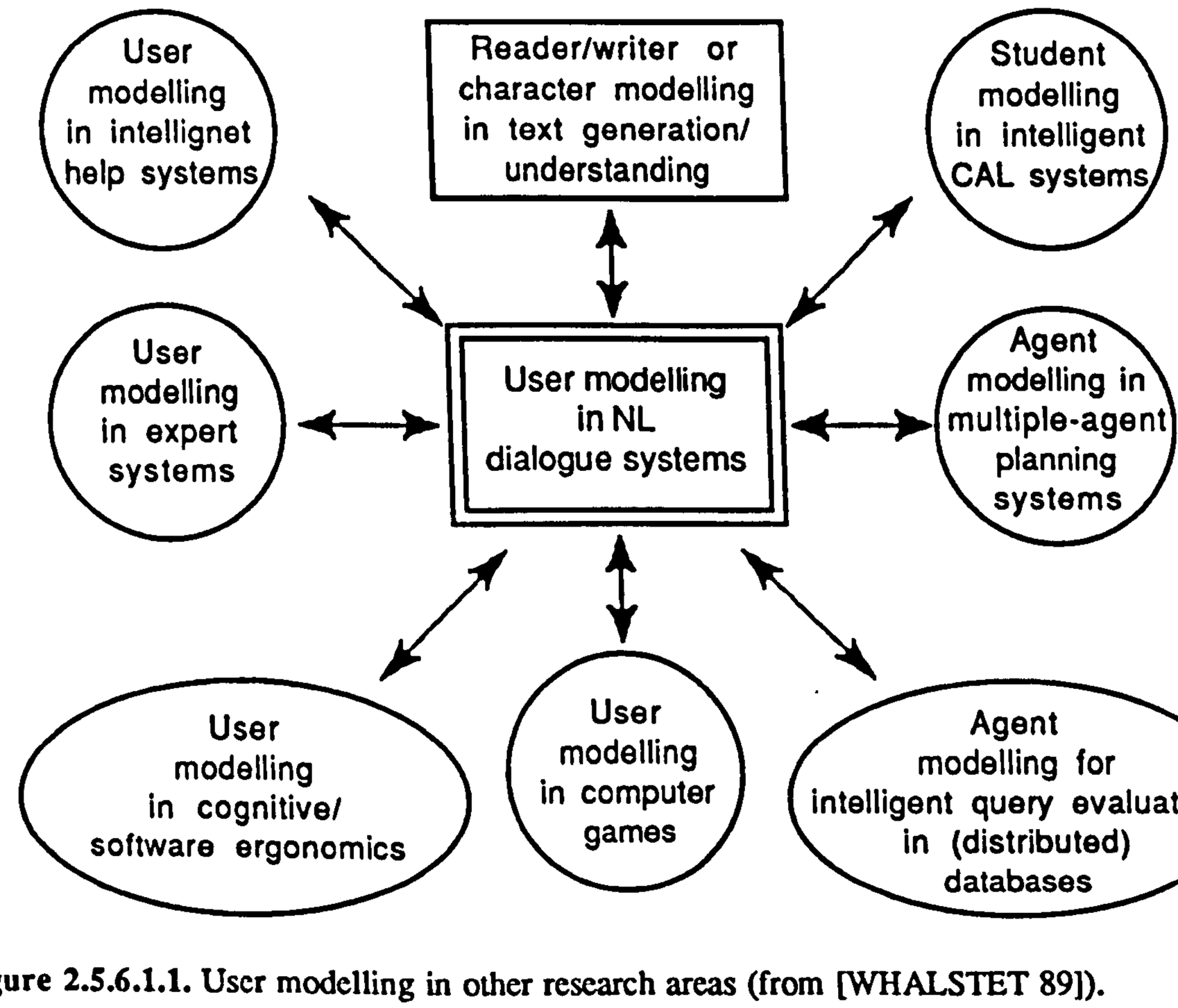


Figure 2.5.6.1.1. User modelling in other research areas (from [WHALSTET 89]).

A more recent application of user modelling, outside that of traditional AI research, has been in the field of human computer interaction. A User model (UM) in the context of

HCI has a slightly different interpretation, defined in terms of current cognitive psychology by Gent as, "a kind of mental model which persons usually form of things and people with whom they interact". [NORM 86] identifies three different types of models that are currently referred to as user models in HCI research:

i "an individual's own personal idiosyncratic model;

ii a generalised 'typical user' model that the designer develops to help in the formulation of the design model;

iii a model that an intelligent program constructs of the person with which it is interacting."

Contrastingly the AI definition does not encompass psychological issues such as mental entities but is usually [WAHLSTER 89] confined to the third type of model.

Formal representations have been proposed and are now generally accepted as a means of describing user's and task structures. These structures include:

- state transition networks,
- formal grammars, and
- production systems.

A substantial body of research has been carried out in the field of dynamic user modelling and cognitive ergonomics. This thesis does not attempt to cover the psychological aspects of user modelling in any great depth and the reader is referred to a recent publication, [KOBSA 89], which provides a comprehensive introduction into this area of research.

To summarise, [WAHLSTER 89], provides convenient definitions of both a user model and the process of user modelling which will be used within this thesis:

> *A **user model** is a knowledge source in a natural language system which contains explicit assumptions on all aspects of the user that may be relevant to the dialogue behaviour of the system. These assumptions must be separable by the system from the rest of the systems knowledge.*

> *A **user modelling component** is that part of a dialogue system whose function is to incrementally construct a user model; to store, update and delete entries; to maintain the consistency of the model; and to supply other components of the system with assumptions about the user.*

Based on this philosophy of modelling the user's beliefs and expectations a number of intelligent front ends are in existence. KNOME (KNOwledge Model of

Expertise) developed at the University of California, Berkley is the user modelling component of a UNIX consultation system (UC) [CHIN 87]. The purpose of KNOME is to determine, from a natural language dialogue, the user's level of experience within the UNIX domain and to customise a template for a particular user in order to [CHIN 87]:

"1) avoid telling the user something that the user already knows,

2) tailor explanations to the user's level of understanding,

3) utilize the user's background knowledge in interpreting what the user says".

While obviously useful, systems such as KNOME are still text based employing quasi-natural language dialogues and are obviously either incomplete in there vocabulary or incur a number of the computational overheads of natural language processing, although Chin's third point is a means of reducing deviance in the dialogue.

The process of design decision making involves visual as well as verbal (textual) descriptions. An adaptable user interface suitable for communication in design must therefore facilitate data acquisition in many different forms. Most UIMS, IUMS and IFEs are implemented within a single language

2.5.6.2. ENCODING KNOWLEDGE

There are many forms of knowledge drawn upon during the design process; absolute factual knowledge and more abstract or subjective rules referring to human nature, each requiring different forms of representation which may be broadly categorised in to the two groups [RADFORD 90]:

1) highly structured knowledge

2) loosely structured knowledge

The way in which the knowledge of the user's task is encoded determines the type of dialogue that may be supported. For example highly structured knowledge often implies procedural dialogues while a loosely structured form of representation facilitates flexible and natural volunteering of information. The latter is infinitely more difficult to achieve than the highly structured (hierarchical) method as multiple entry points must be supported and closure paths to other states must be provided. [ALTY

83] proposed the use of path algebras as a means of validating dialogue control networks to ensure that all possible eventualities (closure), resulting from multiple entry points, are taken into account. Although a little obtuse, Alty's "Path Algebras - a useful CAI/CAL Analysis Technique" provides a complete description of the application of path algebras to dialogue control networks. Once a formal specification of the UI has been generated the software implementation of the interface may be designed in a similarly formal manner. As a result the user-interface should also be more reliable. Several other methods of formalisation have been adopted such as the Z computer language [SUFRIN 86] used to formalise the behavioural aspects of the interface, and an algebraic formalism to describe a serries of interactions for a system (described in DIX and RUNCIMAN, 1985).

These methods aim to provide a finite description of dialogue behaviour. Sutcliffe [SUTCLIFFE 83] suggests that these guidelines (derived from psychology) "are heuristic and dependent on context for interpretation. This makes formalisation in their current state of precision practically imposible". Harrison and Thimbleby [HARRISON 85] have attempted to provide a reliable and context free formalism (Generative User Engineering Principles (GUEPS)) based upon generalised principles of interaction. Therefore rather than a single rule set which attempts to encompass every aspect of interface design, GUEPS, derived from general human factors principles, provide rule sets (or cause and effect statements) for each aspect of the user interface. These principles once generalised may then be encoded in an abstract model. Such an approach results in a modular knowledge based system which may be extended to accomodate dynamic user modelling.

For each of the categorisations of knowledge structure a further distinction may be drawn; indeterminate (for want of a better word), high level surface knowledge [HART 82], and deep, low level, irrefutable knowledge.

Traditional programming languages cope adequately with the latter while surface knowledge in dealing with more abstract notions is best encoded in a rule based system.

2.5.7. ARTIFICIAL INTELLIGENCE (INTELLIGENT KNOWLEDGE BASED SYSTEMS (IKBS)) IN DESIGN CONSULTANCY

Unlike traditional programming methods which are utilised to encode 'mathematical models' and the like, experts systems or intelligent knowledge based systems encapsulate the knowledge of an expert human consultant. The knowledge, held as data to improve maintenance, is typically represented as a collection of logically related rules. As a result of the highly structured nature of rule based systems the dialogue often follows procedural question and answer data acquisition sessions with the user.

2.5.8. MULTI-LEVEL KNOWLEDGE SYSTEMS

Traditional AI systems (production rule schemes) are effective in representing expert knowledge in broad domains. As Weis indicates; there are many problems for which additional knowledge, not represented by these surface models, is required. Hart [HART 82] describes a multi-level expert system consisting of surface level models with deep models of reasoning, the objectives of which are three-fold [LANSDOWN 86] (in [JEON 86]); to provide:

i) more force to explicit explanations arising from the knowledge-based system

ii) a better distinction between objective and subjective knowledge; between gueswork and more fundamental reasoning; between those elements that arise from the laws of nature and those derived from human laws

iii) more knowledge and power.

The traditional interpretation and classification knowledge-based systems (CASNET and MYCIN, used in medical and PROSPECTOR in geological consultation) are implemented as evidence gathering systems, offering interpretations when sufficient information has been submitted by the user. The knowledge elicitation procedure often takes the form of a highly structured and systematic dialogue with the user. Evidence is often in the form of personal observations but is also supplemented by instrument data. This type of information is usually interpreted or filtered through the user and is typical of many closed systems.

[WEIS 82] describes a multi-level expert system for oil well log analysis, figure 2.5.8.1, whereby information from monitoring devices is submitted directly to a central interpretation model.

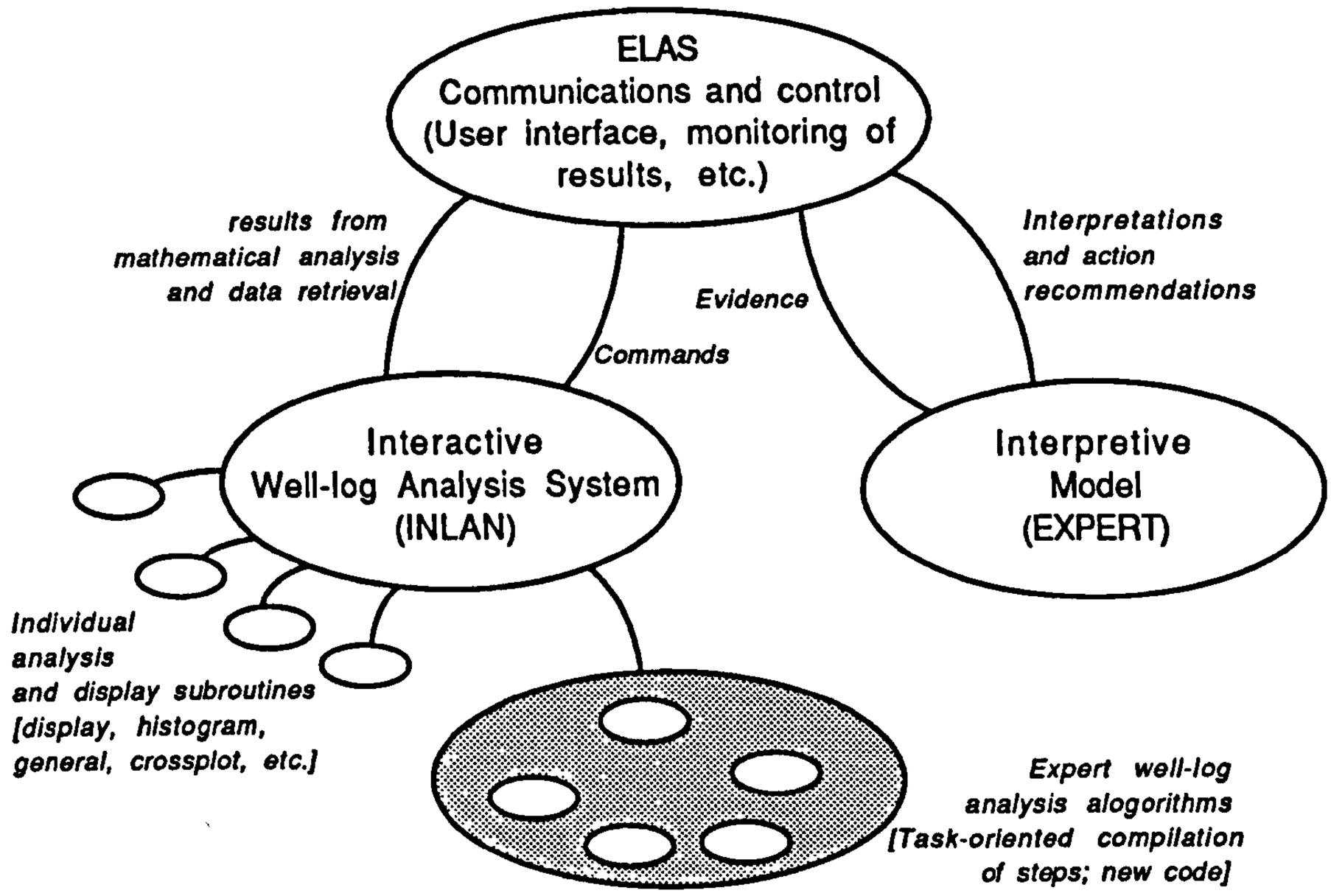


Figure 2.5.8.1. Expert Log Analysis System (ELAS): a multi-level expert system comprising surface level models with deep models of reasoning [Weis, Kulikowski, Apté, Uschold 80]

The surface model is of the production rule type while the deep model is a purely mathematical description of a physical object.

By extending this principle it becomes possible to integrate contributions from other resources or from other sub-disciplines directly in to a high level model acting as a coordinator. Any design aid must be capable of representing the dynamically varying nature of the design process.

Expert systems tend to be either forward or backward chaining systems. The solution to complex problems requires a more flexible framework. In order to provide a suitably powerful and flexible architecture, non-procedural oportunistic knowledge acquisition mechanisms are required.

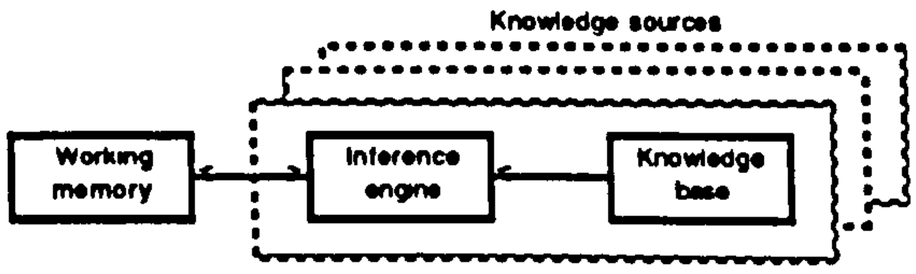
2.5.9. BLACKBOARD SYSTEMS

A result of AI research, blackboard systems have evolved as general problem solving frameworks, enabling many knowledge sources to participate in the formulation of solutions to complex problems [ENGLEMORE 88]. Traditional knowledge based

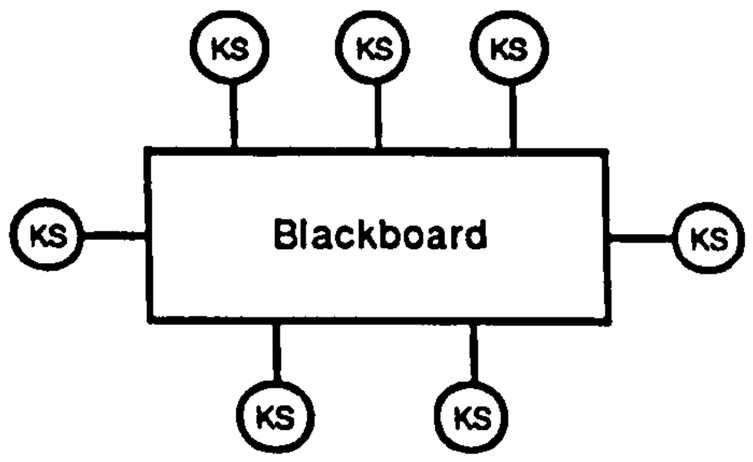
systems, figure 2.5.9.1, consist of an inference engine, knowledge bases, and a working memory for manipulating the emerging solution.

Figure 2.5.9.1. Example of the classical expert system structure

In response to the need to make knowledge engineering and problem definition more modular a common work area is provided which is monitored by several domain related knowledge based systems, figure 2.5.9.2 a and b.



a) A rudimentary blackboard model



b) independent knowledge sources.

Figure 2.5.9.2. Typical blackboard model comprising a central blackboard containing a global data base and a number of [ENGLEMORE 88].

"As there is often no control flow within this scheme of operation, the knowledge sources are self activating and respond to changes in the state of the blackboard." [ENGLEMORE 88].

Although the origins of the blackboard system stems from the early developments of the Defence Advanced Research Projects Agency (DARPA) Hearsay II speech understanding system in 1971, [NII 86] attributes the conceptual development of blackboard system to Allen Newell:

> *Metaphorically we can think of a set of workers, all looking at the same blackboard: each is able to read everything that is on it, and so judge when it he has something worthwhile to add to it. This conception is just that of Selfridge's Pandemonium (SELFRIDGE 59): a set of demons, each independently looking at the total situation and shrieking in proportion to what they see fits their nature ...*
>
> [NEWELL 62]

In order to overcome the problem of multiple resources all screaming at once, blackboard models require a scheduling protocol.

2.5.9.1. BLACKBOARD DATA STRUCTURES

The main function of a blackboard is to hold on a blackboard panel (a reserved section of volatile memory) "computational and solution state data" [ENGLEMORE 88], generated and manipulated by knowledge sources. Information required for the formulation of a solution, such as input data from sensors, partial, alternative, and final solutions together with control data, is held on a blackboard panel as objects.

Objects may be arranged hierarchically into levels of analysis, figure 2.5.9.1.1, each object containing property definitions stored as attribute values pairs, defining the vocabulary of the solution space. Named links are also used in blackboard systems to define relationships, "part of", for example, between objects on different levels or "next-to" type relationships between objects on the same level.

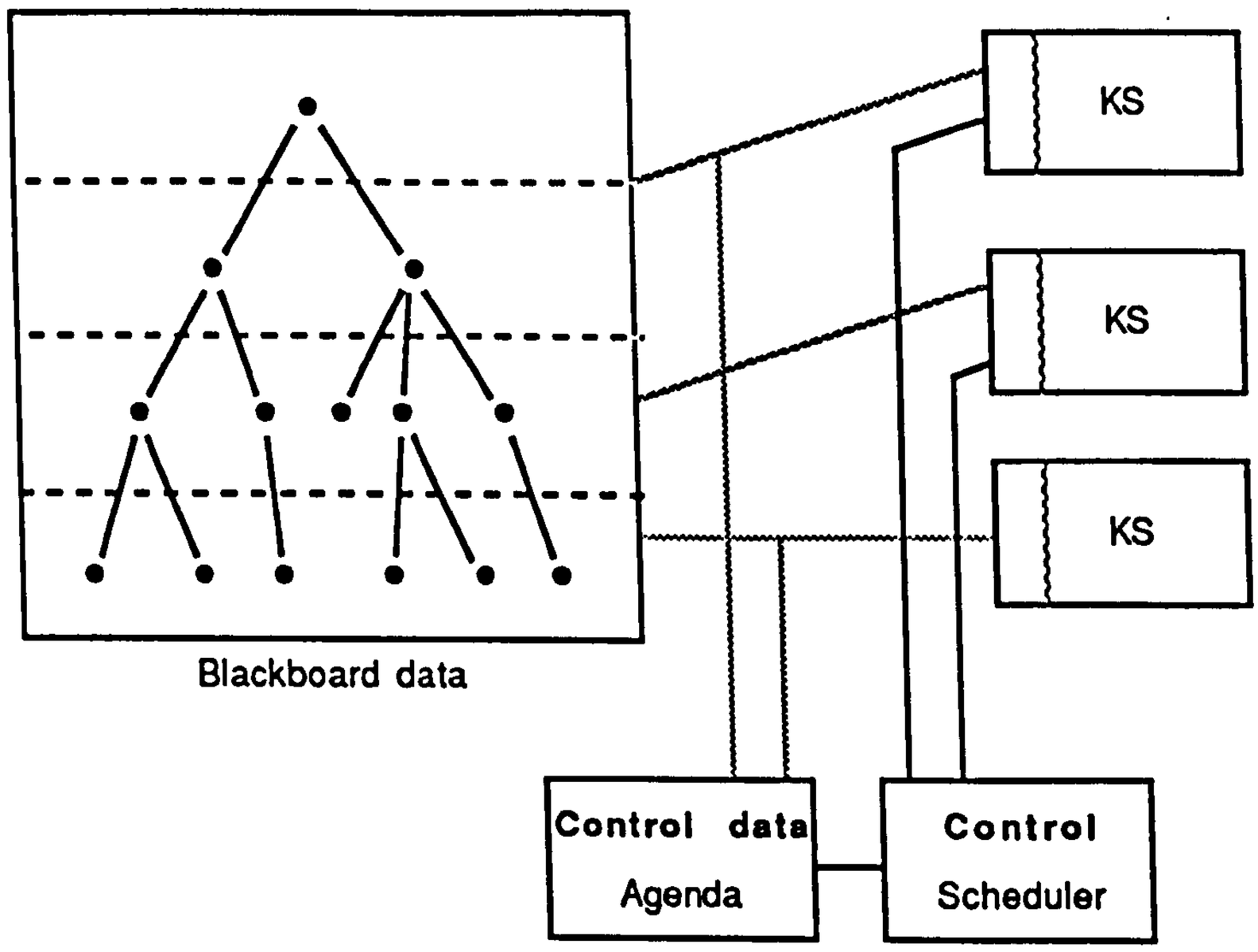


Figure 2.5.9.1.1. Blackboard control modules and oportunistic knowledge sources supported by hierarchical object layers (from [ENGLEMORE 88])..

A solution space may also be arranged into multiple hierarchies (first used in the CRYSALIS system, Englemore, Feigenbaum, Johnson and Nii, 1983) by means of stacked blackboard panels.

2.5.9.2. CONTROL AND DISTRIBUTED PROBLEM SOLVING

Although the concept of distributed problem solving is not a new one; E-Mail, for instance has given rise to shared authoring with sub-tasks being assigned to individual computer users often separated by many miles, blackboard systems provide a mechanism for orchestrating (or focusing the attention of) several problem solving knowledge sources within a single (or perhaps multiple) domains.

Typically a blackboard system has a series of control modules which monitor activity in the various levels of the solution space (or solution islands). The purpose of the control modules is to apply one of three control strategies, focusing the attention of an appropriate knowledge source, in order to resolve a pending problem [ENGLEMORE 89]:

i) decide which knowledge source to activate next

ii) decide which solution islands to pursue next

iii) decide which knowledge sources to apply to which objects

Knowledge sources are therefore said to respond oportunistically to changes in the state of the solution space.

2.5.9.3. KNOWLEDGE SOURCES IN DIALOGUE SYSTEMS

State transition networks are generally used to orchestrate computer initiated dialogues with an end-user. These are procedural in nature and result in hierarchical (context sensitive) vocabularies; the user must adhere to the dialogue sequencing encoded with in the system. Procedural networks are also inappropriate when concurrency or parallelism is required for problem solving. Oportunistic approaches may be adopted whereby information (concepts) may be exchanged between knowledge sources in an undefined order enabling dialogue(s) between knowledge source(s) (the user is viewed as a source of domain knowledge) to take place in an unrestricted, free flowing oportunistic manner.

Such an approach, offering multiple dialogue entry points must requires loosely structured knowledge bases.

Although conceptually generic, the major flaw with many blackboard systems is that they are generally task specific; varying degrees of domain specific knowledge is embedded within the control modules of the package (HARPY, for example). This is often done to optimise the performance of the application. In the case of the HARPY

system [Bolt, Beraneck and Newman], explicit knowledge was encoded into the blackboards transactional mechanisms to meet the near run-time identification of words, required by the original DARPA project brief; in so doing rendering the system inappropriate for general use. There is obviously nothing wrong with optimization provided that it is generally applicable.

This view of the blackboard system, with oportunistic knowledge sources monitoring a central data area, fits perfectly into the model of design assistance with a number of collaborating design consultants monitoring the development of a product. The emerging solution is generated incrementally resulting in reduced computational commitments. Owing to the oportunistic nature of the knowledge sources both forward and backward chaining may be intermingled. The major benefit of this is that [FEIGENBAUM 89], "... in contrast with most generate-and-test procedures, a complete test solution does not have to be built before making a decision to modify or abandon it".

2.5.10. A GENERIC SOLUTION

The advent of multi-tasking workstations and in particular the UNIX operating system, developed to improve software portability, with concurrent processing and inter-process control (IPC) mechanisms enable applications to communicate to each other provided a common syntax has been established. However such advantages have yet to be exploited in providing integrated design environments for building design.

It is clear that there are many well established design principles and methodologies that may be used to successfully develop a user-friendly interface.

Interface design, however, is a prototypical activity with new styles and methodologies introduced with each new application. Any solution must therefore take account of changing beliefs, methodologies and technical advances. This shift in standards also applies to building design and the infra-structure must be capable of accommodating these transitions in a totally transparent manner to isolate the user from any inconsistencies that may arise from the substitution of design tool modules.

One solution has been developed which encompasses all the issues involved in application development and usage (in the context of building design and appraisal) in an open and extendible manner.

2.5.10.1. THE INTELLIGENT FRONT END

The Intelligent Front-end (IFe) is a general purpose, intelligent user interface management system. In the context of human-computer interaction a system may be said to posses a degree of intelligence if it has the ability to interpret information from a range of different users and infer meaning in a particular domain of discourse while in the context of task analysis and solution planning intelligence may be defined as, the ability to apply what is currently known (or acquire appropriate information) in order to solve a particular problem

The system is composed of a series of independent knowledge sources attached to a central, dynamically partitionable blackboard. In addition to maintaining data representing the current state of the model under development (in a conceptual form of attribute/value pairs), the blackboard is also a communications centre, responsible for updating each of those clients expressing an interest in a particular area.

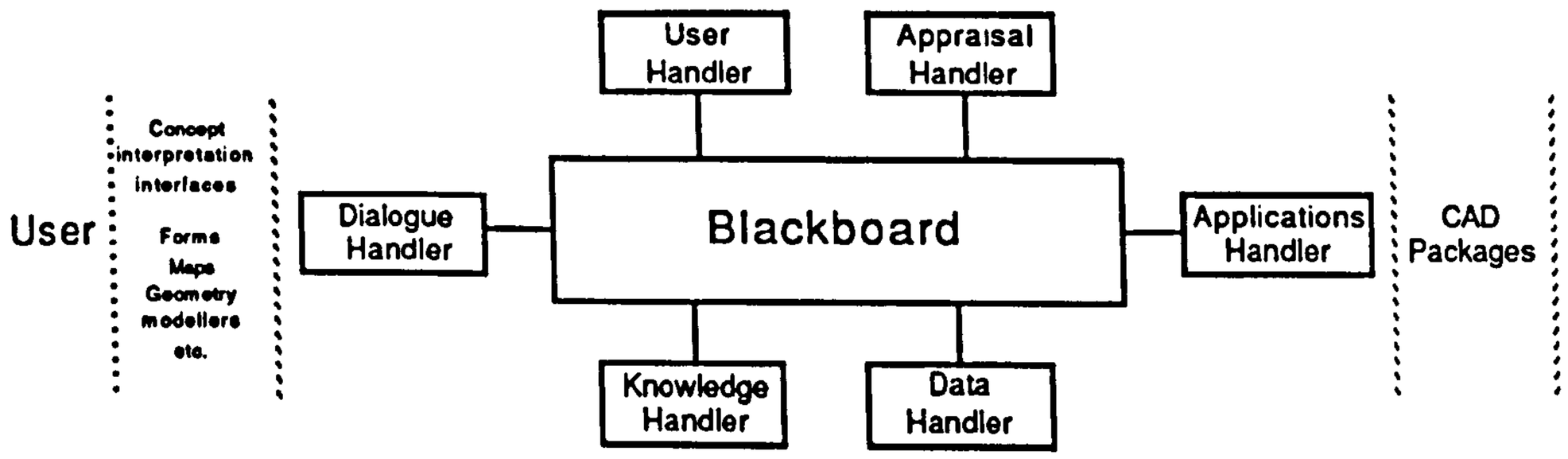


Figure 2.6.1.1. Current IFe Configuration [IFE 89]. The knowledge sources to the left are concerned with user dialogue while those to the right support the application environment.

2.5.10.2. IFE SYSTEM MODULES

The IFe system architecture is composed of a number of cooperating modules supported by a central communications module, the blackboard, illustrated in figure 2.6.1.1; each module operating concurrently. Although the blackboard may support any type of knowledge source, the following generic modules are required regardless of operational context. These cooperating modules are:

- **A Blackboard** [MacRandal], which, unlike the traditional "pigeonhole" view, serves both as a storage device and an inter-resource notifier; and therefore acts as a communications centre between the various IFe modules or clients. As yet the existing configuration does not support scheduling.

- A **knowledge handler** to interpret utterances from the user and, by inference, complete the current product description model in sufficient detail for the target application program.
- A **Dialogue Handler**, which, although it is documented [IFE 89] as being responsible for conversing to the user in a manner suited to the users level of experience and conceptual vocabulary, merely translates messages from a neutral, internal format to that required by its clients, namely; interaction modules such as the forms package [RUTHERFORD].
- A **User Handler** to track the user's progress and ensure that the system responds in a manner suited to the conceptual awareness of the user.
- An **Appraisal Handler** to coordinate the performance assessment methodologies.
- A **Data Handler** which is responsible for creating a product description, from information supplied by the user, in a format required by the application program(s) to which the IFe is interfaced.
- An **Application Handler** [RUTHERFORD] to orchestrate an application program against the selected performance methodology, and passing the required product description data.

Although within this framework developments have been undertaken in many of the modules (knowledge bases and utilities) to varying degrees, the author has made specific contributions to dialogue handling by the design and development of a multi-faceted user interface (forms) and the management of application programs by the resource handler. Other tools such as the browser and a generic multi-representataional three-dimensional viewing and manipulation environment (amongst other things) have also been developed by the author and will be described within the context of intelligent design assistance.

The actual communication mechanisms for implementing and orchestrating a dialogue with the user will now be discussed in turn.

3. THE USER INTERFACE

3. THE USER INTERFACE

A user interface is a communications buffer acting as an intermediary between a user and computer. It is the only means by which the operator and computer communicate with each other. Interfaces are inherently interactive mechanisms and should facilitate a continuing dynamic dialogue between the user and application programme.

The power and flexibility of a particular application is governed by the language embedded within its interface, which in turn, largely depends upon the nature and sophistication of the application program.

The quality of the language employed, in terms of completeness, and the ease with which the user can acquire a reasonable level of competence is often a major concern for application specialists and has lead to numerous philosophical debates regarding the appropriateness of a particular style of dialogue over another for a particular task. In general however, computer initiated, question and answer dialogues, for instance, are more appropriate for novice or occasional users; while user initiated dialogues such as command languages are more suited to experienced users of task oriented software.

3.1. COMMUNICATION

Communication is "the exchange of meanings between individuals through a common syntax of symbolism" [ENCYCLOPEDIA 74].

3.2. COMMUNICATION IN DESIGN DECISION MAKING - A GENERALISED MODEL OF COMMUNICATION BETWEEN TWO COGNITIVE SYSTEMS

In design the communication of design concepts (mental models) is of paramount importance. In the example bellow, figure 3.2.1, cognitive system 'A' wishes to describe a mental model to cognitive system 'B' in sufficient detail to enable 'B' to analyse and comment upon it.

The optimum approach is firstly to establish what concepts the receptor (B, figure 3.2.1) is familiar with within the context of the problem domain. In reality this is often achieved by establishing the receptors educational/occupational background and making the assumption that a certain basic level of understanding is attainable by both cognitive systems.

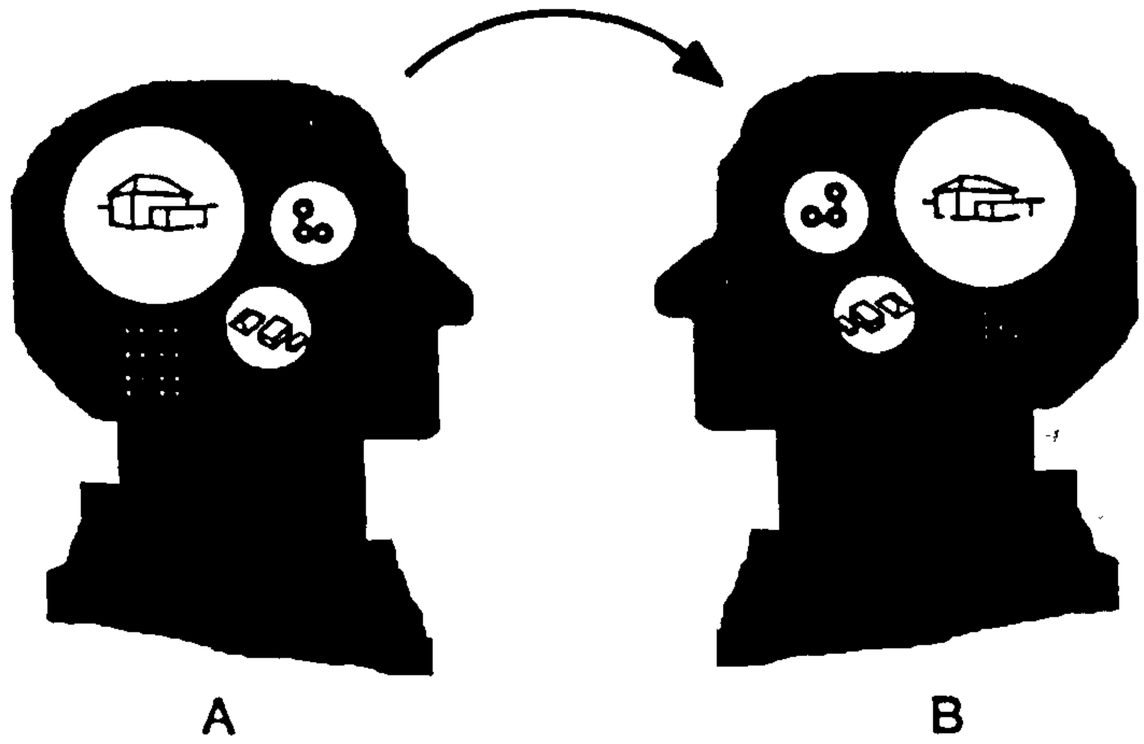


Figure 3.2.1. Concept Sharing between two cognitive systems

The mental model is then decomposed into elemental instances of concepts that the receptor 'B' is familiar with together with a communication strategy or template, logically relating the instances together (figure 3.2.2). The model is then transmitted through an appropriately expressive range of media as a logical sequence of symbols.

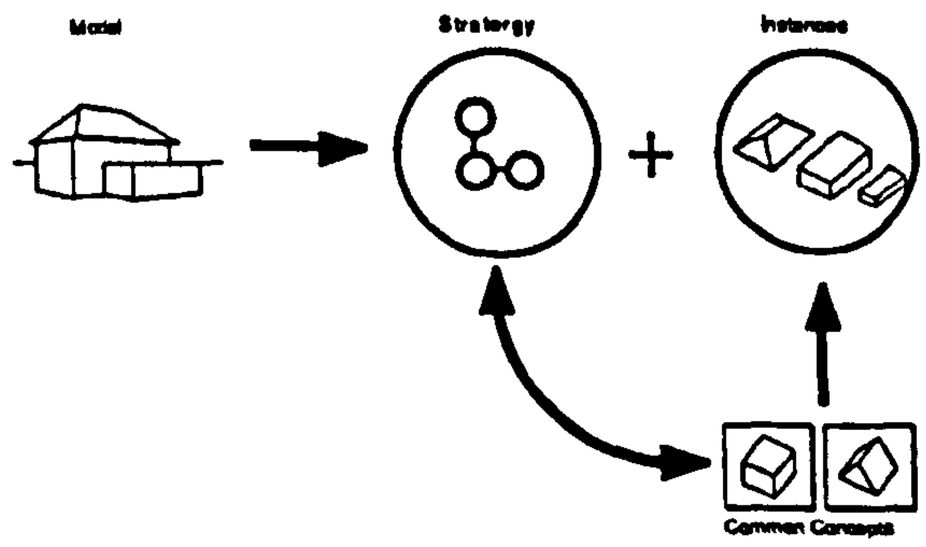


Figure 3.2.2. Strategy for the conceptual decomposition of mental models.

The receptor 'B' interprets the conceptual model filling in the slots with instances of familiar concepts, thus constructing a similar mental model. Models in both 'A' and 'B' will differ according to 'B's experiences and the precision of the initial model description. Differences between the two models (figure 3.2.1) may be resolved by the refinement of instance attributes; based upon individual knowledge and experience.

Many different strategies for one particular model are possible. No two people think alike and therefore many different descriptions may occur for the same model.

The strategy adopted depends upon the degree of commonalty between both parties level of understanding.

Problems begin to arise when a particular concept is not understood. This is resolved by continual decomposition until the sum of the elemental parts is understood. Crucial to this progressive convergence on understanding is the process of feedback.

In order to communicate an idea or concept, the recipient cognitive system must already understand the idea or concept or be sufficiently familiar with elemental components.

A mental or conceptual model may therefore be defined as the decomposition of ideas and concepts into logically related and familiar elemental concepts.

It is important to note that the definition of a conceptual model applies only to the formatting of information for the communication of ideas between two cognitive systems. No attempts are made to describe how concepts are stored in, or retrieved from human memory.

3.3. THE PROCESS OF COMMUNICATION - A GENERALISED VIEW

The actual process of transferring information and ideas between cognitive systems may be summarised as follows, figure 3.3.1:

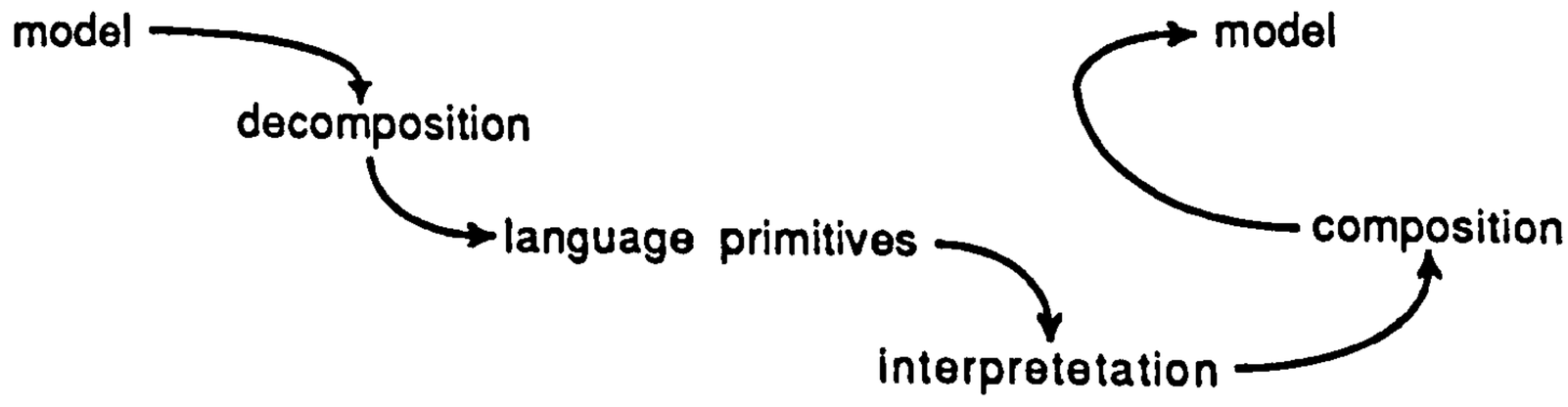


Figure 3.3.1. Transferral process

3.4. HUMAN INFORMATION PROCESSOR

In terms of memory, the process of communication may be viewed as the transferral of data from the long term memory store of one cognitive system to that of the recipient system via the short-term (working) memory of both, figure 3.4.1:

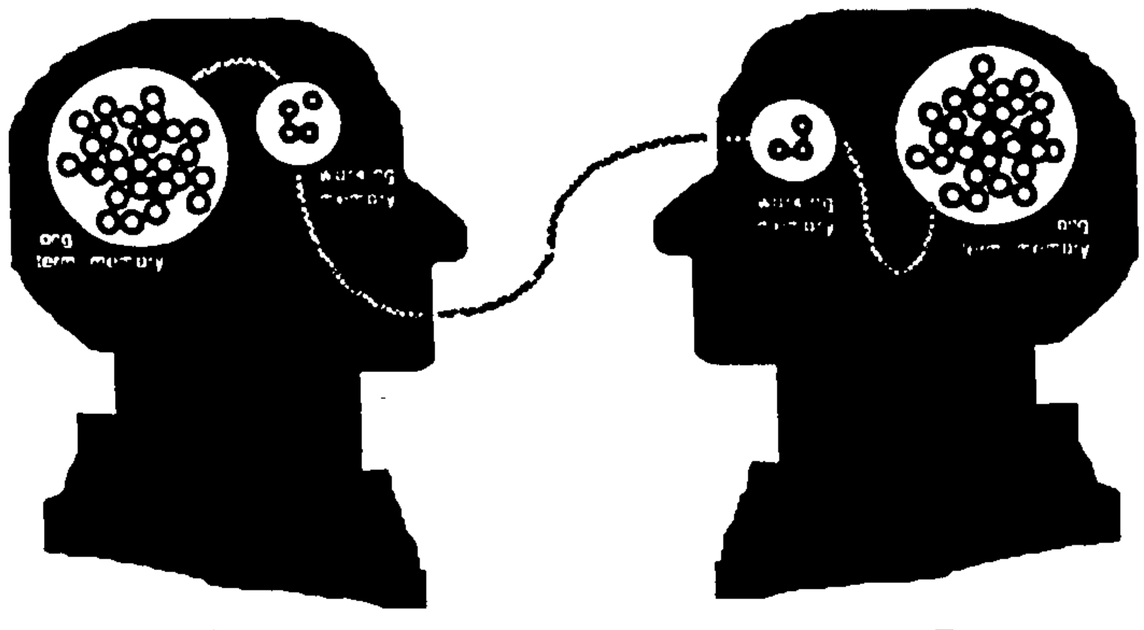


Figure 3.4.1. The relationship between memory regions and communication.

Considering communication in this form is important as there is a limit to the capacity of the short term memory region in man that severely affects his/her ability to learn or communicate [MILLER 57]. Although it is not the purpose of this thesis to investigate cognitive psychology or cognitive ergonomics, it is necessary to consider some aspects of human information processing to establish what mechanisms are at play during the process of human-computer communication.

3.5. USER PSYCHOLOGY

Spurred on by increasing pressure from users, contemporary user-interface design has triggered off a wealth of new and fresh ideas about man as an interacting part of larger systems [TOFFLER 74] and has opened up debates on virtually every intellectual discipline including man's decision making strategies, the way he learns, and the way in which he remembers. "Above all it has highlighted man's inability and unwillingness to adapt quickly enough to new and unfamiliar situations and environments, to cope with the increasingly rapid influx of information such changes impose" [TOFFLER 74]. This unwillingness may also be a measure of the amount of importance placed upon a particular aspect of technology; man will only adapt if his existence is threatened.

3.5.1. USER ADAPTABILITY

Man, as a cognitive system, is not infinitely adaptable, at least not in the short space of time imposed by the abnormaly rapid and immense acceleration in the rate of technological change currently experienced in today's computing world. "No

hyperbole can realistically describe the extent and pace of such change" [BENNING 73]. Although the biologist Julian Huxley has suggested that 'the tempo of human evolution during recorded history is at least 100,000 times as rapid as that of pre-human evolution', 'there are a few intrinsic characteristics of the inner environment of thinking man that limit the scope of thought to the shape of the problem environment' [SIMON 69]. Toffler suggests that if the current trend in the rate of technological change continues "man's inner equilibrium will be disrupted so greatly that by the turn of the next millenium, millions of psychologically normal people will experience an abrupt collision with the future".

3.5.2. INFORMATION ACQUISITION

Whether or not such predictions become reality, man's capacity to learn, and ultimately his ability to adapt to new and unfamiliar situations, in a restricted period of time, despite almost unlimited semi-permanent memory, is severely restricted by the amount of information that can be held and manipulated in the forefront of the mind at any one instant [SIMON 69]. Independent research, carried out in an attempt to find empirical verification of the effective extent of man's adaptability has revealed several limiting properties of man's inner environment that effects human cognitive behaviour [SIMON 69] and ultimately the response to unfamiliar concepts.

Experimental observations [MILLER 56] suggest that up to seven (typically four) short-term (rapid access) memory locations are set aside for simple problem solving and knowledge acquisition. The great difficulty in acquiring knowledge about new concepts is associated with the time required to transfer items of information from limited short-term memory to the long-term memory store. Unfortunately the time required to complete this information transferral is not directly related to the amount of data being moved.

In terms of recall rather than recognition, Simon suggests that "subject meaningfulness, similarity, and familiarity play extremely important roles in the time taken to actually fixate a single element of data in the more permanent regions of memory".

Ebbinghaus and Hull-hovland (in [SIMON 69] page 78), while investigating learning speed, illustrates the importance of subject meaningfulness by commenting (upon observations made) that the time taken per word to learn a sequence of prose is one third of the time taken to learn a sequence of unrelated words. Citing an experiment from B.R. Bugelski's "Presentation Time, Total Time, and Meditation in

Paired-Associative Learning" it is also suggested that a time of between ten and fifteen seconds is required to permanently fixate a single new element of information (a maximal sub-structure of stimulus [SIMON 69]), in memory.

3.5.3. CLOSURE

In order to reflect and respond to the inadequacies of human information acquisition, it is important to limit the extent of the user's task by presenting sequences of small manageable tasks. The user-interface must clearly define the user's goal within a small problem area and thus help the user achieve mental relief or closure [MILLER 56].

3.5.4. SUBJECT MEANINGFULNESS AND FAMILIARITY

The importance of meaningfulness and familiarity in the context of a particular problem domain is clearly illustrated in Simon's "Sciences of the Artificial". A single, unfamiliar element such as 'QUV' (to use his example) consists of the three chunks 'Q', 'U', and 'V', while the word 'CAT' in the context of a computing environment, for instance, may be associated with the command meaning 'CATALOGUE'. In another situation this may be interpreted as a small furry animal with four legs, a tail and whiskers. In any context such a data element is said to consists of a single chunk because of its high associative value. Being a highly familiar unit it is transferred to permanent memory more quickly than 'QUV' would be. Taking a further illustrative example of the importance of familiarity and meaningfulness in dialogue (interface) systems and communication in more general, it would be extremely difficult to fixate (in memory) the command 'ISOLG', taken from the INTERNO menu driver of ESP[1], given that it is short for 'Opaque surface short-wave flux'. There seems to be no obvious direct relationship between the command and its effective meaning, unless perhaps if you are a building and plant engineer. For the same reason, it is found that the human cognitive system has great difficulty in dealing with tasks that involve numerosity judgement and discrimination [SIMON 69], since it is virtually impossible to find familiar patterns within sequences of digits.

---

1 The example is taken from Appendix 4.1, "Interactive commands of the ESP menu drivers", page 4-32, "ESP - A building and plant Energy Simulation System", reference manual, Version 6, Release 2, Clarke, McLean, 1987. Many other examples of poor mnemonics may be found in this section.

3.6. COMMUNICATING WITH COMPUTERS AS COGNITIVE SYSTEMS

The basic philosophy of communication is true for CAD systems as much as it is true for human interaction. The great difference between human and computer interaction is that there is an opportunity within a CAD package to explicitly display the extent of the systems' conceptual vocabulary, enabling the user to structure a mental model accordingly, thus increasing efficiency by saving time resolving ambiguities between the two model systems.

In contemporary software design, this is often achieved by utilizing direct manipulation of graphical representations of data elements, operations and concepts. However, the majority of CAD systems cannot handle multiple conceptual representations and therefore users must adapt their way of thinking and design processes to match those of the package or rather those of the software designer.

The user-interface is often the product of the application developer who has an introspective and idiosyncratic view of how the package will eventually be used resulting in esoteric application programs. The advantage of UIMS philosophy is that it enables the design of the user-interface (perhaps by an ergonomist) and the development of the application source code to take place independently of each other.

Often the extent of a systems' vocabulary may be limited or different, and perhaps unfamiliar, 'words' may be used to describe the same concept. Such problems result in the system being deemed inadequate. Frustration occurs when a system continually fails to recognise a concept. For novice users this is increasingly frustrating as they often assume the recipient system to have the same level of understanding.

The actual expression of ideas is not only a means of communication but also necessary to clarify meaning and intent. Therefore in some ways interacting with any form of computer system is like communicating with oneself, owing to the level of cognitive activity required to continually map concepts from one level of abstraction to another. Often the user expects the computer to know what he/she knows.

3.6.1. A TRADITIONAL VIEW OF COMPUTER APPLICATIONS

Although a number of attempts have been made to implement voice activated user-interfaces (HEARSAY, for example), the type of dialogue between a human and a computer system is usually non-verbal. The most common form of human computer dialogue is text oriented, taking the form of a command or restricted form of natural language. Figure 3.6.1.1 illustrates the typical, cyclical nature of text oriented human-compter dialogue systems.

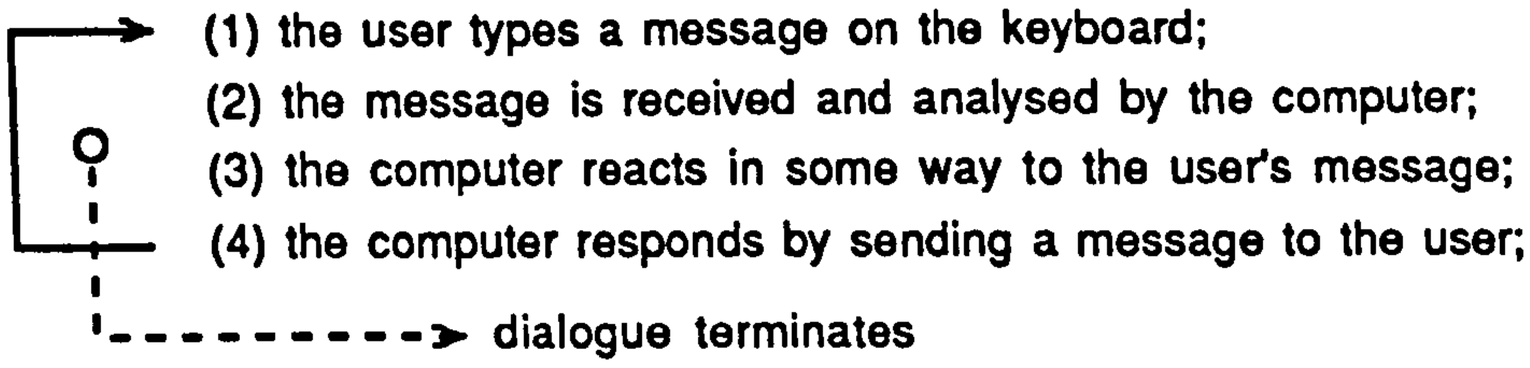
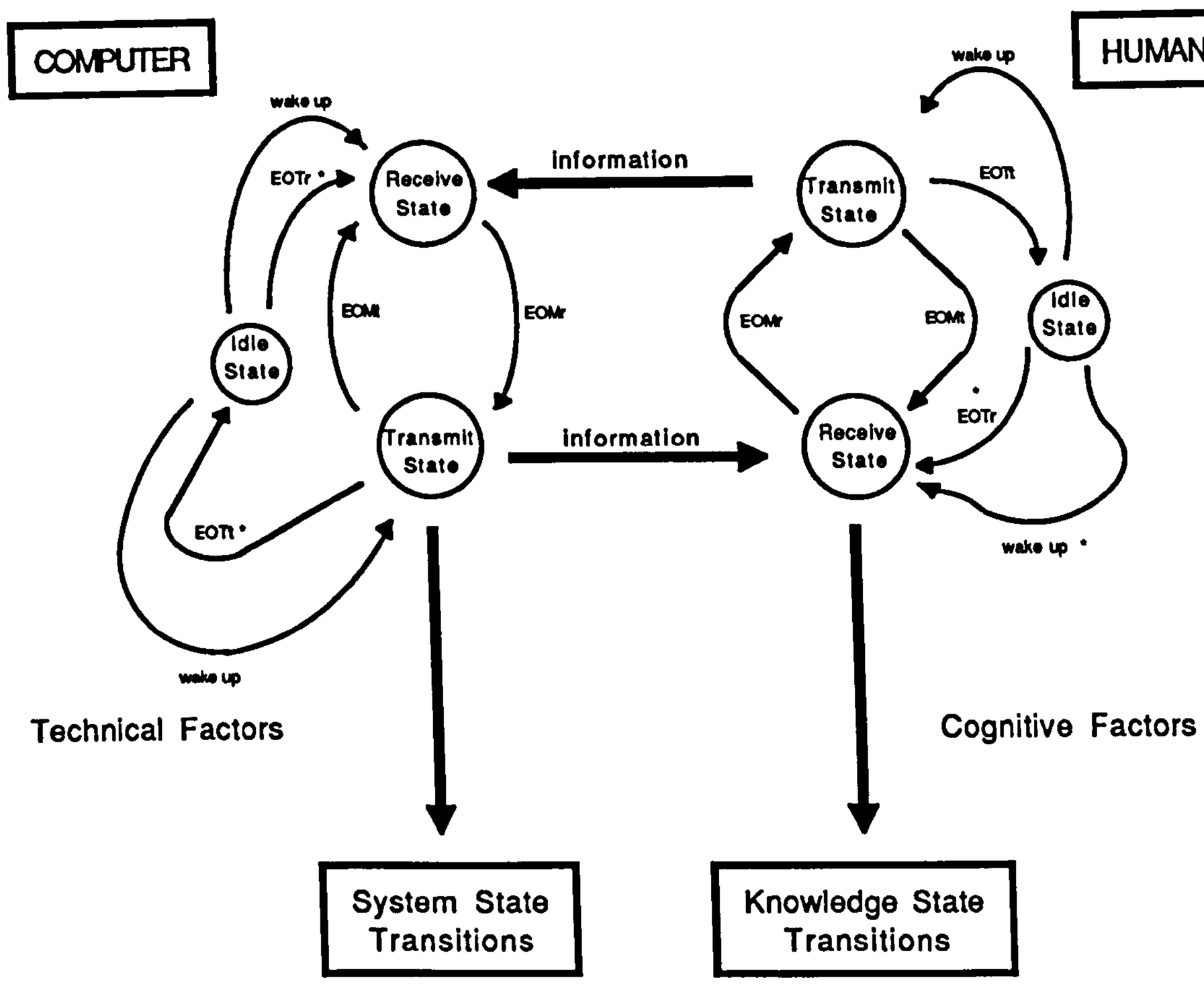


Figure 3.6.1.1. The cyclic nature of human-computer dialogue (from [BARKER 89]).

The control of the dialogue oscillates between computer and user; each becoming idle while the other is active analysing incoming information. Figure 3.6.1.2, illustrates the transition between the three states; transmit, receive, and idle, during this communication process.



EOT - End of Transimission
EOM - End of Message
(t - transmit; r - receive)
* dependant upon system semantics

Figure 3.6.1.2. The oscillatory nature of human-computer dialogue (from [BARKER 89]).

The illustration is a little unnatural in that, for example, there is no means available for any one party to interrupt the other and most forms of natural communication are bi-directional involving many communication channels (gesturing, sound and pictorial information).

Traditional CAD packages employ static conceptual representations and often uni-directional dialogues, targeting software at a particular class of user. In such instances the resulting system may either be too complex for all users to use or may be too inflexible by making general assumptions about the user's class or level of expertise. Such an approach may also force a user to think and structure their work in a particular manner (figure 3.6.1.3) - effectively creating a restrictive environment not conducive to design.

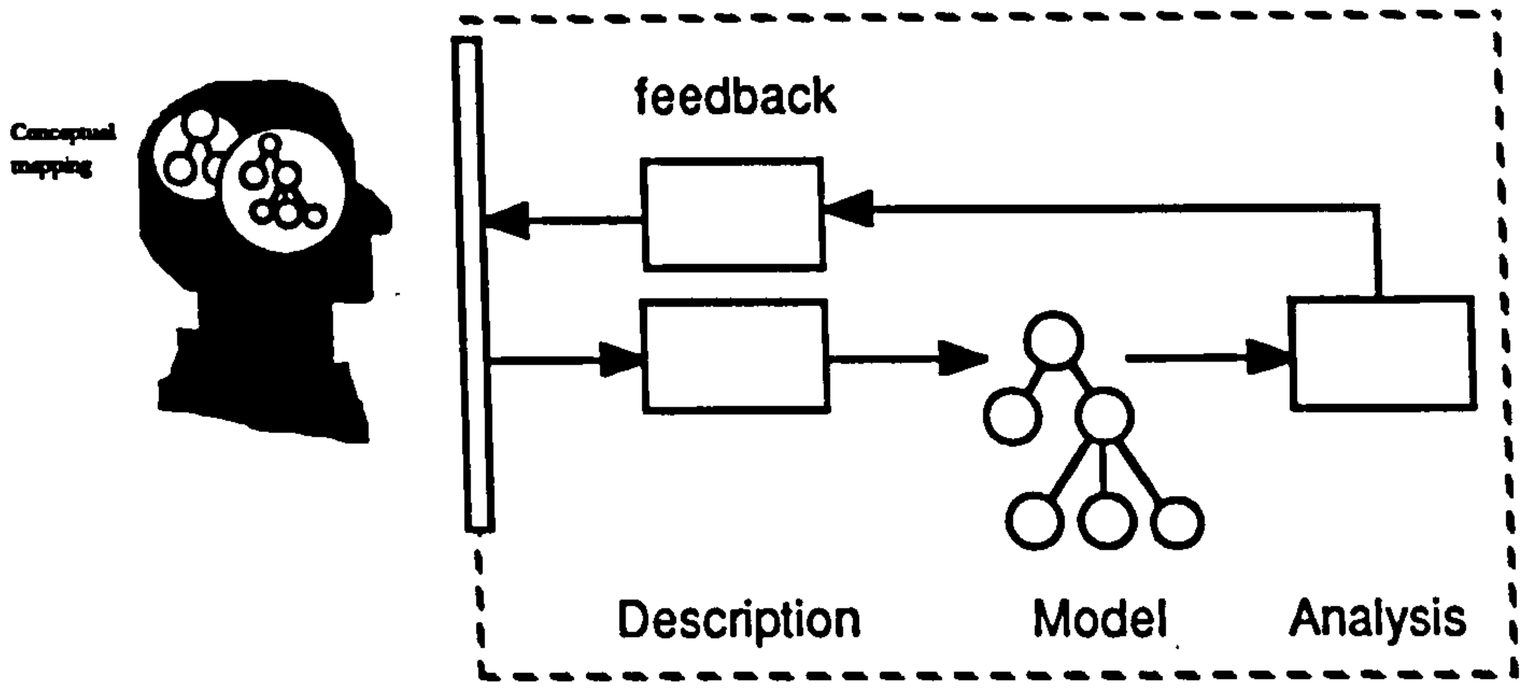


Figure 3.6.1.3. Communication overloading (Cognitive saturation).Semantics and interface intertwined.

By concentrating on one aspect of design decision making within a particular domain such systems cannot be classed as comprehensive design tools as they do not fulfil the first requirement of design process, namely creativity which, it is suggested, is supported by the unrestricted use of an unlimited range of design tools within a single environment.

3.6.2. DIALOGUE STYLES AND INTERACTION MECHANISMS

There are wide and varied range of dialogue styles and interaction mechanisms available for use in interactive systems development; falling into one of two categories:

- user initiated dialogues, such as command and natural language, are propagated by by the user, while
- computer initiated dialogues tend to fall into the procedural, question and answer category.

Each of the two basic dialogue categories are appropriate for particular situations but neither capable of handling the diversities of communication that exist between a range of user types.

Hybrid dialogues combine the features of both user and computer initiated dialogues. From the point of view of communication the combination of these two basic mechanisms results in a more natural style of interaction. An example of a hybrid dialogue would be one where either the computer sets a domain for discourse by prompting, perhaps offering several alternatives. Either of the two participants would then volunteer information, within a particular frame, when it became appropriate to do so.

3.6.3. THE DIALOGUE DILEMMA

Current user-interface design methodologies, in particular the UIMS solution, for providing user-friendly interfaces introduces two potential risks:

(1) adopting an inappropriate interface, using the lowest common denominator user level, and therefore,

(2) encapsulating the power and flexibility of applications programs within a system denying the user full access to the methodologies within the application

User friendliness is a highly subjective issue. In the context of human communication Aristotle writes:

> *Spoken words are the symbols of mental experiences and written words are the symbols of spoken words. Just as all men have not the same writing, so all men have not the same speech sounds, but mental experiences, which these directly symbolise, are the same for all, as so are those things of which our experiences are the images.*
>
> [ARISTOTLE]

This statement, reinforcing the author's view), simply highlights the distinction between language primitives being less portable than the universaly applicable domain knowledge. The separation of user-interface and domain-knowledge is relatively easy to achieve, and, while the domain knowledge remains common to all types of user, (and therefore unchanged) the user interface must, on the other hand, be capable of representing and communicating the "mental experiences" of many different users by offering appropriate symbolic representations of those conceptual models for each of the anticipated class of user.

There is a current trend in HCI research to categorise users into two groups: expert and novice[2] although Chin [CHIN 89] places users into the extended categories of: novice, beginner, intermediate, and expert. Therefore in addition to making allowance for the user's conceptual perception, a conflict exists in dialogue design; between making the interface easy enough for novice users to learn, understand, and remember, while ensuring that it is responsive and sophisticated enough for expert end-users [EDMONDS 81].

> *Experts are fundamentally different from novices; not only do they know more, they know differently. Experts perceive their field in terms of rich interrelationships rather than isolated facts and can make use of far more information than can novices. The transition from novice to expert involves fundamental cognitive shifts, analogous to those regressive stages in children's cognitive development.*
>
> [NEWTON 86].

The transition from expert to novice is described by Dreyfus (1985) as follows:

> *A novice applies context-free rules to a problem; as he learns from experience he develops rules and strategies which work, and learns when they can be applied; a true expert has solved so many problems that he is able to match a problem with a prototypical one and produce a solution without recourse to rules or strategies.*

In the case of the novice a certain amount of guidance is required to assist in his/her decision making process while interaction between experts must be sufficiently flexible so as not to obstruct the thought processes of the more proficient designer.

Regardless of a persons level of expertise, particular tasks require that people work at different levels of abstraction (from conceptual to detailed). In conceptual design many logically related issues are defined but often not attributed until later in the design process. While during detail design the issues previously defined are attributed.

Current software accommodates both of these levels of abstraction by dividing interaction into two processes:

(1) the definition of objects (concepts) in abstract terms

(2) the detailed refinement of object instances by attribution

The degree of freedom offered for the specification of objects is application and dialogue dependant; rigid conceptual definition (procedural question and answer

---

2 The term novice is used in the context of computer modelling and does not refer to the user's level of expertise in their own field of research, where they might be expert.

dialogues found in simulation packages), free specification, (user initiated dialogues in drafting packages, for example). Often the definition and attribution are inseparable as in the case of drafting packages.

The detailed refinement of object instances requires the input of specific attribute values relating to the design solution. As indicated in chapter 2, one of the major problems end-users face in using complex modelling software is the sheer amount of data required to describe a building and manipulate the model [IFE 89]. Gathering and entering this data is a time consuming and error prone activity. In many situations the designer is unable to specify all the information required to run the application program. Traditional software systems offer little in the way of assistance or intelligent defaults. The typical procedural question and answer dialogues employed provide little in the way of error recovery and backtracking. There are currently no CABD software packages available capable of providing contextually relevant defaults or assistance or more natural, non-procedural interaction control.

Consequentially, there is an obvious need for the user-interface to dynamically follow the shift in the user's level of expertise and facilitate the use of different styles of interaction to match the user's conceptual vocabulary and the task abstraction level, providing intelligent, contextually sensitive defaults and assistance.

3.6.4. COMMAND LANGUAGE INTERFACES

Command languages have been widely used in human-computer systems stemming from the availability of the keyboard. Essentially a command (an alpha numeric string, terminated by a suitable symbol, such as a carriage return) is typed at the keyboard. The string is the parsed and and analysed by control software embedded within the application code. The application is responsible for checking the syntactic and semantic validity of the command within the context of the application domain.

Often commands are context sensitive leading to a form of meta-notation [BARKER 89] composed of reserved symbols and words. The position of keywords and arguments is often critical and the user must remember the correct syntax and ordering. In order to alleviate the burden on a user's memory a the natural language solution attempts to equip the computer system with a more comprehensive vocabulary and knowledge of grammatical constructions.

3.6.5. NATURAL LANGUAGE - A COMPLETE SOLUTION

One of the aims of artificial intelligence in HCI is to enable the user to communicate with application programs in a natural and familiar language, using multi-input devices (keyboard, pictures, speech, etc). While this is, theoretically, an ideal solution such an approach has to be capable of interpreting all the syntactic and semantic constructions that occur in natural language and encompass a wide range of grammatical deviances. "As lexicons, grammars and semantic knowledge bases increase in size to accommodate wider classes of user" [LEHMAN 88] and more grammatical deviances are searched failing exact parsing, not to mention errors in input, a significant increase in response time is inevitable owing to the resolution of unavoidable ambiguities.

Most natural language interfaces are coupled to speech recognition systems (HEARSAY II, HARPY) which have to take account of numerous other aspects of communication including: the psychology of the speaker, semantics, rules of discourse, syntax, lexicons, the prosodic system, the phonemic system, and the speaker's articulatory apparatus [NEWELL 75, ERMAN 88] in addition to eliminating extraneous sounds, and all to be achieved in near real-time. In text oriented natural language systems the resolution of any ambiguity in the dialogue that may arise will undoubtedly involve a significant amount of time wasting user/computer dialogue, and ultimately involve more typing [SOMERVILLE 85].

This form of personification, it is suggested, also leads the user to believe that the computer system is capable of recognising and interpreting everything that is said. This is obviously not the case and the user will undoubtedly soon looses confidence in the dialogue. This is also a criticism of verbose or literal (graphical) interfaces such as HyperCard where anything within an image may be sensitized.

3.6.6. ADAPTABLE HUMAN-COMPUTER INTERFACES AND USER MODELLING - A LANGUAGE VIEW OF COMPUTER APPLICATIONS

Despite the claims (propagated by the term) of natural language interfaces, early attempts were tailored to specific domains of discourse ranging from pay-roll data acquisition to aircraft maintenance and on board voice activated weapon control systems. The rather unnatural dialogues initiated by these systems prompted, in the early eighties, the development of a new class of task oriented dialogue system (based upon user modelling) capable of actively participating in a conversation with the user

and predicting, from the user's questioning strategy, their intentions within a particular domain of discourse (KNOME [CHIN 87]for example).

Therefore, rather than expecting the user to modify their conceptual perception and adapt to a particular style of dialogue or interface or even expect a developer to provide an interface capable of complete linguistic coverage a more sensible and feasible approach is one of producing an interface capable of adapting to the user's own linguist usage patterns.

In order to achieve this more natural form of conversation an internal representation of the users conceptual awareness together with their goals and beliefs [WAHLSTER 89] and a strategy for determining and modifying the model is required.

An approach similar to that adopted in person-person communication [ROSC 83] is employed, whereby human categorisation, or stereotypes, are used to organise inferences concerning other people.

Once a user has been associated with a particular stereotype, by closely observing and monitoring feedback from the user, matching responses to reference templates within the stereo-type, it is possible to predict and anticipate a user's expectations. This is known as user modelling.

Any user interface developed must therefore provide mechanisms to enable a user model to dynamically modify the style of dialogue rather than the suspended-time editing approach of some UIMS.

3.7. AN ADAPTABLE USER INTERFACE IN CONTEXT

Figure 3.7.1, below illustrates a system whereby the user's conceptual model is represented within the computer environment, thus enabling the user to communicate in a manner suited to his/her level of conceptual awareness. The conceptual models of a number of stereotypical user types may be 'coded' and dynamically interchanged at the dictates of a user model as discused in chapter 2.5.6.1. This achieves the separation between language primitives and generic domain knowledge.

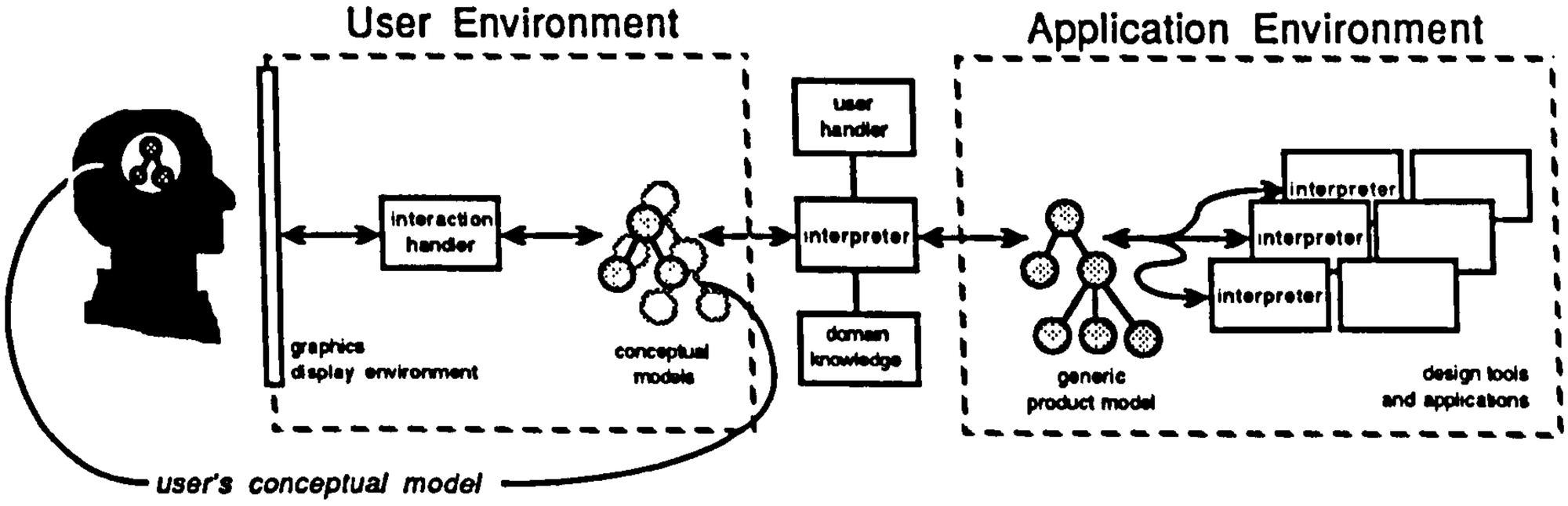


Figure 3.7.1. User adaptable system (Streamlined information transfer)

By combining a language approach to user interaction providing basic language primitives together with user modelling to control and manipulate these primitives a truly adaptable graphical user-interface becomes feasible.

In this model of communication the elemental communications or language primitive is the concept. The properties of a concept are totally generic:

- A concept is a sufficiently high level abstraction common in all design areas.
- Concepts are universal - they may be exchanged between systems.
- A concept may be interpreted differently by different systems.
- A concept may have many different representations depending upon the contextual language employed.
- Any system employing conceptual models is capable of representing any product.

Any approach adopting conceptual models as a medium for communicating information will apply equally well in other domains. The major problems of this approach are that:

- users expect recipient systems to have the same level of conceptual awareness as they do,
- frustration occurs, leading to rejection, when the recipient system fails to understand a particular concept,
- the majority of CAD systems, unlike the human cognitive system, are not designed to learn or acquire knowledge about new concepts or accept decompositions of a given concept.

It is important to note that no attempt is made to synthesise human cognitive processes within the computer. As True (1975) suggests, "The basic objective is not to cause the computer to simulate human conceptualisation and the cognitive processes leading to communication. After all, the inside of a computer is still different from the human mind as it was when Simon and Newell (1964) described it. Instead, in considering the man-computer partnership, and given the current states of knowedge about the functionings of man and the computer, the objective is simply to design the "computer-understood" command language to be more conducive to man's conceptualisation and formulation". Here there is an oportunity to develop dynamic, idiosyncratic dialogues (up to a point) with a range of users. In this situation Guedj (1980) suggests that "the role of the computer is to accept expressions with their intended meaning ... and tell in such a way that the human reciever (or group) finds the computer message sufficiently intelligible and important so that the dialogue may progress". Therefore the main form of communication from a computer system is more related to the concept of telling [LEWIS, COOK 69].

3.7.1. SEMANTIC VARIATION

In normal conversation the resolution of ambiguities between conceptual models is achieved by re-description and re-phrasing. Crucial to this iterative process is feedback. In any form of communication feedback (continual acknowledgement and questioning) is essential in order to perpetuate the dialogue; ensuring that concepts are understood.

It is particularly necessary in a design environment where new concepts are being introduced and are continually evolving. Cultural differences between individuals must also be accommodated.

If a static descriptive language is employed to communicate concepts, in situations where ambiguities arise, such languages become inappropriate; failing to provide alternative descriptions and therefore the receptor fails to extract meaning and the dialogue fails.

This is an inadequacy of traditional graphical interface design methodologies. Only in some natural language dialogue systems [TAILOR] "is the acquisition of an appropriate bias for inductive concept learning" [UTGOFF 82] catered for.

As building design utilises a rich graphical vocabulary natural language is obviously inappropriate; an adaptable graphical, user interface is therefore required.

If such an interface is to accommodate a range of dialogues it must also exhibit the flexibility of natural language. In order to achieve such a free transferral of information and meshing of conceptual models between different classes of user and computer models, the interface must be capable of two things:

- *dynamic re-description* by further conceptual decomposition (elaboration) or contraction,
- *dynamic re-phrasing* by using different "words" or symbols to describe or present both the meaning and value of a particular concept.

Both of these processes have not yet been represented by contemporary interface design which has until now been concerned more with providing tools and libraries to aid software engineering.

3.7.1.1. RE-DESCRIPTION OF CONCEPTUAL MODELS.

The re-description of a conceptual model is achieved by replacing a single concept with a meta-concept, figure 3.7.1.1.1, containing any number of logically related concepts and meta-concepts.

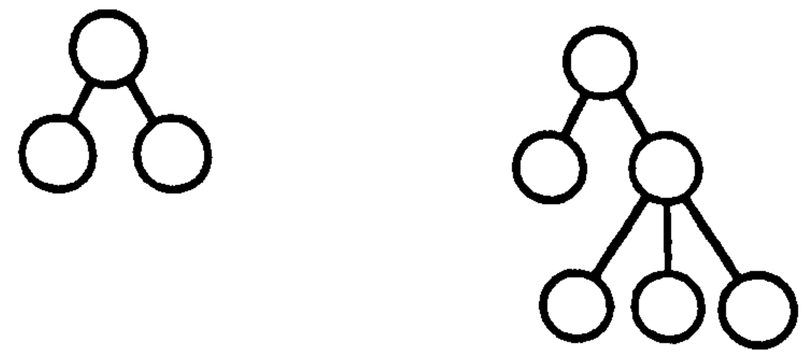


Figure 3.7.1.1.1. Expansion (elaboration) and contraction of conceptual models.

By employing dynamic memory allocation and de-allocation this may be achieved, thus enabling the interface to respond to more or less detailed descriptions; allowing the model to 'evolve' naturally. The model would expand or contract as dictated by a user model (monitoring the user's dialogue), guided by knowledge of the domain.

3.7.1.2. RE-PHRASING

Given that a concept is a single element of data the meaning and value of which may be interpreted in different ways by different cognitive systems, for a concept to be understood by a cognitive system it may be necessary to decompose it, as above, or to provide an another interpretation of its meaning or value. This is achieved by employing a generic form of storage (character string) and by attaching an appropriate interpreter. The interpreter extracts the data form the concept and presents it in a particular way. Since the storage format employed for the concept is standardised many different interpreters may be developed and interchanged dynamically to suite the conceptual vocabulary of the user (figure 3.7.1.2.1).

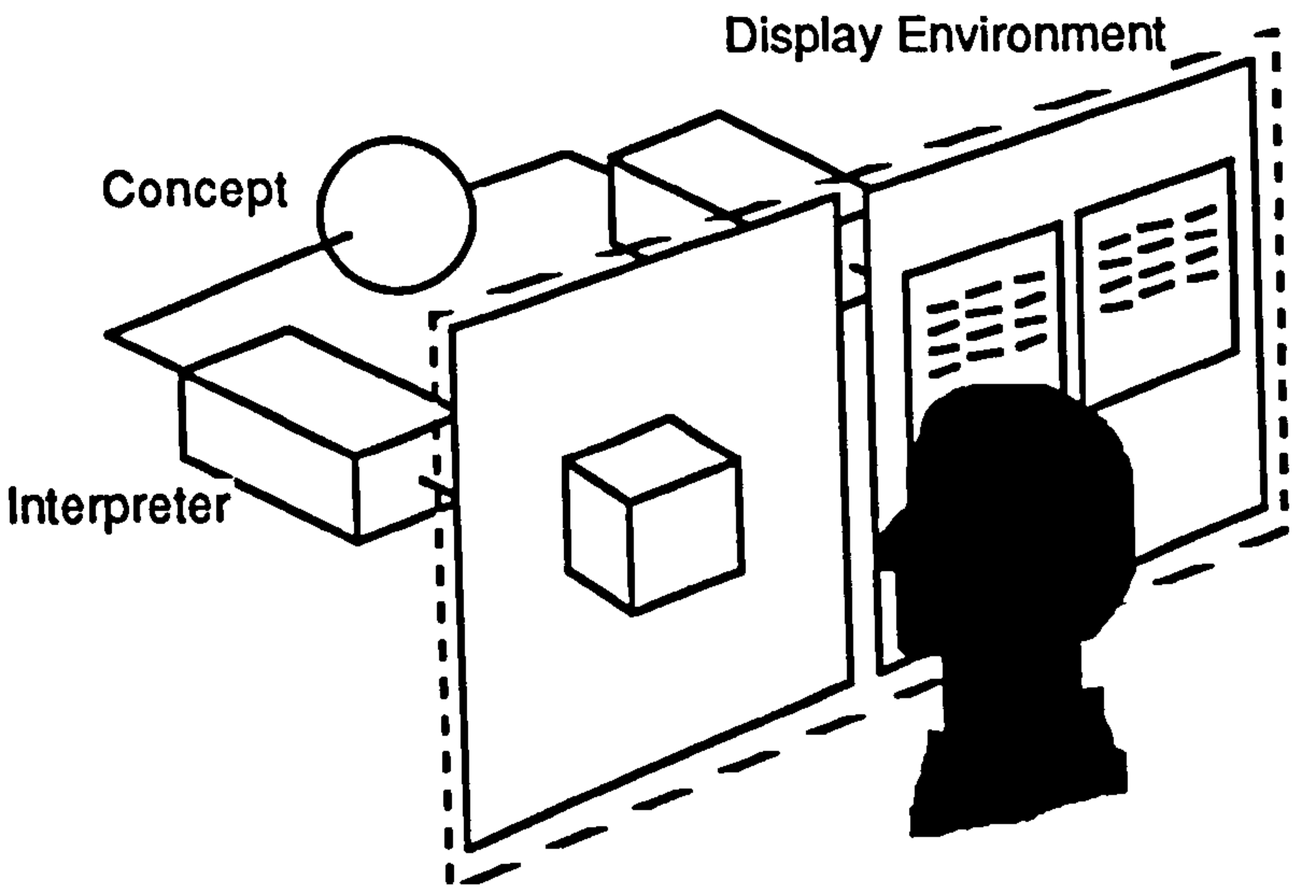


Figure 3.7.1.2.1. Conceptual re-phrasing by the dynamic substitution of concept interpreters resulting in a multi-representational system.

The interpreter may employ different words to describe a concept or may employ a particular style of symbolic representation. In addition to presenting ideas to the user in different formats, the interpreter also provides the user with different interaction mechanisms for manipulating the concept value, the most appropriate being chosen for a particular job.

A single system may also have multiple interpreters for the same class of concept. For example detailed and schematic representations of objects within a modeling packages.

By combining a dynamic n-level hierarchical architecture with a morphological descriptive mechanism it becomes possible to represent or describe anything. This

approach has been utilised in the development of a dynamic multi-representational / mult-faceted user-interface (forms, chapter 4); adopting a restricted form of natural language as opposed to a fully natural language with all the ambiguity resolution that entails.

Another function of the concept interpreter is to format events specified by the user (operations on the value of the concept) into a natural or neutral language utterance. The user's conceptual model is then translated into one that may be understood by an appraisal package or packages. The actual mechanisms for implementing a dialogue system in this manner are discussed in chapters 4 and 5.

To summarise, less static representations of the user's conceptual vocabulary and perception are required. Greater flexibility in the dialogue system is achieved by dynamically changing the conceptual template (user conceptualization). While templates provide the basic structure of a conceptual model the re-descriptive and re-phrasing mechanisms, described in this chapter, enable the model to be fine tuned or customised to the needs of an individual user. The issues involved with choosing alternative descriptions of concepts is discussed further in chapter 5.

The three basic components of a dialogue system are:

1) a dialogue toolkit containing a number of generic language primitives,

2) a knowledge base containing rules and knowledge of the user, as an interacting part of the system, in order to orchestrate instances of language primitives

3) knowledge of the users task and domain of discourse.

This three level view fits conveniently into the general IFe infra-structure, illustrated in chapter 2.

3.7.2. A GENERIC ARCHITECTURE

The strategy is one of representing and manipulating conceptual models. Conceptual models, by definition are hierarchical and therefore a network of nodes is used to represent such a model in the working memory of the user-interface. By adopting a language view of the user-interface as opposed to considering the functional requirements of interface design tools (UIMS) a generic hetra-archical framework for implementing an adaptable user interface has been identified.

An object oriented approach has been adopted whereby a generic object, figure 3.7.2.1, with which the user interacts, formats events from the user into a high-level neutral language utterance which is then passed out of the interface environment to the working memory of the IFe for interpretation.

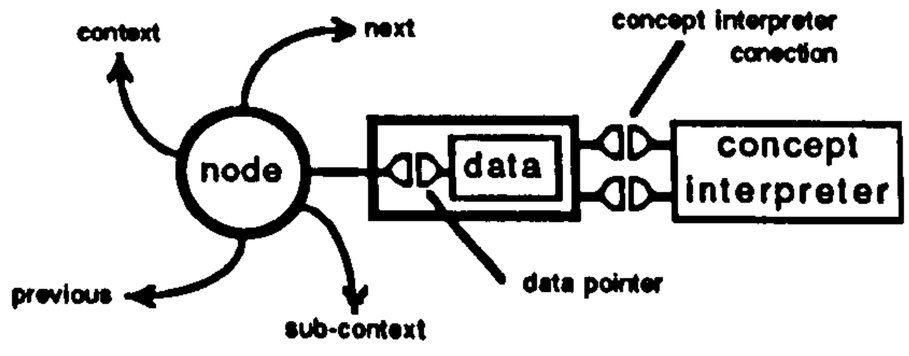


Figure 3.7.2.1. A generic concept interpreter.

Each concept category is represented by a concept interpreter which, in object oriented terms, contains methods for both displaying and modifying the information embodied within the concept. From the systems viewpoint, each concept interpreter responds and behaves in exactly the same manner; allowing concept interpreters to be interchanged dynamically whilst maintaining a generic command syntax. This ensures that, regardless of the interpreter employed to represent a particular concept the utterances from the user in the form of events remain constant. Therefore unlike traditional approaches to interface design the controlling knowledge base will remain unaffected by any changes made to the interaction layer.

In addition to the pointers required to establish a relationship between other nodes (context, sub-context, next and previous) a node contains pointers to:

- a concept held in an instance library (a simple linear linked list), figure 3.7.2.2, and
- an interpreter, figure 3.7.2.1, which is used to both display and manipulate the contents of the concept.

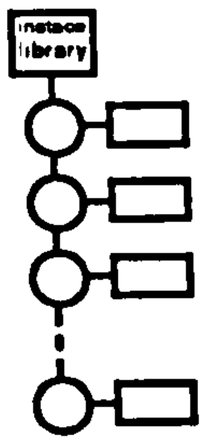


Figure 3.7.2.2. Instance library.

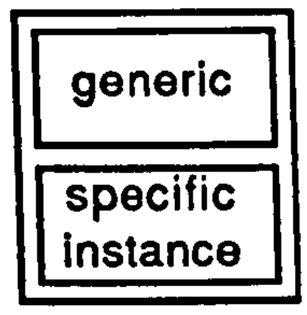


Figure 3.7.2.3. Generic and specific instance data.

Data is categorised into two sections, figure 3.7.2.3:

- generic data: this is specific to the display environment and the physical properties of graphical primitives used to represent concepts, and
- instance data: concept name, value, defaults, examples.

3.7.2.1. INTERFACES

Conceptually the concept interpreter plugs into the concept data cell and extracts information, as required, from both regions. As will be discused further in chapter 4, there are two aspects to systems capable of representing conceptual models:

1) description (or specification) of user templates and,
2) the dynamic manipulation of those models.

Owing to the different systems involved in the design and manipulation of the end user-interface, a different interface is required for each. These are identified and discussed in the following chapter. However, in order to facilitate both of the requirements, identified above, three interfaces are required, illustrated in figure 3.7.2.1.1:

1) a template interface to facilitate the initial definition of the conceptual model (held in a file),
2) a presentation interface which enables the end-user to access and manipulate the value of the concept,
3) a command interface facilitating the manipulation of the conceptual model by an external knowledge source using a series of generic control mechanisms.

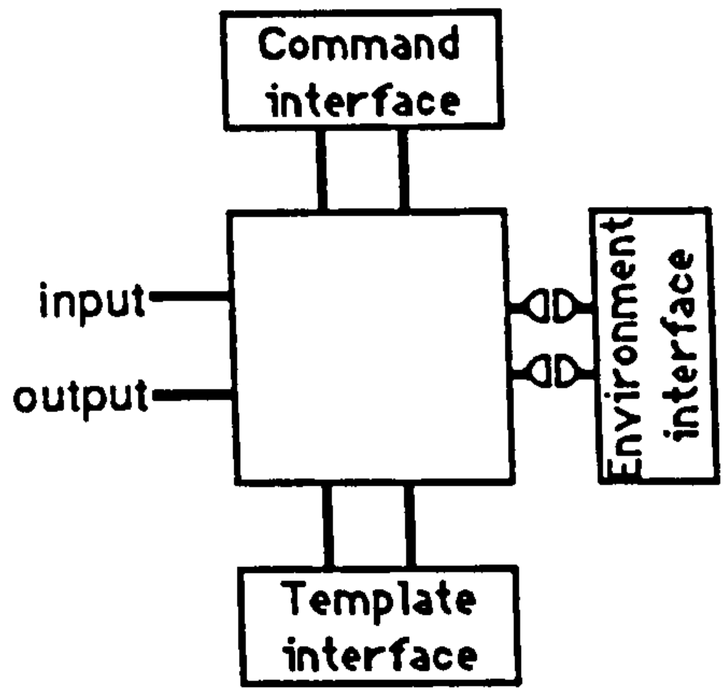


Figure 3.7.2.1.1. Conceptual interpreters.

The relative properties of each of these interfaces is discussed in more detail in the following chapter. Interpreters fall into one of two categories:

i) *Concept* interpreters: those representing individual concepts, and

ii) *Meta-concept* interpreters: those representing meta-concepts; in which case the reference to data points to a list of logically related concepts or meta concepts.

Using combinations of these two entities conceptual (hierarchical decompositions) models may be constructed, figure 3.7.2.1.2.

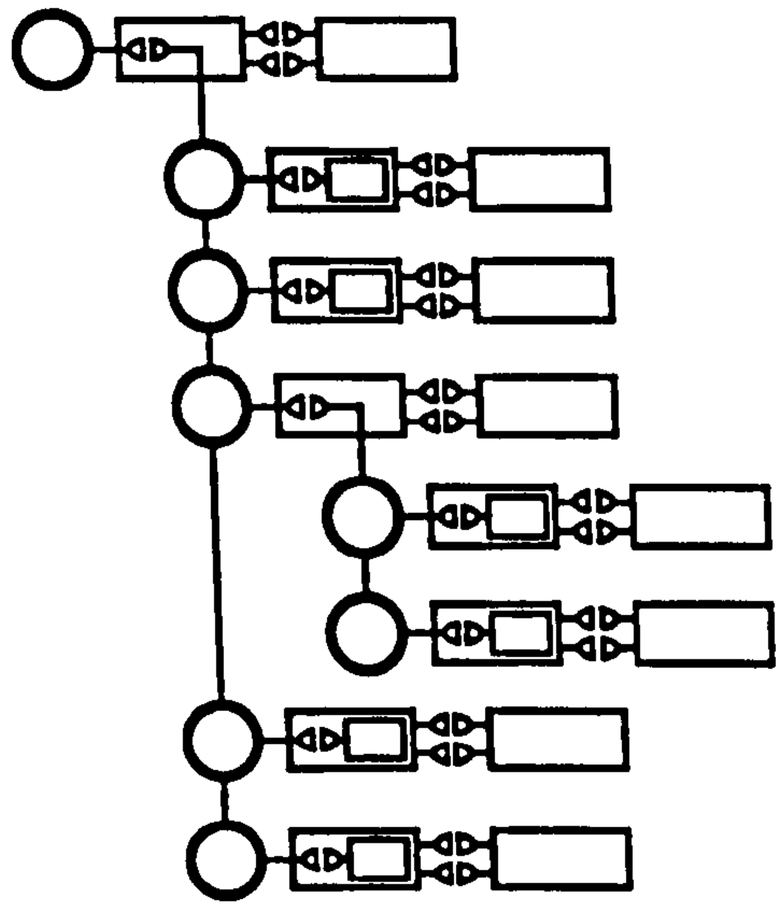


Figure 3.7.2.1.2. Dynamically alterable n-level hierarchical architecture

The actual internal data structure for this type of user-interface is therefore partitioned in to two:

i) an instance library, figure 3.7.2.2, containing the actual concepts, and

ii) a template stack, figure 3.7.2.1.3, with each node pointing to an entity in the instance library.

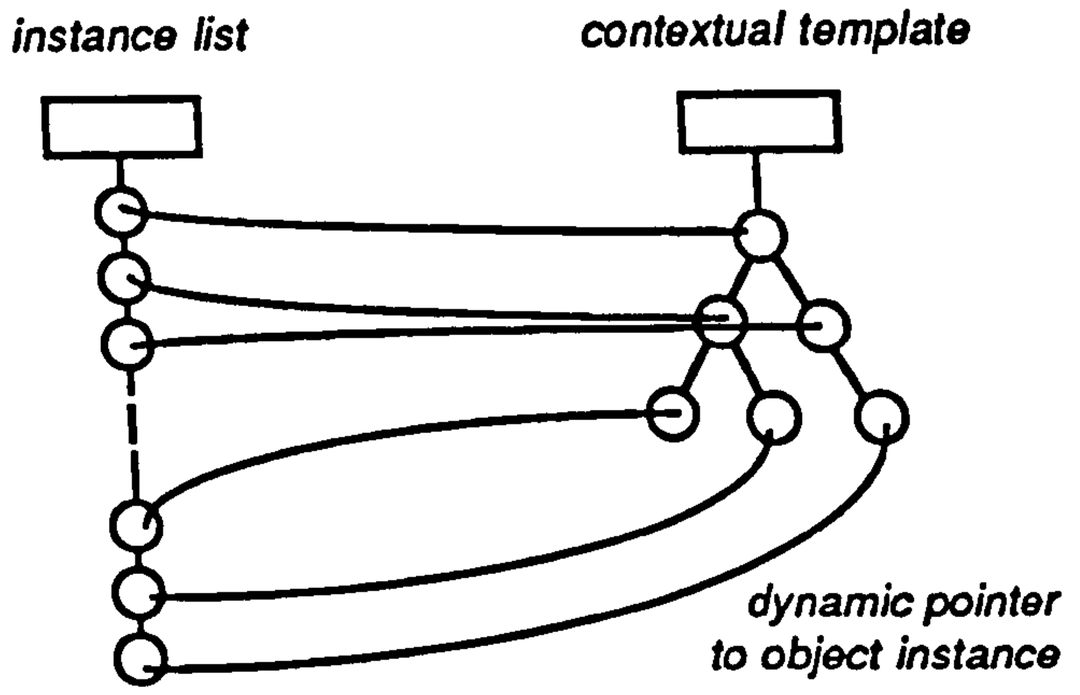


Figure 3.7.2.1.3. Contextual concept template and instance list

Templates may be dynamically configured and substituted; enabling re-description. It is also feasible to employ a number of templates simultaneously, each providing a different interpretation of referenced instance data thus resulting in a multi-representational system. This is also useful for assembling 3D geometrical mechanisms Appendix E.

In addition to the substitution of templates (resulting in a re-description mechanism), this approach also enables the dynamic exchange of interpreters at each nodal point thus facilitating re-phrasing (semantic variation) as outlined in Section 3.7.1. This results in an infinitely extendible architecture which is capable of evolving with the user's expectations or even design solutions.

3.7.2.2. CONCEPT INTERPRETERS

In essence concept interpreters map the data associated with a particular instance of a concept to the display environment allowing the user to edit it. The style of presentation depends upon the:

- hardware platform
- required contextual functionality of the interface, whether: 2d/3d graphical, or 1d text oriented interaction.

3.7.2.3. A BASIC TAXONOMY OF CONCEPT INTERPRETERS

By careful categorisation of conceptual interfaces multiple interpretation of concept values are also possible thus enabling the required re-phrasing mechanism.

For the purpose of describing a building model to a simulation package, conceptual interpreters may be categorised into two categories (representing the two extremes of user types) outlined below together with their corresponding graphical representation:

i) free response
ii) restricted response

The actual functional characteristics of these interpreters are identified and discussed fully in chapter 4.

3.7.2.4. INTERFACE PORTABILITY

On of the main objectives behind the development of user interface management systems is that of software portability.

In terms of the graphical properties of the window environment of a particular hardware device, meta-concept interpreters provide graphical environments to support concepts.

The interface of the top level node or meta-concept interpreter effectively points to window manager of the current hardware platform. By utilising the functionality of a high level graphics library it is feasible to port a graphical user-interface based upon this architecture to other hardware platforms simply by modifying the parent interface; by making calls to other graphics function libraries.

3.7.2.5. GENERIC CONTROL MECHANISMS

The basic mechanism brought in to play during communication is the setting of a context for discussion which is achieved by focusing the attention of the recipient system towards a particular set of concepts. In terms of a graphical user interface this maps to concepts being displayed and hidden.

It is also necessary to modify the value of a concept. This is achieved by 'setting' concept values. Clearing a concept is achieved by setting it's value to null.

Therefore the basic operations required by an interface capable of representing conceptual models are:

- hide / display a concept
- set the value of a concept
- query the contents of a concept
- dynamic substitution of interpreters and templates
- dynamic modification of generic attributes
- dynamic specification of instance attributes

As attributes are modified the attached interpreter is notified and the display is updated immediately.

Although a hierarchical model is illustrated, there is no need to follow this structure as the control mechanisms for focusing the users attention upon particular concepts and meta-concepts is under the control of external knowledge bases and not hardwired into the interface.

The template may be manipulated dynamically. Therefore unlike suspended time editing found in some UIMS the interface is capable, under the control of a user model, of adjusting to the requirements of the end-user during interaction.

In terms of notifying knowledge bases of events from the user, events are formatted into an utterance of the form (figure 3.2.7.5.1):

concept:event/method:value

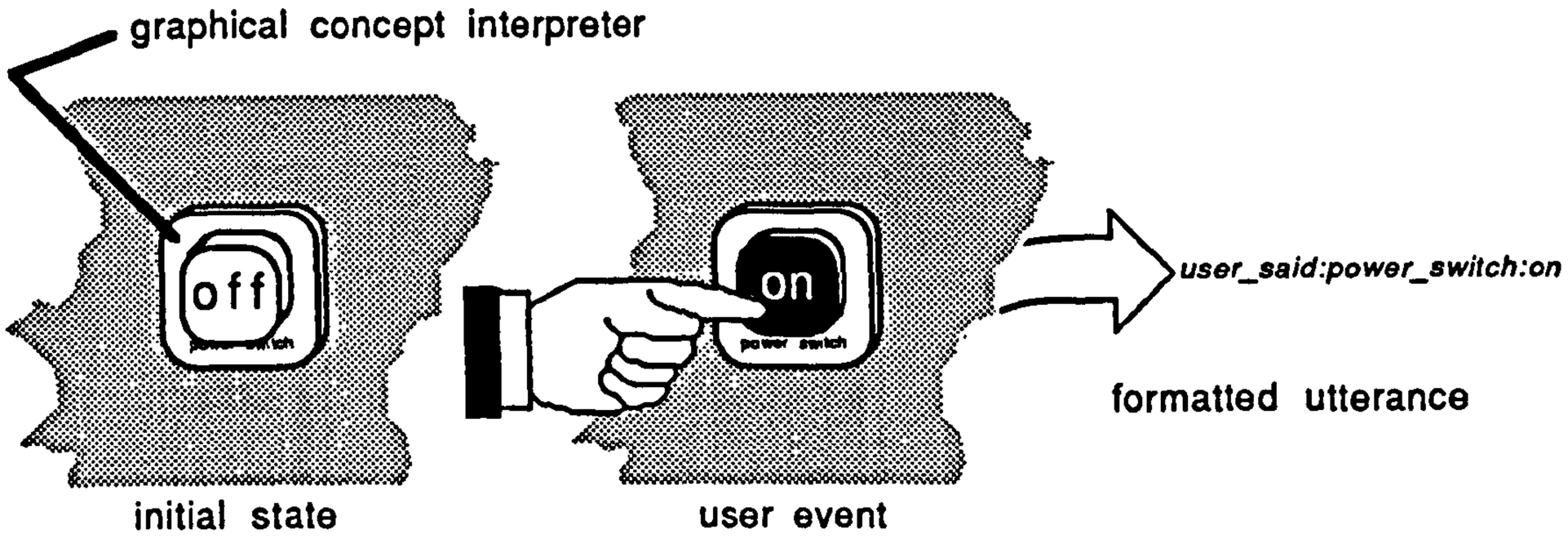


Figure 3.2.7.5.1. Formatted user utterance; concept:event:value.

In a generic form, events reported by a concept interpreter, should include the notification of:

- modification (user has set the value of the concept)
- requests for assistance

Other forms of event such as "plan to" and "not plan to" [SZEKELY 90], figure 3.2.7.5.2, should also be accommodated to allow the user model to anticipate and respond appropriately to user events. Changing the value of a particular concept may have a number of implications elsewhere in the system. A `plan to' modify (indicated by a positive selection) may result in a warning being issued or the suspension or termination of an application program.

Figure 3.2.7.5.2. Direct manipulation of concepts: "plan to" event tracking. Aborted operations would be interpreted as "not plan to".

For the purpose of this investigation, rather than considering the language primitives used by applications and users, a more formalised language approach was adopted (see Communication protocol, chapter 4.8.4). The protocols and

corresponding mechanisms for controlling a dialogue with the user, in the context of a form filling package, are discused in chapter 4.

3.7.2.6. MAINTAINING THE FOCUS OF DISCUSSION AND DIRECTING INPUT

Owing to the asynchronous processing activity within the various knowledge sources of such a dialogue system, the current focus of attention of the application environment and that of the user may at some time be different. For this reason the interface must be capable of maintaining the focus for each participant. This is achieved by the use of two concept pointers, figure 3.7.2.6.1.

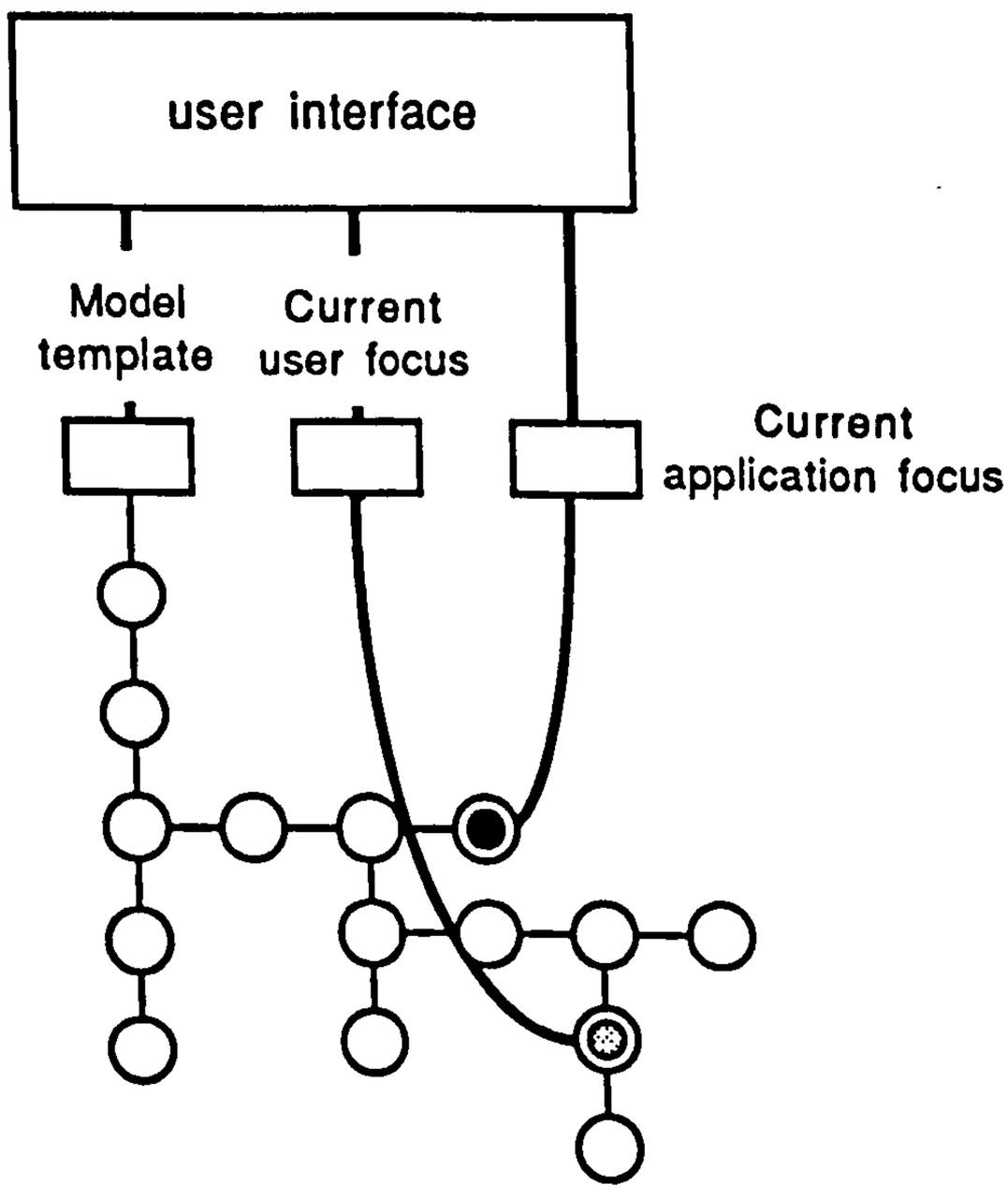


Figure 3.7.2.6.1. Current domain and user focus pointers.

3.8. SUMMARY

From the identification of an appropriate model of human communication, a generic framework, for constructing and manipulating graphical user interfaces, has resulted into which many different interpreters may be inserted and manipulated by domain knowledge.

The architecture is device independent and may therefore be implemented on any hardware platform providing a generic control mechanism for device dependant primitive objects Owing to a generic communications protocol and a number of generic control mechanisms (chapter 4.8.4), systems developed using this approach become plug compatible and may be linked together; providing tailored solutions to specific tasks.

This generic architecture has been used in the development of two different categories of interface:

- Forms: form filling text and 2d graphical oriented user-interface, (chapter 4).
- A 3D viewing and manipulation environment in which concepts may also be configured with articulatory control mechanisms (Appendix E).

The primitives for each interface are described in detail in the following sections outlining the design methodologies and technical problems associated with implementation.

3.9. AN APPROPRIATE METAPHOR

Traditional styles of computer dialogue (text and sound based interfaces) are serial. Barker suggests that these communication channels, while useful, have a limited communication capacity.

Owing to the different domains of activity associated with design processes more comprehensive communication mechanisms are required. Higher bandwidth communication channels, involving the use of pictorial and multi-media information, are seen as a natural extension to text oriented interaction providing a powerful means of communication.

Sommerville [SOMMERVILLE 85] suggests that majority of problems associated with natural language dialogues may be resolved by adopting a graphical rather than linguistic vocabulary, relying upon human abilities of pattern recognition coupled to a pointing device. This would, it is suggested, help the user to communicate instructions and perhaps information within a consistent symbolic system.

In order to optomise the communication and assimilation of information [BARKER 89], multi-media interfaces (Xerox and Apple) utilise several communication channels in parallel.

Once an appropriate conceptual model has been selected, a lexivisual presentation system such as this, combining both text and pictures, provides an efficient and effective mechanism for communicating a wide range of concepts; stimulating and orientating a user's cognitive activity [BARKER 89].

One of the most effective and natural-looking means of achieving a consistent form of interaction is the desktop metaphor; one of the greatest proponents of which are Apple with the Apple Macintosh. This metaphor (devised by Xerox), constructed around the graphical representation of familiar real-world objects (icons) which are manipulated using an appropriate pointing device, such as a mouse, on another graphical entity, the desk top, or window, provides the basic building blocks for developing "user-friendly" interfaces. By manipulating icons on the desk-top simple operations may be performed.

By creating a virtual reality within the computer, the user is sufficiently stimulated to draw upon real-world experiences. This has the obvious benefit that the interface is easy to learn and remember.

The most important visual device in this virtual world is the cursor. As the cursor is the main focus of the user's attention (the eye following the selection process), this graphical data element may be re-defined dynamically, providing a mechanism facilitating feedback; indicating permitted operations and system performance reports.

The dynamic modification of visual images is an ideal means of achieving semantic variation; providing alternative descriptions for a given concept. The desk top metaphor bridges many cultural and linguistic barriers where conventional interfaces have failed. The issues of subject meaningfulness and familiarity apply equally, perhaps more so, to graphical data elements such as icons. It is important that the visual cues used to represent operations are carefully chosen. A prime example is the Pafec dogs drafting system which has recently been given an icon driven command menu[3]. The poor choice of images result in many contemplative hours trying to establish what operations these buttons invoke.

The principles behind the desk top metaphor have been utilised in the development of the forms package (the main user-interface employed by the IFe), chapter 4, supporting a lexi-visual style of dialogue facilitated by the provision of

---

3 Note Image unavailable owing to technical problems.

graphical display and manipulation interpreters which support graphical feedback from other resources (such as image browsing tools, perspective image generators and the like) and allow the user to directly interact with graphical representations of concepts.

4. THE FORMS PACKAGE

4. THE FORMS PACKAGE

The forms package is a general purpose, dynamic graphical user-interface toolkit employing a form filing metaphor.

4.1. WHY A FORM FILLING METAPHOR?

Forms are a special type of communication medium primarily used to communicate between an organisation and an individual, utilizing a visual rather than verbal language, employing entities such as questions (with or without answer options), answer spaces, explanations, and redirection instructions. Although form-filling is a particularly convenient way of expressing a complex request to a computer it is an activity that has rarely been used as an alternative means (to other forms of interface) of commanding an application. [FROHLICH 86].

A screen or paper based form is essentially a template that facilitates the entry of data through a number of data entry slots (or fields) [BARKER 89]. This type of interface is often used in text retrieval systems and primarily forms packages have been confined to simple warehouse invoicing tasks and other, more traditional, business applications, such as stock control systems (OfficeTalk [ELLIS 80]), where the computerized forms are used to prepare information prior to submission to an application for processing. The results of the user's input are often displayed, some time after, either on the form itself, or by means of some other type of alphanumeric feedback. Examples of screen based form filling packages are Fillin [WARTIK 86], SQL*Forms [ORACLE 87], and FormsDesigner [DIGITAL 88].

The majority of forms packages are therefore merely interfaces to data bases, enabling the user to create and access information by filling in the appropriate form. Despite the limited facilities these systems offer it is 'the simultaneous presentation of options and free response answer areas on screen-based forms' (Frohlic) which makes form-filling suitable for many different types of application; offering the user with an alternative to menu selection and command language interfaces, resulting in the flawless entry of data.

4.2. DESIGN METHODOLOGY

A user-interface capable of managing hierarchical conceptual models (concepts and meta-concepts) was required. The analogy of stacked and nested forms containing logically related fields satisfied the requirement in a manner that users would find familiar.

The Forms package is based upon a generic architecture derived from a language view of the user-interfaces, describe in chapter 3, and is a means of representing and presenting concepts at any level of detail. Both the object management functions and the concept interpreters are written in C, employing object oriented programming methodologies. The forms package was developed to meet the following requirements

- manage hetra-archical conceptual models
- facilitate dynamic (run-time) manipulation of concepts and meta-concepts allowing the dialogue with the user to be adapted during interaction
- provide developers with a clean separation between application and the user-interface together with providing high level dialogue control mechanisms; thus ensuring re-use of domain knowledge.
- isolate hardware dependencies within a single module and by utilising a high-level graphics toolkit ensure portability across a range of hardware platforms.

4.3. PORTABILITY CONSIDERATIONS

One of the primary concerns when developing the forms package was the issue of portability. Although the forms package has been developed within a Unix environment the most obvious limiting factor to software portability is that relating to graphical input and output (I/O).

4.3.1. WINDOW ENVIRONMENT

At the initial stage of the project there where no 'stable', truly portable toolkits or libraries that fitted the requirements of the project. Window managers such as X and NeWS were still being evaluated and modified and only available for beta test sites. A stable development base was required. A major decision was made to adopt a graphics toolkit developed by the informatics division of Rutherford Appleton who are concerned with developing the user interface. The ww function library was developed in order to isolate software development from 'the ever changing base of machine and window manager' [MARTIN 87].

4.3.2. TOOLKIT LAYERS

WW is intended for the development of high level tools for window management (and has even been used as a base for an implementation of GKS [MARTIN 87]). It exists in two distinct layers:

- a high-level function layer providing rigid functionality (file system browsing, text editors)
- a low level interface providing simple line drawing and rasterop between bitmaps [MARTIN 87].

Having some experience with Sunview, Whitechapel graphics, and a little working knowledge of X and NeWS, this low level layer is relatively high-level considering all that it hides and provides a consistent interface between numerous platforms.

Having access to the source code was also a great benefit; enabling a number of the supplied high level functions to be modified to suite a particular requirement.

Although, ww offers an ideal development platform, the author was conscious of the need to isolate code from this layer to enable other toolkits to be used. The only really specific use of ww, apart from bitmap operations is the use of the 'tx' (text editor) function set. These calls have been isolated within a number of macro definitions and therefore should be relatively easy to replace.

4.3.3. HARDWARE PLATFORMS

WW enables the developer to access the base graphics of the following systems:

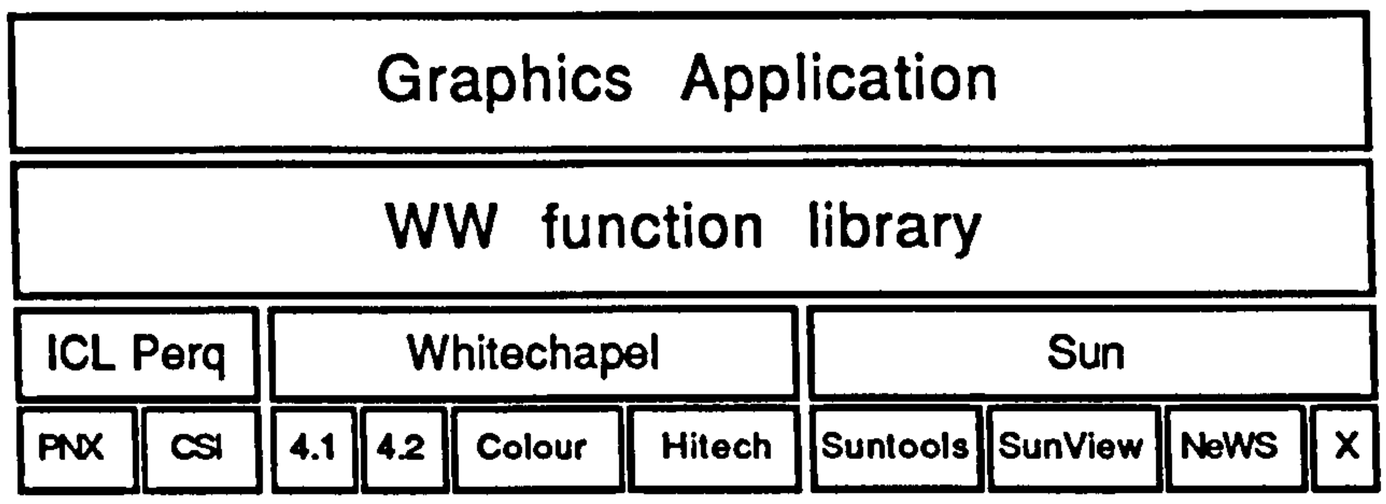


Figure 4.3.3.1. Toolkit layers

During the research period the development of the forms package was undertaken on Sun workstations operating with system IV and V. Towards the end of the project a whitchapel Hitech workstation running the X window manager was acquired and the forms package successfully ported with relative ease. It has not been possible to port the application to other platforms owing to the availability of hardware. None of the network transparent mechanisms of X where explored although this is intended for future research.

4.3.4. HIGHER LEVEL TOOLKIT

An object oriented toolkit layer, written in C++, has also been designed for future developments. This layer provides a high level of abstraction an example of which is the image class with the following methods:

```
image->rotate
image->flip_horizontal
image->flip_vertical
image->save_to_file
image->read_from_file
image->shift
image->magnify
```

4.3.5. COLOUR

Colour is an important aspect of symbolic representation/presentation. Ideas may be communicated more readily using colour or other sensory stimuli (sound, smell).

The use of colour within the forms package has been withdrawn for a number of practical reasons:

i) The use and interpretation of colour is highly subjective. Although physical properties of fields (size and position) may also be regarded as subjective, it is suggested that colour is more culturally emotive and perhaps should be avoided.

ii) A practical limitation of the use of colour results from the use of virtual screens (bitmaps) for individual forms. Introducing colour increases the demand upon application memory (eight bits per pixel as opposed to one bit). As the creation of forms and fields is dynamic, the allocation of large areas of memory significantly reduces interaction and system response times.

iii) A final and perhaps the most significant reason for no longer supporting colour is that, by writing a series of test programs (colour ramping for example) it was discovered that ww's representation of colour was inaccurate, inconsistent and therefore unreliable.

One justification for using colour would be for displaying colour images and for distinguishing between object primitives. A simple grey-scale is used for form backgrounds to visually highlight conceptual boundaries.

4.4. ARCHITECTURE

The key to the Forms package lies in its data oriented architecture. This is based around a dynamically alterable n-level hierarchy which is used to build a conceptual model of a user's mental perception of a system, described in chapter 3. Each node in the tree structure, inheriting attributes from its parent, points to an instance of one of two basic entities:

a field — which is used to represent concepts. The value of a concept is displayed and manipulated by means of an appropriate interaction mechanism, and,

a form — or meta-concept which facilitates the conceptual classification of a number of logically related fields and forms.

Thus using combinations of these two basic elements complex models may be constructed by hierarchical decomposition, effectively using nested and stacked forms. This is particularly useful for defining and limiting a user's task to a particular problem area and ensuring task completion or closure [MILLER 57].

Both forms and fields are essentially modules containing pointers to a private data area and to methods for displaying and manipulating this data. These methods are collectively referred to as a concept interpreter.

Each field has an identity tag (or label) associated with it. This tag represents the actual concept (or meaning) while the data entry slot or concept interpreter represents the concept's value. The field identifier also enables the user to access on line assistance and default values.

The value of a concept is stored as a character string (a series of character tokens) and therefore may be interpreted in a number of ways, simply by substituting one concept interpreter for another. Taking the simplest of examples first, figure 4.4.1 illustrates a basic field consisting of a concept together with it's concept interpreter, positioned on a form.

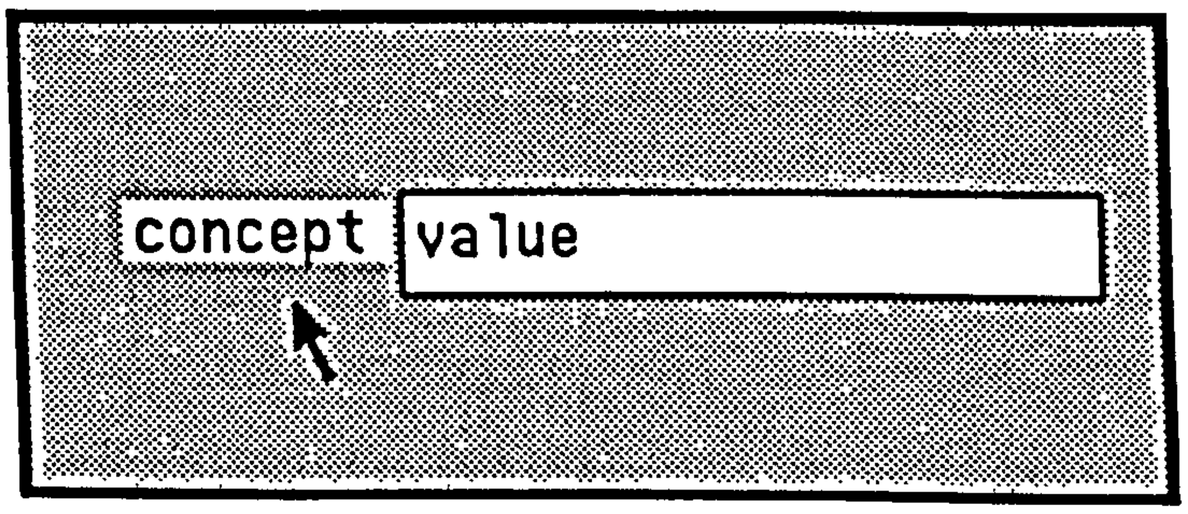


Figure 4.4.1. Concept and interpreter

Screen based forms may be categorised in to:

i) static - rather like paper based forms with simple keyboard transcription mechanisms, and

ii) reactive - responding to input events and guiding the user through pre-defined sequences of forms (if deemed necessary by a user model) either by re-direction instructions or by display sequencing.

As a consequence of implementing reactive form sets, there are essentially two distinct aspects to the forms package:

i) form creation: the definition of conceptual templates

ii) form manipulation both by end-user and application program

This has resulted in the development of three distinct interfaces for each of the users of the forms package:

i) an *end-user interface* for the eventual user of the application program.

ii) a *proforma interface* for the interface designer

iii) a *command interface* for the application developer

As so many issues involved with the development of each of these interfaces are interrelated it is difficult to choose an appropriate starting point. One of the aims of the forms package is to allow the user to manipulate concepts. As concept manipulation is a significant aspect of the forms package, the mechanisms for end-user manipulation will be considered first.

4.5. CONCEPT INTERPRETERS - A FRAMEWORK FOR CONSISTENCY

Consistency in the user-interface has been a major concern for many developers of toolkits and application software. The forms package has three user interfaces:

i) Proforma interface

ii) End user interaction interface

iii) Manipulation interface

Note that command consistency between application programs depends entirely upon the proforma designer and knowledge engineer and not upon the functionality of the forms package.

Since all entities within the forms package are derived from a generic concept class, regardless of their properties each will inherit generic behavioural characteristics and therefore result in complete internal consistency.

Another criteria, which is satisfied by the use of a generic object class, is consistency in the command language, declarative syntax and user interaction layer.

The design considerations for each of these interfaces are described.

4.6. A GENERIC CONCEPT INTERPRETER

Regardless of it's value a concept is represented by either a multi-line text string, figure 4.6.1, or a bitmaped image.

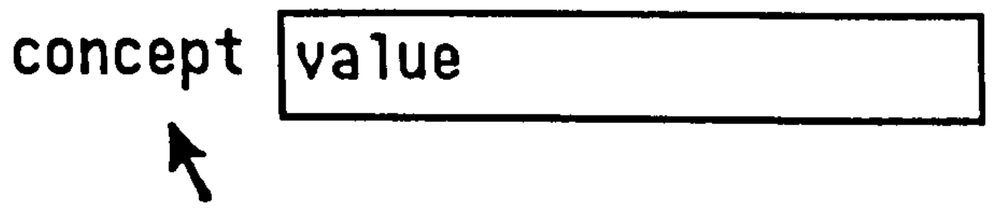


Figure 4.6.1. Concept and basic free response interpreter

4.7. END-USER INTERFACE - COMMON CONCEPT EVENTS

The end-user of the forms package interacts with forms and fields by means of the mouse and keyboard. Irrespective of the concept interpreter, a number of common activities involved with the manipulation of concepts and meta-concepts has been identified:

- selection or focusing
- requests for assistance
- manipulation of concept values

Ignoring variations in concept interpreters, the only means of interacting with a concept is by means of the mouse. Therefore, access to mechanisms fulfilling the general requirements of interaction with a concept is provided by means of two pop-up menus, associated with each field label. Assuming a three button mouse, these pop-up menus are activated by the left and right mouse buttons.

As concepts and meta-concepts are essentially derived from the same object class (class concept) represented in the forms package as a field label and the background of a form, a number of common activities are made accessible by means of a common pop-up menu.

select	select the form or field for input
help	obtain help information
description	obtain a description of the concept or meta-concept

The first menu, figure 4.7.1, therefore includes items for selecting (or activating) the concept, requesting assistance and retrieving default and previous concept values. The latter being useful as an error recovery mechanism

As this menu allows the user to access information regarding the operation of the forms package together with interpretations of the meaning of the concept this menu is activated with the left mouse button since this is the most commonly used button, particularly by novice users. It is also important that the first few user actions result in valid non-destructive events otherwise frustration will occur and the user may loose interest.

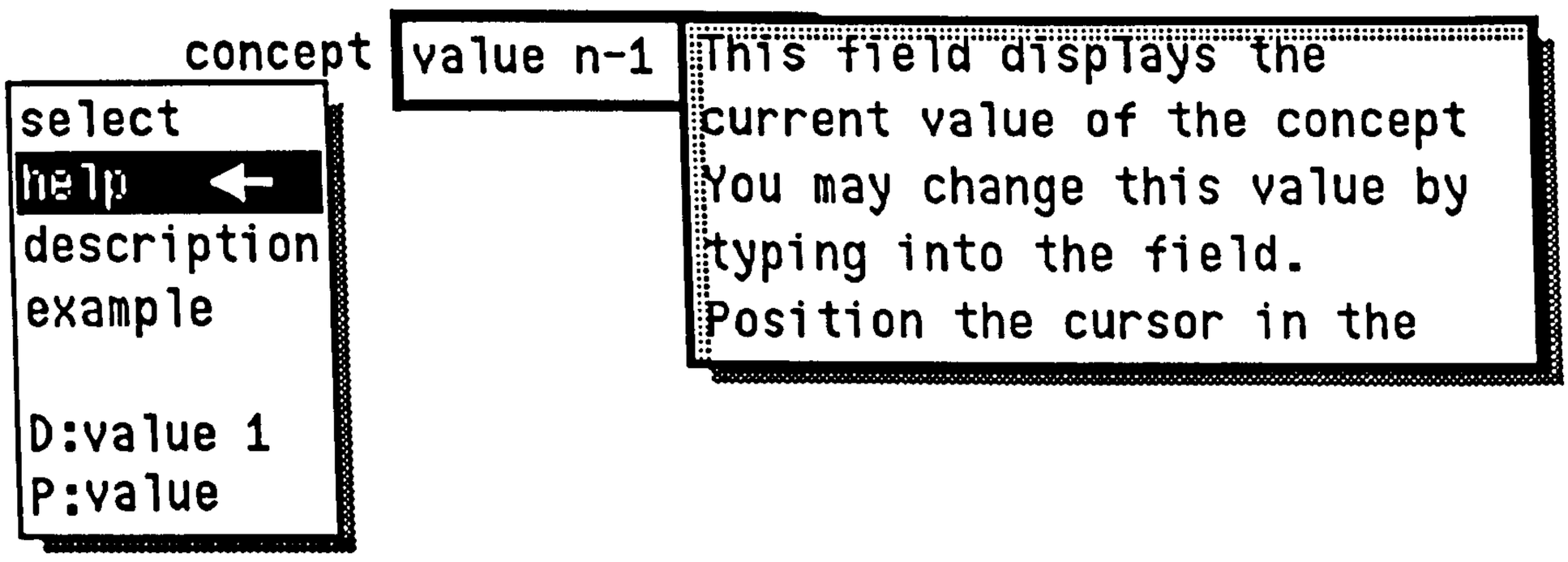


Figure 4.7.1. Standard menu and help window

The second menu (4.7.1.5.1), activated by the right mouse button is a concept specific menu which can be configured dynamically, during run time, with contextually relevant values.

4.7.1. DOMAIN INDEPENDENT RESPONSE (COMMAND MENU)

The command menu, figure 4.7.1, is divided into three sections: selection, assistances, data retrieval, enabling the user to issue a number of elementary requests. The order of items has been chosen in descending "destructiveness". The least destructive placed at the top. On selection the cursor is always placed to the left of the least destructive item; the help item.

As the items on the command menu are common to all concepts, the menu may also be activated from the background of a form.

4.7.1.1. SELECTION ITEM

A concept may be selected for input (keyboard or mouse) by clicking any mouse button in the bounding rectangle of the concept interpreter or by selecting the *select* item from the concept menu. When selected the field is highlighted and any other currently active field is de-selected, figure 4.7.1.1.1.

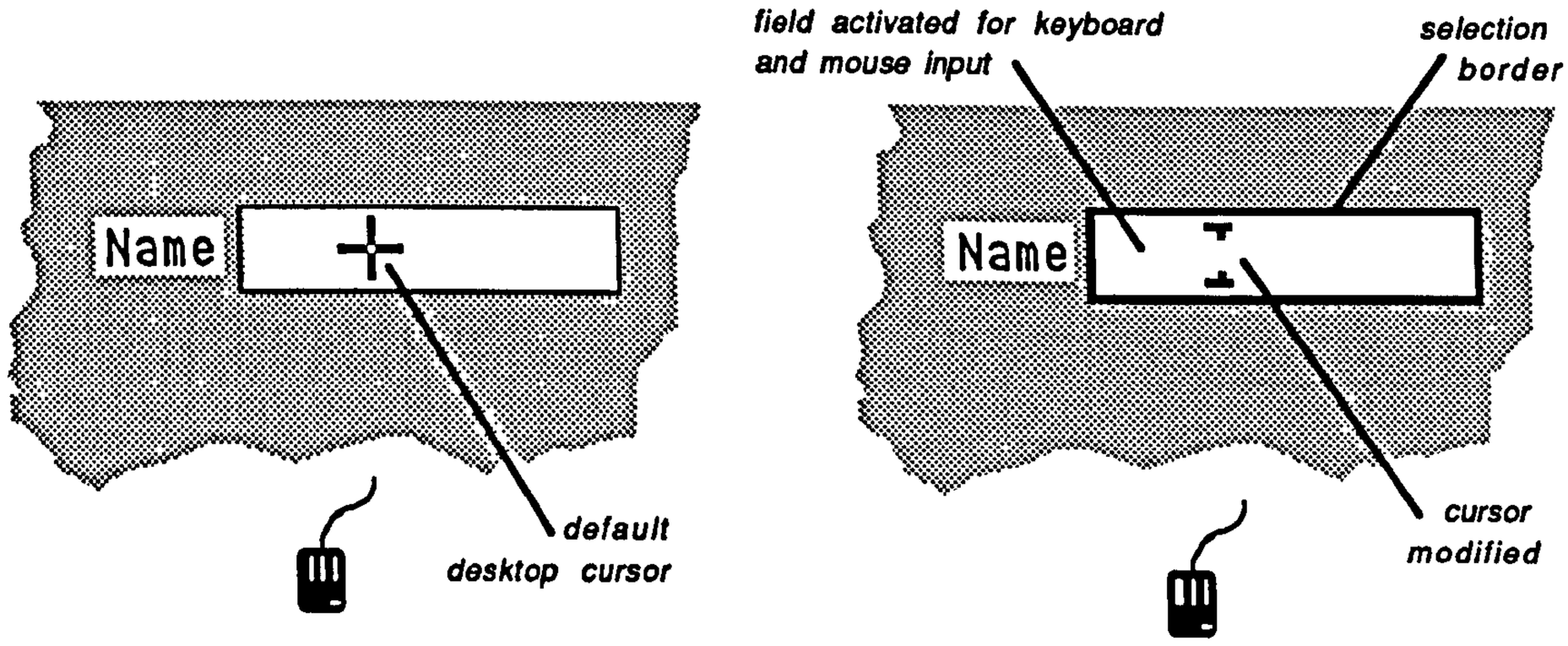


Figure 4.7.1.1.1. Field selection.

This enables the user to select the field for keyboard input other than clicking the cursor in the concept interpreter.

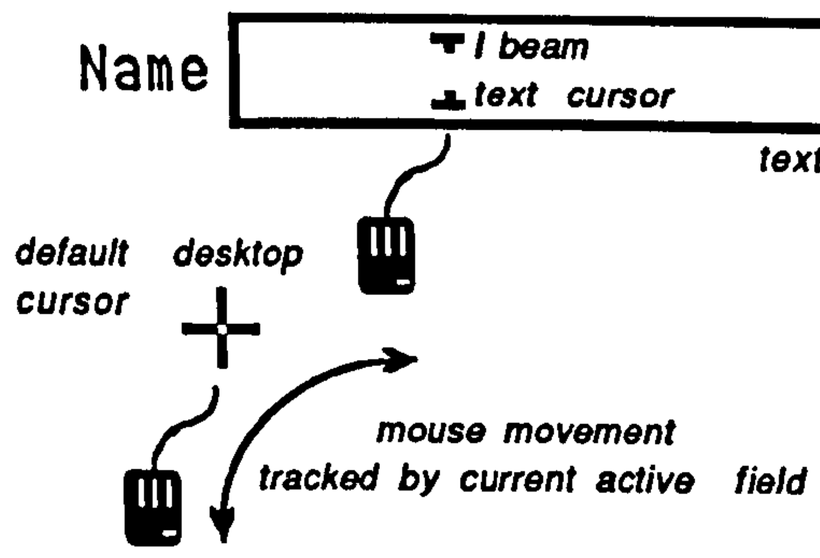


Figure 4.7.1.1.2. Tracking the cursor.

When selected the cursor is tracked by the currently active field and is modified when it wanders in and out of the field's bounding rectangle, figure 4.7.1.1.2. In order to facilitate multiple cursor patterns within a single window, ww enables cursors to be stacked n-levels deep. Unfortunately when managing multiple desktops, window leave and enter events pop the cursor stack. When the field is eventually de-selected an attempt is made to pop a (NULL*)cursor_stack resulting in a buss error. In order to by-pass the problem cursors are explicitly set avoiding the need for a cursor stack.

4.7.1.2. ON-LINE ASSISTANCE ITEMS

This section of the menu may be defined by the application developer. The names are taken from an initialisation file, which ensures that all command menus behave in exactly the same manner. Two default items are provided at run time:

i) help, and

ii) description.

Any data, defined in the proforma template, associated with either of these field attributes is displayed in a pop up window. As the contents of the help and description items refer to the value of the concept, when selected the window is placed to the top right hand corner of the field interpreter so as to avoid its contents being obscured, figure 4.7.1.2.1.

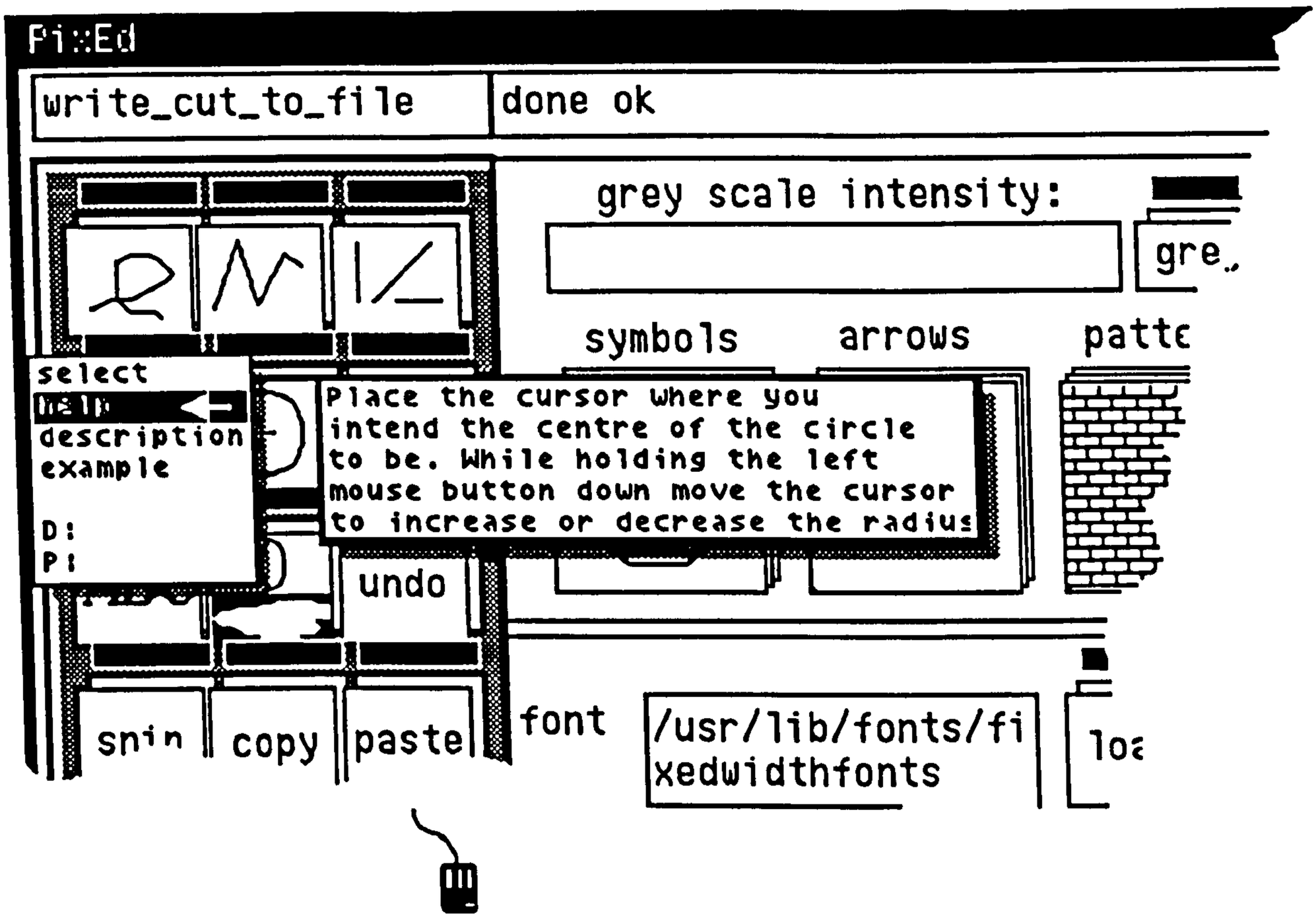


Figure 4.7.1.2.1. On-line assistance for individual concepts. Note that the help window is only displayed once a positive selection from the concept menu has been made (ie the concept menu is de-activated before the help window is displayed. This image is taken from the PixEd (Pixel Editor) interface, Appendix F.

Following the *'modeless'* philosophy of the interface, this window does not lock other events out, although to avoid confusion only one display window may be open at any one instant.

4.7.1.3. THE HELP WINDOW

Help and concept descriptions are defined in the proforma template and displayed in a pop-up window when the item is selected.

As either of the help or description items is selected a message of the form:

concept?USER_QUERY?item_name,

is formatted and written to standard output. Thus allowing the parent process, or other monitoring resources, to respond appropriately by overwriting information defined in the proforma with contextually relevant information.

It is important that the help window remains active until the user wishes to remove it. The window, when displayed, must not lock the user out and has been

designed as a *modeless* object allowing the user to move, scroll, and destroy it, while still allowing the user to select and enter data into other fields.

4.7.1.3.1. Moving the help window

The help window is automatically positioned to the top right of fields (so that it does not cover the contents of the field), and in the centre of forms. In some situations the user may wish to move the window out of the way so that the information may be referred to while dealing with other concepts. This may be achieved by holding the middle mouse button down over the window and dragging it to a new position. When initially displayed a copy of the area beneath the window is copied and pasted when the window is closed or moved.

In order to eliminate any flicker that may occur as a result of the continued bitmap pasting during a move operation the window's bitmap is stacked with the WWPUSHOFF flag. This enables changes to be made to the default window's bitmap without them being seen until the contents of the window is restored by another call to bmstack with a WWPUSHON flag. The effect is a flicker free movable object.

When the window is moved away from its initial position it is important the user is aware of what concept, the information displayed, refers to. Therefore, lines from the corners of the window, are rubber banded to the corners of the field while being dragged. The field label is also inverted when the mouse enters the window to ensure relevancy.

The window may be moved by clicking the left mouse button within the bounding rectangle of the help window and dragging it to the desired position. The cursor pattern changes to a double box to confirm the operation.

4.7.1.3.2. Scrolling the help window

To prevent the help window occupying the entire form the depth of the window is restricted to an arbitrary number of lines of text.

The maximum depth of the window is defaulted to five lines (this may be changed to any depth (see customising the forms package, section 4.9.2). The width of the window is determined by the maximum line length in the text block. When the number of lines in the text block exceeds the maximum defined depth, the window becomes scrollable. To indicate to the user that additional information is available a virtual shadow, cast by an imaginary light source (positioned to the top left of the

window) is pasted along the top and left edge of the window, figure 4.8.1.1. The shadow is achieved by ORing over an appropriate greyscale pattern once the text has been printed.

The actual scrolling action was initially a pan operation, with the width of the window also being defaulted to an arbitrary length. However panning text is a rather un-natural activity and so the width of the help window is dynamically configured to the maximum line length within the text block; resulting in a more natural, "page like", vertical scrolling action.

Scrolling is achieved by holding the middle mouse button down, within the help window, and moving it in the required direction. The permitted scrolling direction is indicated by changing the cursor pattern, figure 4.7.1.3.2.1.

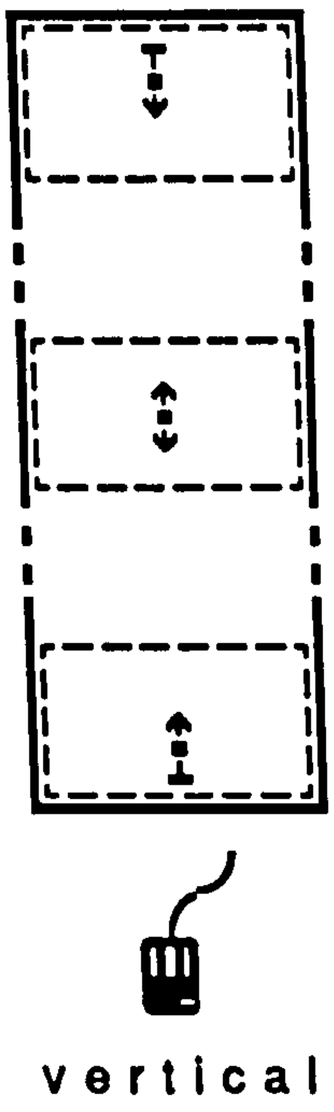


Figure 4.7.1.3.2.1. Help window scrolling cursor patterns

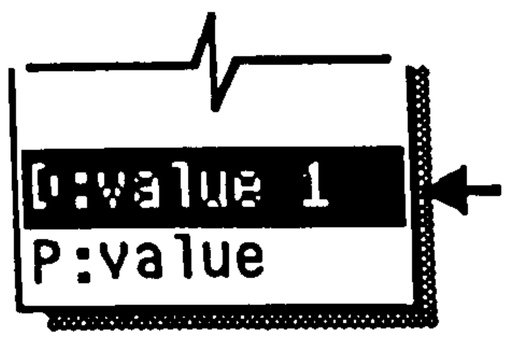
In addition to mouse activation, the help window may also be invoked by an application or knowledge resource. This enables the type of assistance to be tailored to a particular state. The mechanism for achieving this (chat user) will be discussed in a later section.

4.7.1.3.3. Destroying the help window

By clicking the right mouse button within the bounding rectangle, the help window may be destroyed.

4.7.1.4. DEFAULT VALUES AND ERROR RECOVERY

Each concept may have one default value which is displayed on the command menu as



If this item is selected the concept's default value is transferred and interpreted by the concept interpreter.

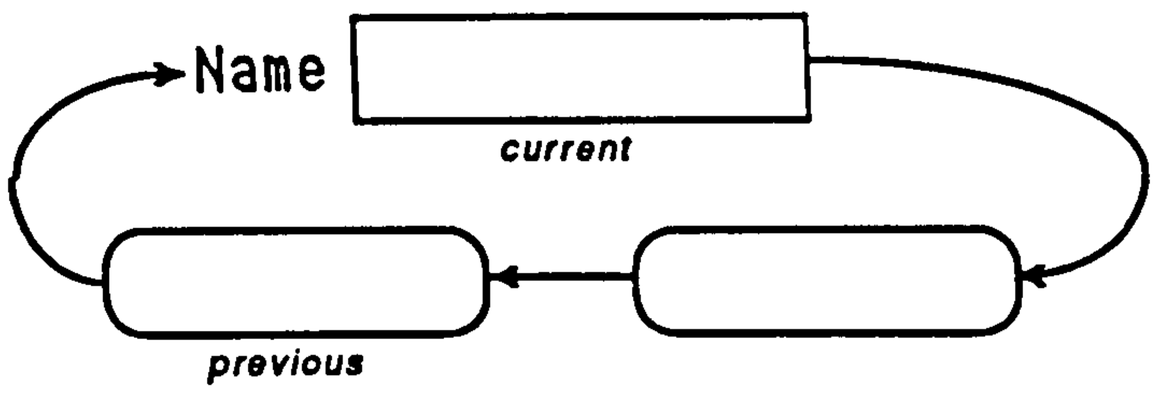
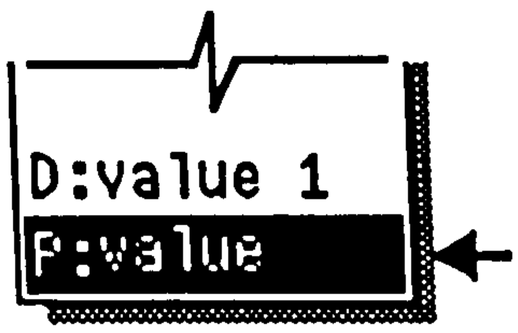


Figure 4.7.1.4.1. Transferring data to and from the previous value store.

The current concept value, before over writing is placed in a buffer, figure 4.7.1.4.1, and displayed on the command menu as:



for previous value. This allows a single level of error recovery to be achieved although it is envisaged that previous values may be stacked onto a list.

As with the assistance requests, messages of the form:

concept:USER_REQUEST:default_value

concept:USER_REQUEST:previous_value

are formatted and written to the standard output stream, enabling any monitoring resources to respond. This enables contextually relevant defaults to be supplied, and facilitates a greater depth of error recovery.

4.7.1.5. DOMAIN SPECIFIC RESPONSE

Contextual, domain specific, alternative values may be offered to the user by means of a second pop-up menu which is activated by depressing the right mouse button while the cursor is positioned over the field label, or concept identifier. The menu items are usually defined in the proforma template definition, figure 4.7.1.5.1, but may also be dynamically configured by knowledge bases and applications to provide contextually relevant values.

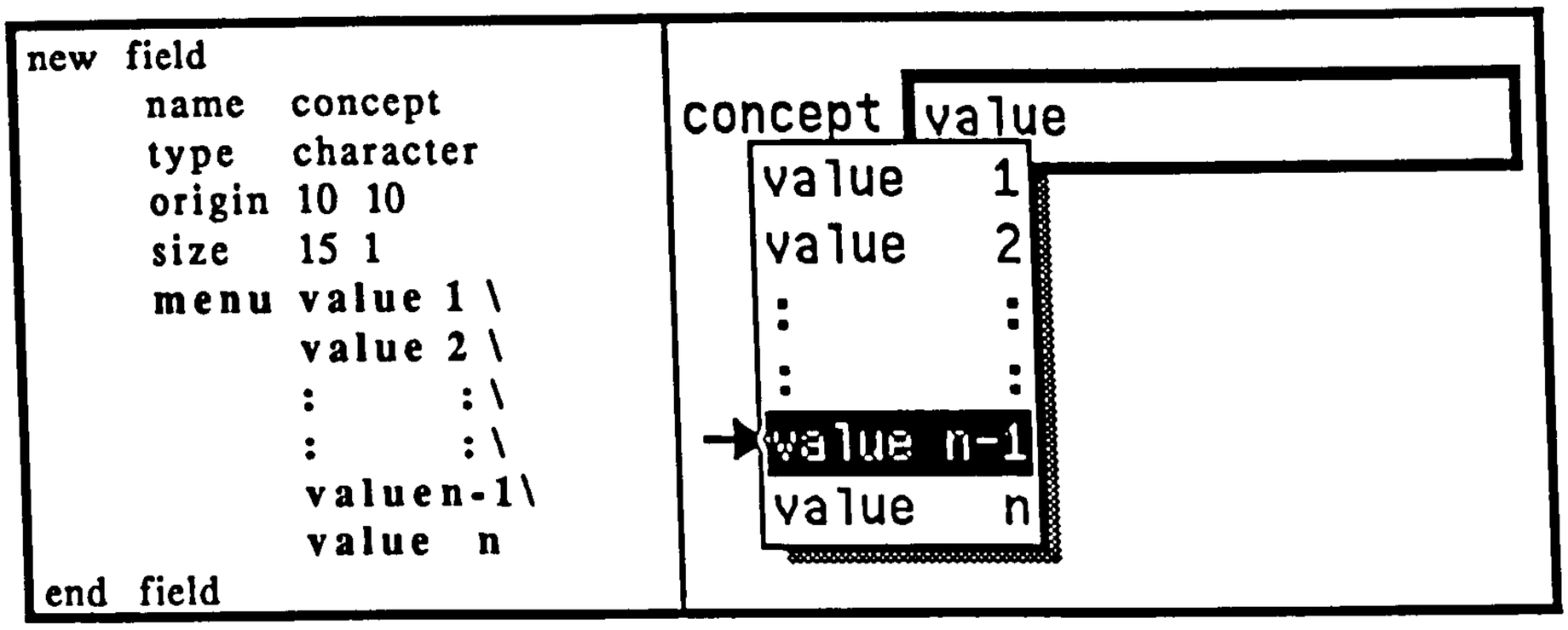


Figure 4.7.1.5.1. Dynamic menu

If there are no alternative values defined for a particular concept the action taken by the forms package is to respond with an invalid mouse selection error; indicated by a cross cursor, figure 4.7.3.1.1.

When a positive selection from this menu has been made the field is selected, if not already, and the item is then displayed by the interpreter. The previous value may be retrieved from the the standard concept menu associated with the field label.

4.7.1.6. POPUP MENUS

The interaction by means of pop-up menus has prompted an investigation into event interpretation.

The pop-up menu is a "high-level" tool provided by WW. Although this function enables items in a list to be chosen using cursor selection the functionality of the menu routine required modification. The menu routine returns when the button state changes; this may be a release or an additional button. Menu items should only be selected when all buttons are released; i.e. the user returns to the stable state before the invocation event.

4.7.2. INTERPRETING MOUSE EVENTS

Mouse events may be categorised into two distinct groups:

i) positive (button down) events, and

ii) passive (movement) events with all buttons up.

Positive events are those which invoke a process such as:

- selecting a field,
- dragging an icon,
- drawing a line,
- popping a menu,

while passive mouse events are those such as mouse movement, these are typically managed by mouse tracking routines, changing cursor patterns over active (hot or sensitive) regions of a display.

Positive events posed a slight problem within the forms package. Although positive events are used to select and activate forms and fields for input, two possible alternative actions may be taken on selection:

i) on selection the event activates a form or field and is immediately processed, or

ii) a positive event is held until the button is released or the combination of buttons held changes.

Once a working prototype of the forms package had been implemented, these two issues where investigated. Through use, the first option proved to be rather aggravating with menus popping up on form and field selection.

From personal observations the act of selecting a field is merely to address a new concept. The user should not be forced in to a situation, by a pop-up menu, for instance, requiring a decision for which the user is unprepared. Therefore the second method; "one event, one process", which is far more consistent, is the method currently implemented.

4.7.3. FEEDBACK AND ACKNOWLEDGMENT

As the cursor is the focus of the users interest this is used, by changing the cursor pattern, to indicate permitted operations such as text editing, or to indicate heavy computational activity or initialisation (by showing an hourglass).

4.7.3.1. INVALID MOUSE EVENTS

Not all areas on forms or fields respond positively to mouse events. To indicate invalid mouse events a cross cursor is displayed, figure 4.7.3.1.1.

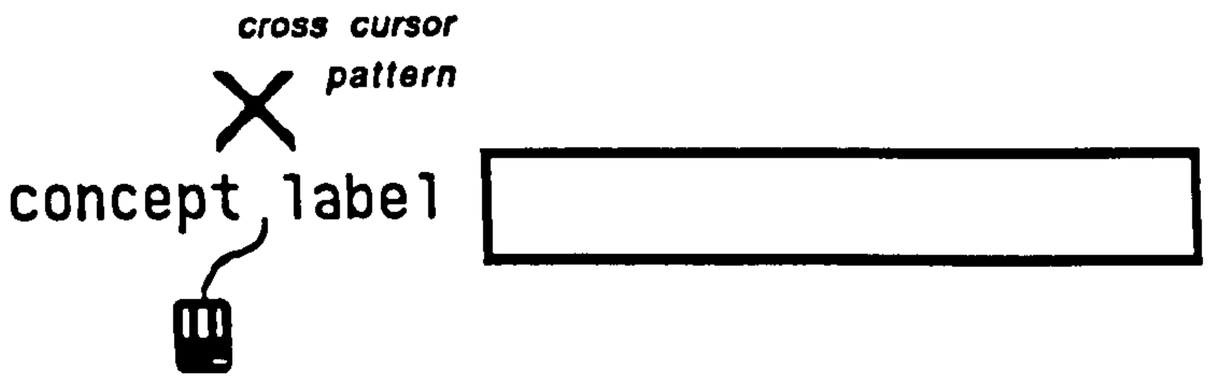


Figure 4.7.3.1.1. Invalid mouse event.

Other warnings such as a flashing screen and/or bell sounds are not used as these cause unnecessary alarm and are annoying to other users.

4.7.4. CONCEPT INTERPRETERS - THREE BASIC TYPES

In order to reflect the capabilities of the two extremes of user types (expert and novice) within the IFe two categories of concept interpreter have been provided:

i) free and

ii) restricted response.

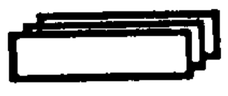
Three basic field types exist; each with its individual style of presentation; providing a visual cue to its function:

4.7.5. VISUAL CUES

The following visual cues are used to indicate, to the end-user, what mechanism is being employed for input:

 free response

 restricted response (Boolean)

 restricted response (multiple choice)

Knowledge of a particular class of user and their task (held in a user model) is used to dynamically switch between the above input and presentation modes.

The following interpreters are broad set of primitives used within the IFe. A more complete guide to special purpose dialogue design may be found in [MARTIN 73, EASON 75].

Where the above visual convention has not been rigourously observed, the function of the interpreter is obvious by the content of the field (menu, date, etc). Each of the concept interpreters is now described.

4.7.6. LABEL - GENERIC CONCEPT IDENTIFIER

This field simply displays text or an image and responds only to mouse input; activating the two general pop-up menus described in section 4.7. and 4.7.1.5.

4.7.7.  FREE RESPONSE

Free response areas provide the user with the maximum input latitude and are therefore a more appropriate form of input for expert users performing complex tasks. This is the least structured of responses and perhaps more prone to miss-communication between systems [MILLER 76]. For this reason free response areas should be used in the gathering of unstructured information such as: Name:

Free response functionality is provided within the forms package by text field and graphics fields.

4.7.7.1. TEXT FIELDS

A basic text field, figure 4.7.7.1.1, enables the end-user to interact with a concept value, by directly editing the character token string. A number of simple editing facilities are offered as illustrated by figure 4.7.7.1.1 and .4.7.7.1.2

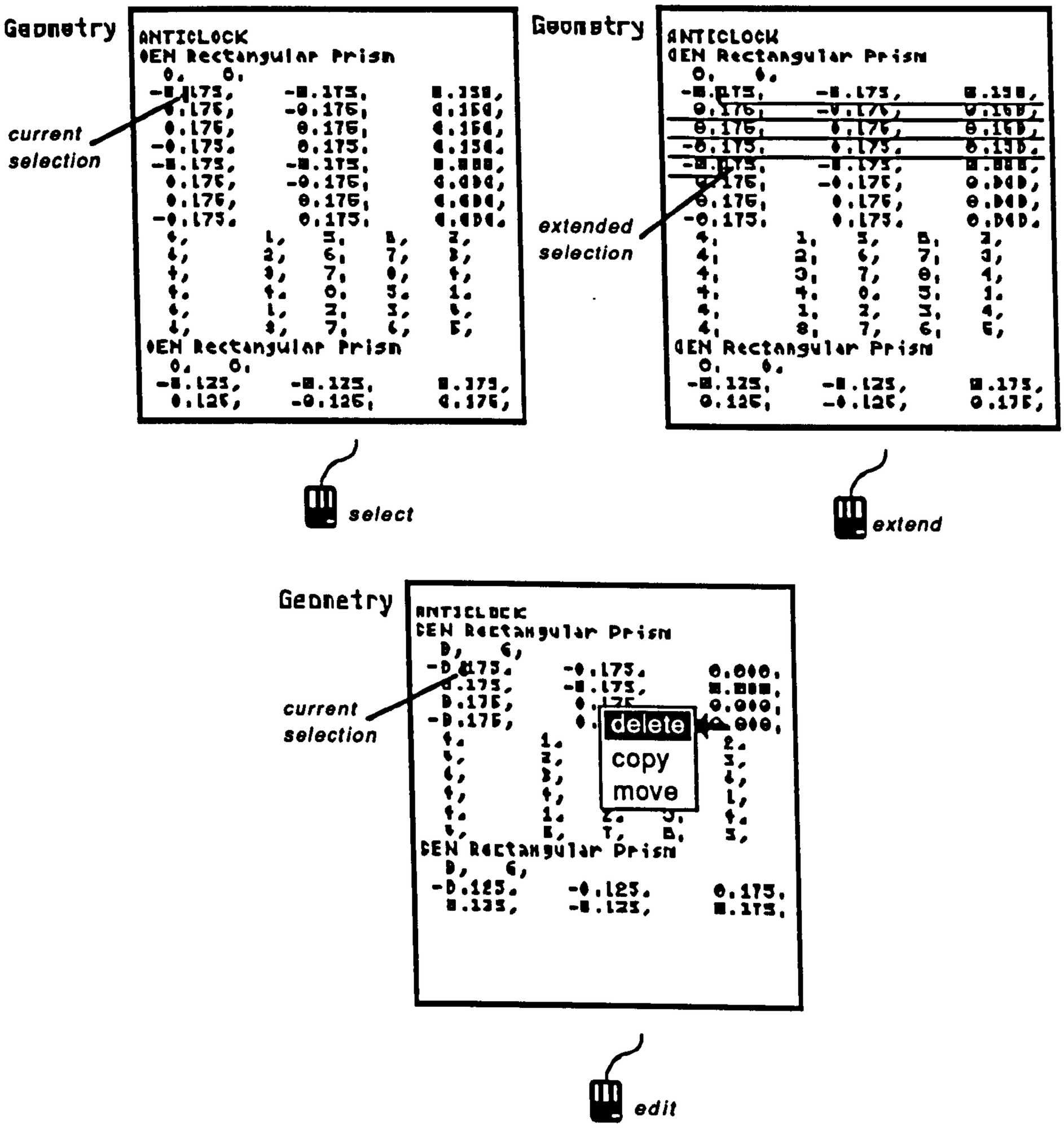


Figure 4.7.7.1.1. Simple text editing, using ww tx function set.

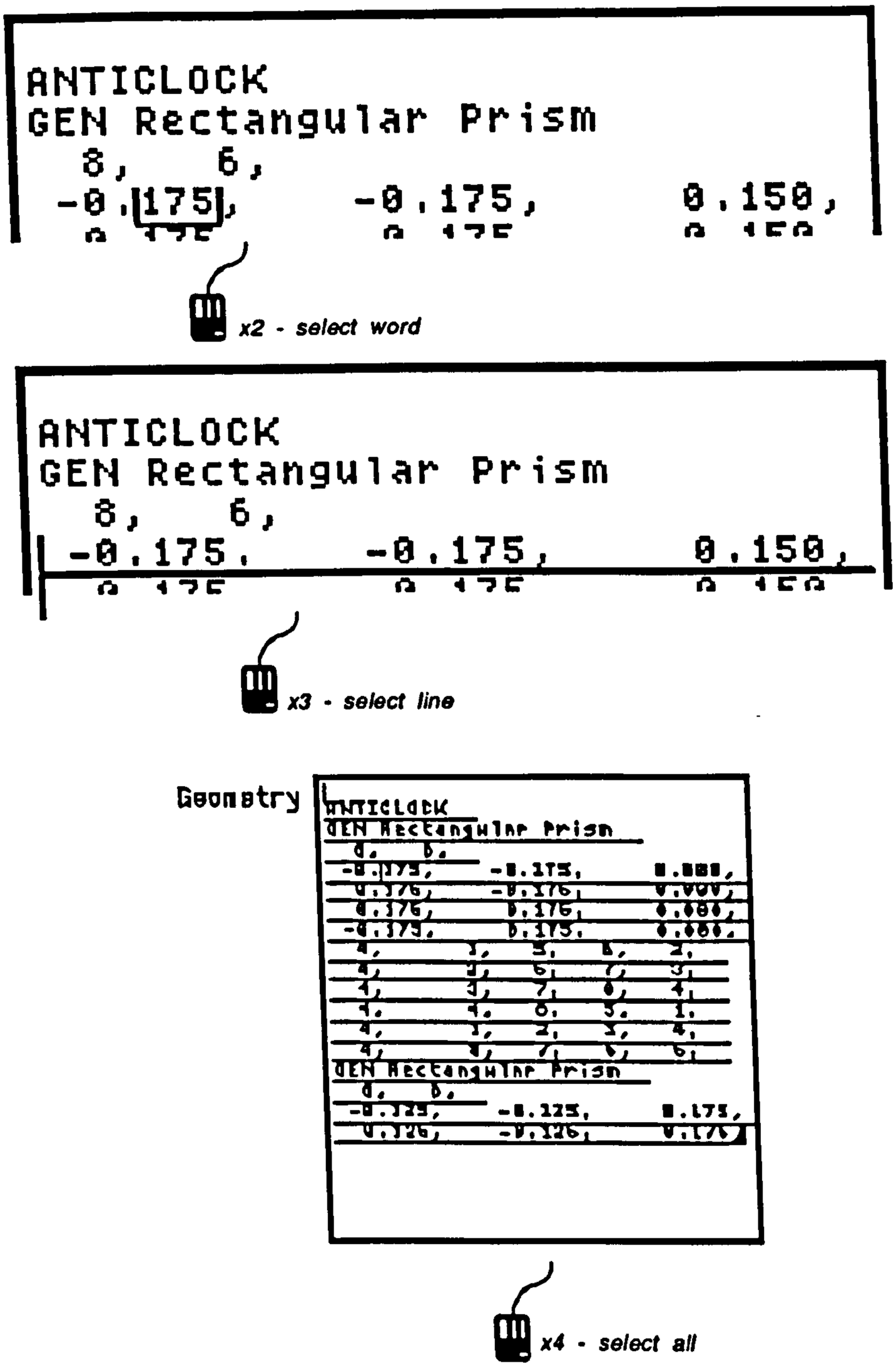


Figure 4.7.7.1.2. Selection accelerators

A number of magic control characters are trapped for editing:

^U - delete line

^W - delete word

4.7.7.1.1. Derived Character validation fields

In order to ensure the flawless entry of data a number of character validation methods have been defined to restrict input a particular type of data string. Each may be interchanged. The following interpreters are derived from the basic text field and inherit the behavioural and functional characteristics of the generic text field class. The currently implemented character validation methods are:

character	This is a free response text editor. All characters are accepted.
alpha	Input is restricted to ASCII characters <a-Z>.
alpha-numeric	Input restricted to ASCII characters <a-Z> and numbers <0-9>.
integer	Only numbers <0-9> accepted.
floating point	This interpreter allows only one decimal point to be entered between characters [0-9], therefore only floating point numbers are accepted.

Character validation may be extended for each of hte above types as described in 4.9.3.7.

4.7.7.1.2. Cutting and pasting from other text fields

When the contents of a text field is selected (either totally or partially, figure 4.7.7.1.2.1.a), it may be cut or copied into another field using the editing menu.

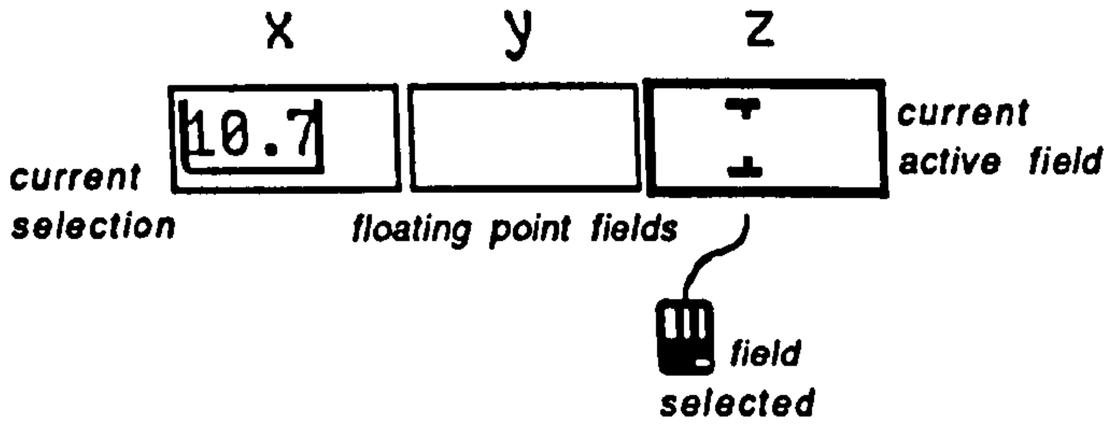


Figure 4.7.7.1.2.1.a. Selecting text to copy into another field.

When the editing menu is activated in a field other than the one containing the current selection, the selected text is validated against the method of the target field.

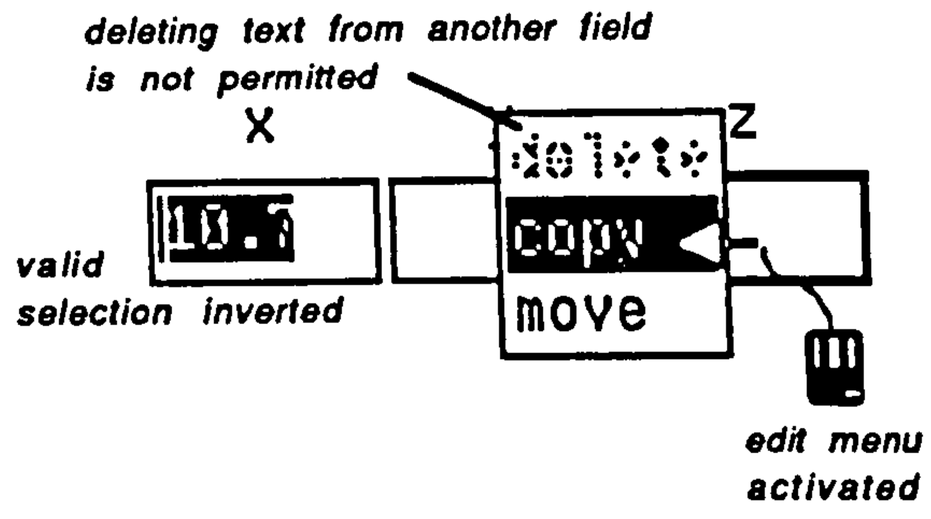


Figure 4.7.7.1.2.1.b. Activating the edit menu

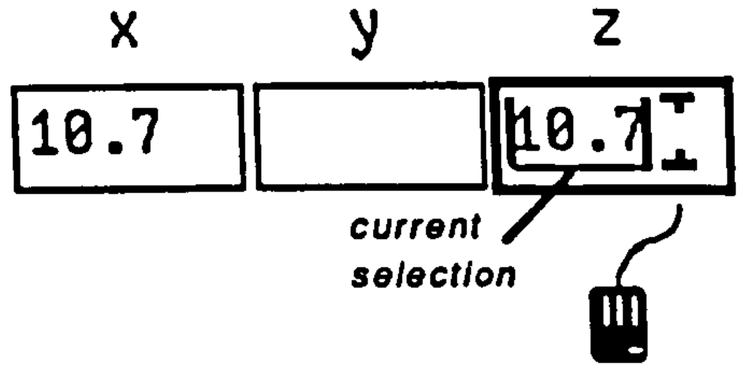


Figure 4.7.7.1.2.1.c. Selected text copied into current, active field. Note that the current selection has moved to the current field.

If it is a valid selection the text is inverted, figure 4.7.7.1.2.1.b, and may be cut or copied to the required field, figure 4.7.7.1.2.1.c. Otherwise the selection is made in the new field.

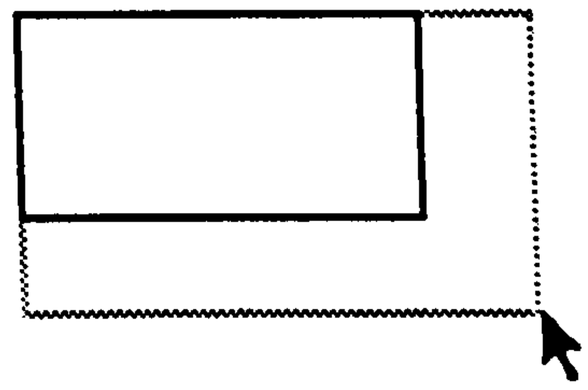
4.7.7.2. ▭ GRAPHICS FIELD

The graphics concept interpreter was originally intended to just display static bitmaped images. However, in some applications it is often necessary to interact with graphical objects. Although, during the development of the graphics interpreter, the author has deliberately avoided Apple's HyperCard approach to sensitized screen images (suffering from the same problems of natural language interfaces; ie the user expecting too much), the functionality of the graphics field is to be extended. It is anticipated that the interpreter will provide an object drawing layer similar to that found in applications such as MacDraft.

Operations performed on graphical entities will result in formatted utterances of the form:

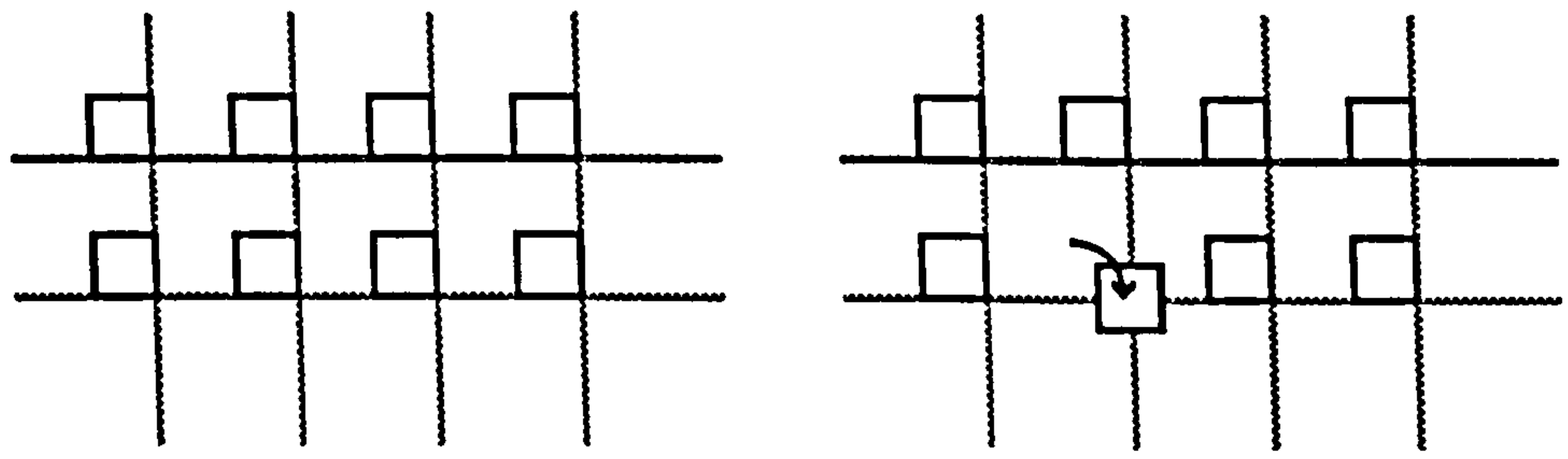
concept:attribute:new_value

eg:



room:size:15 7

This will enable domain specific knowledge to be applied directly to the operations performed on graphical representations of objects. For instance a structural grid may be displayed, as in figure 4.7.7.2.1a.



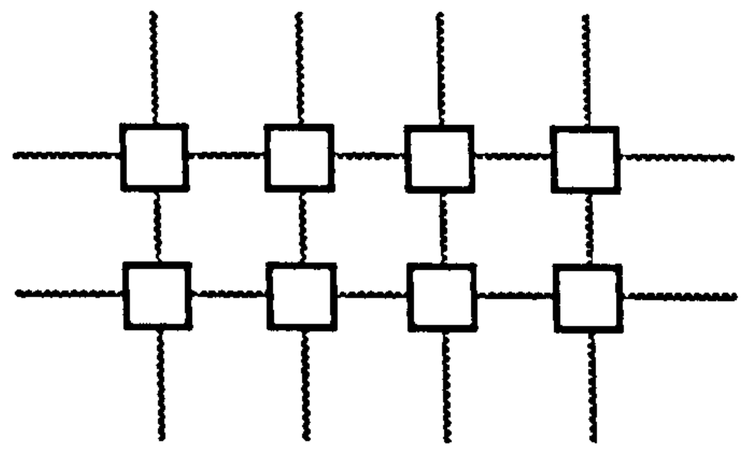
a b

Figure 4.7.7.2.1a and b. Manipulation of symbolic diagrams. Single instance of an object class is modified.

The user may specify, by dragging an iconic representation, a new column position, figure 4.7.7.2.1b. The corresponding utterance would be:

column_n:origin:x y

Explicit knowledge of the domain and task may establish the offset of the centre of the column to the intersection of grid lines and modify, by shifting, the position of all objects in the same class (column) by the same amount [BRIDGES 89], figure 4.7.7.2.1c.



c

Figure 4.7.7.2.1c. All objects re-positioned.

The recognition and generation of symbolic diagrams of this form is a convenient vehicle for implementing stylistic knowledge. The proportions and juxtaposition of volumes and components may influenced by encoded knowledge of a particular architectural style. A large organisation, for instance, may have a particular house style which if represented as a series of formal rules would enable all designers and technicians to work within the same shape grammar. This area of research is left for future investigation.

4.7.8. RESTRICTED RESPONSE

Restricted response fields limit the user's options to a number of pre-defined alternatives. This type of mechanism is therefore useful for novice users and for enabling the user to direct the dialogue by selecting appropriate topics. In the forms package there are two types of restricted response fields:

i) Boolean, true/false interpreters, and

ii) multiple choice fields.

4.7.8.1. BOOLEAN FIELD

A Boolean interpreter simply allows the value of a concept to be toggled between two pre-defined states. In the forms package this type of field is referred to as a button.

4.7.8.1.1. Button

Button fields are indicated by two overlapping planes, figure 4.7.8.1.1.1, representing the two value states of the concept.

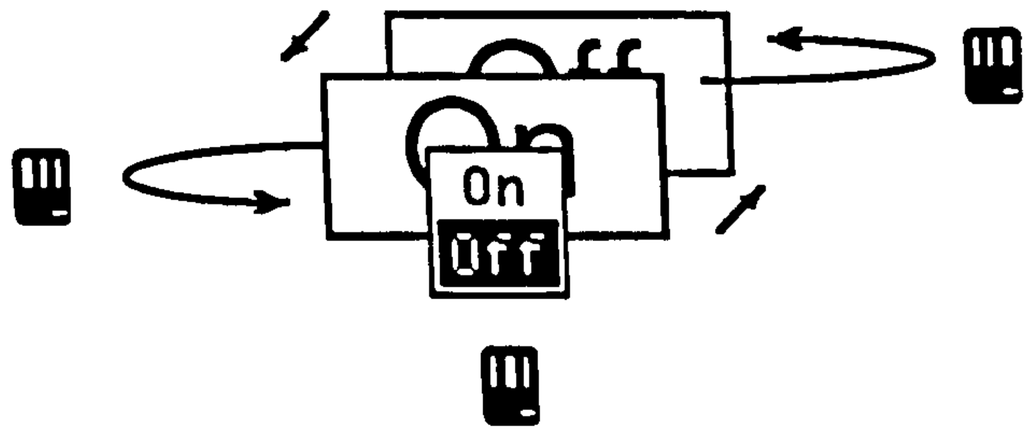


Figure 4.7.8.1.1.1. Conceptual view of a button field as a two state (Boolean) selection field.

Figure 4.7.8.1.1.2, illustrates the possible mouse interaction with a button field. A left button depression cycles the value counter-clockwise while the right button cycles the value in the opposite direction. The direction is unimportant for a Boolean field, but is implemented to ensure consistency with other forms of restricted input field. The middle mouse button activates a pop-up menu, indicating in textual form the alternative values. The cursor is automatically positioned over the alternative value, figure 4.7.8.1.1.1.

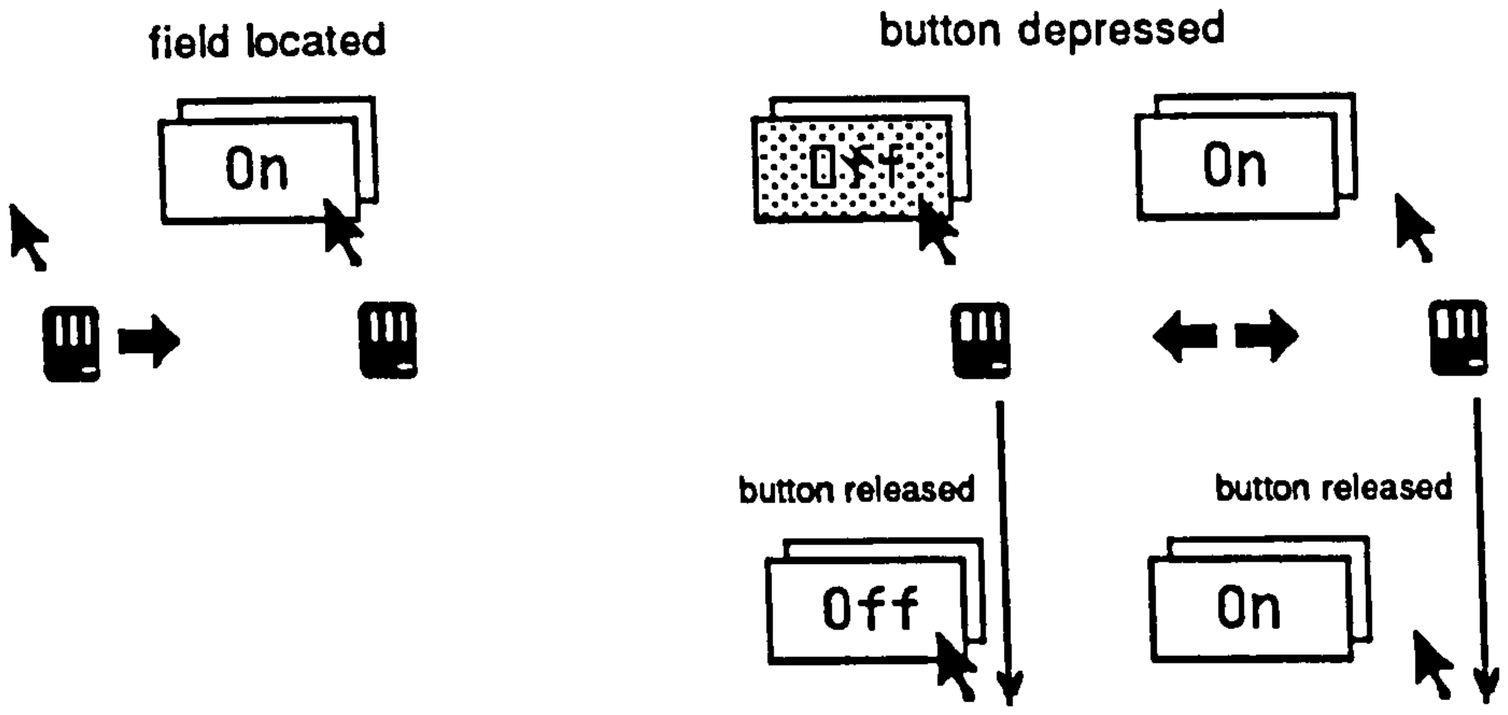


Figure 4.7.8.1.1.2. User interaction with button field.

The design of the button field was carefully considered to reflect the user's actions. When a button field is selected the alternative state is displayed and highlighted, figure 4.7.8.1.1.2a. The contents of the field remain obscured until the button is released. This is done to ensure that the user is fully aware of the implications of the selection. Moving the cursor out of the bounding box, while the initial button remains depressed, returns the contents of the field to it's original state, thus enabling the user to abort the selection safely, figure 4.7.8.1.1.2c

4.7.8.1.1.1. Obscuring concept values

Obscuring information is used in many situations as will become apparent (as in the case of the button field, above) and is achieved by ORing over a greyscale pattern. The process of obscuring concept values is used to indicate to the user that the value of a concept has changed or is about to change as a result of some user initiated event (with or without the new value being made visible). Although the process is described as obscuring concept values, it is important that information remains legible.

A greyscale pattern is defined as a percentage from 0% (white) to 100% (black). Four patterns (the most regular) out of a possible (theoretical) 100 patterns, illustrated in figure 4.7.8.1.1.1.1 ORred over a Helvetica narrow font together with a circle, where investigated for their appropriateness.

Figure 4.7.8.1.1.1.1. % grey scale pattern ORed to obscure text and images

The more dense patterns (50% and 25%) are more likely to obliterate information, while regular square grids (25%) are more likely to cause bit alignment problems; note the heavy top and bottom to the circle with the 25% pattern and the loss of text definition with 50% and 25% greyscale.

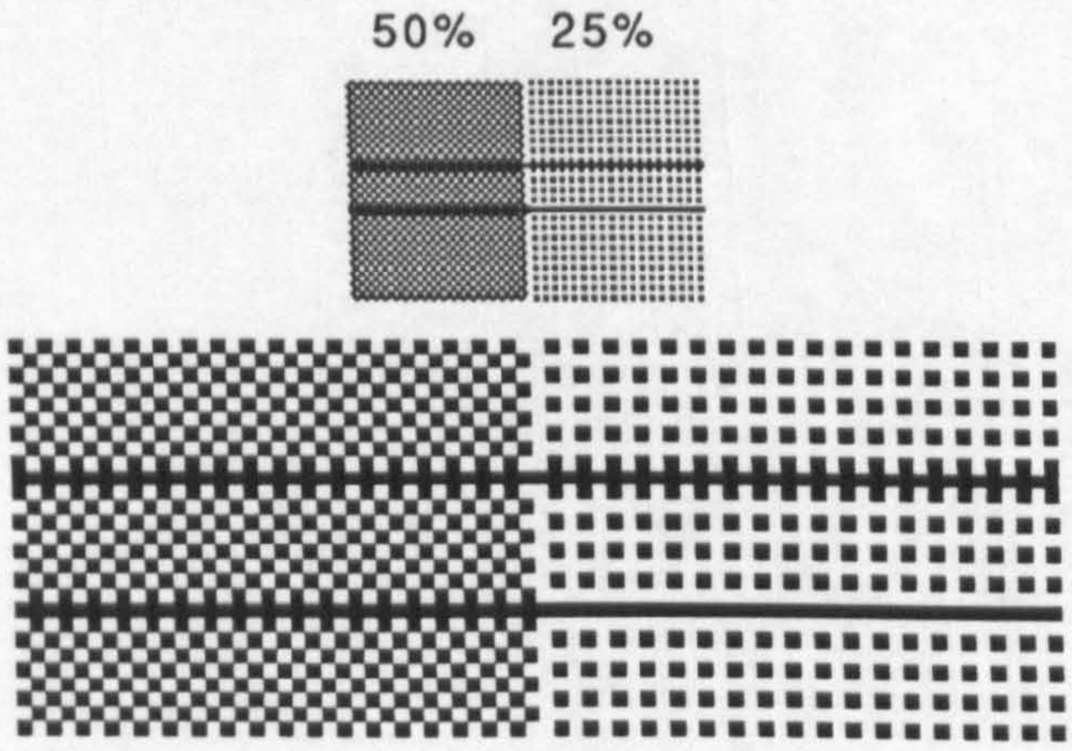


Figure 4.7.8.1.1.1.2. Bit alignment. Two lines offset by 8 pixels ORed over a 50% and 25% grey scale pattern. Note the fuzzy edge to the second line over the 25% pattern.

Theoretically, the more densely packed the bit pattern the more consistent the effect. Figure 4.7.8.1.1.1.2, illustrates two lines offset by 8 pixels ORed over a 50% (diamond grid) and a 25% (square grid). The quality of the two lines is more

consistent with the denser 50% pattern. However, the fonts used for text output are continuous shapes spanning several pixels in each direction.

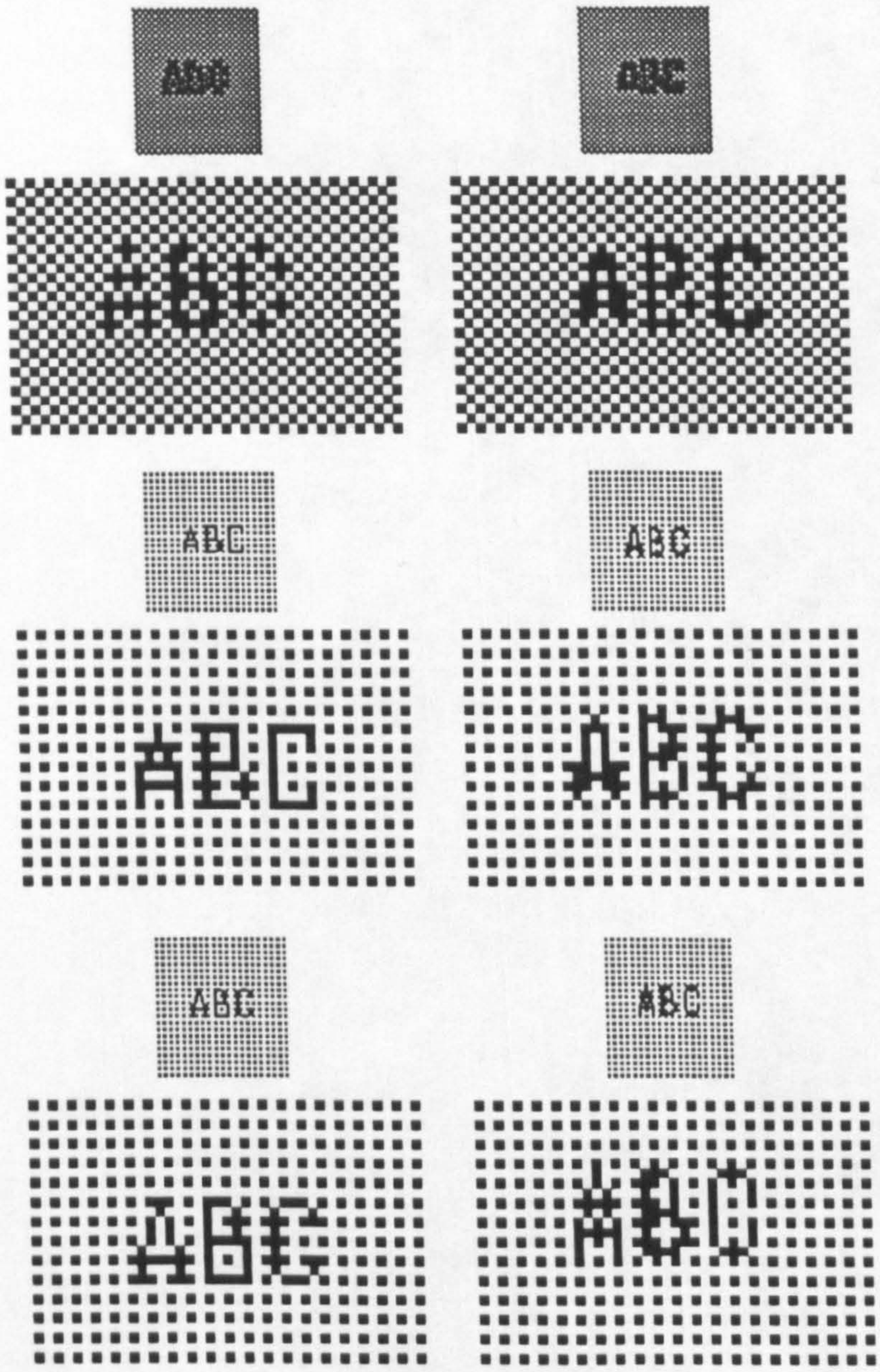


Figure 4.7.8.1.1.1.3. Alignment of text characters.

Figure 4.7.8.1.1.1.3, illustrates the effects of offsetting characters by a single pixel. Although some positions are acceptable ww does not facilitate automatic bit alignment which would be necessary for guaranteed consistency.

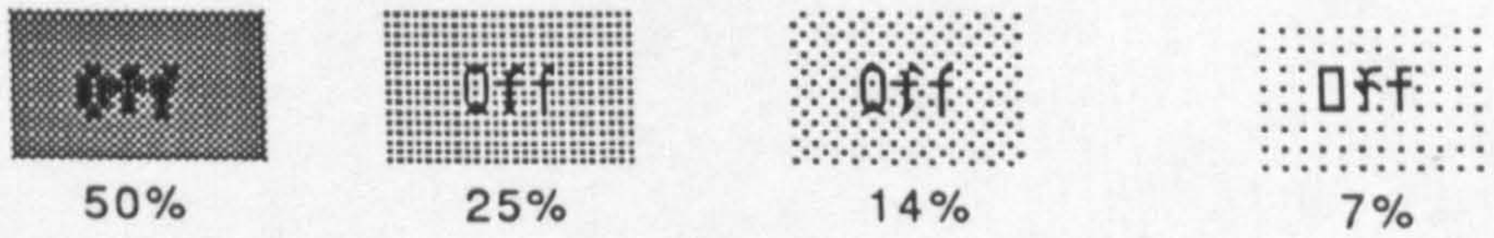


Figure 4.7.8.1.1.1.4. Bit alignment and registration.

Figure 4.7.8.1.1.1.4,illustrates the four most regular greyscale patterns ORed over the word "Off". The 50% and 25% patterns are obviously inappropriate. The 7% pattern is too open and causes a dazzling effect, the regular grid is also more likely to cause bit registration problems, figure 4.7.8.1.1.1.2, and interfere with vertical and horizontal lines (note the squareness of the letter 'O'). The 14% greyscale pattern, on the other hand, is a regular pattern composed of two offset square grids and therefore less likely to interfere with vertical and horizontal lines. The effects obtained using the 14% pattern are more consistent and is therefore used to obscure concept values throughout. WW does attempt to allign greyscale patterns during a rasterop using the WWREGISTER flag, however, it is difficult to predict how fonts are treated.

The button field is essentially a selection mechanism. The field will toggle between the first two items on the menu only. If only one item is specified the item will be inverted on selection and the formatted outputs are:

field name:USER_SELECTED:on

field name:USER_SELECTED:off

It is hoped that a background image may be specified using "background" followed by the name of an exrep image file. Text and images would be over printed. The background image may contain a mask - this would be useful for producing buttons with rounded corners, for instance.

The relationship between the attributes and the behavioural characteristics of button fields are indicated in the following figures. Attributes of interest are highlighted in bold text.

Proforma declaration	Event reported:(on selection)	
	switch:USER_SET:expert	switch:USER_SET:novice
new field name switch type button origin 10 1 size 10 1 label string User level label position fit above menu expert\ novice end field		

Figure 4.7.8.1.1.3. Button field: menu items as text

Proforma declaration	Event reported:(on selection)	
	switch:USER_SET:expert	switch:USER_SET:novice
new field name switch type button origin 10 1 size 64 64 pixels label string User level label position fit above menu expert\ novice end field		

Figure 4.7.8.1.1.4. Button field: menu items as images

Proforma declaration	Event reported:(on selection)	
	switch:USER_SET:A	switch:USER_SET:B
new field name switch type button origin 10 1 size 64 64 pixels label string User level label position fit above menu expert=A\ novice=B end field		

Figure 4.7.8.1.1.5. Button field: Aliased menu items

Proforma declaration	Event reported:(on selection)	
	switch:USER_SET:off	switch:USER_SET:on
new field name switch type button origin 10 1 size 10 1 label string User level label position fit above menu expert end field		

Figure 4.7.8.1.1.6. Button field: Single menu item inverted on selection

4.7.8.1.1.2. Selection dilemma

A fundamental dilemma was encountered when designing the button event handler namely; when the user selects the button is the current state being selected or the next state?

The answer to the problem depends upon the context the button is used in. Buttons may be used to acknowledge the current state of an event or may be used to present an alternative. It is up to the designer of the proforma template to decide which is appropriate for a given task.

Another issue associated with button selection is the notification response. Should changes of the concepts value be notified immediately the field is selected or after field de-selection; allowing the user to 'test alternatives?

Both options are supported. By specifying a selection border of zero any change in the state of the field value is reported immediately. This is useful for actual buttons. Selection border widths other than zero implies that the field must be de-selected before the event is reported. This facility is useful (for error prone novice users) where the state of a button may affect other processes.

4.7.8.2. MULTIPLE CHOICE

Multiple choice fields are appropriate when the value of a concept must be restricted to two or more alternative values.

4.7.8.2.1. Button

The button field, simply by extending the number of menu items defined in the field declarations, may be used as a cyclical selection mechanism, as illustrated in figure 4.7.8.2.1.1.

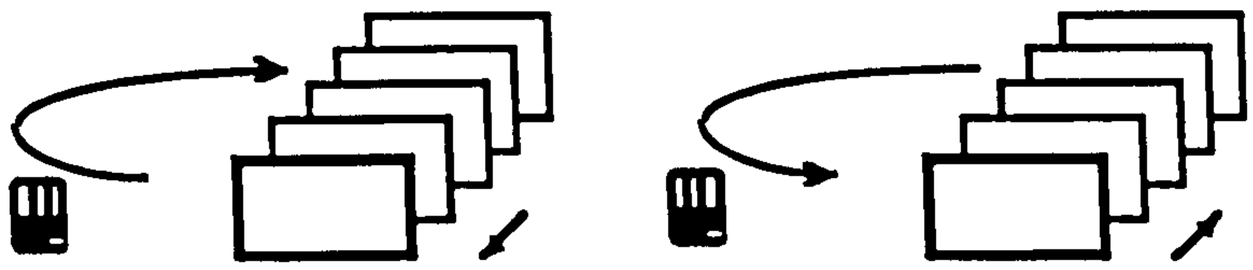
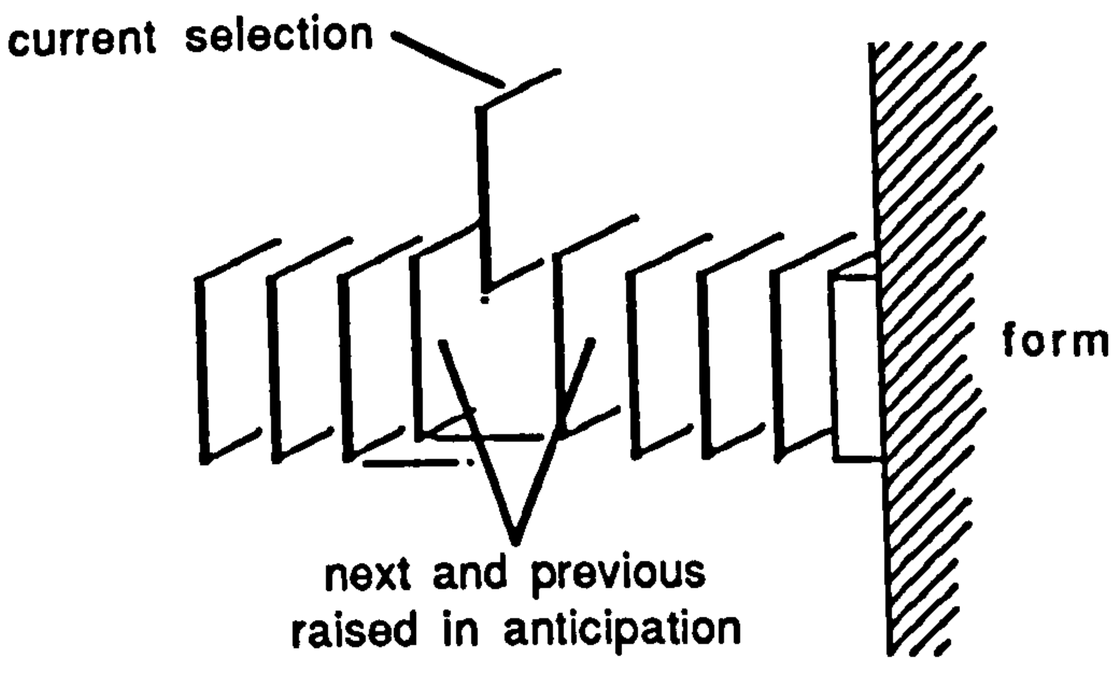


Figure 4.7.8.2.1.1. Conceptual view of a button field as a multiple choice selection mechanism.

A more obvious and useful interaction mechanism for multiple choice selection is the pop-up menu. From experience, the number of alternative values for some concepts (such as entries from a database, eg. building materials) is often substantial. In order to accommodate large number of items scrolable pop-menus would provide a solution. Rather than implement another pop-menu, a cascading pop-up menu has been designed and implemented.

4.7.8.2.2. Cascading popup

Menu items are displayed in a cascading image stack. This field interprets border as the separation between items on the stack (min 1 pixel max 1/4 field size) and therefore enables large numbers of items to be condensed into a relatively small region of the screen. Conceptually, this field may be visualised as a three-dimensional stack of bitmaped images, figure 4.7.8.2.2.1, which cascade out of the screen with a positive mouse event in the bounding rectangle of the interpreter.



To aid traversal the next and previous items are raised in anticipation of the next selection; providing a previewing mechanism.

Figure 4.7.8.2.2.1. Cross section through image stack

A button down event on the field or label pops the item stack and places the cursor at the current value.

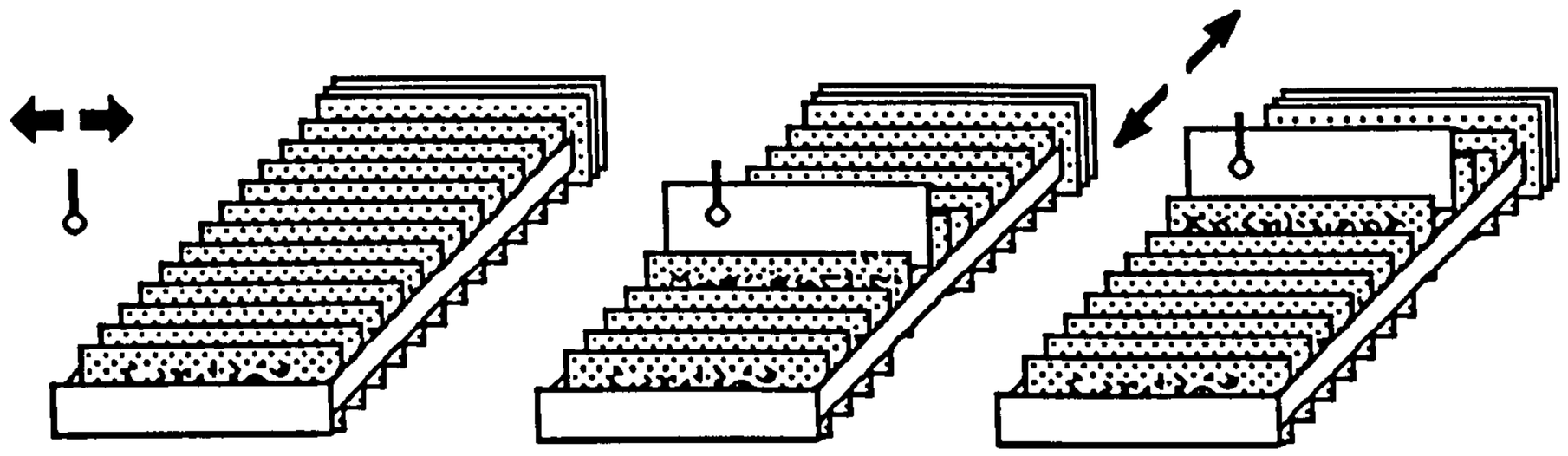


Figure 4.7.8.2.2.2. Cascading pop-up field; item selection.

Items are selected by moving the cursor back and forth along the stack. A positive selection is made if the button is released while an item is raised. Moving the cursor outside the boundary of the image stack aborts the selection preserving the current value on button release, figure 4.7.8.2.2.2.

As the label may be obscured by the stack (if positioned below), it is displayed at the front of the image stack, effectively creating a virtual draw front.

The following figures illustrate the relationship between the attributes and visual characteristics of this field type.

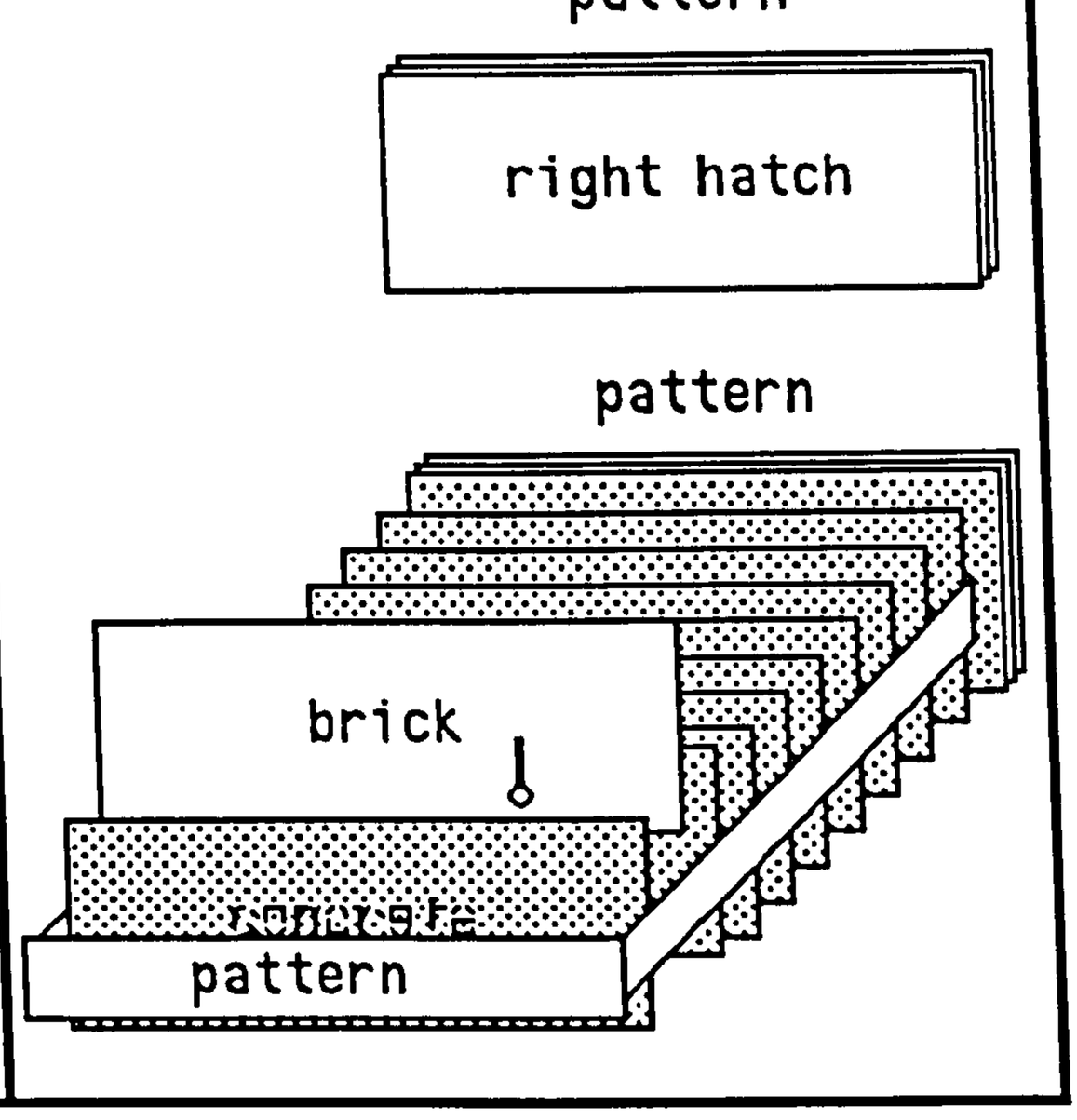
Proforma declaration	Event reported:(on de-selection) pattern:USER_SET:brick
new field name pattern **type popup** origin 10 1 size 10 2 label string pattern label position fit above **menu right hatch** **left hatch** **concrete** **brick** **earth** **marsh** end field	

Figure 4.7.8.2.2.3. Popup field: menu items as text

Graphical images may be substituted for text simply by creating an exrep bitmap image with the same name as the item and setting the forms image directory accordingly.

Proforma declaration	Event reported:(on de-selection)
	pattern:USER_SET:brick

```
new field
    name            pattern
    type            popup
    origin          10 1
    size            64 64 pixel
    label string    pattern
    label position  fit above
    menu            right hatch\
                    left hatch\
                    concrete\
                    brick\
                    earth\
                    marsh
end field
```

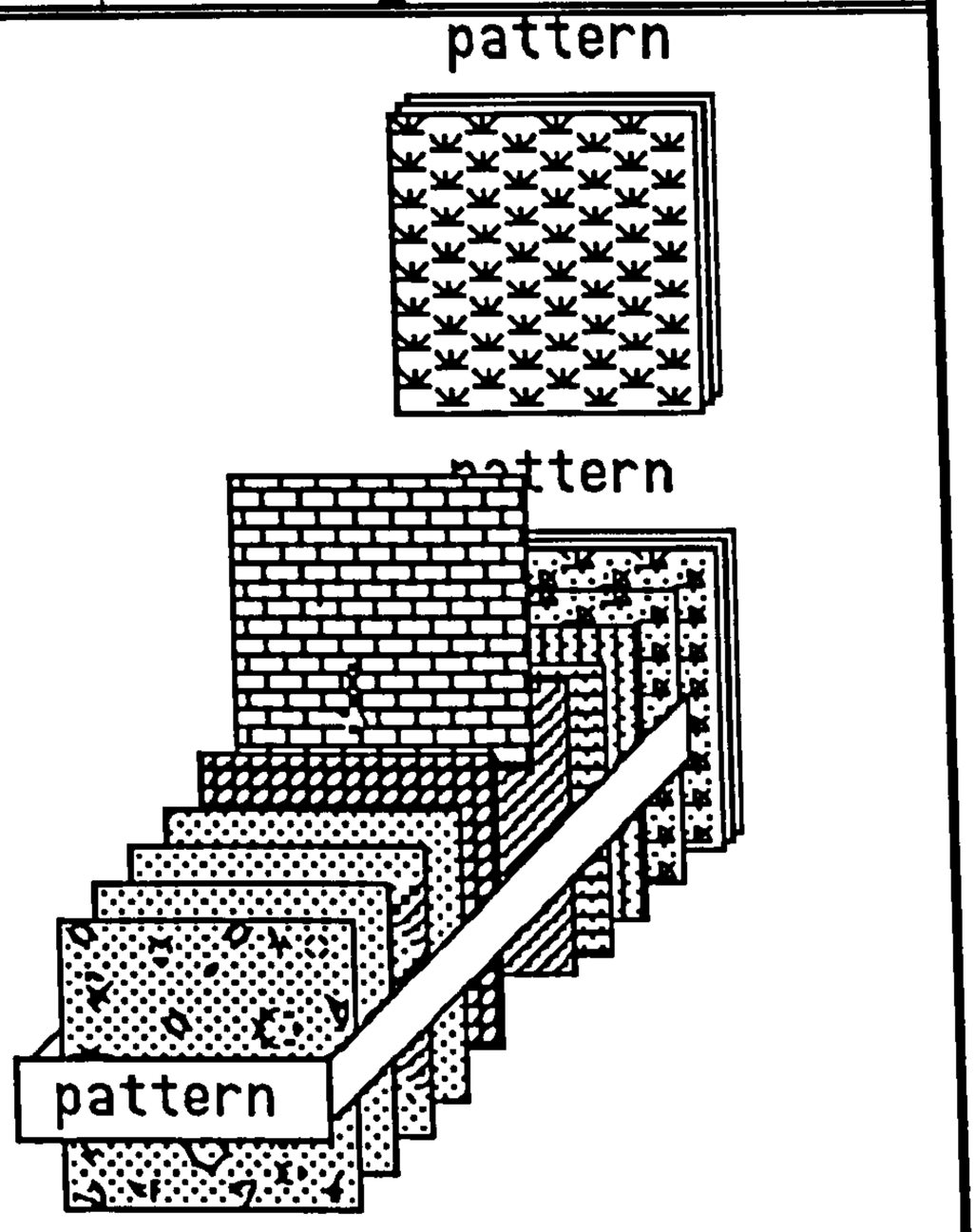


Figure 4.7.8.2.2.4. Popup field: menu items as images

As well as being a multiple choice field, the cascading pop-up is also useful for containing and retrieving information. For example (see figure 6.10.2.1), the results from a simulation may be displayed as a series of images and retrieved by the user simply by flicking through the stack.

It is envisaged that forms may also be stacked and retrieved using this mechanism.

Selecting the *previous* entry or using the right *alternatives* button over the label raises and positions the cursor over the appropriate entry. In order to perform this function the cursor is re-positioned by shifting it's offset accordingly. The actual cursor position is unchanged and results in premature (unexpected) de-selection (window leave events) if a cascading field is positioned near to the window border. With the current implementation of ww it is not possible to access low level cursor control mechanisms at the window manager level.

4.7.8.2.3. Menu

This field type displays the field menu as a list, figure 4.7.8.2.1. As the menu is changed so the field's depth (later option for width) will change dynamically in accordance with the number of items.

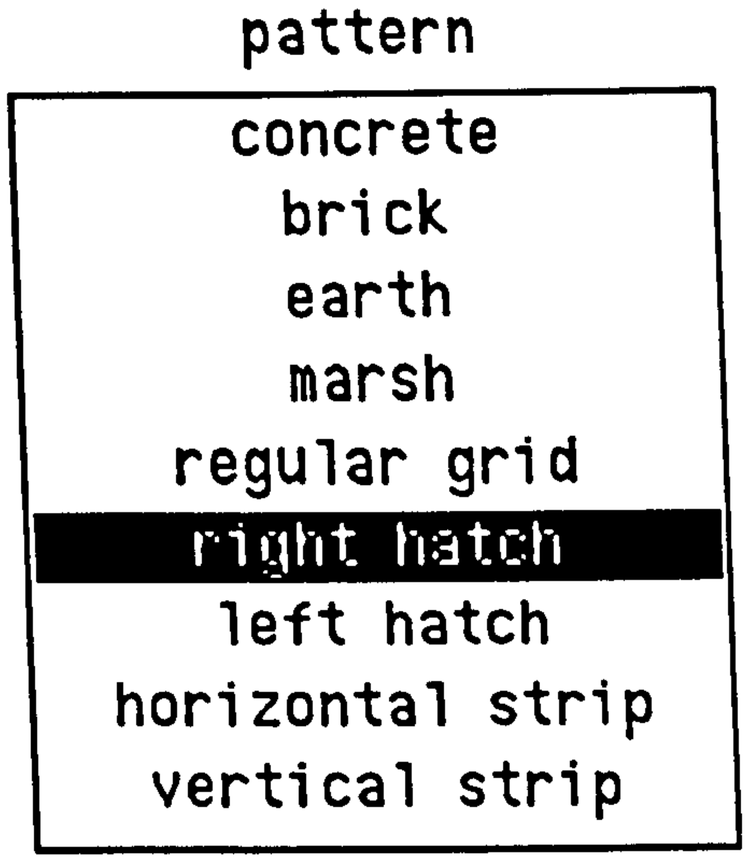
Proforma declaration	Event reported:(on de-selection) pattern:USER_SET:right hatch
new field name pattern type menu origin 10 1 size 15 0 label string pattern label position fit above menu concrete\ brick\ earth\ marsh\ regular grid\ right hatch\ left hatch\ horizontal strip\ vertical strip end field	

Figure 4.7.8.2.3.1. Menu field: menu items as text

The same rules for reporting the selection of items for button fields applies for the menu interpreter.

4.7.8.2.4. File browsing

In many applications it is often necessary to enter the names of existing files. In order to simplify the task of accessing files an interface to the UNIX directory structure is provided by a file browsing field. This is based upon the tx_tree file scanning function set within the ww library. Two pop-up menus are displayed, one for directories (.. indicating the one above) and the other displaying the names of files within the current directory. The user is able to traverse the UNIX directory structure simply be clicking a second mouse button over a selected directory.

The functionality of the browsing menu is accessed by the right mouse button and is similar in nature to the domain specific menu, providing alternative concept values. In the ww implementation the menus are activated by the left mouse button. In order to make the function consistent within the forms primitive set, the mouse

event handler was modified enabling alternative values (files) to be selected using the right mouse button.

In terms of keyboard interaction, the field is similar in functionality to that of a character field except that all white space characters are trapped and replaced by an underscore to avoid problems with the UNIX interface. File names are also restricted to fifteen characters in length, again for system compatibility. The popup menu associated with the label will scan the directory system relatively or absolutely (selected from a choice menu activated by the middle mouse button over the field label, figure 4.7.8.2.4.1).

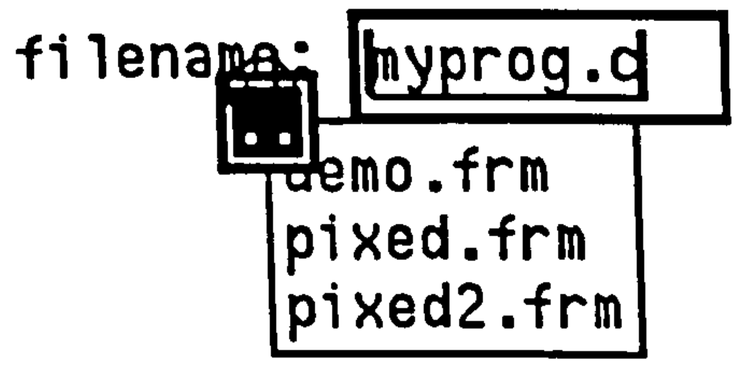


Figure 4.7.8.2.4.1. File browsing field. Note files are selected by placing the cursor over the text string. Directories are shown in a separate panel and may be entered by pressing the middle button while holding the right button down.

The files are cached and therefore the directory must be re-scanned before newly created files are accessible from the menu. This facility is also accessed by means the middle mouse button, figure 4.7.8.2.4.2.

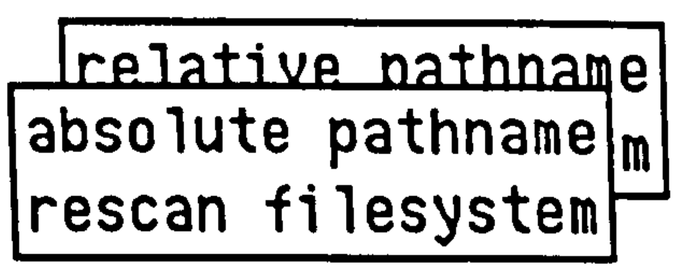


Figure 4.7.8.2.4.2. Re-scanning and search specification menu.

4.7.8.3. SLIDERS

Sliders are useful for restricting a user's entry to a range of numeric values. Restricting the attribute set proved to be troublesome for a number of field interpreters; particularly sliders.

For purely investigative reasons it was felt that the appearance of a slider should represent data pictorially; eg a thermometer for setting or displaying temperature.

In order to achieve a certain level of generality the following set of attributes where deemed necessary:

image
background
image mask
positive / negative hit zones
....etc

Although other field types would ignore this superfluous information, this attribute set was too specific.

A compromise (a fudge) was derived. A special "image" file containing the afore mentioned attributes together with exrep images was derived. The format for this file is outlined below.

image
background image
masks
hit zones etc

The slider field has been temporally withdrawn until time permits to tackle this problem again. A simplified slider would perhaps be sufficient for the time being.

4.7.9. OBSCURE FUNCTIONALITY

As a result of the object oriented approach adopted, it is relatively easy to add new concept interpreters.

A number of domain specific concept interpreters have been developed; including a date field and a vector image display field, and have been integrated within the set of interpreters owing to their generality within a particular domain.

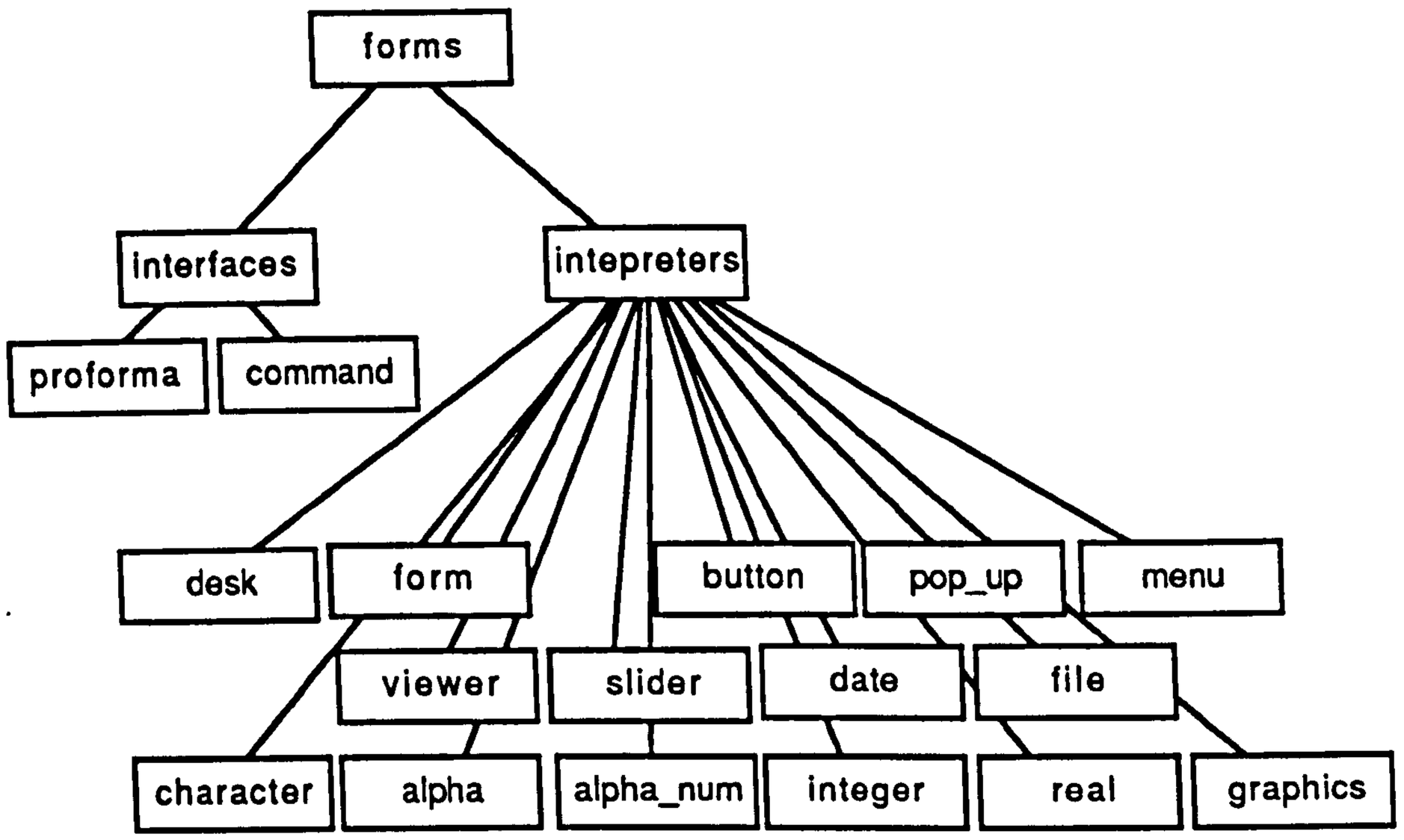


Figure 4.7.9.1. Forms directory structure.

Figure 4.7.9.1, illustrates the source code file structure for the forms package, indicating the separation between the forms package and the concept interpreter primitive set.

4.7.9.1. DATE

The concept of time is used extensively in the field of building simulation. It is often not necessary to specify simulation constraints in terms of hours, minutes and seconds. Therefore, although a date field does not immediately appear to be a common interpreter type, the concept of date is sufficiently generic to be included within the forms primitive set. A time field is also being developed.

In order to avoid ambiguity, which may result from an unrestricted form of data input, the date field restricts entry of dates by means of a popup calendar, figure 4.7.9.1.2, which is essentially an interface to the Unix *timeval* structure. The initial

value, unless otherwise specified, is set to the current (system clock) date in American format. The date is displayed in a fixed size box in the current data font in one of two basic styles:

i) style European -> day:Month:year

ii) style American -> Month:day:year

Dates may be specified, when using the proforma or command interfaces, as a series of numbers separated by colons ":" or as numbers but with the month specified by a character string. If this method is used only three characters are checked and the first character must be an upper case character. Care must be taken when mixing formats, figure 4.7.9.1.1.

Data	Numeric interpretation	American interpretation	European interpretation
8:2:1964	8:2:1964	Aug:02:1964	08:Feb:1964
Aug:02:1964	8:02:1964	Aug:02:1964	00:Feb:1964

Figure 4.7.9.1.1. Date format specification

Date: 1:12:1989	Numeric representation
Date: 01:Dec:1989	European format
Date: Jan:12:1989	American format

The date field does not support a numerical representation, figure 4.4.7.9.1.1, owing to the potential ambiguity which may arise when interpreted by the user.

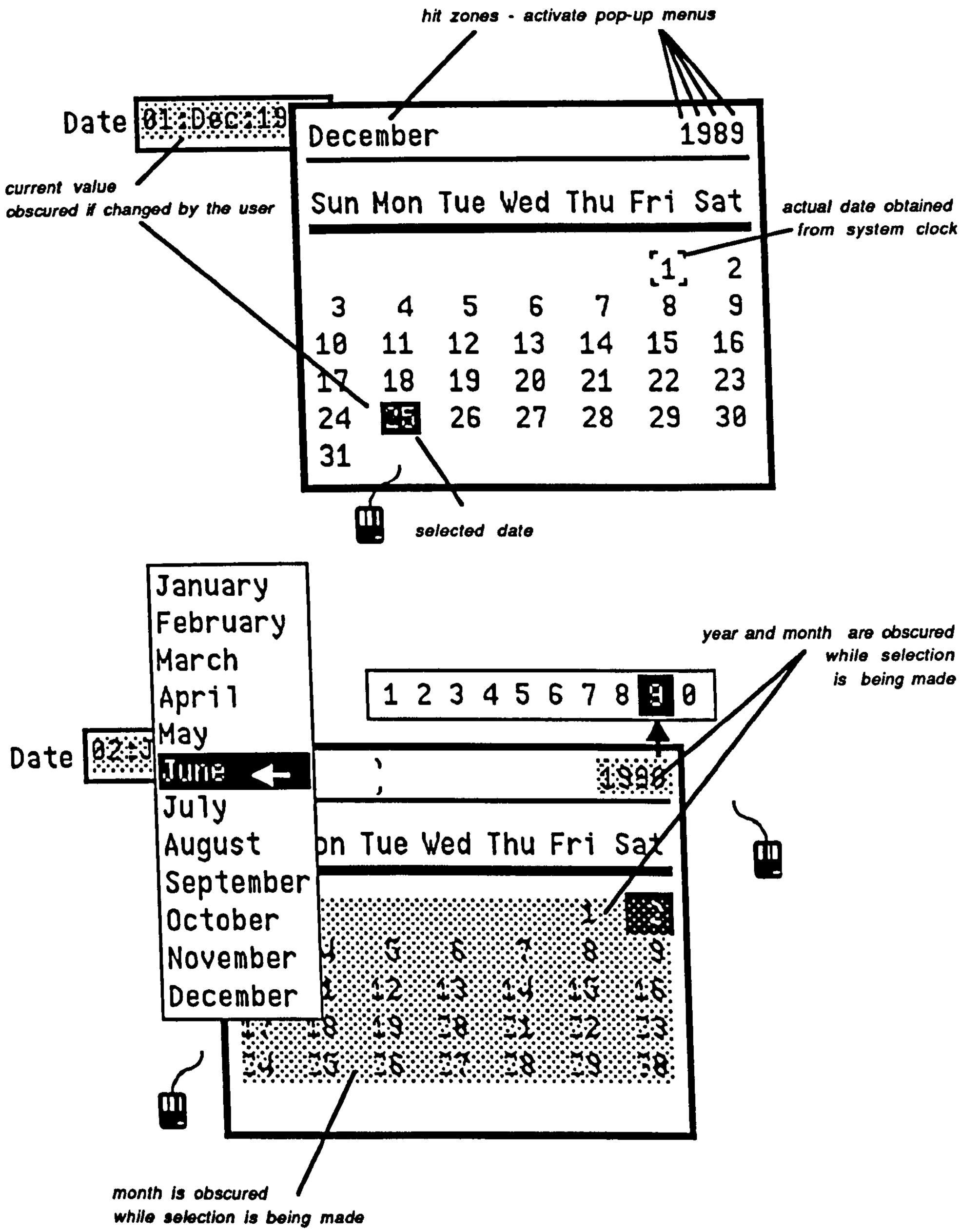


Figure 4.7.9.1.2. Date field: Popup calendar.

Figure 4.7.9.1.2, above, illustrates the user interaction mechanisms supported by the date field.

4.7.9.2. VIEWER DISPLAY FIELD

A vector image interpreter has been encoded owing to the constant use of perspective images generated by an in house application program. Although the graphics field is capable of responding to move and draw commands, the number of vectors making up a perspective image is often too great to transmit down a UNIX pipe; resulting in blocked pipes and unacceptable response time. Therefore, rather than writing a rasteriser to a neutral format bitmaped image (again increasing system response time) a field, figure 4.7.9.2.1, capable of reading and displaying a single ABACUS viewer picture file was developed.

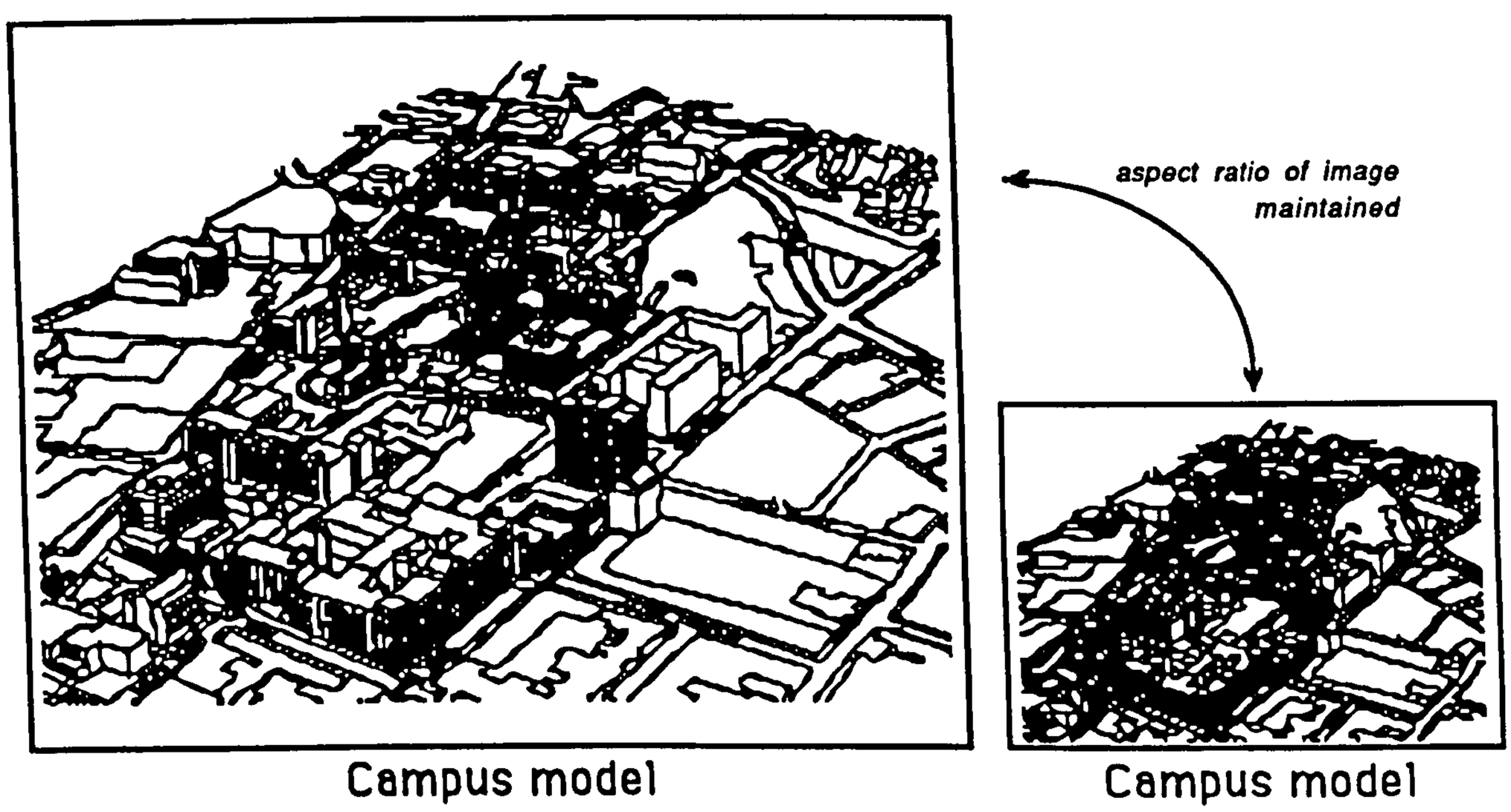


Figure 4.7.9.2.1. Viewer display field.

The display methods scale the image to fit the bounding rectangle of the field, maintaining the aspect ration of the original picture.

It is possible to include several perspective views of a model within a single picture file. Although currently, it is only possible to display the first image specified within the file, it is anticipated that later versions of the this interpreter will adopt the characteristics of the cascading pop-up field and thus enable the user to access several images from the same display area.

Using combinations of these primitive concept interpreters it is possible to construct an interfaces for the description of a building product model. This is discussed further in chapters 5 and 6. In order to logically related and arrange concepts together a meta-concept interpreter has been developed.

4.7.10. META-CONCEPT INTERPRETERS

Meta-concepts are groups of logically related concepts and meta-concepts. Within the forms package a *form* is used to display and allow the end-user to manipulation meta-concepts.

4.7.10.1. FORMS

A form may be thought of as sheet of paper on which fields are printed. The categorisation of concepts (and meta-concepts) is achieved by nesting forms. As concept (fields) may be created dynamically during run-time, the eventual number of concepts on a form is un-known (and likely to be many) a form must therefore be capable of growing in size to accommodate an increasing number of concepts. Initially this may seem straightforward. However, there is a limit to the physical dimensions of the screen which introduces the need for scrolling forms.

Scrolling in current interface toolkits (SunView, MacApp) is limited to panels containing text (scrolling menus) or graphics. Where other entities such as buttons and other selection devices are placed on panels they are only used to extend the size of a single window. The forms package in dealing with multiple levels of conceptual decomposition must accommodate this by ensuring that forms may be nested to any level.

The need for n-level nesting has resulted in a particular graphical structure for a form, figure 4.7.10.1.1. A form is composed of a bitmap page on which graphical entities such as fields (indicated by shaded primitive shapes) may exist, and a window onto the form's page, which is itself a graphical entity and as such may exist on other forms. Thus forms containing concepts may be nested to any depth.

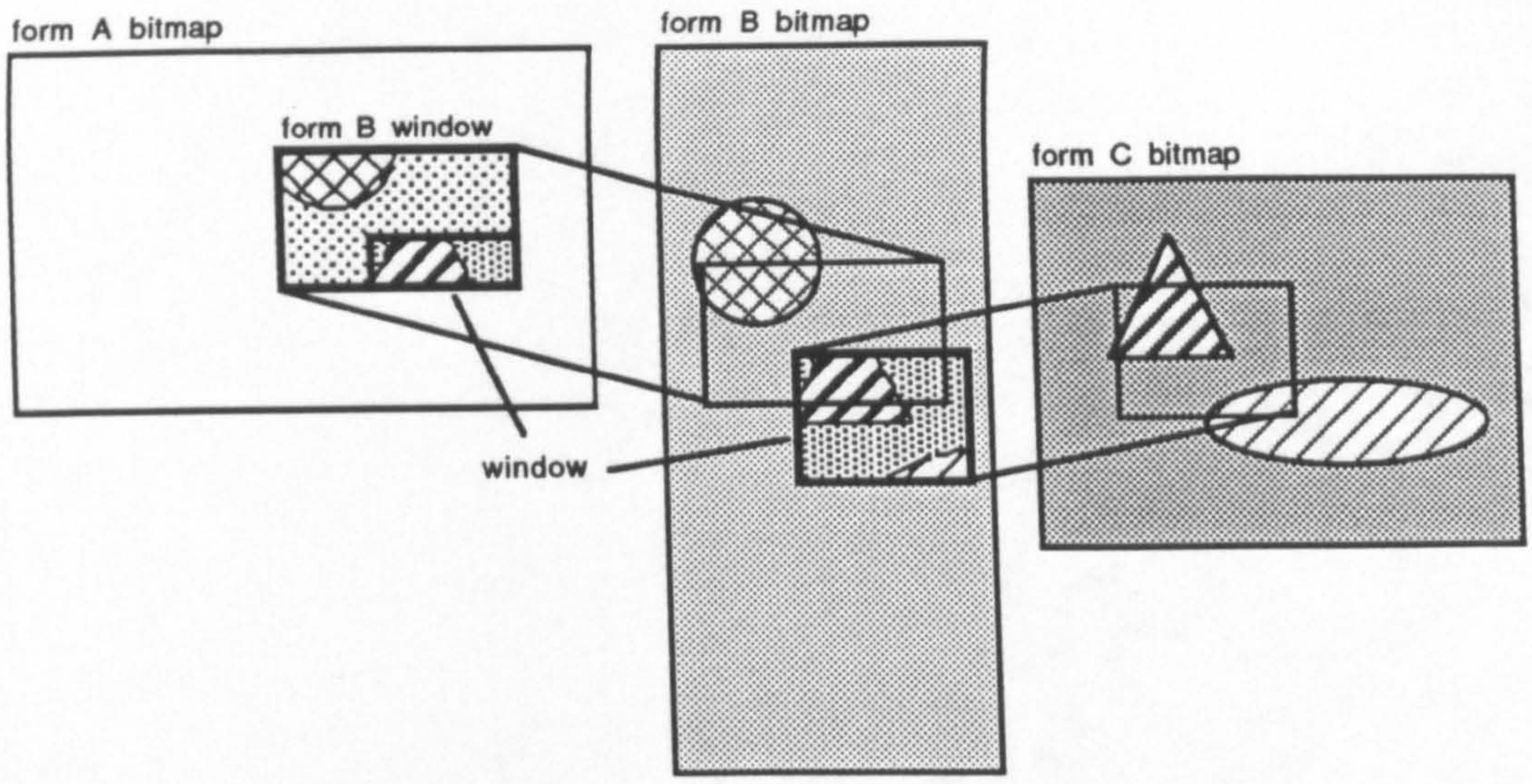


Figure 4.7.10.1.1. Conceptual view of a form.

4.7.10.2. DYNAMIC MEMORY ALLOCATION AND (RE)SIZING

In order to facilitate the introduction of new concepts during run-time execution, dynamic memory allocation is used extensively throughout the forms package to create new forms and fields. The only limiting factor affecting the number of concepts which may be represented by the forms package is determined by the memory capacity of the hardware platform.

The position of fields and forms cannot be predicted. Two options are available for sizing forms: either all forms are created large enough to accommodate any field or the form is re-sized as new fields (concepts) are introduced. The later is the preferred option as this ensures that the memory requirements of the forms package are kept to a minimum. Figure 4.7.10.2.1., below, illustrates the re-sizing procedure.

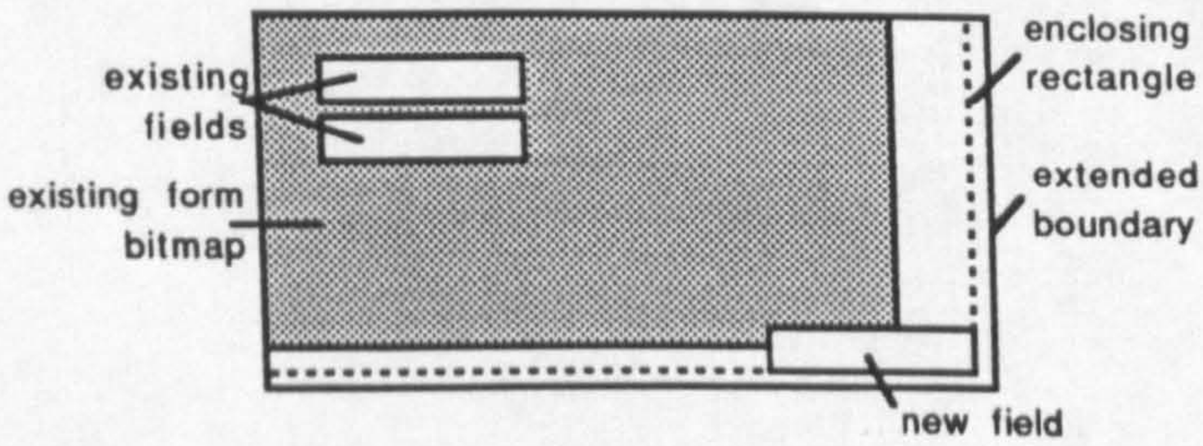


Figure 4.7.10.2.1. Dynamic (re)sizing. The rectangle enclosing the existing form and new concept is extended in the direction of the enclosing rectangle for user comfort. The extended boundary rectangle is used to allocate a new bitmap.

It is not yet possible to re-allocate bitmaps in the same sense as character strings in C can be re-allocated (extending the array length but preserving the contents of the string). Therefore in order to extend the size of a form's bitmap a rectangle, enclosing the original form boundary and that of the newly introduced field, is established. To ensure that the field is comfortably accommodated the enclosing rectangle is extended by a proportion of the fields font size. This extended rectangle is then used to allocate a new (clear) bitmap which replaces the existing form bitmap, figure 4.7.10.2.2.

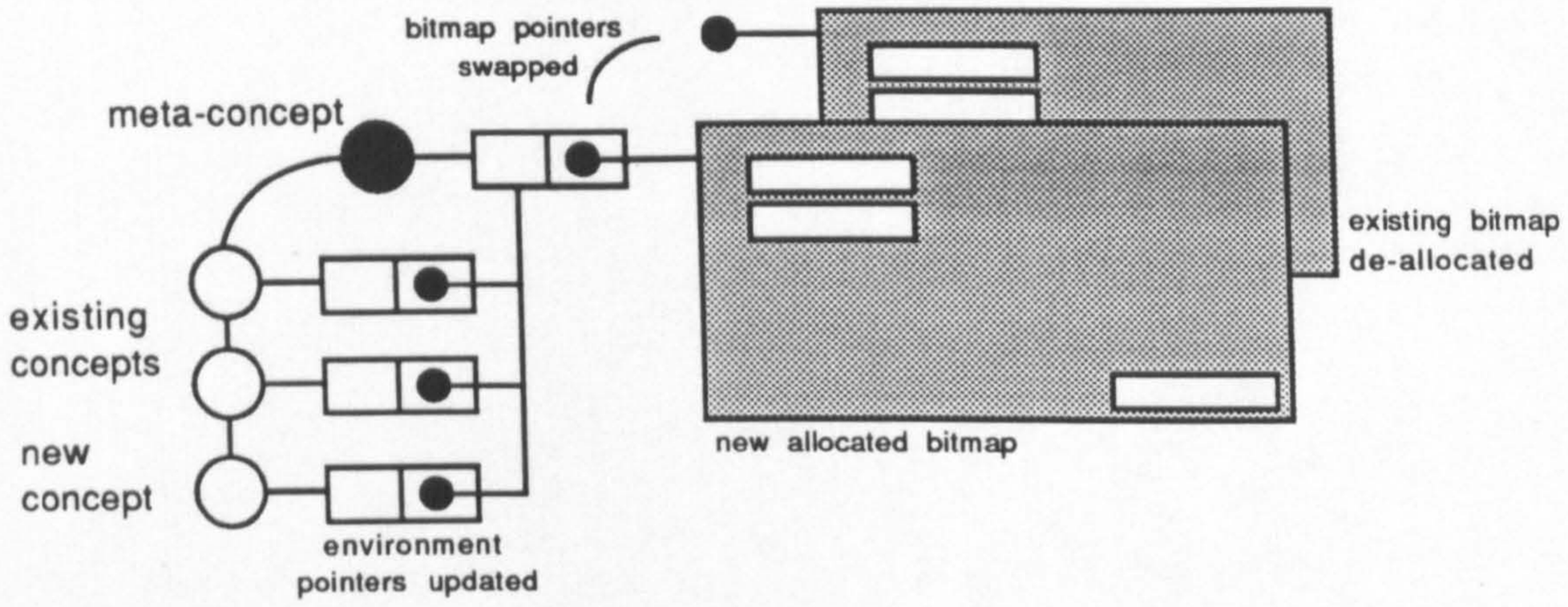


Figure 4.7.10.2.2. Dynamic substitution of bitmap pointers.

Once substituted the memory tied up in the original bitmap is freed and each descendant of the meta-concept is informed of the change and re-display themselves on the new bitmap. With a form containing many fields the process of re-displaying all the fields takes an unacceptable length of time. Therefore to improve the response time when new concepts are introduced the original bitmap image is copied onto the new bitmap once background shades have been pasted. It only remains for the new concept to display itself for the first time. Although the process has not been timed there is a significant improvement in response time.

4.7.10.3. SCROLLING

Once re-sized the form bitmap is no longer the same size as it's window and therefore the form may be scrolled. This is achieved by simply moving the form's window across it's page.

Although scroll bars (to aid absolute positioning and indicate the extent of the user's task) have not yet been implemented scrolling control is achieved using the middle mouse button which when depressed indicates the possible directions of

scrolling by modifying the cursor pattern. The current cursor patterns relating to form scrolling are summarised in the following diagram, figure 4.7.10.3.1.

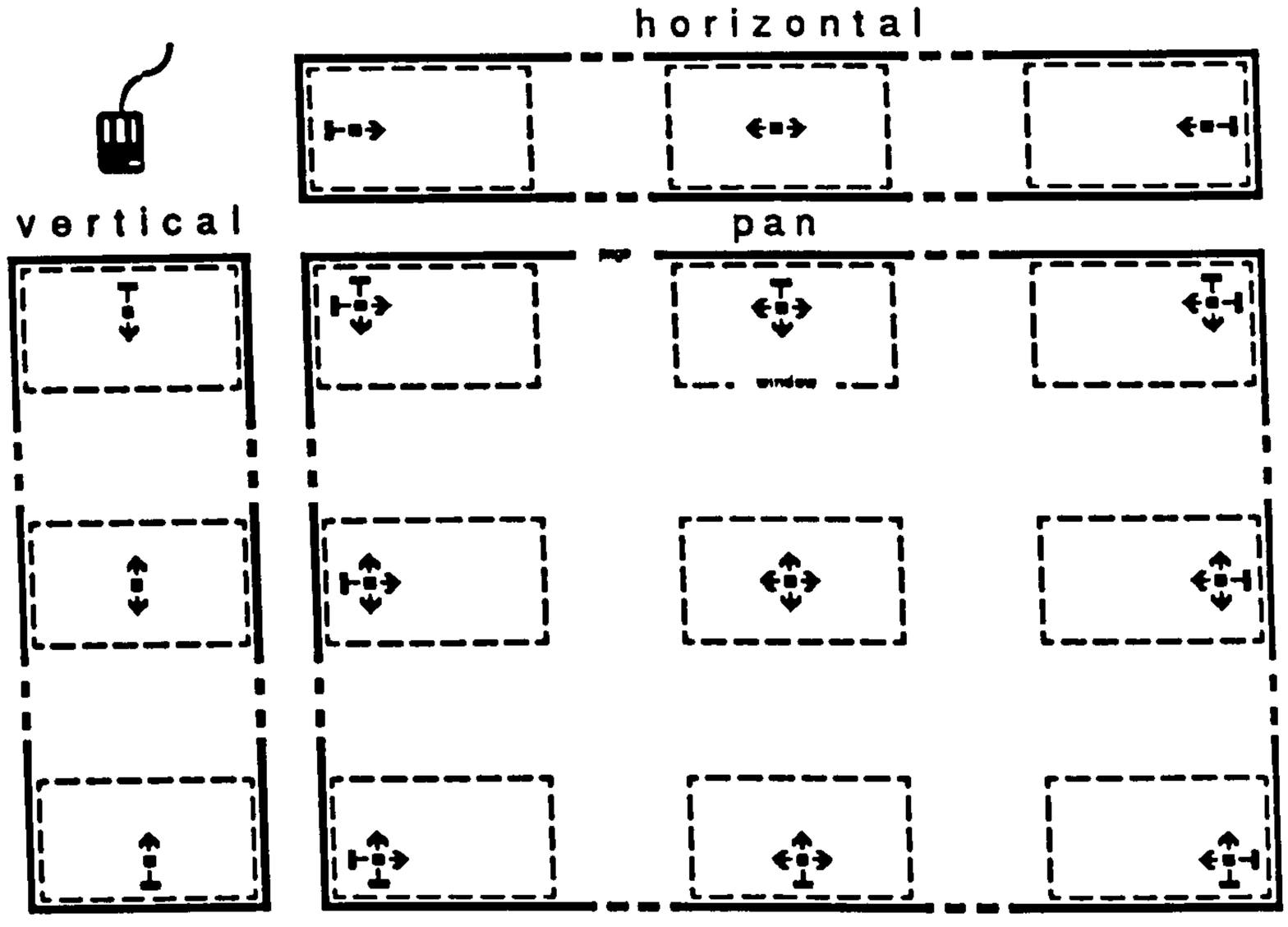


Figure 4.7.10.3.1. Cursor patterns indicating scrolling direction for form page/window variations.

4.7.10.4. DESK

A special meta-concept interpreter known as a desk facilitates the mapping of forms onto the window managers desktop and hence to the user. The desk is simply a form (domain meta-concept) in the shape of a window and as such it is possible to re-size it's bitmap and therefore is also capable of scrolling.

4.7.10.5. REFRESH

Forms provide the basic environment for fields to exist. In addition to displaying a border around a form or field, the entire contents of a field or form is refreshed, on selection, to the top-level desktop, figure 4.7.10.5.1. This ensures that the user is aware of the current value of the chosen concept.

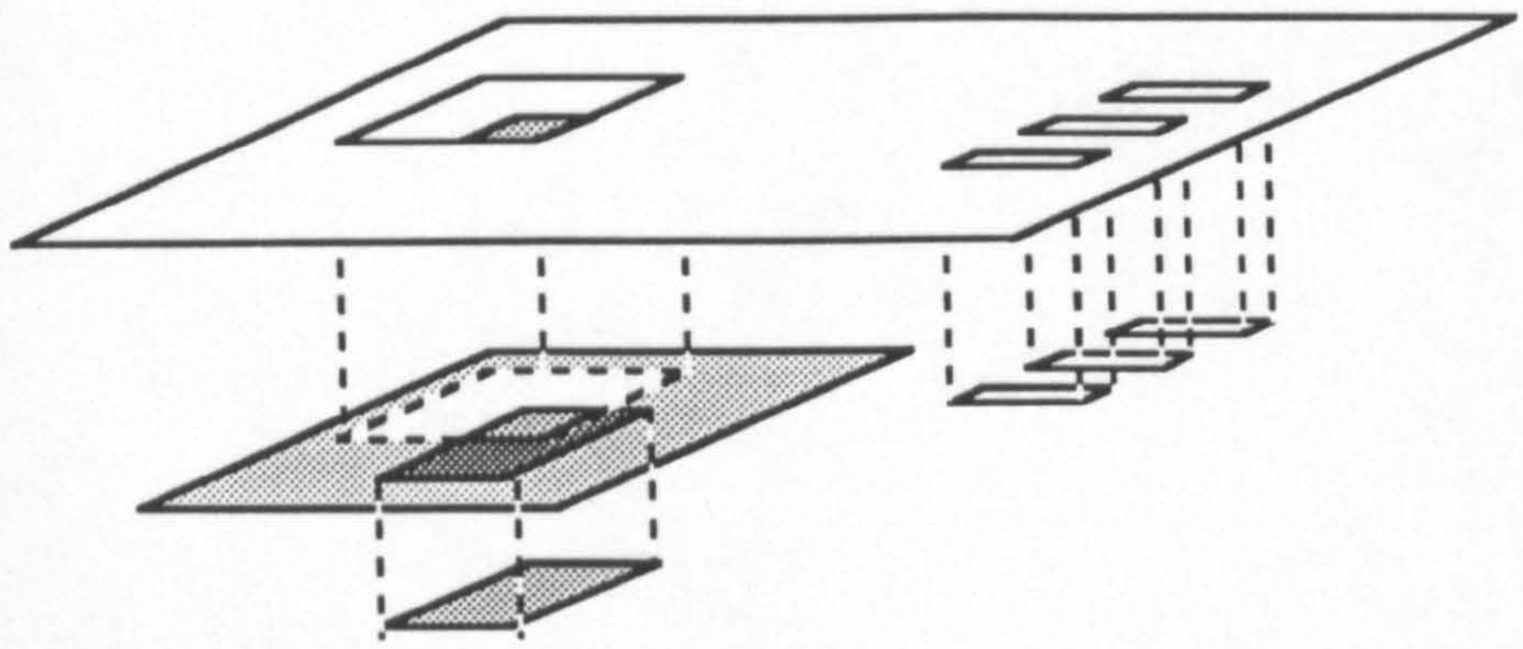


Figure 4.7.10.5.1. Form stack

Refreshing provided an additional unforeseen bonus with respect to overlapping forms. When two or more forms overlap a selection event in any of the form boundaries will make that form active and therefore pop it to the forefront, figure 4.7.10.5.2. The result is a simple window manager.

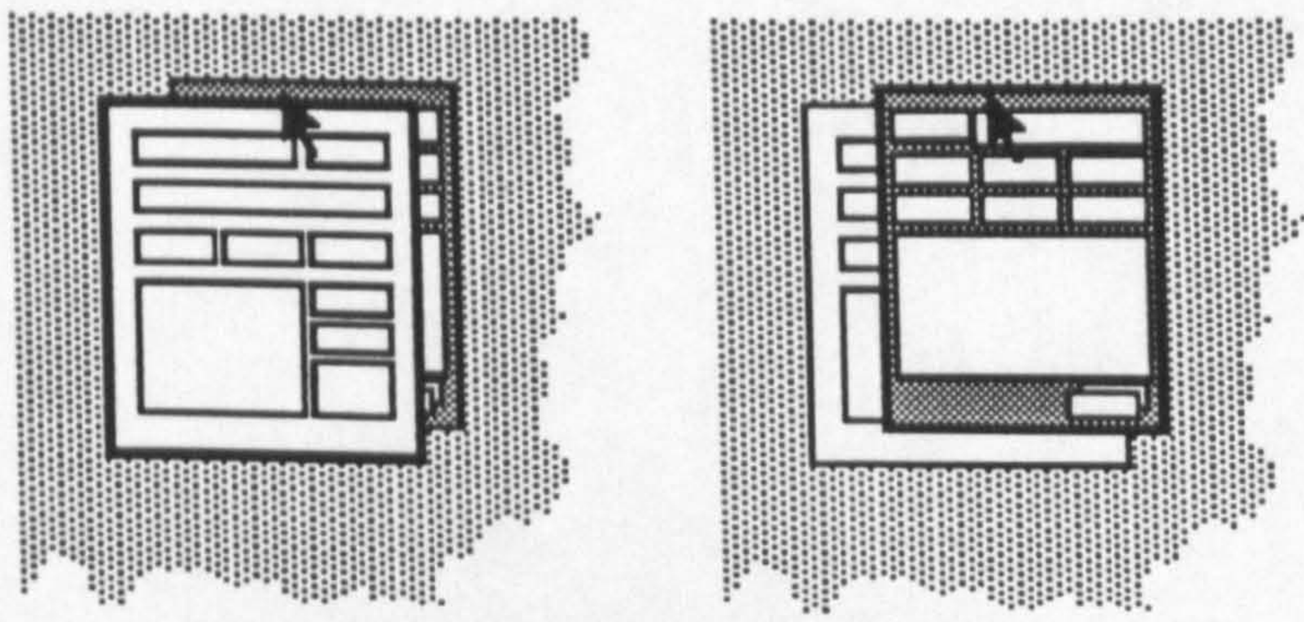


Figure 4.7.10.5.2. Popping forms.

4.7.10.6. REMOVING DISTRACTIONS

In situations where the acquisition of large amounts of data is required the user may feel overwhelmed or distracted by additional task un-related forms or fields. Two methods for removing these distractions have been provided:

i) iconsing and

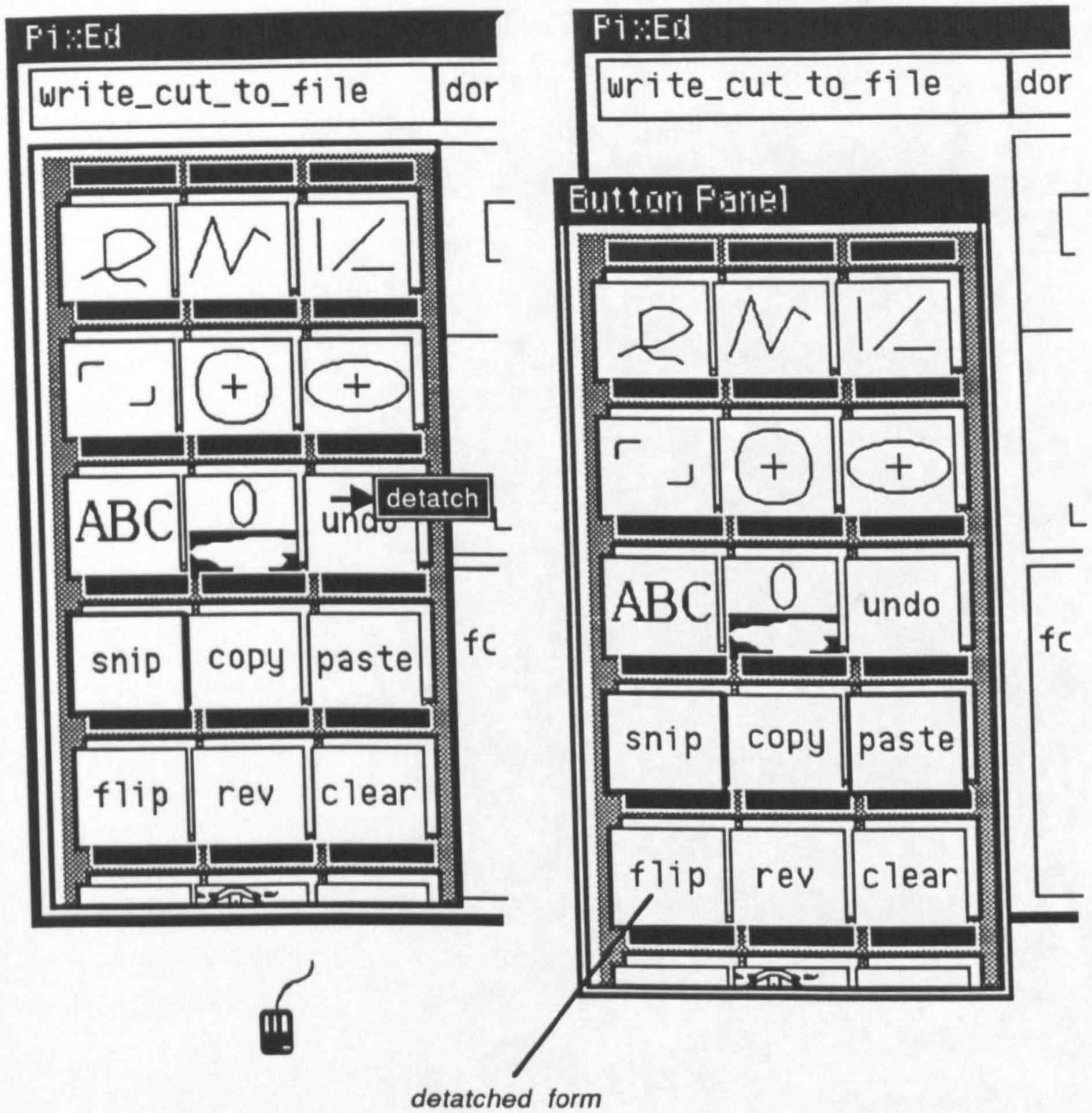
ii) detaching forms.

4.7.10.6.1. Iconising

In addition to the scrolling mechanism forms may be iconised into smaller regions of the screen. In addition to increasing the available screen real estate it allows the user remove selected meta-concepts from view to concentrate on other issues.

4.7.10.6.2. Detaching a form

Another method of selectively addressing concepts has been provided. By means of a positive mouse event inside and near to a form's bounding rectangle an option to detach the form is given, figure 4.7.10.6.2. When selected the form is detached from the main desktop and transformed into a window, allowing the main window to be iconised.



detatched form

Figure 4.7.10.6.2.1. Detaching a form.

Although it is not possible to re-attach a form to a desk, once implemented it would allow the user to export and import information to and from other applications.

4.7.10.7. TYPES OF INPUT.

There are two basic forms of input:

i) selection using pointing devices and

ii) keyboard input.

4.7.10.8. DIRECTING INPUT

All form oriented input is achieved by means of the mouse and keyboard. The mouse is used to direct input by selecting or focusing a concept. When selected the field is said to be *active*. The current input status of a field is indicated by a selection border which is pasted around the field, figure 4.7.1.1.1.

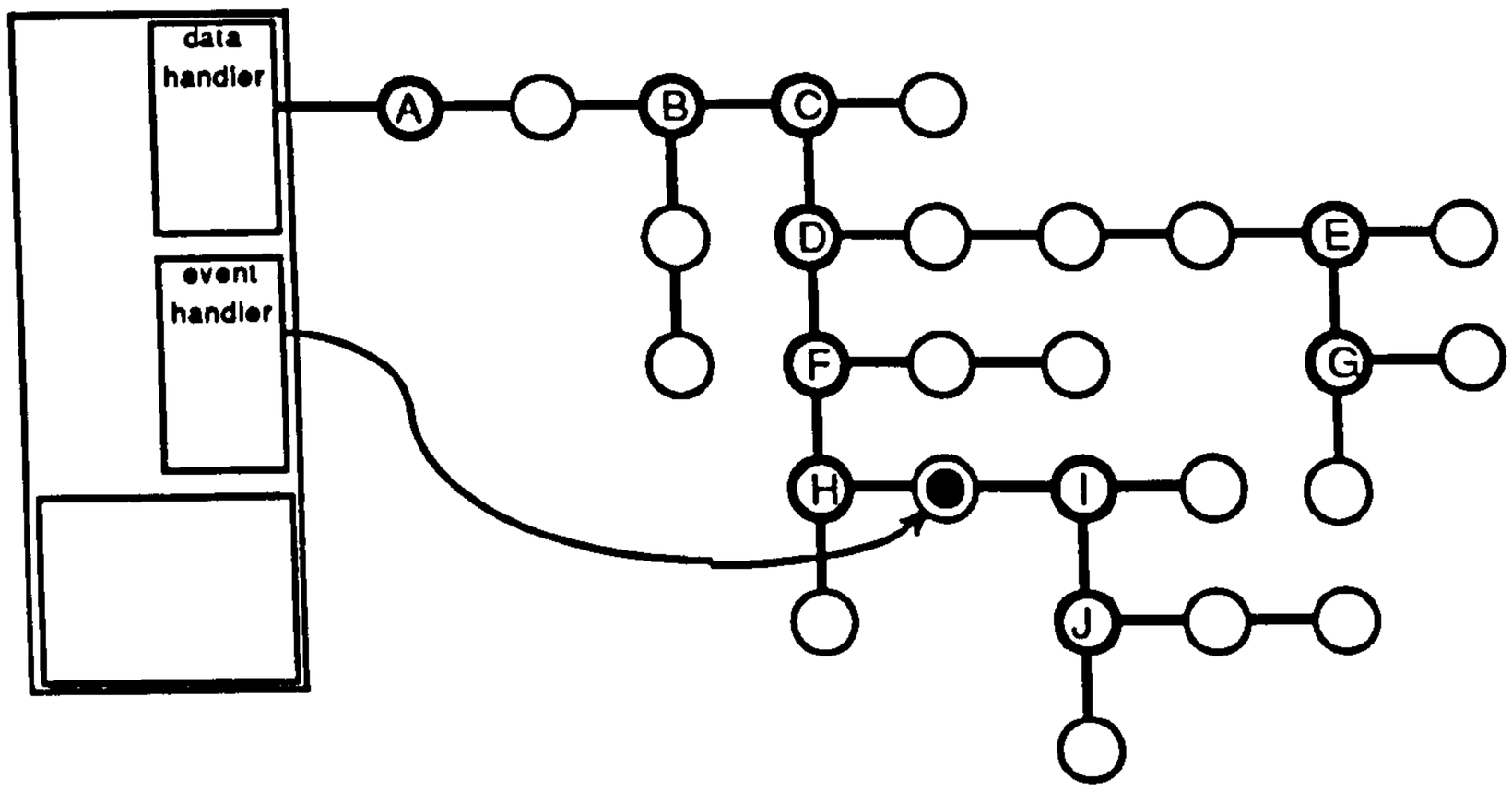


Figure 4.7.10.8.1. Directing input events to a field or form.

Any keyboard or mouse input is then directed to the currently active field, figure 4.7.10.8.1. The field may or may not respond to keyboard input depending upon the type of concept interpreter/interaction device is employed.

In order to select and activate a form or field for input a number of selection mechanisms have been developed driven by both keyboard and mouse. Note that a three buttoned mouse is assumed:

Directing input to a concept has presented a number of interesting problems relating to the general management and handling of hierarchies.

4.7.10.8.1. Selecting and de-selecting fields using the keyboard

There are a number of conceptual problems involved with hierarchical navigation using the keyboard; ascending, descending and traversing. The following keys and control characters are used to perform these special functions and never reach field interpreters:

left arrow	select next form/field
right arrow	select previous form/field
up arrow	select parent form
down arrow	select the first field of a form

The above characters are rather keyboard specific control characters. A mechanism is therefore provided to change these characters. Navigation characters may be re-defined in the forms resource file (section 4.9.2, customising the forms package).

4.7.10.8.2. De-selection using the keyboard - a dilemma

The de-selection of fields using the keyboard was considered at length and and still causes anguish; the final solution may not be perfect.

The main issue is which keyboard character should be used to de-select a field. On many systems (Macintosh application programs, for instance) the return key is used as this is a natural terminator for users familiar with the keyboard. However, this restricts keyboard entry to single lines of text and would be inappropriate if the user is asked supply a list of items or a description, where carriage return is used to delimitate the end of a line Special concessions could be made for multiple line input fields but this would introduce a basic inconsistency into the end-user interface. Another key is therefore required for de-selection. Keyboards on Sun workstations have an additional new-line key which is used for this purpose. De-selection and new-line keys may be re-defined in the forms resource file.

A decision was taken to use the return key for multi-line input and the new-line key for de-selection. This may be considered as the wrong choice. However, it was considered that most users of computers are familiar with a typewriter analogy and therefore would expect the return key to move the character cursor to a new clear line. The concept of field selection has no natural keyboard extension and therefore it would not matter which key was used.

Owing to the graphical nature of the forms package a user's first encounter with field selection is by mouse. It is often the case that this remains the sole means of field selection and form navigation for the user.

Although it is not the aim of this research to investigate interface ergonomics a simple experiment could be employed to find empirical evidence which would indicate the effectiveness of various keys for de-selection. The first criteria of such an experiment would be that users are unaware that such a feature was under investigation. The experiment would rely of the forms package writing an entry into a pre-defined file each time a user selected and de-selected a field during an interaction session with a range of applications. The context of the interaction session, the user's experience and knowledge of the forms package together with concept names and the number of lines entered must be taken into account and therefore an entry should contain the following information:

	concept	field type	N°.lines	selected by	de-selected by	last key before de-selection	elapsed time (s)
a)	quit	button	NA	mouse	mouse	NA	25.2
b)	name	alpha	1	mouse	nl	return	
c)	vertex [1]	real	1	left arrow	left arrow	0	

The nature of the requested data input is important. If a field requires keyboard entry the likelihood of another text field being selected by the keyboard is much higher than it would be for a graphical field.

The interpretation of the data is not straight forward as many aspects are involved in a users choice of field selection, for instance:

- The first field is most likely to be selected by the mouse
- Fields such as buttons containing images are more likely to be chosen using the mouse.
- Some users may not be aware of the different navigation mechanisms
- Some users may not like using the keyboard or mouse.

Many factors have to be considered and the problem is left for others to figure out; Part II "The User and the Usage of Interactive Computer Systems" [BAECKER 87] and [POLSON] provide an insight into some of the physical and psychological guiding factors involved in designing user interfaces.

4.7.10.8.3. Selecting a field using the mouse

Form and field selection using the mouse is much more straight forward. A special search algorithm was developed to optimism selection response time by anticipating the user's next selection. The algorithm works on the assumption that the user's next selection will be within the same context as the current one; i.e. the anticipated field will be one on the current form which is searched first. The algorithm is essentially a recursive tree walking function which passes a message to each node in turn. In the case of field selection the message passed contains the cursor coordinates at the instant the event occurred. Each form/field evaluates the information by performing a simple *boxinside* test. If the test is evaluated as true (the cursor position is inside the bounding rectangle) the message is passed down until the target field is found and then processed, otherwise it is recursively passed up to parent of the current form repeating the test procedure.

Depending upon the result of the boxinside test a node will activate or de-activate itself for input with one of the following methods being fired to update the display:

true	node->display_select	paste border and refresh field contents
false	node->display_deselect	restore original border.

The selection process is best illustrated diagrammatically. Figures 4.7.10.8.3 1.to 4.7.10.8.3 7, illustrate an number selection possibilities together with the corresponding focus transition.

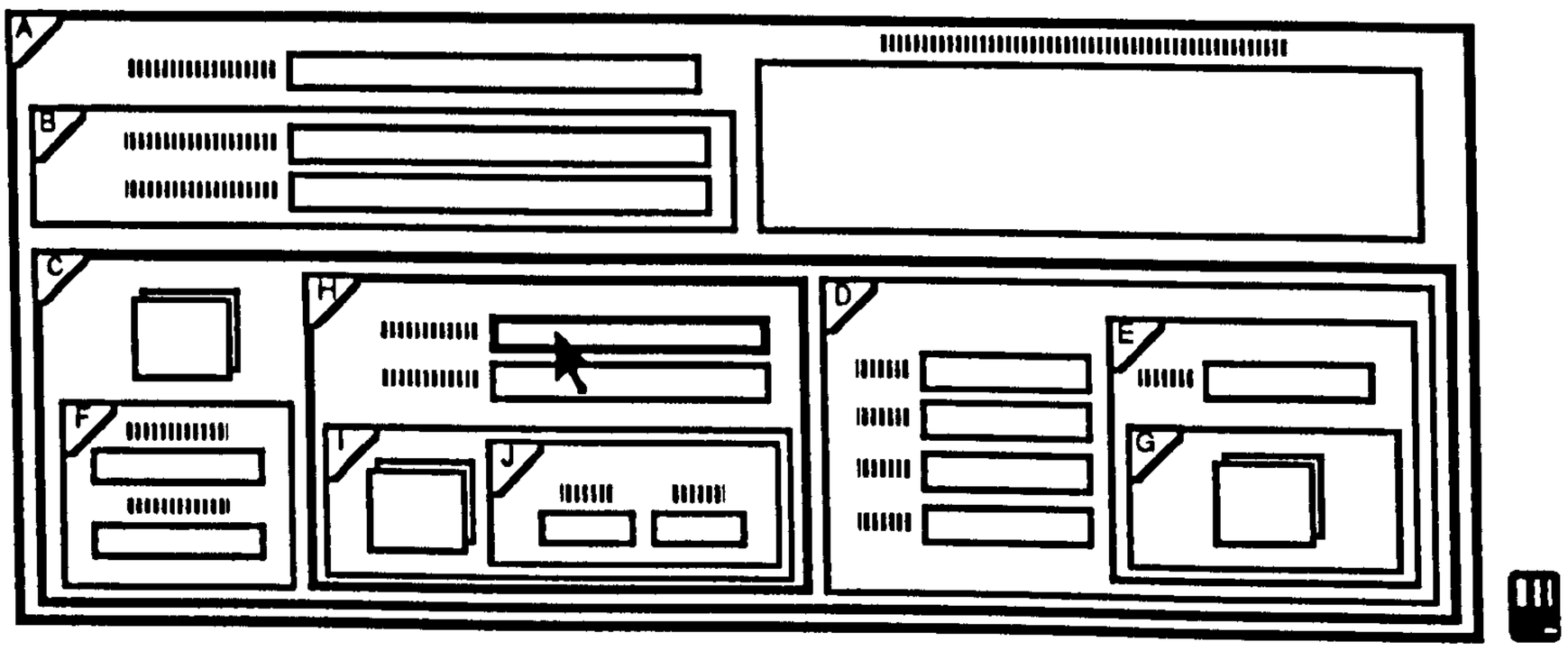


Figure 4.7.10.8.3 1. Current focus H1

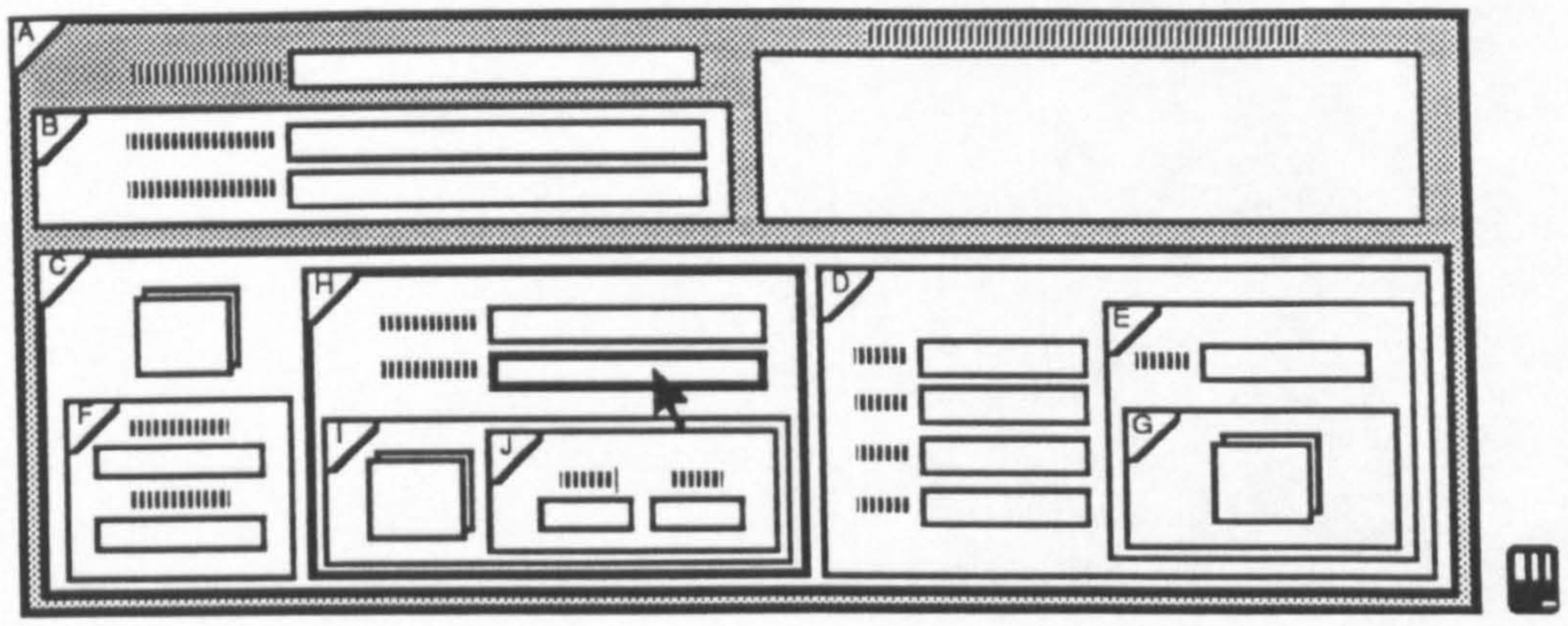


Figure 4.7.10.8.3 2. Target Focus:H2

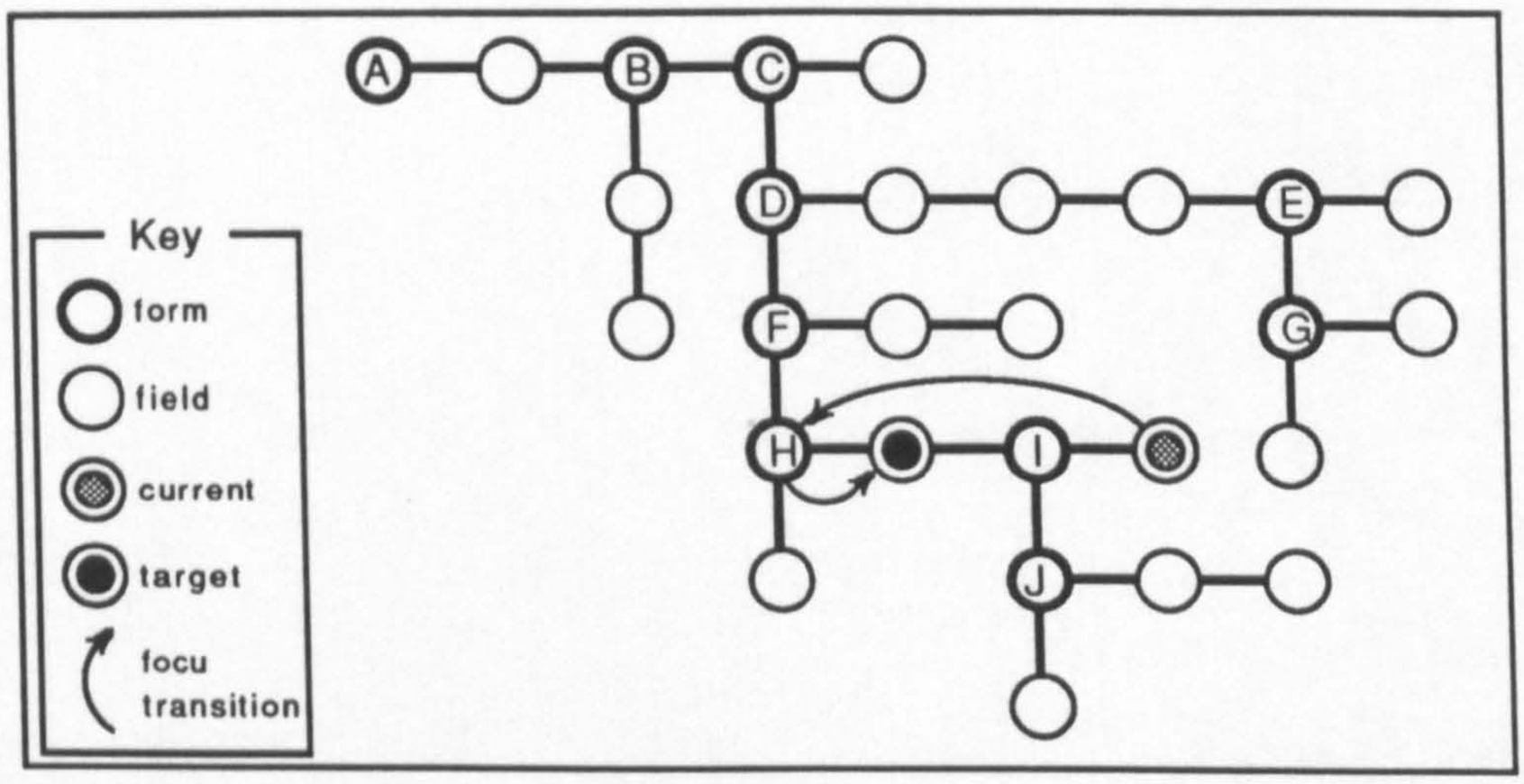


Figure 4.7.10.8.3 3. Transition between current focus H1 and target focus H2.

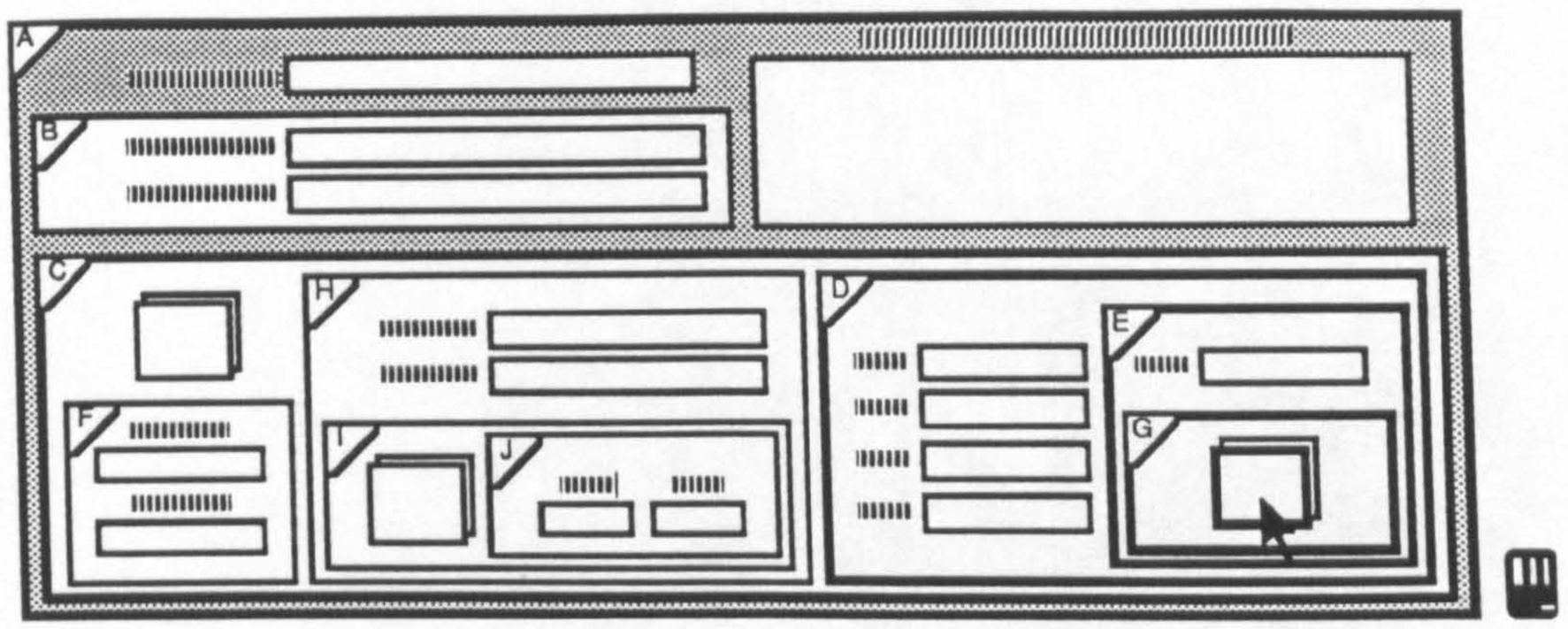


Figure 4.7.10.8.3 4. Focus G1

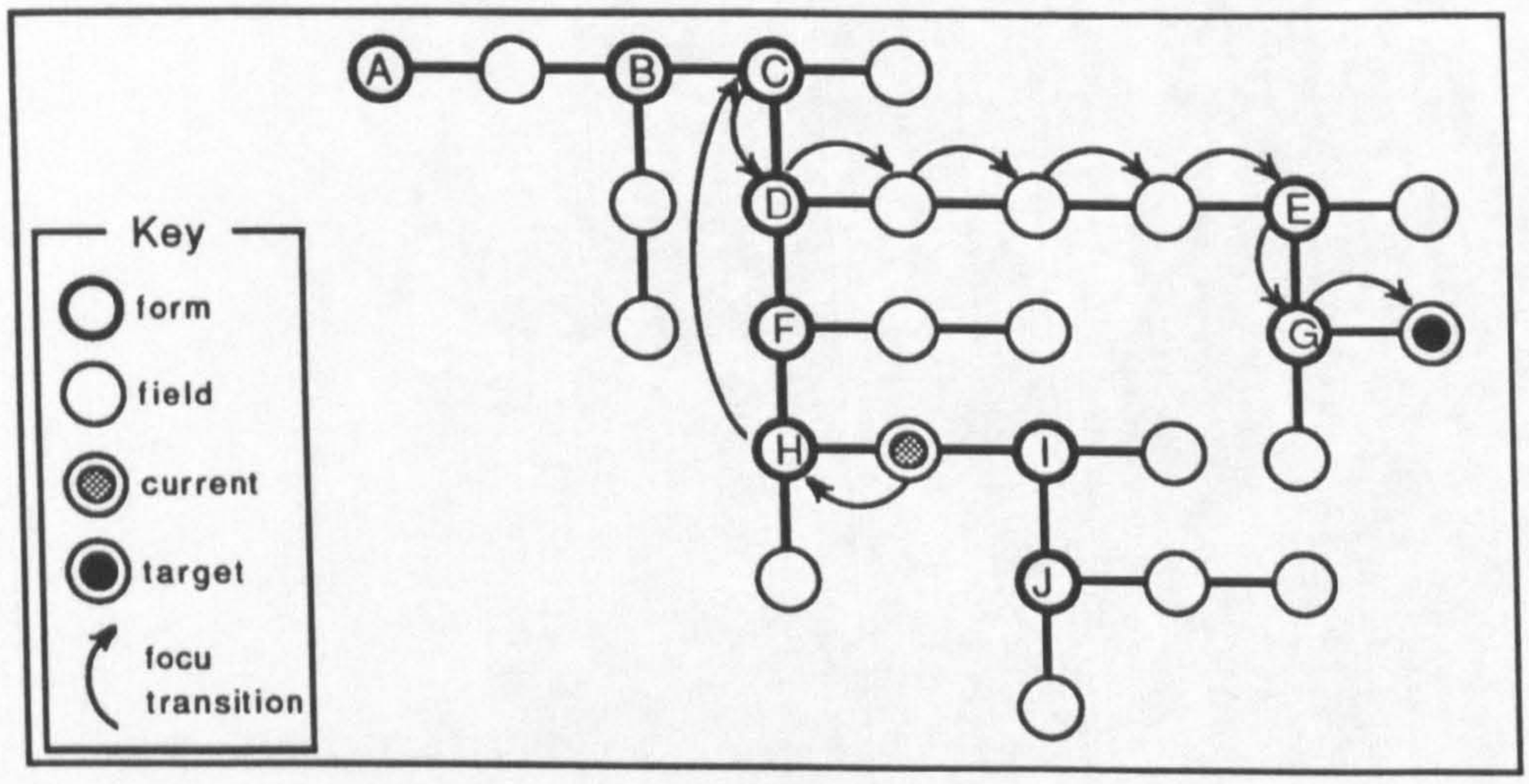


Figure 4.7.10.8.3 5. Transition between focus H2 and G1.

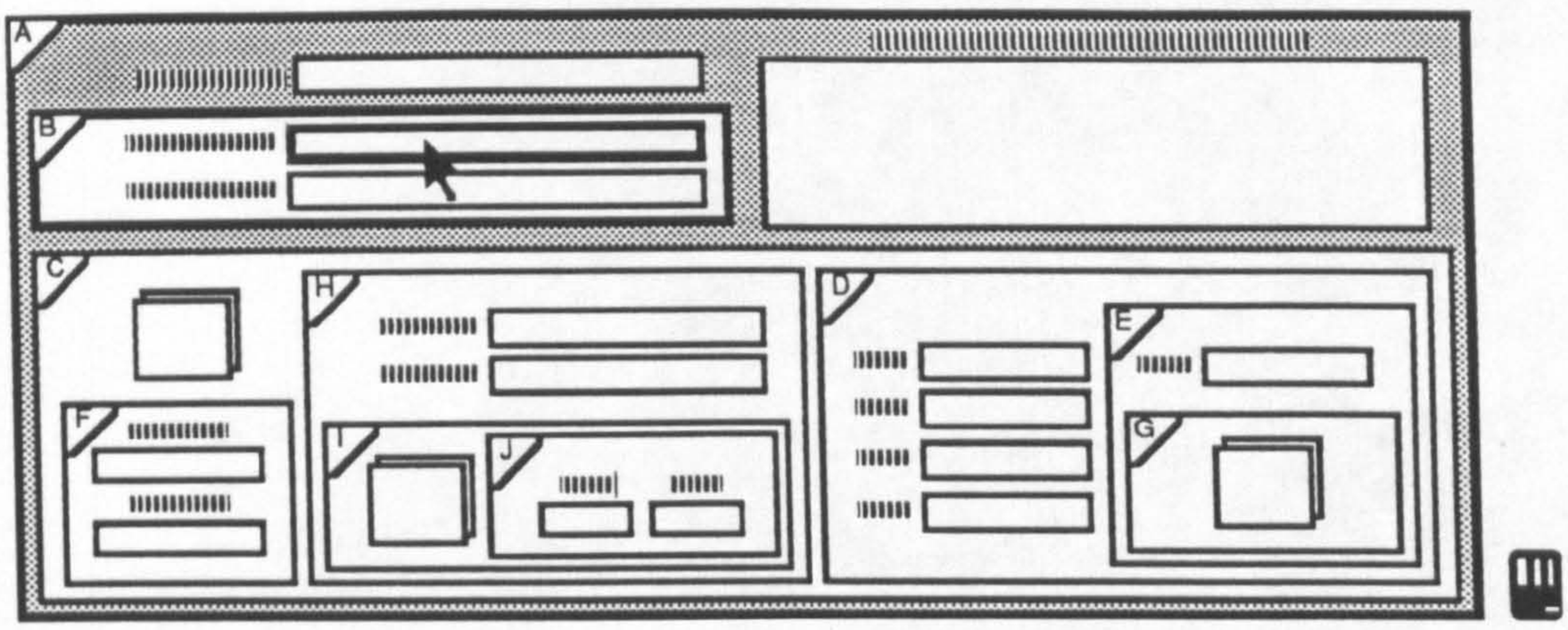


Figure 4.7.10.8.3 6. Focus B1.

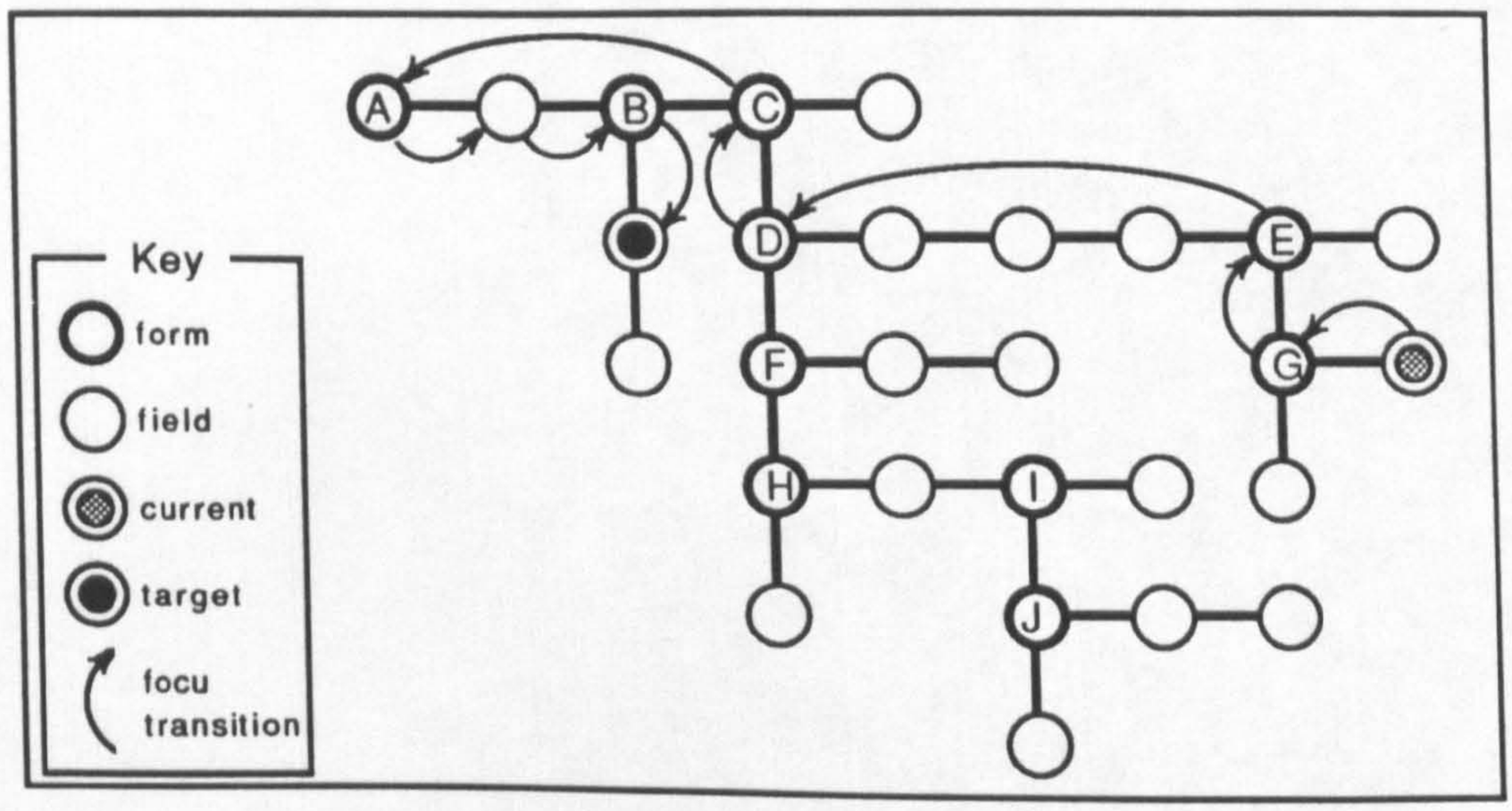


Figure 4.7.10.8.3 7. Transition between focus G1 and B1.

Note 'de-select' message issued ascending and, 'select' message issued descending the tree.

4.7.10.8.4. A brute force method - depth first (Linear) search

Another mouse driven selection algorithm was also tested. It operated by exhaustively running through a linear list of fields and performing a *boxinside* test for each field rectangle. The technique worked on the assumption (as a result of the order of field definition) that the last successfully tested field was the target field, figure 4.7.10.8.4.1.

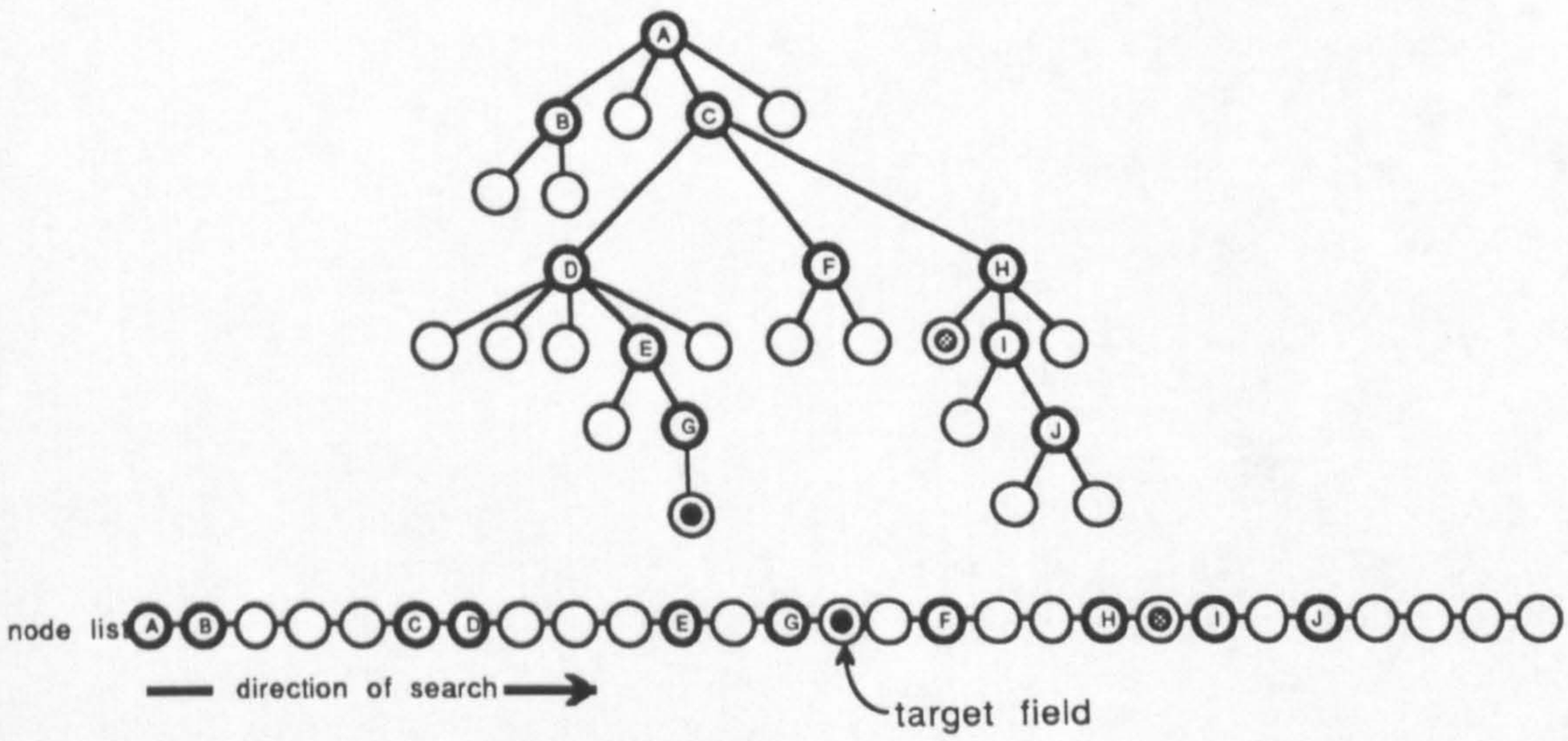


Figure 4.7.10.8.4.1. Linear node list. Note the field order, the last field on the list is placed on form A.

This worked well but, owing to the nature of the selection feedback mechanism (highlighting all levels above the selected field), rather than de-selecting all fields to the top level desk and then highlighting all fields down to the selected field; resulting in a great deal of flashing form borders, an additional step was added.

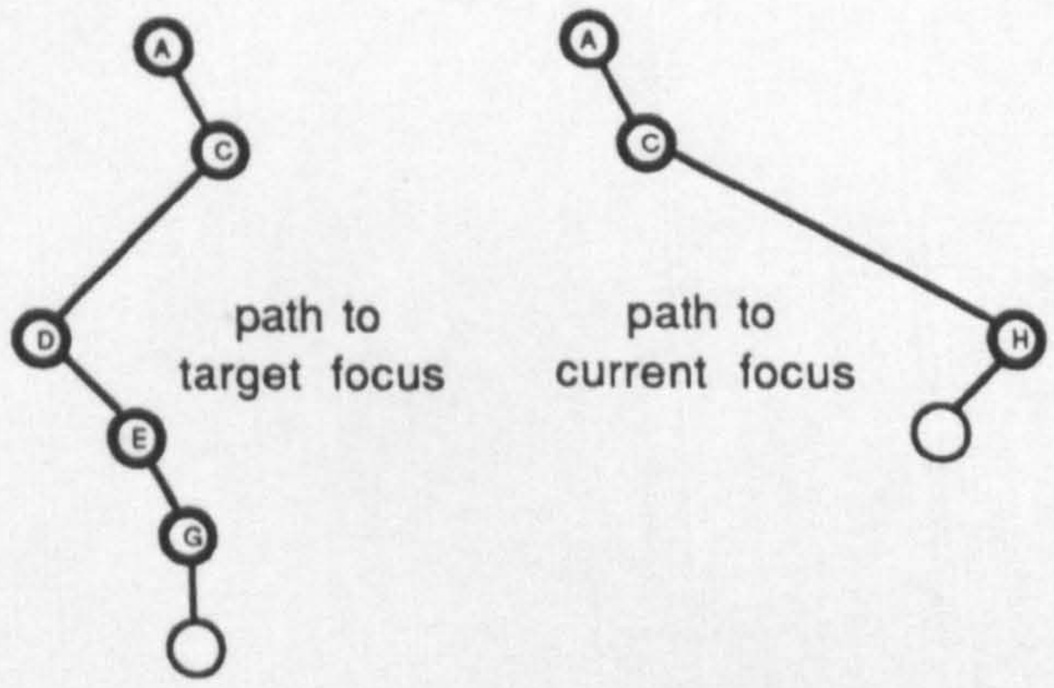


Figure 4.7.10.8.4.2. Absolute paths of current and target fields traced back to root (desktop).

This involved backtracking from current and target fields to the top of the field hierarchy, figure 4.7.10.8.4.2 and merging the paths; subtracting the common

elements up to the point of intersection, figure 4.7.10.8.4.3. De-selection and selection messages are then passed to each node.

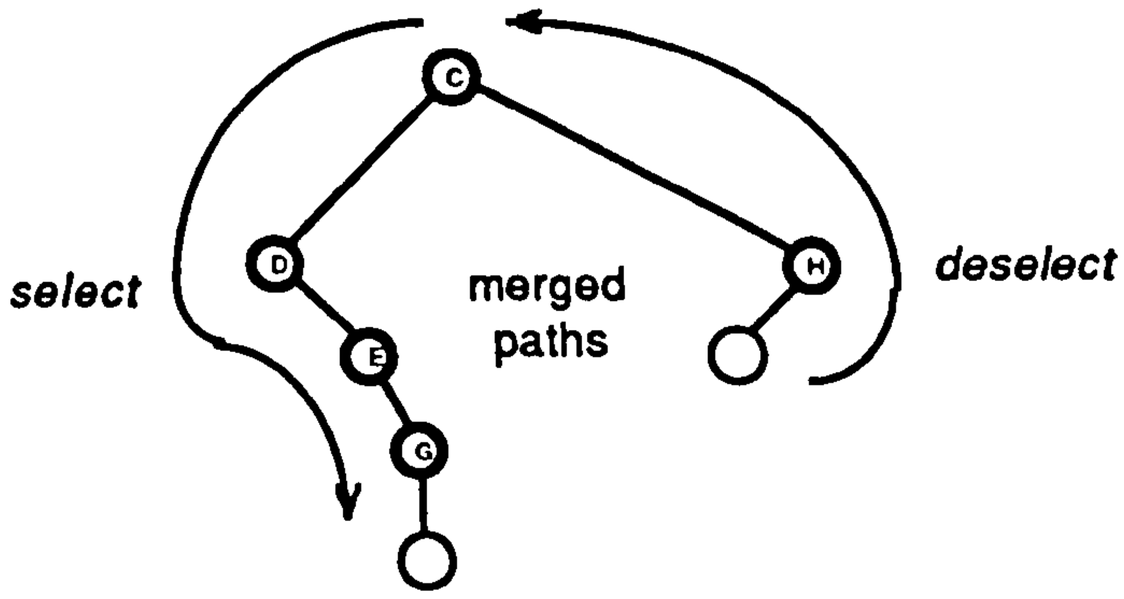


Figure 4.7.10.8.4.3. Common path between current and target fields. This path is used to de-select up to the intersection and select fields from the intersection to the target field without having to pass through the desktop.

Both methods work well, although the linear search method test all nodes to the left of the target field and therefore becomes less efficient the further right the target node is; there is no advantage gained by selecting fields on the same form. Fourteen searches are made for the focus transition between fields H1 and H2 with the linear search; only two using the recursive method. The relative recursive method performs the search and de-selection and selection functions in one step and is the one currently implemented.

4.7.10.8.5. Implied selection

When a user types data, in a situation when no there is no active field in a selected form, all of the user's efforts are lost. One solution, provided that the cursor is placed over a field, would be to automatically select and activate a field as soon as keyboard events are received.

This method has been implemented to an extent, but rather than any keyboard character selecting a field (causing complete havoc), the space bar activates a field for input and is interpreted as the user taking a deep breath before continuing with data input. This method of selection is only applied in situations where there is no currently active field.

4.8. APPLICATION INTERFACE

Most of the issues involved with user interaction have been discussed in the previous sections. The other main aspect of the forms package is the manipulation (by an external knowledge source) of the conceptual model.

A user interface management system should separate application code from user interface code; with each section being written separately [PRIME 88].

In contemporary UIMS the presentation, dialogue and application layers are eventually merged, usually as a result of compilation and linking. By virtue of a multi-tasking Unix environment, by employing processing forking, the forms package exists as a separate process passing and receiving messages to and from another concurrent application process through a single communications gateway, figure 4.8.1. Messages are interpreted by both the knowledge source and the forms package; each responding appropriately.

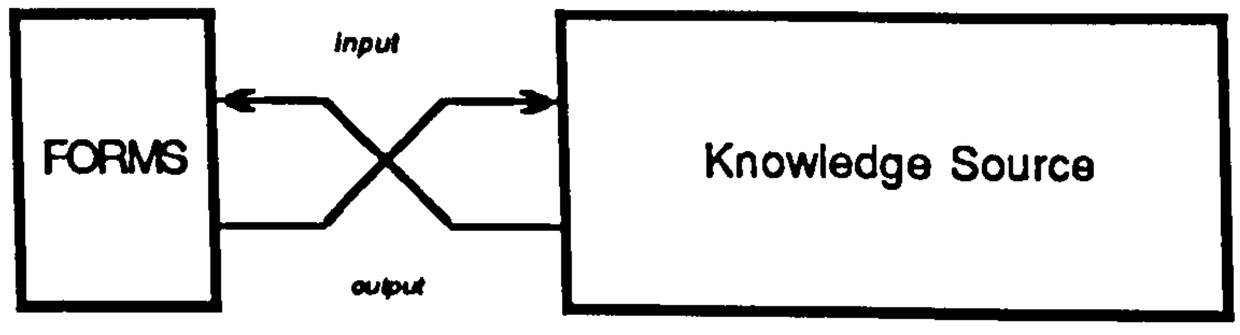


Figure 4.8.1. Connection between the forms package and an external knowledge source; achieved by connecting the outputs to the inputs of each process.

The actual communication between the application and forms packages is via a UNIX pipe.

Connecting the forms package to application programs requires a standard neutral language with which to communicate with application objects and operations and enable the application program to communicate to the user. In order to achieve a high degree of independence between the user-interface and the application (knowledge source) the language must enable a knowledge source to communicate to the user in abstract terms. Equally the language should enable the interface to be plugged onto many different types of application and must support two levels of operation by the provision of:

i) a high level language for communicating concepts to the user, and

ii) an interface specific language or manipulating the interface at a lower level of abstraction.

An interface has been created to handle the connection and event handling between the application module and forms package. These functions and macros

provide high level communication mechanisms and are fundamental to the perception of the forms package as a natural language interface. The routines (described in Appendix A together with a number of example applications) are held in the "inform" library which, as well as being integrated into the forms package, must be compiled into the application. The inform application interface consists of two parts:

i) an output (notification handler), and

ii) an input (control) handler.

4.8.1. NOTIFICATION

Natural language interfaces theoretically facilitate unrestricted dialogues. Unfortunately the practical overhead of resolving ambiguities increases message traffic and therefore reduces the efficiency interface. By restricting user input to a small but equally expressive number of constructs a natural language dialogue is achieved without any great computational overhead.

Messages (user events) from the forms package to the application layer are formatted into neutral language utterances using the protocol:

concept:utterance_type:value

Utterances have been categorised into four groups:

i) *user_set* the user has set the value of the concept

ii) *user_request* a request for help, description, or an example (issued from the concept menu).

iii) *user_action* describes what the user is doing; Ie moving the mouse over a field, or selecting a field

iv) *user_error* an error, with respect to interaction with a particular concept interpreter has occurred: eg invalid character typed, or invalid mouse event.

The most common utterance is the *user_set* event. When a field, in the forms package, is de-selected, if it's value has been modified by the user, the new, modified value is written to the standard output stream (stdout) in the following formatted utterance:

concept:USER_SET:value

Although the forms package has been specifically designed for the IFe, owing to the distribution of labour a slight incompatibility exists between the forms package and the IFe system. However, how the utterances from the forms package where formatted was not too important since the dialogue handler would perform the mapping between the high level neutral language of the IFe (see chapter 6) and the more specific control mechanisms of individual interfaces and interaction devices. Any discrepancies between the forms package and the IFe system are resolved by the dialogue handler which converts events from the forms package into a more natural, neutral language utterance of the form [MAC RANDAL]:

dialogue handler user_said concept value

Simply by modifying the output handler of the inform library, the forms package is capable of transmitting utterances of this form.

To facilitate a two way dialogue with the user a number of control mechanisms are required. These are managed by the input handler of the inform library.

4.8.2. CONTROL

The high level language is most important for ensuring complete separation and independence between the user interface and knowledge sources. The basic requirements of this high level language are to simply communicate to the user in abstract (non interface specific terms). The most obvious and fundamental operation required of the forms package was a facility for directing, or focusing and de-focusing the user's attention towards a particular concept. Also concept values (default and alternatives) must be set and retrieved and help messages passed to individual concepts.

The manipulation of fields and forms is controlled by a simple command syntax. The categorisation of commands is described below.

4.8.2.1. CONTROL METHODS

During the development of the command language interface it was necessary to identify and classify the types of operations that where likely to be used during the course of interaction with the user.

As a result of the object oriented approach, adopted for the development of the forms package, a specific protocol for passing messages has evolved in which a reference to an instance of the generic object class together with the required method are identified by name and passed an item of data. The protocol therefore takes the following form:

concept:method:data

A concept in this form is best visualised as a *software ic* in which the software *chip* represents an instance of a class of object and the *pins* are access points to methods. Messages are parsed by event handlers and the appropriate method is fired, figure 4.8.2.1.1.

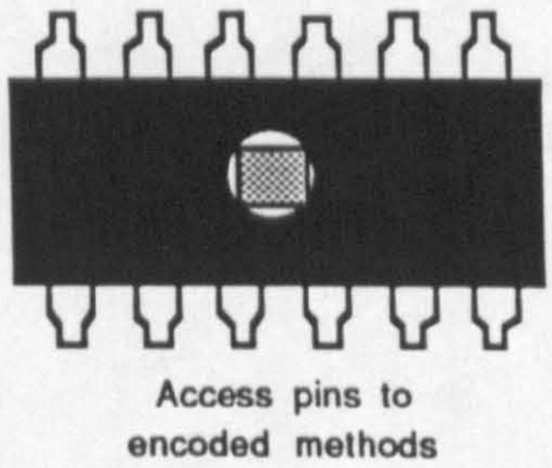


Figure 4.8.2.1.1. Software IC.

This view is extended further in chapter 7 into a dynamically configurable data cell which the user defines and attributes and is used in a dynamic 3D object manipulation and viewing program, Appendix E.

The range of commands has been kept to a minimum and consist of either attribution, manipulation or state query commands.

The following control mechanisms for the communication and manipulation of conceptual models together with methods for the attribution of fields have been identified. Where an operation has two potential states commands used to invoke the operation have been defined symmetrically (eg on/off).

mechanism	parameter	description
hide / display	<rate>	focus/de-focus the user's attention towards/away from a concept. These two commands may be given an integer parameter which is interpreted as a fade rate (0 - 100 steps from full image saturation to background, and vice versa). This is useful to prevent large areas of the form "flashing on and off" which may cause unnecessary discomfort (see 4.8.3).
set current	<value>	overwrite the contents of the field. An optional +, preceding the value, may be used to append to the current concept value.
set previous	<value>	This is useful when restoring a proforma from a previous dialogue session.
set default	<value>	offer the user a contextually relevant default value
on / off		enable/disable field editing by the user. This is particularly useful when a item of information supplied by the user is being processed by a knowledge source or if the information is a non-editable statement from the knowledge source. Allowing the user to change such a concept's value would result in inconsistency. The knowledge resource may change the value while the field is "off". This prevents the user tampering with data while it is being modified by the knowledge resource.
contents	<concept>	obtain the contents of a concept. If the named concept is a form, the contents of all fields contained upon it are reported. A -r flag may also be supplied in which case all concepts are queried recursively.
select		activate a concept for user input; ie force the user to address the concept.
highlight	<concept>	make a particular concept stand out; ie indicate a number of concepts which must be addressed (before aborting a session). This is achieved by hatching the background of the concept interpreter. Flashing or the use of colour would be more suitable but would require an interupt handler for each field in the main event loop.

In addition to these control methods, all field attributes (described in the proforma interface) are accepted so that the physical characteristics of specific fields may be modified dynamically if necessary. This lower level command language also includes the following mechanisms:

mechanism	parameter	description
load	<proforma>	load a template or control file, from the current focus
store	<proforma>	saves the current template from the current focus to a file. This is useful for saving the current session or when designing a proforma interactively.

A number of response mechanisms have been provided for user_request and user_error events; these are:

mechanism	parameter	description
chat_user	<text>	Display the text message in the pop-up help window next to the named concept.
ignore help		do not issue help requests, use proforma definitions
error cleared		only one error is issued at a time to prevent overloading or flooding the application with errors such as invalid characters being entered into a concept interpreter.
ignore errors		do not report errors
ignore events		do not report user events

A number of macros have been defined around these methods resulting in a high level language based on the neutral language defined MacRandal (see chapter 5 & Appendix A).

4.8.3. SOFT FOCUS

Focusing and de-focusing (concepts and meta-concepts) is used extensively throughout the IFe. In order to minimise the visual impact caused by large areas of the screen "flashing on and off" a soft focus mechanism has been provided. Simply by passing an integer value (between 0 and 100) as an argument to the hide and display commands, forms may be melted into the background of their parent . This is achieved by an iterative series of raster operations, illustrated in figure 4.8.3.1.

The initial display image is cleared by COPYing over the background image. For each iteration a mask (a bitmap containing a greyscale pattern) is generated and COPied onto a temporary working bitmap. The initial foreground image is then ORed on top of the temporary bitmap and the greyscale pattern XORed out with the mask. The resulting bitmaped is finally ORed over the display image. The process is repeated until the final temporary bitmap image is clear. A complete cycle takes the display image, from full image saturation, to the background image.

The reverse effect (a crystalising image) may be obtained, simply by decrementing the greyscale intensity at each iteration.

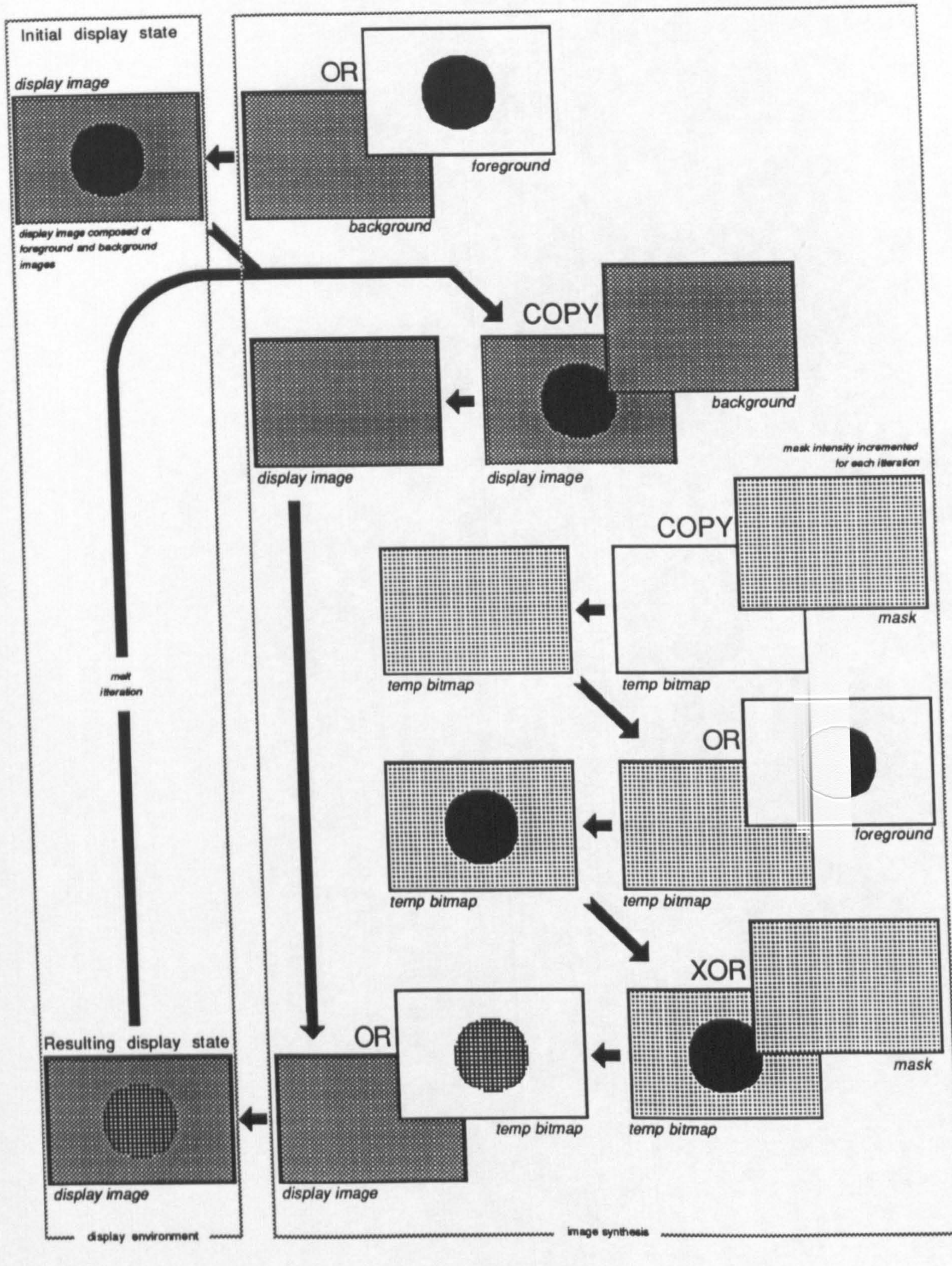


Figure 4.8.3.1. Bitmap operations for soft de-focus. For soft focus the mask intensity is decremented at each iteration.

The overall effect is best demonstrated interactively but is illustrated below as a series of frames, figure 4.8.3.2.

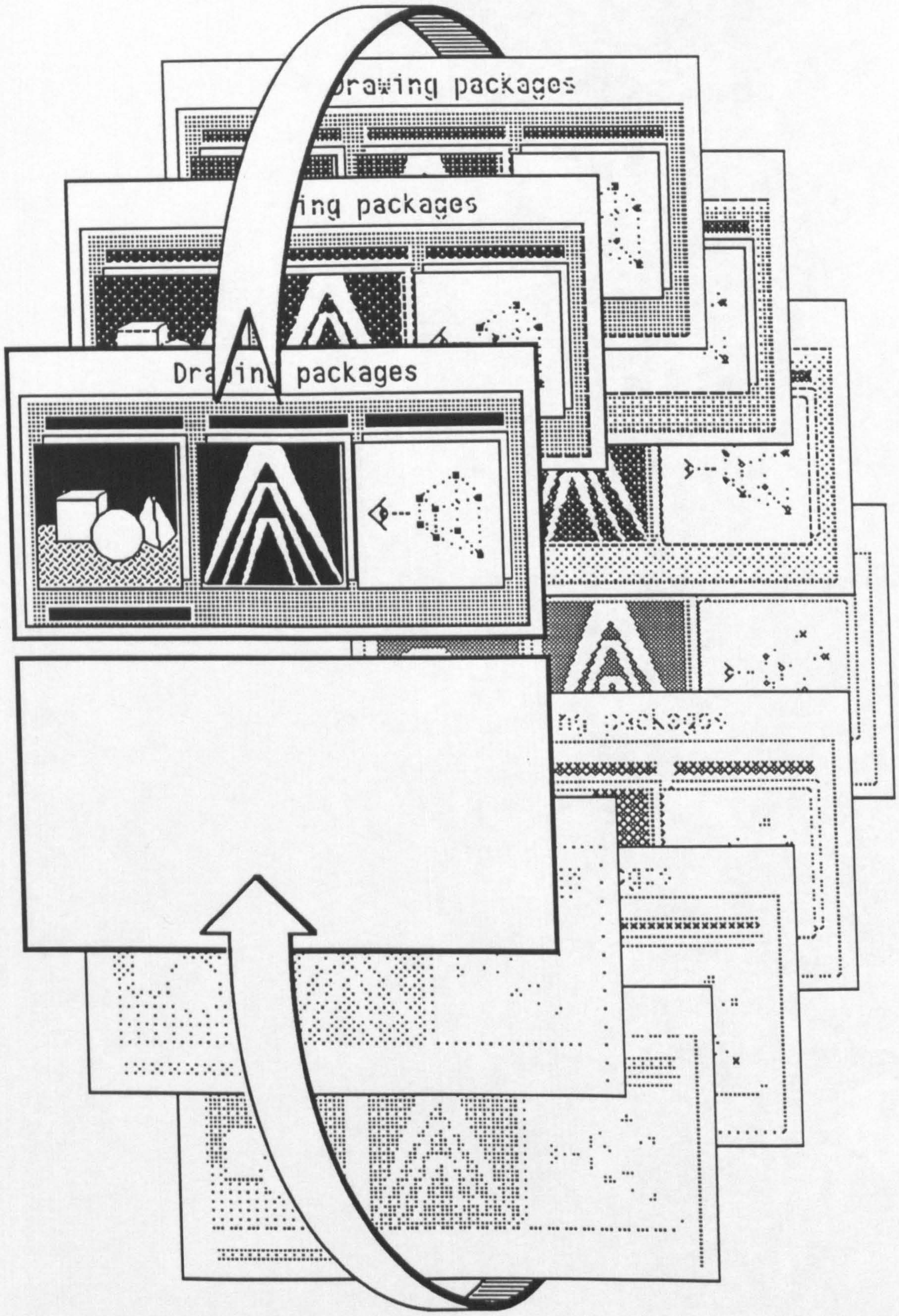


Figure 4.8.3.2. Soft focus and defocus.

4.8.4. A COMMUNICATIONS PROTOCOL FOR UNRESTRICTED DISCOURSE

In order to pass messages between each of the modules in particular to the user a special communications protocol was devised. The basic principle is that described for person-person communication, section 3.2.

The communications protocol used by the system is equally important. It's structure represents events from the user.

<address or name of primitive> <method> <data>

The address is used to identify the "primitive" from which the data originates by the application of a particular method. For example the forms package transmits messages of the form:

user_name	user_set	fred
date_started	user_query	help
geometry	user_action	dc-focused

The modules developed within the IFe system all respond to the same protocol. The implication of this is that just as the event is transmitted from a module it can also respond to the same event. It is merely a question of who is sending the event.

The primitives (described in chapter 4) employed by the system are therefore important in that they dictate the type of dialogue that is possible. The primitive element chosen is one identified as being common to all design domains and fundamental to human communication. The primitive with respect to the overall system will be referred to as a concept although for each of the modules a different viewpoint may be used (ie field in the forms package). This therefore results in a totally generic communications protocol.

4.8.4.1. ADDRESSING CONCEPTS

As already emphasized, concepts by their definition are hierarchical and this has lead to a particular form of addressing. The protocol for addressing a concept within the generic interface architecture has been adopted from the UNIX directory structure as this is a convenient means of navigation; taking the form:

/meta-concept/concept

which enables absolute reference to concepts, as illustrated in figure 4.8.4.1.1, and is therefore referred to as *absolute addressing*.

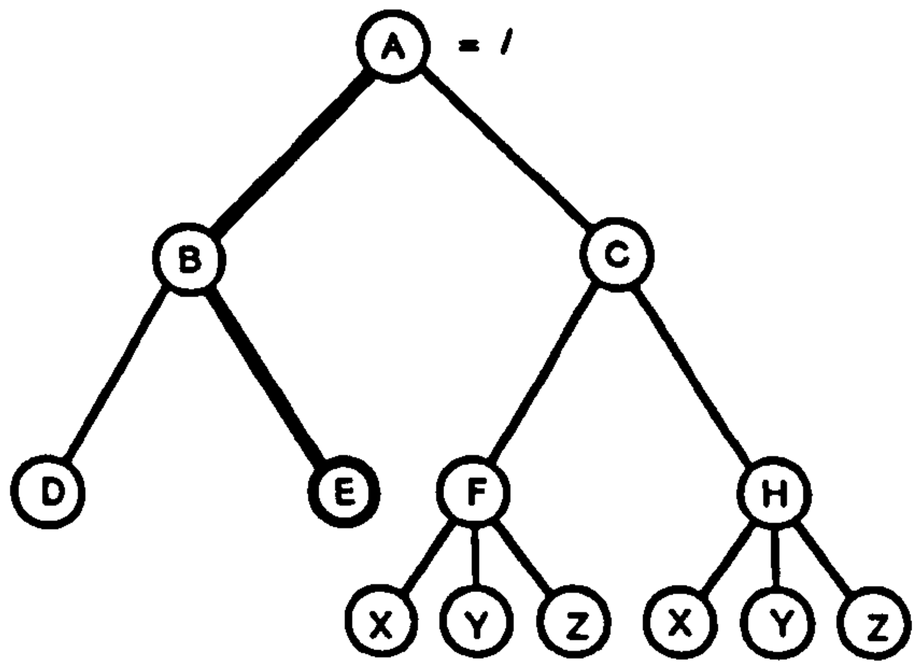


Figure 4.8.4.1.1. Absolute addressing /A/B/E

In normal conversation this form of addressing concepts (or focusing) would be an abnormal means of communication. It would be strange to constantly refer to an entity by its class, sub-class, and instance name. Usually, once the context of discussion has been established (class, sub-class) only the instance name is used. For this reason an additional means of addressing concepts is provided:

@concept

As this is a means of identification by means of a single symbol, rather than an absolute address this is referred to as *symbolic addressing*.

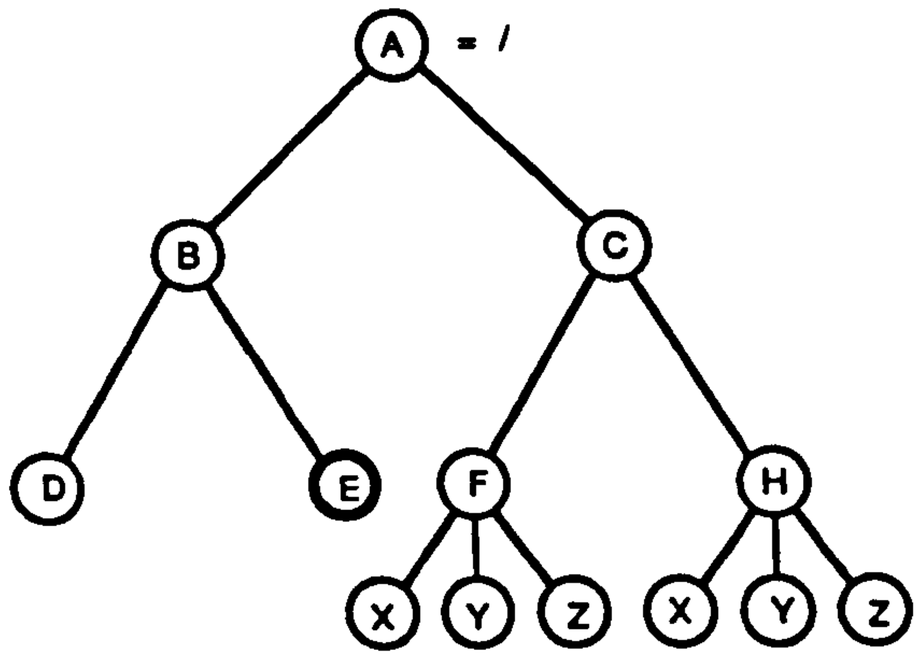


Figure 4.8.4.1.2. Symbolic addressing @E

Symbolic addressing will address the first occurrence of 'concept' and therefore to pin point a particular concept with absolute certainty all concept names must be unique. In many situations this is impractical, for instance the concept of geometry involves the reference of many vertices. Ensuring that every body had a different vertex reference id, for example, would introduce unnecessary complications and management problems.

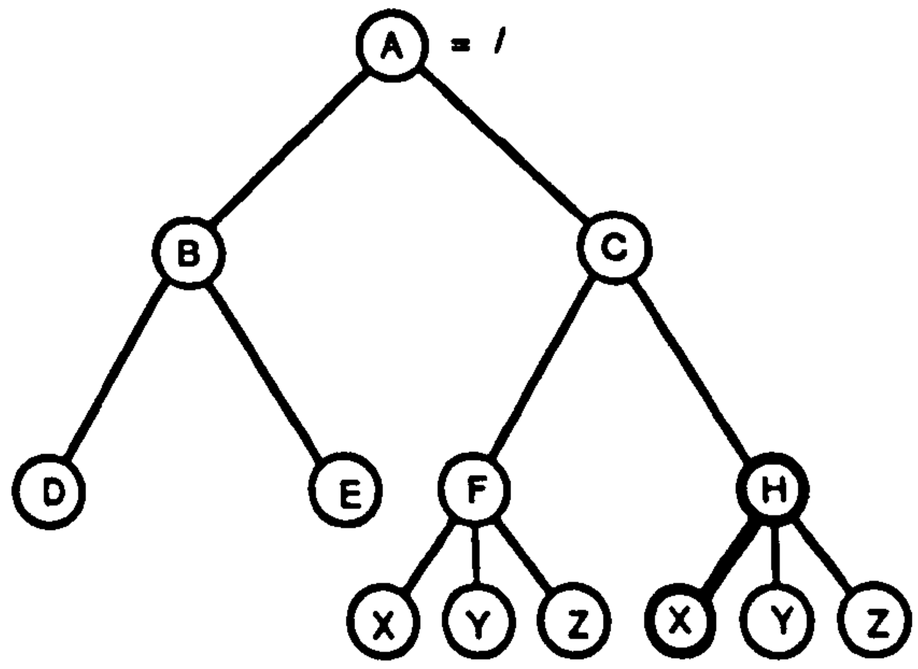


Figure 4.8.4.1.3. Absolute symbolic addressing @H/X

Therefore in order to accommodate multiple occurrences of concept names a combination of absolute and symbolic addressing are used to narrow down the search resulting in a combined *absolute symbolic addressing*, figure 4.8.4.1.3:

@meta-concept/concept.

Once a concept has been located it becomes the current application focus. An additional means of navigation is included (again based on the unix directory navigation mechanism) enabling concepts to be addressed relative to the current focus. This is *relative addressing*, figure 4.8.4.1.4:

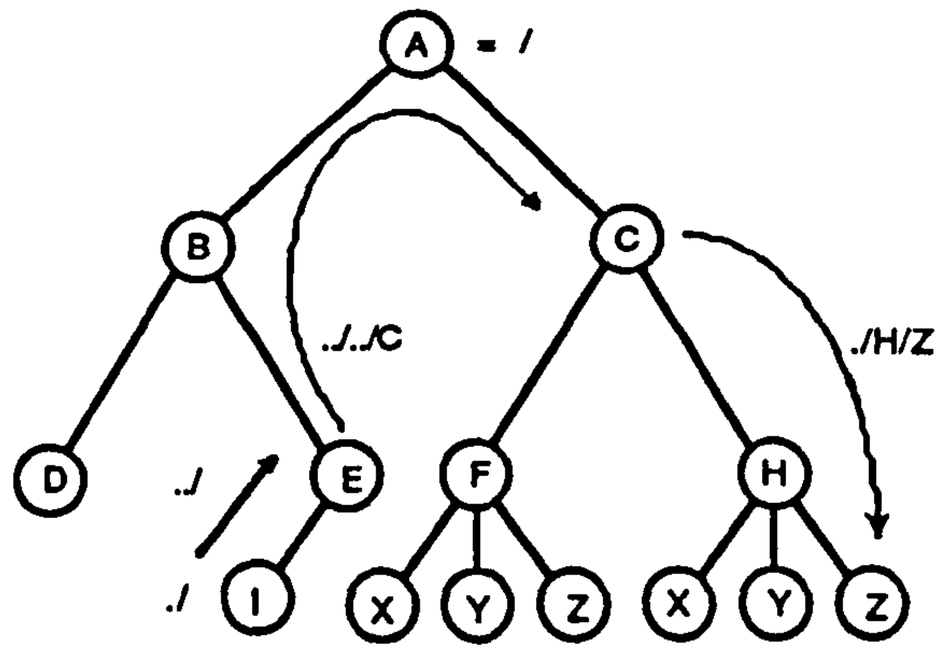


Figure 4.8.4.1.4. Relative addressing; / represents the current focus, ../ parent concept, ../../ two levels up, ../../meta-concept/concept, ./meta-concept/concept descend into a meta concept.

Relative addressing is particularly convenient in situations when the current focus is no longer at issue. The focus may therefore switch safely to the current context of the current focus as the conversation is most likely to continue along the same lines.

The choice of addressing has important implications for the way in which the corresponding user model and the knowledge bases are structured. Employing abstract "relative" addressing suggests that the structure in both the knowledge base, user model and interface need not be the same or follow the same principles. For instance the user's conceptual model (represented in the user interface) may be strictly hierarchical to aid navigation, while the knowledge base may simply contain discrete packets of knowledge and transition networks employed to navigate between concepts. Symbolic, absolute symbolic and relative forms of addressing are therefore employed throughout the IFe with the following advantages:

- Complete independence between user interface and domain knowledge sources; no knowledge, other than conceptual vocabulary, of other knowledge sources is required resulting in a high degree of modularity and therefore extendibility.
- appropriate forms of knowledge representation for particular tasks may be employed without affecting the overall system.
- opportunistic knowledge sources and interfaces; the user dialogue may be more fluid and natural (less procedural). The user may randomly address any domain of discourse.
- improved knowledge engineering; knowledge sources (in particular knowledge bases) may be structured as collections of logically related

packets of information, which may be added, modified, and removed with ease leading to re-usable knowledge which may be applied to numerous other domains of discourse.

Although any combination of the above methods for addressing concepts may be adopted for sending messages, when utterances are formatted and passed out of the forms package an absolute address is used thus ensuring absolute precision. It is the responsibility of the parent process to extract the concept name. The following messages illustrate a typical interaction session:

user	/building/geometry/zone_name:user_set:kitchen
	/building/bld_focus/materials:user_set:on
application	@geometry:hide:10
	../:load:materials
	./materials:display:10

4.9. PROFORMA INTERFACE

Forms (meta-concepts) are defined by a template file (called a proforma) containing the definition of field primitives (concepts) together with the physical attributes and properties of their relative concept interpreters using a simple declarative syntax; describing each physical attribute.

It is inappropriate to deal explicitly with abstract notions of concepts and mix physical property declarations. Therefore the analogy of form filling is carried through to the declaration and creation of form sets and fields.

One of the main objectives of the forms packages is that it should facilitate different interpretations of the same concepts using interchangeable concept interpreters. In order to achieve this dynamic interchange and provide a clean interface between the forms package and a knowledge source it is necessary that concepts and met-concepts are manipulated in a consistent manner. This in turn implies that all concept interpreters should respond to the same control mechanisms. This includes the attribution of their physical (graphical) characteristics. Therefore a generic set of attributes and control instructions has been identified.

4.9.1. DECLARATION OF A CONCEPT - PROFORMA FILES

The proforma contains the physical attributes of forms and fields. Each attribute together with its associated value must be on a new line. The proforma specified as an argument to forms must begin with the description of a desk (window). The interface uses a simple syntactic analyser comparing the first character string with an internal list.

For optimum performance the following protocol should be observed:

<attribute name>\t\t\t<value>

where the attribute and its value are separated by any number of tabs `\t' placed hard against the end of the attribute name. Without a tab the Forms package attempts to analyse the current line by searching through it's internal attribute list counting the number of words in each, extracting the same number of words from the current line and then comparing the two. This procedure obviously takes longer to initialize the proforma and therefore the user is encouraged to optomise the template. Lines failing the first analysis are reported as optimization errors in the console window together with the current profroma name and line number.

In order to achieve a high level of generality, field definition has been restricted to a small set of generic attributes. As forms and fields are all rectangular, the physical characteristics of both of these entities may be described using the following generic set of attributes:

attribute	value options	description
new/end	desk form field	create/complete a new desk or form (meta-concept) create/complete a new field (concept).
name	any unique character string	assigns the name of a domain concept to a form or field
type	the name of a concept interpreter (see concept interpreters for list).	specifies the manner in which the concept value is presented to the user and the degree of freedom the user has in manipulating it.
start	visible hidden	defines the initial (start-up) state of the from or field (whether the concept is visible or hidden).
origin	x y	defines the position of the fields concept interpreter in relation to the parent form (meta-concept) origin
size	l h	the size of the concept interpreter display area.
label string	any unique character string	defines the name of the concept the user deals with

All fields and forms are defined in this manner. Any additional attribute that is added to this set must be accommodated by all concept interpreters. The above table represents the basic attributes necessary for the definition of a conceptual model. A complete set of attributes together with valid options is provided section 4.9.3.2.2 which describes complete form specification.

It is important to note that the type attribute only applies to field definitions; defining by name, what concept interpreter to employ.

The range of the currently available interpreters is categorised into two groups, concept and meta-concept. Interpreters from the same group may be interchanged with each other but not with interpreters from another group.

category	interpreter	description
meta-concept	desk	Graphics window
	form	scrollable region on which fields areplaced
concept	character	accepts all ASCII characters
	alpha	characters a-Z only
	alpha numenc	a-Z and 0-9 only
	integer	0-9 only
	real	o-9 and a single`.'
	button	toggle
	popup	cascading image stack
	file	unix file browsing field
	date	pop-up calander
	graphics	graphics display and interaction field
	slider	not implemented
	viewer	displays a single viewer picture file

4.9.1.1. BUILDING A CONCEPTUAL MODEL

A hierarchical structure may be defined in two ways; by:

1) defining instances of objects and linking them together with a template
2) defining both object instances and the relationship between them in one stage.

The first instance is useful when libraries of objects have been defined. The template merely points to a reference of the library object. Objects within the same template may be interchanged. Alternative templates may also be interchanged.

The second method is useful when only one solution is being defined. This method is employed in the definition of proforma templates.

As fields exist on forms, a form must firstly be defined followed by field definitions. Thus:

```
new form
  name   user_details
  origin  0 0
 size       50 10
 end form
  new field
          name    user_name
          type
          origin
          size
          label string      name
          label position   left
 end field
```

would result in the form "user_details" on which the field "user_name" is placed at 3 characters from the top and 15 character units from the left edge of the form.

Fields and forms are defined relative to the previously defined form (referred to as the current parent form). This enables forms to be nested. In order to change the current parent form (provided that it has already been defined) the following attribute must be specified:

```
parent form        @form name
```

The top level form or desk is a window and has a special name "/" (meaning root). Both "/" and the root's user defined name may be used. "/" is particularly useful when fields and forms must be positioned on the top level desk:

```
parent form        /form/sub form
```

Another facility is the *include file* command. This will load the named proforma template and provided that it does not contain a *parent form* declaration the contents of the template to be included will be placed on the current parent form. This enables forms to be defined as modules and therefore reused for other applications.

4.9.1.2. DEFAULTING

In order to minimise the definition of fields or forms and to provide a consistent appearance all field and form attributes are defaulted at run time.

Every field attribute is defaulted at three different levels:

- hardwired defaults - internal
- system defaults - /usr/lib/formsrc
- user defaults - $HOME/.formsrc

4.9.2. CUSTOMISING THE FORMS PACKAGE

The Forms package, when initialised, attempts to open a resource file in /usr/lib and the users home directory.

The resource file contains a definition of a form and field which are used to default fields and forms when created. For example the following entry in a .formsrc file would default all field types to 3 by 1 integer fields, set the selection borders for all fields to 2 pixels and define the background shade of all forms created to light grey.

.formsrc:

```
form
   name        default form
   origin      0 0
   size        20 20
   shade       light grey
end

field
    name       default field
    type       integer
    origin     1 1
    size       3 1
end
```

Apart from the following attributes all defaulted attributes are overwritten by profroma entries.

- shades
- selection borders
- fonts

4.9.2.1. DEFINING MENU ITEMS

The items generating user requests on the concept menu may be re-defined by the following procedure:

```
concept menu   help\
               description\
               example\
               why
```

It is important to note that when the concept is being re-defined the default help, description, and example entries are overwritten and if required must be re-defined.

4.9.2.2. INHERITANCE

Any unspecified field attributes will be inherited from the parent form. Fields and forms inherit the following attributes from their parent desk or form:

- all fonts
- label position
- foreground and background colours
- shade
- border and selection border

As a result forms and fields may be defined with minimum number of attributes, indicated below:

Minimum requirement for a desk:

```
new desk
        name        desk top
        origin      20.4  10.5
        size        150 20
end desk
```

Minimum required for a form:

```
new form
        name        form a
        origin      0 0
        size        50 5
end form
```

Minimum required for a field:

```
new field
        name            field a
        type            integer
        origin          20.4 10.5
        size            2 3
        label string    field label a
end field
```

4.9.2.3. GENERIC ATTRIBUTES

The following list indicates the current range of field attributes. Specific attribute values and defaults are given where appropriate.

Note: Attributes are shown in bold type. Alternative attribute values are shown in italic and defaults are indicated in bold italic.

The number and position of spaces must be observed closely, although the proforma interface will attempt to handle multiple spaces with a substantial speed penalty. This is reported as an optimization error in the console.

Table 4.9.3.2.1. Field creation attributes

new/end desk	create/complete a new desk
new/end form	create/complete a new form
new/end field	create/complete a new field

Table 4.9.3.2.2. Field attributes

name	<unique character string>	
type	<type>	field data type
	alpha num	alpha numeric field
	character	characters [a-z A-Z] only
	integer	[0-9]* only
	real	[0-9]*. only
	date	input by popup calendar
	clock	not available
	popup	cascading image stack
	button	displays icons and text, images specified in menu
	file	file system search field
	graphics	graphics field
	label	non-editable text field
	slider	dangerous - core dump!
	menu	dynamic menu
	viewer	displays single viewer picture file

start	<state>	field state at start up:
	visible	default
	hidden	field starts invisible
	open	default
	closed	iconized (not supported)
origin	<X Y *pixels*>	Field origin relative to parent in parent character units or optional pixels.
size	<L H pixels>	Field size in character units or optional pixels; eg: size 64 64 pixels
shade	<shade>	background shade for forms:
	white	
	light grey	
	mid grey	
	dark grey	
	black	
	left hatching	
	right hatching	
	intensity <int>	0% 100%
current value	<text>	set the current contents of the field. The data format must match that of the current interpreter
default value	<text>	set the default value on the concept menu
previous value	<text>	although not immediately obvious the ability to define the previous value is useful when when restoring a form set from a previous interaction session.
background colour	<colour>	background colour for desk, forms and fields.
	black	
	red	
	green	
	blue	
	yellow	
	cyan	
	magenta	
	white	default
	r g b	<int int int>
foreground colour	<colour>	foreground colour for text, lines and images
	as for background colour	default ***black***
label string	<character string>	[a-z A-Z 0-9 .<> etc]

label position	<position>	position of label relative to field:
	left	
	above left	
	below	
	right	
	above right	
	below right	
	above	
	below	
	fit above	
	fit below	
	fit left	
	fit right	
label font	<vfont>	font for labels
data font	<vfont>	font for data
application font	<vfont>	font for defaults and application use
description	text block describing field	The text may contain tabs `\t', new line `\n' and line continuations `\'.
help	text block giving help information about field.	
style	<style>	varies for each field type:
	European *American*	date field
	underlined *enclosed*	experimental
menu	<items>	This is useful for buttons where the item name is the filename of an exrep bitmap.
selection border	<w> ***2 pixels***	width of border in pixels around field when selected.
border	<x> ***1 pixel***	width of border in pixels around field.
vertical offset	<n characters>	used for duplication.
horizontal offset	<n characters>	used for duplication.
label colour	<shade>	as for shades.

4.9.3. A GUIDE TO DEVELOPING A PROFORMA USER INTERFACE

The nature of the information that is represented by a form is context related and also depends upon how it is designed. Design guidelines for forms may be found in [RUBINSTEIN 84], [SHNEIDERMAN87], [SMITH 86], [MORELAND 83], and [JENKIN 82] cited in [BARKER 89] "Basic principles of Human-computer Interface Design".

The main principles behind the development of the original Xerox Star user-interface, which have propagated the development of many graphical user interfaces, are useful guideline's and are summarised below:

i) the use of familiar concepts;

ii) the application of seeing and pointing operations rather than remembering and typing;

iii) the utilisation of WYSIWYG technology wherever feasible and useful;

iv) the use of universal or generic commands wherever possible;

v) consistency;

vi) keeping the system simple;

vii) modeless interaction;

viii) and allowing the user to tailor the system.

After extensive proforma definition and interaction with the forms package the following general hints to aid the definition of consistent user interfaces are provided, addressing some the physical attributes in more detail.

4.9.3.1. SIZE

The size of a field or form refers to the length and height of the field in character units or pixels. Two fonts are used for displaying concept values: data font and application font. The larger of the two is used as character units.

4.9.3.2. ORIGIN

The origin of a field or form refers to the position of the top left corner of the field relative to that of the parent form in either parent form character units or pixels. Figures 4.9.4.2.1 to 4.9.4.2.3, indicate the origin and size datums for each of the three response types; free, restricted (Boolean), restricted (multiple choice).

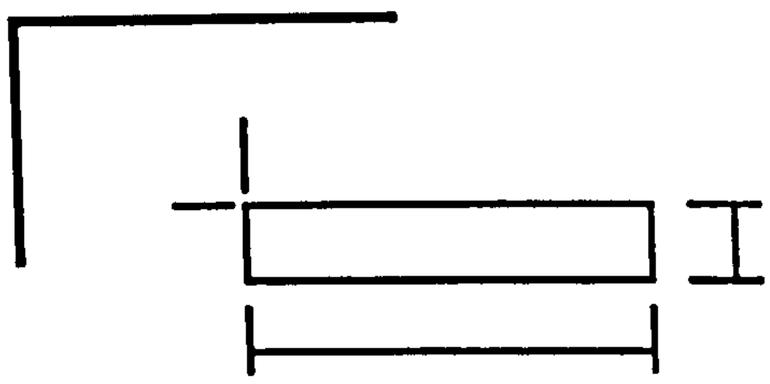


Figure 4.9.4.2.1. Free response origin and size datums.

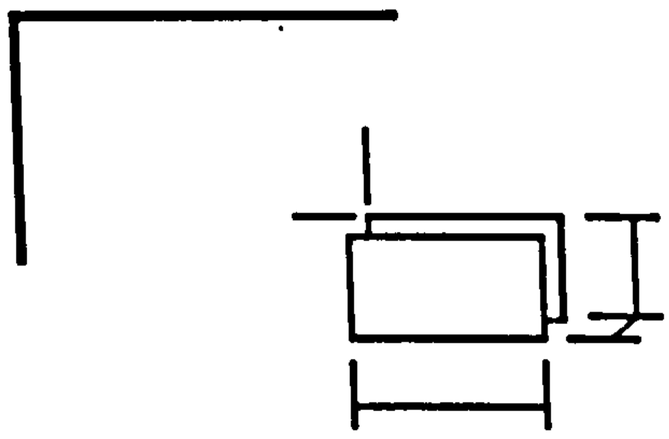


Figure 4.9.4.2.2. Restricted (Boolean) response origin and size datums.

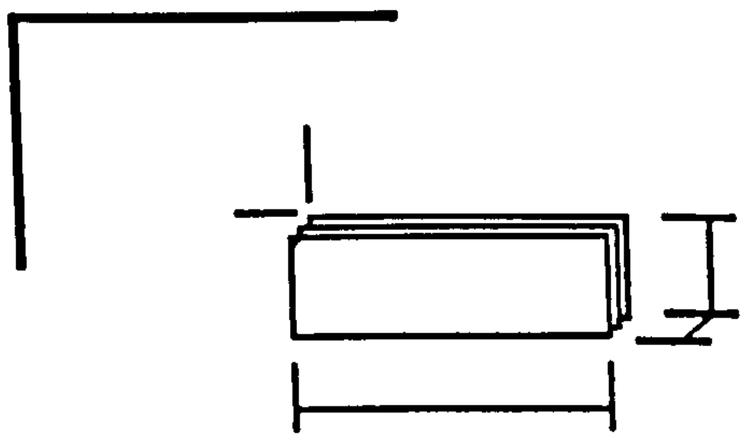
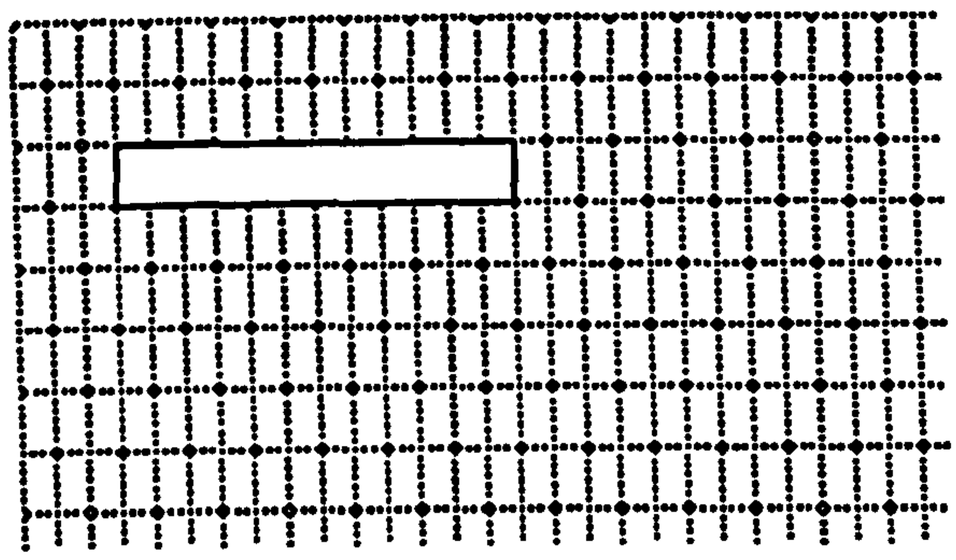


Figure 4.9.4.2.3. Restricted response (multiple choice) origin and size datums.

Note origin and size are defined separately rather than defining an area using top left bottom right; where ambiguities may arise

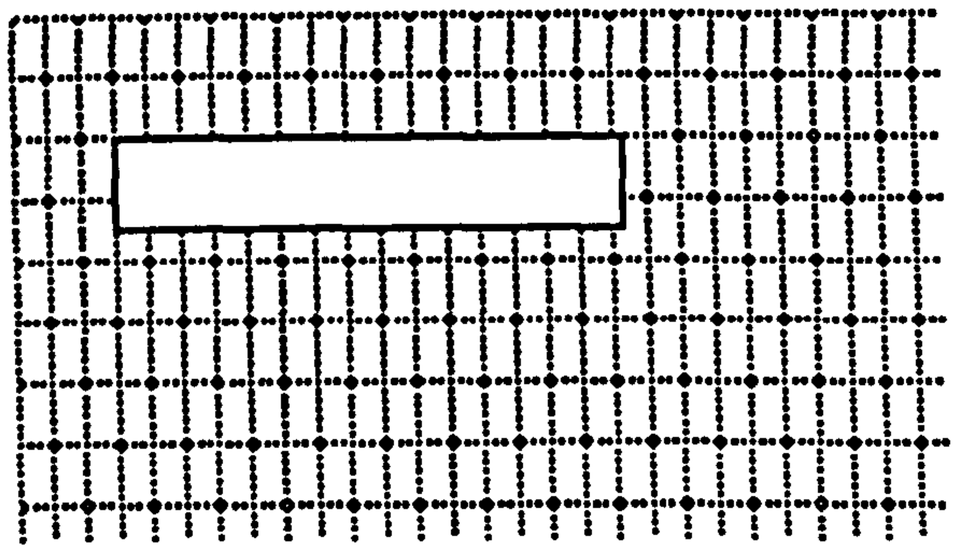
4.9.3.3. POSITIONING FIELDS WITHIN A CHARACTER GRID

When specifying the origin in character coordinates the font size of the parent form/desk is used to set the grid spacing and not that of the field.



Provided the maximum sized font between the application and data fonts is the same as that of the parent form alignment of fields is guarantied, figure 4.9.3.3.1, left.

Figure 4.9.3.3.1. Field position using character coordinate system



However larger font sizes result in alignment problems figure 4.9.4.3.2, left. Positioning fields with a pixel coordinate system ensures precise field alignment regardless of parent font sizes.

Figure 4.9.3.3.2. Field alignment problems

4.9.3.4. LABEL STRING

The field label string is the user's concept identifier. It may be specified as a character string containing any number of characters and new lines. The label may also be a bitmaped image (see image format and icon directories).

Although it is not necessary to specify a label string, it often good practice to do so. In the case of button fields, where the contents of the field is sufficient to describe the concept to the user, the label may be omitted. To enable the user to access the concept menus, the label colour may be specified as black. The label box is set to a proportion of the label fonts size, resulting in a black rectangle, which may be positioned relative to the field.

4.9.3.5. LABEL POSITION

The field label position specifies the position of the field label string relative to the field interpreter. A box is automatically generated for the label string or image and is offset horizontally or vertically from the field, depending upon its relative position, by a distance:

$$d_{horiz} = \frac{f_{width}}{2}$$

or $$d_{vert} = \frac{f_{hieght}}{4}$$

where; f is the current field data font.

4.9.3.6. BORDER AND SELECTION BORDER

The border around the field or form is specified in pixels. On selection the current field and parent forms are outlined by a continuous line of thickness n pixels, specified by the selection border attribute in the proforma template.

4.9.3.7. EXTENDING CHARACTER VALIDATION

The most common field type used is the text field. Each derivative of this field type restricts data entry to a particular type to ensure flawless data acquisition.

eg. type integer will limit data entry to characters 0-9.

For some applications this may be too rigourous; a field containing a telephone number, for instance, would require a space or hyphen between the area code and the remainder of the number. An integer field, while ensuring a valid entry of numbers, would not allow for this directly. In order to extend the range of permitted characters a, the field type declaration should be followed by a `+' and a list of additional characters.

Example:

```
new field
        name    telephone number
        type    integer + -
        :               :
        :               :
end field
```

This would only allow integer values, spaces and hyphens to to be typed into the field. Note the format of the data is not checked. A valid entry from the forms point of view would be:

55 2 44-00, which is a bit silly.

To achieve consistency within a particular application it may be necessary to set all integer fields within a proforma to accept spaces and hyphens. This is achieved by using:

accept integer + -

outside the field description, usually at the head of the file.

4.9.3.8. THE DE-SELECTION DILEMMA

Concept values are only reported when a field is de-selected, by selecting another. To ensure that all values are reported it is necessary to force the user to de-select a field. This is achieved by providing a dummy field usually of type button and containing the word *OK* or *done*.

4.9.3.9. SYNONYMS

One important feature of this declarative syntax is the use of two concept names: one fixed concept name, which is manipulated by domain knowledge, and the other (label string) manipulated by the user. This provides a mechanism for conceptual mapping between two systems (computer and user). The field label is therefore a synonym, a word or phrase that means exactly or nearly the same as another (field name) in the same language [LOD 84](page 606). By enabling the dynamic manipulation of field labels a means of introducing a re-phrasing mechanism is achieved. Another means of re-phrasing is by the substitution of concept (value) interpreters.

4.9.3.10. RE-USABLE CONCEPTS

Where the forms package is to be used by a number of applications it is important to ensure that all concepts are consistent between each application. In order to achieve this and reduce the amount of work involved in defining the interface it is suggested

that common concepts and meta concepts be isolated and stored in separate files for *including* in application proformas. Examples may be file management fields for saving retrieving and deleting files.

4.9.4. COMMAND LINE ARGUMENTS

The forms package may be run interactively from a tty terminal by typing control messages. This is useful during the development of the proforma interface where field positions and sizes may be refined.

In order to invoke the forms package simply type:

forms proforma <options>

Valid options are:

-P proforma directory - look in the specified directory first

-I <Image directory> - search here first for exrep image files.

The above options may also be specified in the environment variable FORMOPS.

4.10. MISTAKES

The forms package has evolved over a period of two years with additions and refinements being made when deficiencies in its functionality where identified. Although the package is fairly robust, complaining only when the defined protocols are not followed, a number of fundamental errors of judgement have been made which have become apparent over a period of time.

4.10.1. SELECTION BORDER

Adding the border to the field boundary creates positioning problems. If fields are too close the select/de-select function may overwrite the borders of neighbouring fields. The selection border would be best accommodated within the bounding rectangle of the field interpreter, and result in a cleaner appearance.

4.10.2. BITMAPS

Using bitmaps as form pages requires an enormous amount of memory. Although this method has numerous implementation advantages and some for interaction (scrolling, pseudo window manager) the drain on memory is unacceptable and therefore a less demanding method for implementing forms has been sought. This involves bitmap clipping, with each form, rather than containing a bitmap, having a clip box onto the window's bitmap. When a form is activated all display updates to the window will be clipped by the form's clip box. A number of experiments have been undertaken and the method works reasonably well. The only foreseen disadvantage with the technique is anticipated to be with scrolling. As the forms window is scrolled, rather than copying another part of the form's bitmap up to the level above, each descendant of the form will have to be shifted. Therefore scrolling response will decrease significantly with increased nesting levels and numbers of fields on each form, whereas the use of bitmaps carries no speed penalty.

4.11. SUMMARY

Since a generic data structure has been developed, all methods are constructed upon a consistent framework and therefore may be interchanged dynamically, resulting in a multi-representational system; enabling real-time metamorphosis. By utilising a rich variety of concept interpreters the look and feel of the user-interface may therefore be modified to suite a particular class of user simply be replacing the interaction module. This may be done at the dictates of a user model. Owing to the modular structure of the forms package it is relatively easy to extend the range of concept interpreters.

The approach adopted during the development of the forms package has also been applied to the visualisation and manipulation of geometrical bodies (Appendix E).

The benefits of formatted natural language utterances, ensure complete independence and therefore re-usable knowledge. The following chapter places the forms package in the context of the IFe for which it was designed, and describes dialogue control mechanisms in more detail.

5. THE INTELLIGENT FRONT-END

Portions of this Chapter contain extracts from the IFE final report [IFE 89], "The application of intelligent knowledge based systems in building design" (Clarke, MacRandal,Rutherford). Individual contributions are acknowledge accordingly.

5. THE INTELLIGENT FRONT END

The discussion has so far concentrated upon the development of an adaptable user interface. The following sections describe in detail the issues involved in implementing a front end to a complex application (ESP) placing the forms package in the context for which it was designed.

The Ife was developed in response to the inadequacies of traditional approaches to user interface and application development and encompasses many disciplines which until now have remained confined to their original domains. The IFe is a synthesis of current IKBS and HCI techniques and methodologies. Using these techniques it is possible to construct a user interface which incorporates a significant level of knowledge in relation to building description (CLARKE)[IFE 89].

The IFe system is composed of a number of cooperating modules, illustrated in figure 2.7.1.1. Crucial to the operation of the IFe are the inter-client communication mechanisms.

5.1. INTER CLIENT COMMUNICATION

When knowledge sources or modules are initialised, at start up, each makes a request for the creation of a working memory area on the blackboard. Individual knowledge sources may also express an interest in other areas. As messages are posted, by individual modules, into their respective blackboard areas, the blackboard informs subscribing knowledge sources of the event. Traditionally this would be achieved by continually polling the blackboard for events; resulting in deadlock. Where two or more resources express an interest in the same blackboard area each is notified in turn. Although it is not necessary for this particular application (owing to the relatively small number of knowledge sources involved), a scheduling mechanism would normally be invoked, giving priority to a particular knowledge source. This is discussed in the final chapter on Intelligent Design Assistance.

Figure 5.1.1, bellow, illustrates the current partitioning of the blackboard and indicates the notification network between each of the modules.

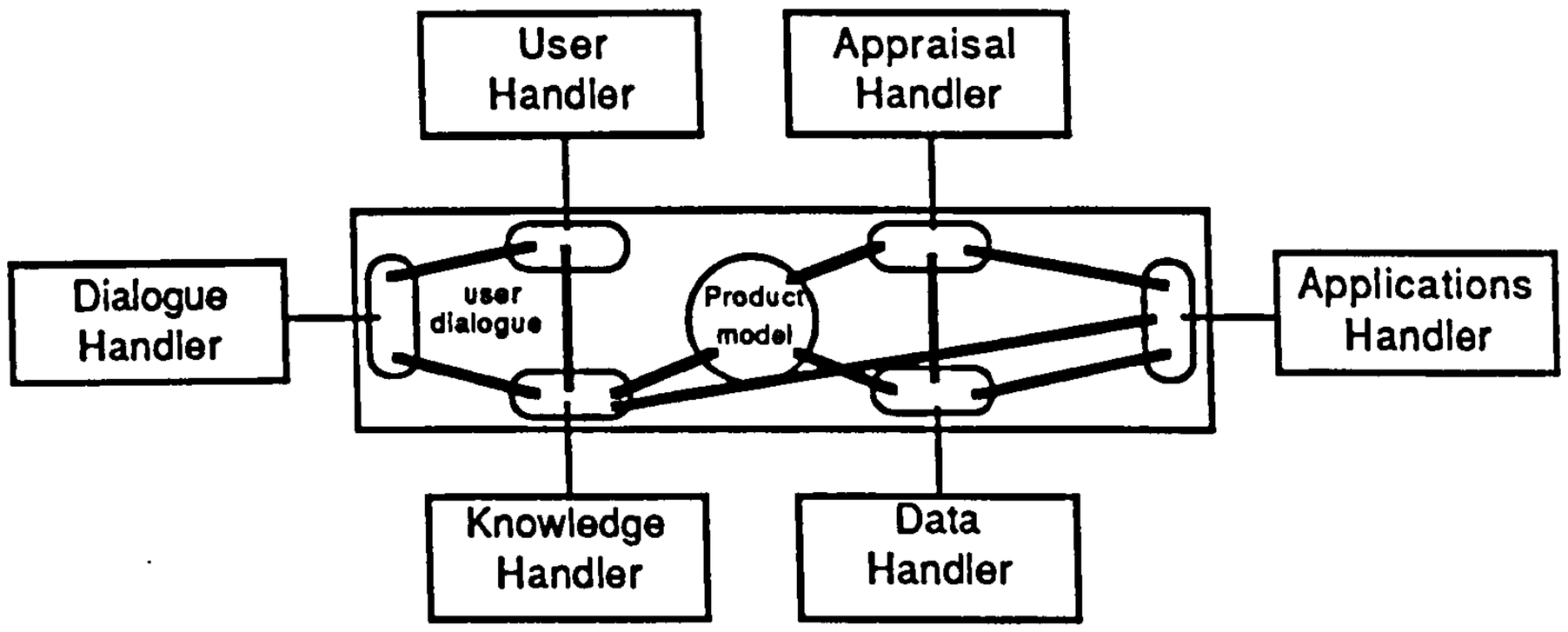


Figure 5.1.1. Inter-client communication: The relationship between IFE modules. Concepts are posted by knowledge sources into their respective blackboard areas. Other knowledge source, expressing an interest in these issues, are duly notified. Note that the relationship between user dialogue and applications is bridged by the product model or central data base.

The knowledge sources, illustrated above, are categorised into two groups:

i) those, to the left of the product model, relating to issues involved with human-computer communication (user_dialogue), and those to the right,

i) concerned with the management of product information and the control of computer applications.

This distinction is indicated figure 5.1.1. Common to both areas is a product model, containing the current description of the user's product.

The above infra-structure is capable of supporting any form of dialogue with the user. However, rather than adopting a natural language dialogue, a more graphical oriented interaction is preferred for the description of conceptual models. For this purpose the forms package, chapter 4, (and any other interaction program) may be coupled to the dialogue handler by means of a Unix pipe.

The purpose of the dialogue handler is to support a number of (possibly simultaneous) dialogues with the user, converting events or utterances between the different systems.

5.2. ORCHESTRATING USER DIALOGUE

A dynamic dialogue with user may be set up by employing one or a number of the techniques and mechanisms already discussed.

The communications protocol (in particular absolute symbolic addressing) described in chapter 4.8.4.1 is fundamental to the operation of the IFe.

Utterances from the user (generated by interaction tools, such as the forms package) are formatted by the dialogue handler and are posted in the form of text tokens to those knowledge bases concerned with monitoring user dialogue. By the same means messages are passed to the interface in order to modify and tailor the dialogue according to a particular stereo-typical template.

5.2.1. A HIGH LEVEL NEUTRAL LANGUAGE

In order to isolate the IFe from application development in other domains and maintain a high degree of independence between application programs and domain specific knowledge sources (which effectively orchestrate these deep models), a standard high level neutral language has been formulated [MacRandal]. As in the case of the forms package the range of methods available for orchestrating an interactive dialogue with the user may be categorised into:

i) dialogue control mechanisms, and

ii) user utterances.

The components of this high level neutral language [MacRandal] and the corresponding commands issued to and from the forms package [Rutherford] are outlined below [IFE 89]:

new_dialog — This is a request to initiate a new dialogue with the user, usually by starting a new interaction program (for example a map utility for inputting locations). The predicate requires three arguments:

- the name of the dialogue utility
- the filename containing the executable (binary or shell script),
- an argument list passed to the program

The complete request is posted to the user_dialog area of the Blackboard and in turn the dialogue handler is notified. In order to invoke a map utility the following message is posted:

< new_dialog ife_map_prog ~ife/bin/map "-s -o -e" >

This is a little too low level and should ideally be in a generic domain related task specification to be solved by another knowledge source. This is discused in chapter 7.

Currently, only one forms process is supported by the dialogue handler as the forms package is capable of supporting multiple dialogues in multiple windows.

focus_user

This is a request to direct the user's attention to a new meta-concept (other current meta-concepts may still be addressed by the user). The intention is to introduce a new domain of discourse by presenting the user with a named dialogue frame. This mechanism is usually invoked in response to the user's request to progress along a particular thread of discourse. A typical example is:

< focus_user geometry >

The request resulting in a the named mata-concept (form) to be (re)displayed and highlighted for easy identification. If the form is not already in memory it is loaded by the forms package from a file of the same name form the current user conceptualisation directory ~ife/lib/uc/?/forms/geometry. The focus user predicate also loads the corresponding knowledge base into memory from ~ife/uc/?/kbs/geometry (see focus of attention chapter 5.6.2).

unfocus_user

This request removes the named meta-concept from the current domain of discourse; the concepts contained within the dialogue frame are no longer accessible for input. This particular command ensures that the user is not overwhelmed by concepts which would otherwise result in NON closure [Miller 57]. In addition it ensures that valuable display real-estate is kept tidy. The unfocus_user predicate is usually invoked by the user when the particular domain of discourse has been exhausted (finished input):

< unfocus_user geometry >

In the forms package this results in the "geometry" form being removed from the screen (hidden),(see focus of attention chapter 5.6.2).

ask_user

When focusing on a meta-concept, some concepts are implicit and will be addressed by the user without further prompting (i.e. the concepts are visible when the form is activated). Other concepts may only be relevant for particular situations (re-description or elaboration) in order to resolve ambiguities or inconsistencies in the blackboard model; the user is forced to address these particular concepts. In order to explicitly define a site location, for example, the following tuples are issued:

< ask_user latitude >
< ask_user longitude >

It is also possible to suggest a default value by adding a further argument:

< ask_user latitude 55.7 >

The forms package responds by (re)displaying the field provided that has been previously defined (usually as hidden).

unask_user

In some cases, a concept that has previously formed part of the dialogue may become irrelevant owing to other information. For instance, if a user requests an annual simulation, it is perhaps inappropriate for the user to specify a start and finish date. In such an instance the concepts are withdrawn from the dialogue:

< unask_user start_date >
< unask_user finish_date >

They may however by re-introduced into the domain of discourse with the ask_user command.

new_query

This is a mechanism by which previously undefined concepts may be introduced to the user. It is necessary to also specify what type of data (syntax checking) is anticipated which is obviously application specific. This additional information may be communicated in two ways. Currently it is achieved by passing a complete description of the field:field name, and type:

< new_query user_age integer >

alternatively this additional knowledge may be held in a dictionary containing concept names, data types and alternative representations useful in re-phrasing. This ensures complete domain independence:

< new_query user_age >
< new_query date_of_birth >

with corresponding dictionary entry:

user_age integer
date_of_birth integer+/ date

The dictionary may also contain preferred screen positions or general locations such as the name of a particular dialogue frame in which case the forms package would automatically position the field.

tell_user

This is a mechanism for setting the value of a named concept. Two arguments are required in addition to the concept name:

< tell_user session_number 2 >

The forms package handles arrays of fields as a simple list. each list element has a general base name and an index number; vertex[4] or zone_name[6] (see duplicate). To address a list element an additional $ is required, for example:

< tell_user zone_name$6 kitchen >

The $ symbol is required as prolog does not permit the use square brackets in argument lists.

suggest_user

Default values for concepts, base on contextual knowledge and in-built knowledge, may be suggested to the user with this mechanism. Rather than explicitly telling the user, it is possible to suggest that an associated value is an appropriate one. The actual value of the concept is not set until explicitly accepted by the user. It is often much more convenient for the user (particularly the novice) to select the default value than enter the same information. For example, to handle the concept of time, the referenced time zone must be known. Assuming GMT is the most likely (deduced from a site environment variable), the knowledge handler may send the message:

- < suggest_user time_zone GMT >

The forms package will display this value in a different font from that user supplied data is entered with in order to indicate that it has been set by the knowledge base. The default value is also stored on a pop-menu so that it may be retrieved if it is overwritten by the user.

offer_user

In some instances, the knowledge handler may be able to make extensive inferences about a subset of possible values for a concept. For instance, if the user enters the name of a material that is defined in one of the available domain related databases, the knowledge handler will extract the relevant material properties, otherwise the information will have to be elicited from the user. One option to the knowledge handler is to supply the user with a list of known materials using:

< offer_user material_type paper,wood,brick,concrete,stone >.

This information is stored by the forms package and becomes accessible to the user by means of a pop-up menu which is activated by a right mouse button event over the concept label.

In summary, the current mechanisms available to communicate concepts to the user are:

Request	Meaning	Example
new_dialog	Switch to a new interaction program	Use the map program
tell_user	Inform the user about something	Set the contents of a concept field
suggest_user	Suggest an appropriate value for a concept	Set or reset a field's default value
offer_user	Present sensible options to the user.	Set menu options
focus_user	Direct the user's attention to a new meta-concept.	Display a form
unfocus_user	Finish addressing a particular meta-concept	Hide a (displayed) form
ask_user	Request a specific piece of information or data	Display a concept field
unask_user	Withdraw a request for data	Hide a (displayed) field
new_query	Ask an unexpected question	Create a new concept field

Utterances from the user are formatted by the dialogue handler and take the form:

user_dialog	user_said	concept	value
user_dialog	user_request	concept	help/description/example/default
user_dialogue	user_error	concept	error.

5.3. BLACKBOARD DATA

Following on from the communications protocol the basic element of information held and manipulated within the IFe is a Tuple:

<concept> <value>

A Tuple consists of a concept together with it associated value. All information is held in this form by the blackboard. In addition to these two elements of data the source of the concept is stored along side the concept and value; enabling information to be traced back to it's origin.

<concept> <value> <origin> <time>

The concept is also time stamped ensuring that there are no outstanding events left on the blackboard. This also enables the user to backtrack through earlier interaction sessions by recovering previous values.

The actual knowledge sources responsible for managing uer_dialog are prolog interpreters (NIP [HUTCHINGS 86]) and are responsible for:

- translating user supplied data in to an internal representation, and for
- transferring data from the short term working memory region of the user dialogue area to the long-term store of the product model.

5.4. ORGANISING KNOWLEDGE

Knowledge sources are oportunistic in nature and as such must be loosely structured to avoid procedural dialogues. A knowledge base, representing a meta-concept, therefore contains discrete logically related packages of knowledge (prolog predicates) regarding individual concepts. A complete description of a particular problem domain may consist of many discrete meta-concepts.

The actual structure of an oportunistic knowledge source is hetra-archical facilitating multiple entry points. The dialogue may be entered at any level of detail, thus enabling a free unrestricted dialogue to be employed within a particular sub-domain. Such an approach enables users to address concepts and meta-concepts in any order; resulting in a system that supports idiosyncratic design procedures (both top-down and bottom-up). It imposes few restrictions upon the knowledge engineer although to support multiple dialogue entry points path algebras (chapter 2.6.6.2) and transition handlers between states must be employed to ensure closure [ALTY 83].

Different people may adopt different conceptual vocabularies and therefore different interface templates and corresponding knowledge bases are required in order to communicate effectively with each type (or class) of anticipated user. These are collectively known as user conceptualizations or stereotypical templates. In order to simplify knowledge management and ensure complete generality with the IFe system (no knowledge of the domain is held within the blackboard or interaction handlers) a systematic approach towards managing knowledge bases and interface templates is required. Figure 5.4.1, illustrates the method of organising knowledge within the IFe and must be observed closely. In addition to illustrating the relationships between knowledge bases the figure also indicates the location of other system related files such as source code and utility programs.

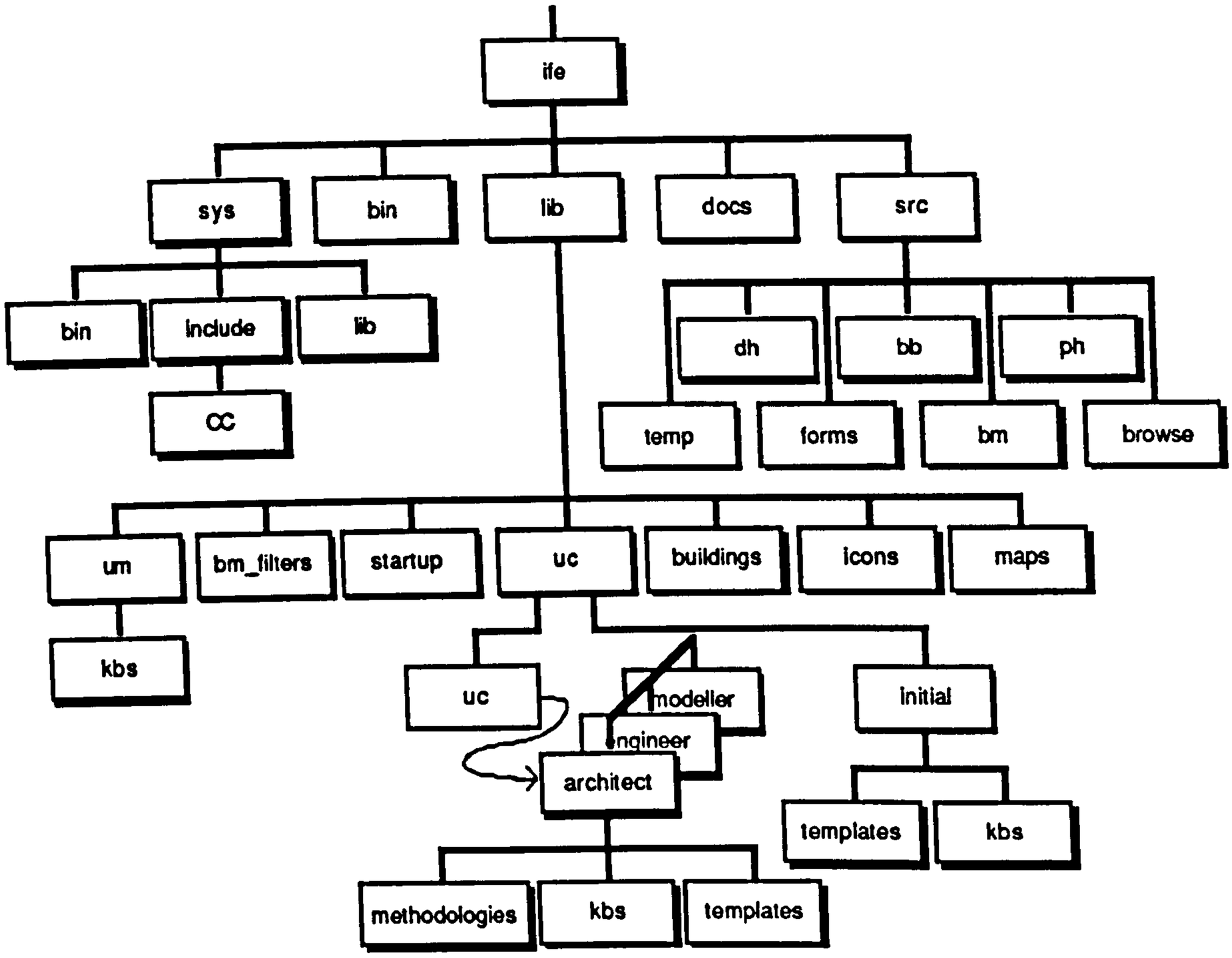


Figure 5.4.1. Current IFe directory structure. The key to the use of multiple user conceptulaisations is the symbolic link ~ife/lib/uc/uc which points to the conceptualisation the user model thinks the most likely.

Each user type is represented by a directory containing sub-directories for knowledge bases (kbs) and proforma templates (forms). A file for each meta-concept within the domain is stored in each of these directories. The name of the knowledge base and that of the corresponding interface template must be the same so that both the knowledge and dialogue handlers can refer to the same meta-concept. So that neither the dialogue or knowledge handler are aware of what user conceptualisation is being employed at a particular time, the proforma templates and knowledge bases are consulted from the ~ife/lib/uc/uc directory, which as illustrated in figure 5.4.1, is a symbolic link to one of the anticipate user stereotypes. This link may be changed dynamically at any instant by the user model (using the change_cpt_set shell script) and therefore, rather than a dialogue following a single thread, an appropriate user conceptualisation may be chosen for individual meta-concepts within the same interaction session.

5.5. BLACKBOARD COMMUNICATION PREDICATES

MacRandal has provided a number of mechanisms, in the form of prolog predicates and utilities, to aid the development of new user conceptualisation. They essentially enable knowledge sources to interact directly with blackboard areas. The current set of mechanisms is included for completeness. In the following list, the variable C is a concept name, V is a concept value and K is a list of key values used to discriminate between a collection of identically named concepts (for example <x_coord, [1,3,4], 15> is the x-value of vertex 4 of surface 3 of room 1)[IFe 89].

Predicate	Function
startup	Starts the knowledge Handler, creates Blackboard areas feedback(C), invokes a predicate feedback(C,_User_level) for the appropriate user level (currently novice or expert). These are supplied in the user conceptualisation alongside the concept.
The following post a Tuple to the "user_dialog" area on the Blackboard	
new_dialog(Name, Command)	Starts a new interaction program called "Name" using the unix command "Command".
focus_user(C)	
unfocus_user(C)	
ask_user(C)	
ask_user(C,V)	
ask_user(C,K)	
ask_user(C,K,V)	
unask_user(C)	
unask_user(C,K)	
tell_user(C,V)	
tell_user(C,K,V)	
suggest_user(C,V)	
suggest_user(C,K,V)	
offer_user(C,V)	
offer_user(C,K,V)	
chat_user(T)	Appends the list of strings T to the concept user_chat
The following predicates post a Tuple to the "u_cpt" (user conceptualisation) area on the Blackboard	
uset(C,V)	Flagged as user set
uset(C,K,V)	
kset(C,V)	Flagged as knowledge base set
kset(C,K,V)	
The following recover a Tuple from the "u_cpt" area on the Blackboard.	
known(C,V)	
known(C,K,V)	All keys must be specified
known(C,W,V)	As known(C,V), but W indicates where value originated.
known(C,W,K,V)	
u_cpt_got(C,V)	user_supplied, invoked when someone else sets a concept value.
The above are macros defined using the following predicates:	
to_bb(A1)	These create and send a Tuple to the Blackboard.
to_bb(A1,A2)	A1 is the Blackboard are to post to.
to_bb(A1,A2,A3)	A3 is a string (usually indicating where the value came from).
to_bb(A1,A2,A3,A4)	A2 is the concept being posted, A4 is a list of keys.
to_bb(A1,A2,A3,A4,A5)	A5 is a list of values
quitrqst	Stops the Knowledge Handler (exit gracefully).
The following are are genral utilities in the file "utilities".	
refresh(C)	Tells the user everything known about the concept.
near()	Compares positions for (close) match - see definition in file.
gen_integer(X,S)	Implements a for loop X = 0,S.
append(H,T,L)	Appends list T to list H resulting in list L.
member(E,L)	Checks if element E is on the list L.

Table 5.5.1. IFe communication mechanisms [MacRandal][IFE 89].

5.6. MONITORING USER DIALOGUE

Events or formatted utterances from the user are posted by the dialogue handler on to the user_dialog area on the Blackboard in the following format:

<user_dialog user_said concept value>

Both the User and Knowledge Handlers are informed of each message and selectively interpret the information. The Prolog knowledge handler essentially matches each incoming concept with predicates stored in the knowledge base, held in memory. When a successful match has been found the predicate is fired, which in turn fires other predicates, perpetuating the dialogue. Data, once formatted, is posted onto the Data (product) area on the Blackboard for selective retrieval by the resource handler (and on to individual application packages) (described in section 7). A full illustrative example is provided in chapter 6.

5.6.1. MODIFYING THE USER INTERACTION

Very little in the way of user modelling was achieved during the project owing the complex nature of developing stereotypes. However the following mechanisms are suggested.

5.6.2. FOCUSING THE USER

Owing to the interpretive language of both the prolog knowledge handler and the forms package, knowledge sources and proforma templates may loaded dynamically and thus facilitates a truly dynamic dialogue.

In order to take advantage of this dynamic environment two prolog utilities (focus_concept and defocus_concept) have been developed, illustrated below, to focus the user's attention towards a particular meta-concept, loading both knowledge base and interface template dynamically.

```
focus_concept(_Focus_type, _Meta_concept):-
        defocus_concept(_Focus_type),                    %% defocus relevant meta_cpt
        focus_usr(_Meta_concept),                        %% enable input, flag as currently
        assert(focus(_Focus_type, _Meta_concept)),       %%  available for input
        assert(called(_Meta_concept)),                   %% has been addressed (data input?)
        name(_Meta_concept, _Mcpt_str),                  %% load handler for this meta-concept
        append("lib/uc/uc/kbs/", _Mcpt_str, _Mcpt_str2),
        name(_Meta_concept2, _Mcpt_str2),
        [-_Meta_concept2].

defocus_concept(_Focus_type):-
        ( focus(_Focus_type, _Meta_concept) ->
                name(_Meta_concept, _Mcpt_str),          %% turn off button; b_ prefix
                append("b_", _Mcpt_str, _Mcpt_str2),
                name(_Meta_concept2, _Mcpt_str2),
                tell_usr(_Meta_concept2, 'off'),
                unfocus_usr(_Meta_concept),
                retract(focus(_Focus_type, _Meta_concept))
        ;
                true
        ).
```

Figure 5.6.2.1. Focus concept: a generalised predicate for focusing the user's attention towards a meta concept

The predicate is used in the form

focus_concept(type,meta_concept)

where; type is the general classification of the meta-concept and in the case of the forms package is used to determine the position in the conceptual model (parent form) of the meta_concept.

Thus by using:

focus_concept(building, geometry)

the user is directed towards the issues relating to a buildings geometry. Issuing the command again with the same concept category:

focus_concept(building, materials)

will result in the current meta-concept (geometry) being de-focused before the materials specification form is displayed. Note that as meta-concepts (forms) occupy relatively large areas of the screen, rather than simply flashing forms on and off (which may cause user discomfort) a soft focus and de-focus mechanism is employed (this is best illustrated with an interactive demonstration of the software but is illustrated in figure 4.8.3.2, for completeness).

The focus_concept mechanism forms the basis for the re-description technique described in chapter 3. A number of approaches to use of conceptual re-description are possible using the inbuilt predicates or macro definitions. Both are now described.

5.7. STRUCTURING A KNOWLEDGE BASE FOR RE-DESCRIPTION.

Re-description is the ability to elaborate (or simplify) upon a particular concept; decomposing a concept into a meta-concept or condensing a meta-concept into a concept.

For example, a site location may be specified either by its:

location_name

or by a coordinate pair:

latitude
longitude.

The knowledge base must contain predicates to handle the components of the meta-concept or as a whole. Therefore, for the example of location, predicates are required for:

- location_name - collective value of
- location_latitude and location longitude

In order to provide an environment capable of re-description rules must be provided to infer the collective value of the concept from the individual elements and vice versa.

```
location(_Name):-
        location(_Name,_Lat,_Long),
        if(location described in detail tell_user(_Lat,_Long)
```

```
location_latidude(_Lat):-
        if(location longitude is know)
        infer location name (latitude, longitude).
```

```
location_longitude(_Long):-
        if(location latitdude is know)
        infer location name(latitude, longitude);
```

The extent of conceptual decomposition is related to the user's level of experience and understanding and may be achieved using dialogue control mechanisms of the form:

describe(location,_user_level)
or describe(location,_dialogue_level)

Mechanisms for switching between verbose and terse descriptions of site_location are illustrated below:

```
describe(location, verbosely):- %% in detail
        ask_user(location_latitude);
        ask_user(location_longitude).
        %% or focus_user(location,coordinates).
```

```
describe(location, tersely) %% in less detail
        unask_user(location_latidude);
        unask_user(location_longitude);
        %% or unfocus_user(location,coordinates).
        ask_user(location_name).
```

5.8. RE-PHRASING

Re-phrasing, utilising a different conceptual vocabulary from that currently employed, may be achieved by modifying the interpretation of the concept itself or that of it's value (or both); both methods are facilitated by::

- the dynamic substitution of concept interpreters
- the re-wording of the field label (or use of graphics).

In order to dynamically substitute a concept interpreter with the forms package the command:

@concept:type:character/button/pop-up

is issued, resulting in:

Figure 5.8.1. Free response

Figure 5.8.2. Restricted response

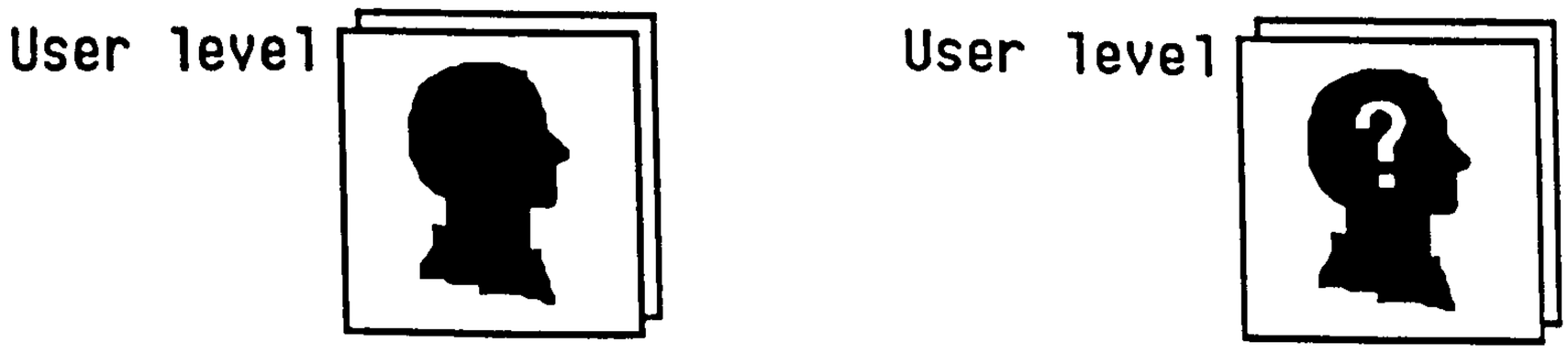


Figure 5.8.3. Re-phrased restricted response

However this is too specific to the interaction device and therefore to maintain the separation between interaction mechanisms and domain knowledge a neutral language, re-phrase macro is defined.

re-phrase(_Concept,_meaning,_user_level)

re-phrase(_Concept,_interpretation,_user_level)

Returning to the example of location

```
re-phrase(location, free_responce):-
        ask_user(location_name);
```

location Paris

Figure 5.8.4. Re-phrased free response field.

```
re-phrase(location, restricted, graphically):-
        new_dialogue(map).
```

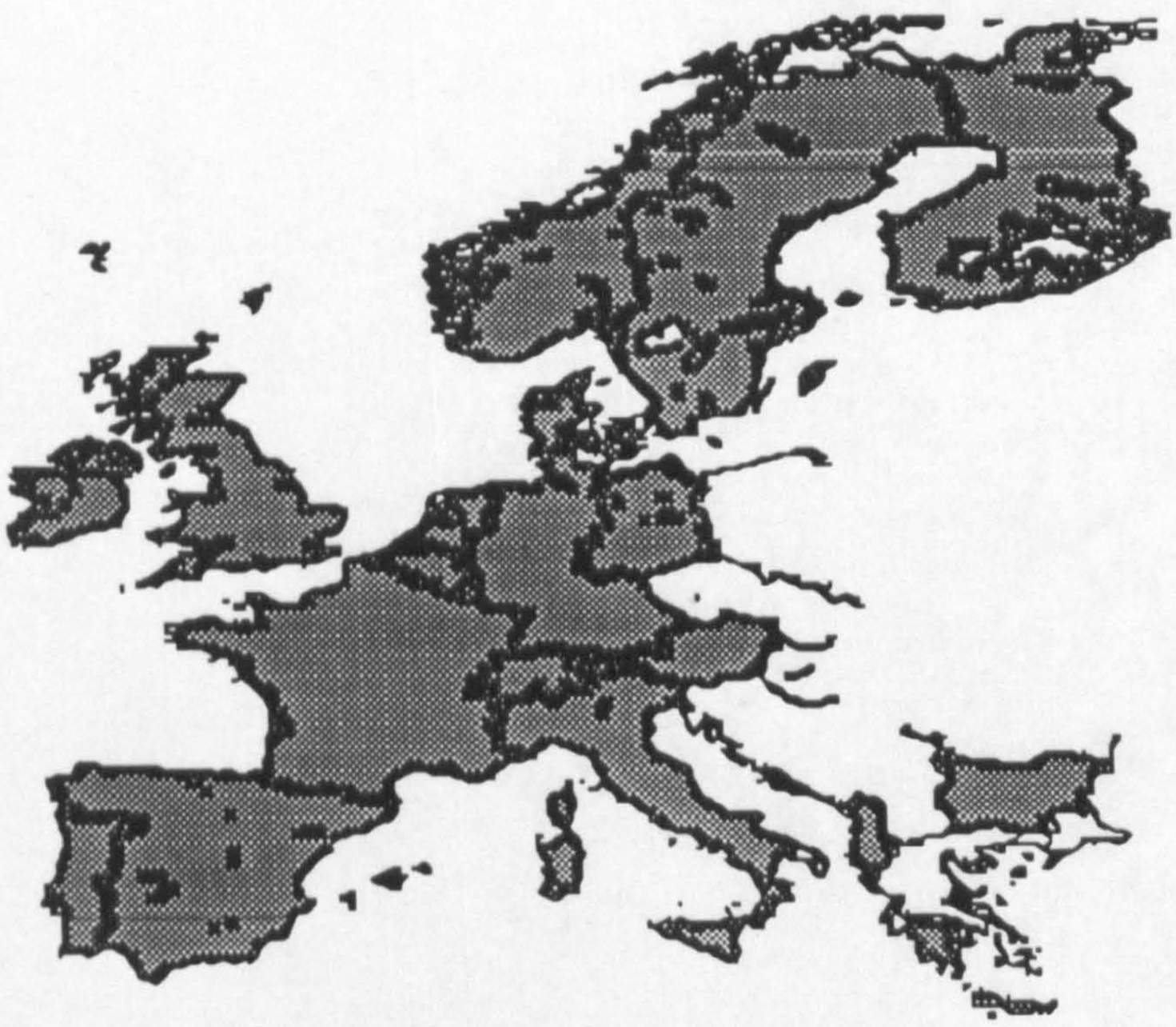


Figure 5.8.5. Location re-phrased by restricted graphical interpretation using map programme

```
re-phrase(location,restricted, textual):-
        restrict_user(location,locations).
```

The restrict_user macro simply issues the command to set the interpreter type of the concept to a menu or pop-up and sets the contents of the menu.

```
restrict_user(concept,list)                          \
        ConForm(concept,SET_TYPE,                    \
                (style == menu)?"menu":popup);       \
        ConForm(concept,SET_MENU,list);
```

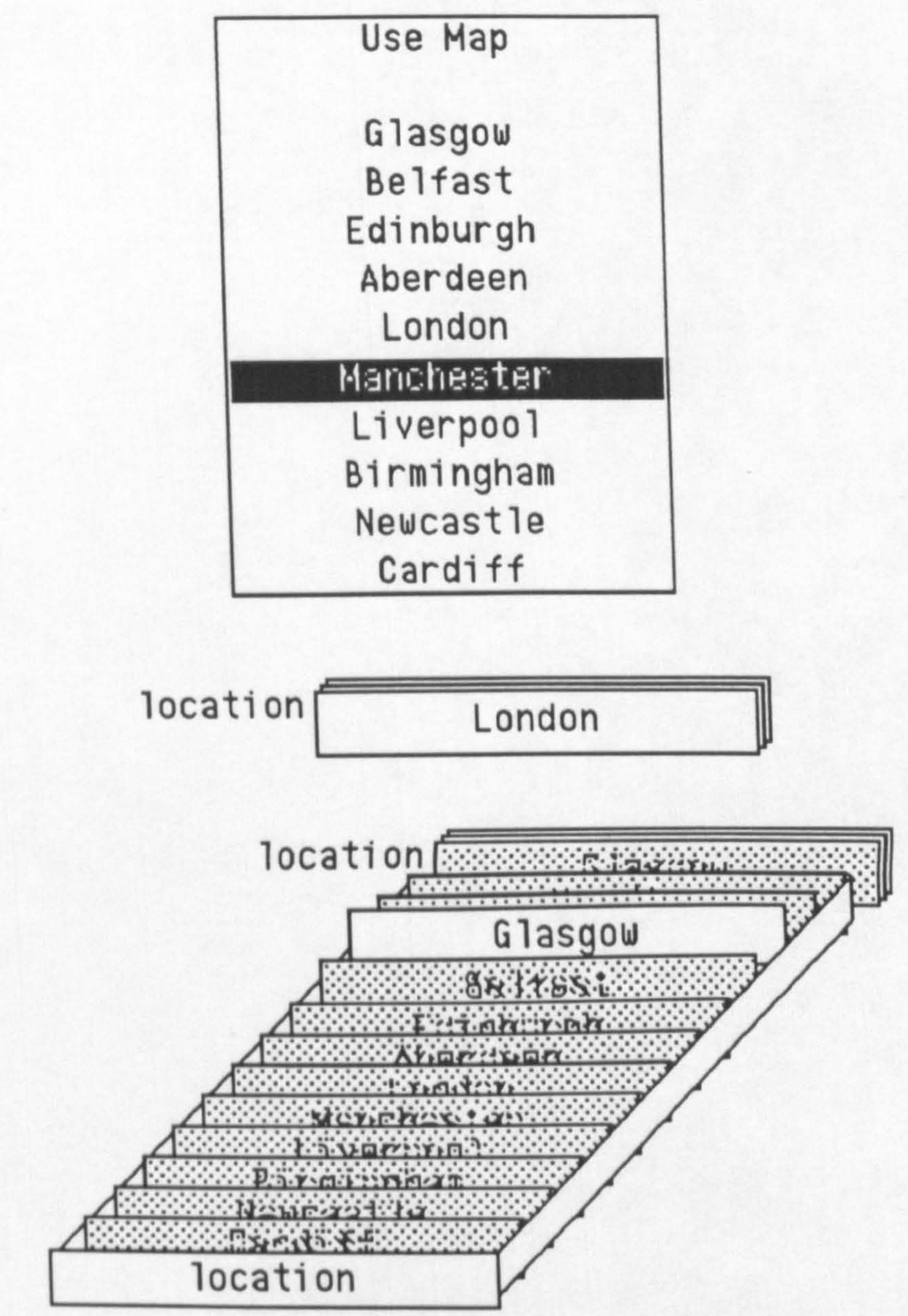


Figure 5.8.6. Re-phrased restricted response fields

Both re-description and re-phrasing mechanisms may be employed at the request of the user_model in response to user actions.

In order to achieve an even higher level of abstraction within the knowledge base it is proposed that dictionaries of selected concepts are used. Dictionaries would take the form of:

concept	meaning		context	associated	interaction	value response	
	terse	verbose		concepts	device	free	restricted
latitude	latitude	latitude	location	longitude	forms	real	
longitude	longitude	longitude	location	latitude	forms	real	
location	location	geographical position of building	bld_spec	latitude + longitude	forms	character	menu ll pop-up
					map		
material	material		construction		forms	character	menu ll pop-up

Table 5.8.1. Suggested dictionary of concepts - + = and, ll = or.

which would be located along side the proforma templates and knowledge bases for each user conceptualisation. The user model would be responsible for selecting an appropriate level of representation for the current users level from the dictionary and posting it to the dialogue handler and, in turn, on to the interaction program.

The fields could be extended to include knowledge of particular applications so that when a particular interpretation is selected the application (such as the map program) would start immediately as a new dialogue.

The forms package requires that the physical properties of a field be specified along side the declaration of the concept. When using the new_query predicate, in order to position the field on a form, application specific knowledge is required, thus rendering the knowledge base specific. This information would be best accommodated within the dictionary as environment preferences.

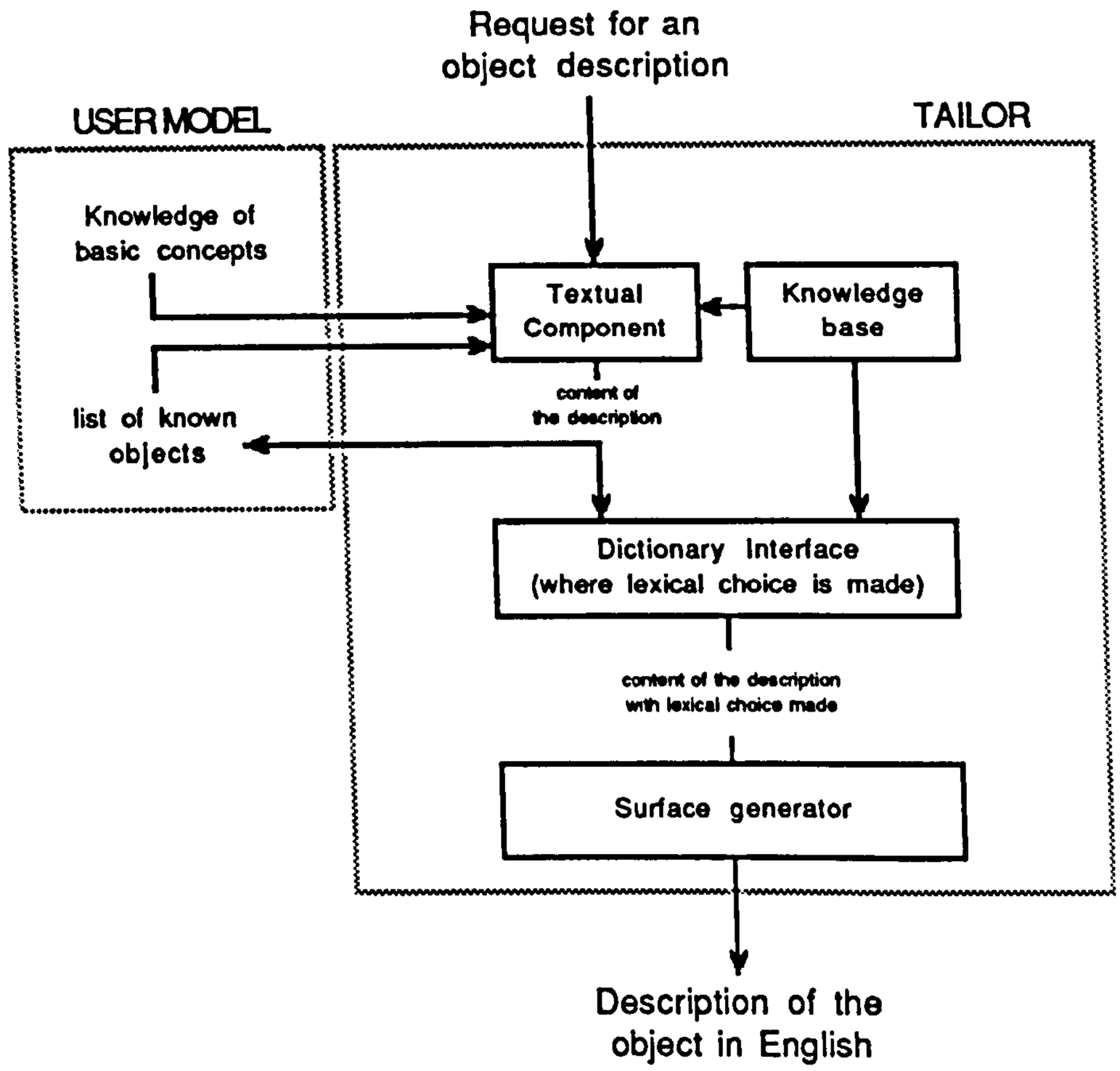


Figure 5.8.7. The TAILOR generatative description system [PARIS 89].

TAILOR [PARIS 89], figure 5.8.7, is a computer system that generates descriptions of object devices using one the two discourse strategies found in text (constituency schema and process trace [PARIS 89]) to construct a description for either a novice or an expert user. Constituency schema (identified by McKeown) is a means of describing an object (or concept) by decomposition into subparts together with a description of each part and is characterised by Paris below:

---

i) [Identify the object as a member of some generic class, using the identification predicate][1]

ii) Present the constituents of the item to be defined (subparts or sub-entities), corresponding to the constituency predicate

iii) Present characteristic information about each constituent in turn, corresponding to the depth-attributive predicate

iv) [Present additional information about the item to be defined, corresponding to the attributive predicate]

---

[1] Steps included in square brackets are optional [PARIS 89].

Using McKeown's notation[2] [MCKEOWN 85] (cited in KOBSA 89) the constituency schema predicates are [PARIS 89]:

{Identification (description of an object in terms of its superordinate)}

Attributive* (associative properties with an entity) / Cause-effect*

Constituency (description of subparts or subtypes.)

{Depth-identification / Depth-attributive

{Particular Illustration / Evidence}

{Comparison; Analogy} }+

{Attributive / Explanation / Analogy}

Figure 5.8.9. Constituency schema after McKeown, 1985 (in [PARIS 89]).

The second discourse strategy, process trace, is a step by step methodology for description and is summarised in figure 5.8.10:

[For each object, given a chain of causal links]

(1) Follow the next causal link

(2) Mention an important side link

(3) {Give attributive information about a part just introduced}

(4) {Follow the subsets if there are any. (These subsets can be omitted for brevity.)}

(5) Go back to (1)

[This process can be repeated for each subpart of the object]

Figure 5.8.10. Process trace (in [PARIS 89]).

The two strategies are interlinked by decision points, figure 5.8.11 and 5.8.12.

---

2 McKeown's notation as cited in [PARIS 89]: "{}" indicates optionally,"/" indicates alternatives, "+" indicates that the item may appear 1 or more times, and "*" indicates that the item may appear 0 or more times. Finally, ";" is used to indicate that the propositions could not clearly be classified as corresponding to one predicate.

Identification (introduction of the superordinate)
if the is no local expertise for the superordinate
do a Process Trace (for the superordintate) before proceeding.
Constituency (description of the subparts)
For each part,do:
If there is local expertise about this part (or its superordinate),
do **Depth-identifcation**
Otherwise do a Process Trace (for the part)
Attributive

Figure 5.8.11. Constituency Schema (with decision points) (in [PARIS 89]).

Next causal link
Properties of a part mentioned during the process trace
If a fuller description of the part is desired,
do Constituency Schema (for the part)

Substeps
Back to next causal link

Repeat for each of the subparts:
If there is local expertise about this part (or its superordintate),
do Constituency Schema
Otherwise do a Process Trace.

Figure 5.8.12. Process Trace strategy and its decision points.

Switching between the two strategies is done if the user's expertise is deficient (deemed by the user model), providing more elaborate information if a particular concept is not understood. The switch between process and constituency strategies does not occur if the user lacks knowledge of basic concepts in the process trace description.

Using these two strategies the TAILOR system can automatically produce descriptions to suite user levels falling between the two extremes expert and novice.

In the context of the IFe, these mechanisms could be added to the user handler providing an automated mechanism for generating several alternative descriptions for a particular concept or object. In the TAILOR system adult[3] and junior[4] encyclopedias, manuals[5] and school text books[6] where used as sources for descriptions of the same objects, providing several stylistic differences addressing the needs of the two extremes of user levels [PARIS 89].

3 The New Encyclopedia Britannica, Collier's Encyclopedia
4 Britannica Junior Encyclopedia, The New Book of Knowledge - The Children's Encyclopedia, The Encyclopedia of Science.
5 A Woman's Guide to Fixing the Car, Weissler, A 1973.
6 Elements of Physics, Baker, D.L., Brownlee and Fuller, R.W.

The alternative solution would be to pre-format descriptions or video sequences for each user type and class.

5.9. USER MODELLING

The user determines the users conceptual vocabulary and adjusts the dialogue accordingly by employing re-description and re-phrasing techniques. User modeling in the IFe has not yet been tackled owing to its complex nature, requiring knowledge of user psychology (which is perhaps beyond the current capabilities of the author) however a number of suggestions using the techniques already described are proposed. These have been implemented to various levels of completeness in order to explore their potential. None of the methods have yet been formally introduced into the IFe User Handler.

5.9.1. MONITORING USER ACTIONS

The user model expresses an interest in the dialogue area of the blackboard and in particular monitors user_action and user_query events.

5.9.2. QUERY EVENTS

If a user is constantly asking for help, descriptions, or examples, their user_level is demoted and the level of assistance, in the form of feedback and defaulting is increased.

5.9.3. ACTION EVENTS

The range of user action events currently reported are the:

- selection of concepts (user has focused attention)
- de-selection of a concept (implied by selection of a concept).
- errors in data specification

5.9.4. RESPONSE TIME

The time taken for the user to respond to a particular concept together with the accuracy of the data, may be an indication of how well it is understood.

Rules may be provided to promote or demote the user's level depending upon the speed and accuracy of response.

5.9.5. ERRORS

Continual data specification errors imply that the style of presentation provided and perhaps the interaction mechanism is inappropriately matched to the user's conceptual vocabulary. It may therefore be necessary to employ a local or global re-phrasing mechanism without affecting the users level.

Also if the user is continually selecting "other" from a particular restricted concept list, a free response area may replace the current interpreter.

For example; changing from a popup or menu field to a text field, would provide the user with an unrestricted means of submitting information but (with the dynamic pop-up menu associated with each concept label), figure 5.9.5.1, provide access to pre-defined alternatives or defaulted values.

This is perhaps an indication of the users increasing perception and expertise within the domain; the user model should perhaps consider a promotion of user_level.

```
.....
describe(location,_dialogue_level),
.....
```

```
describe(location, novice):-
        restrict_user(location,locations).
```

```
describe(location, expert):-
        ask_user(location_name),
        focus_user(location,coordinates).
```

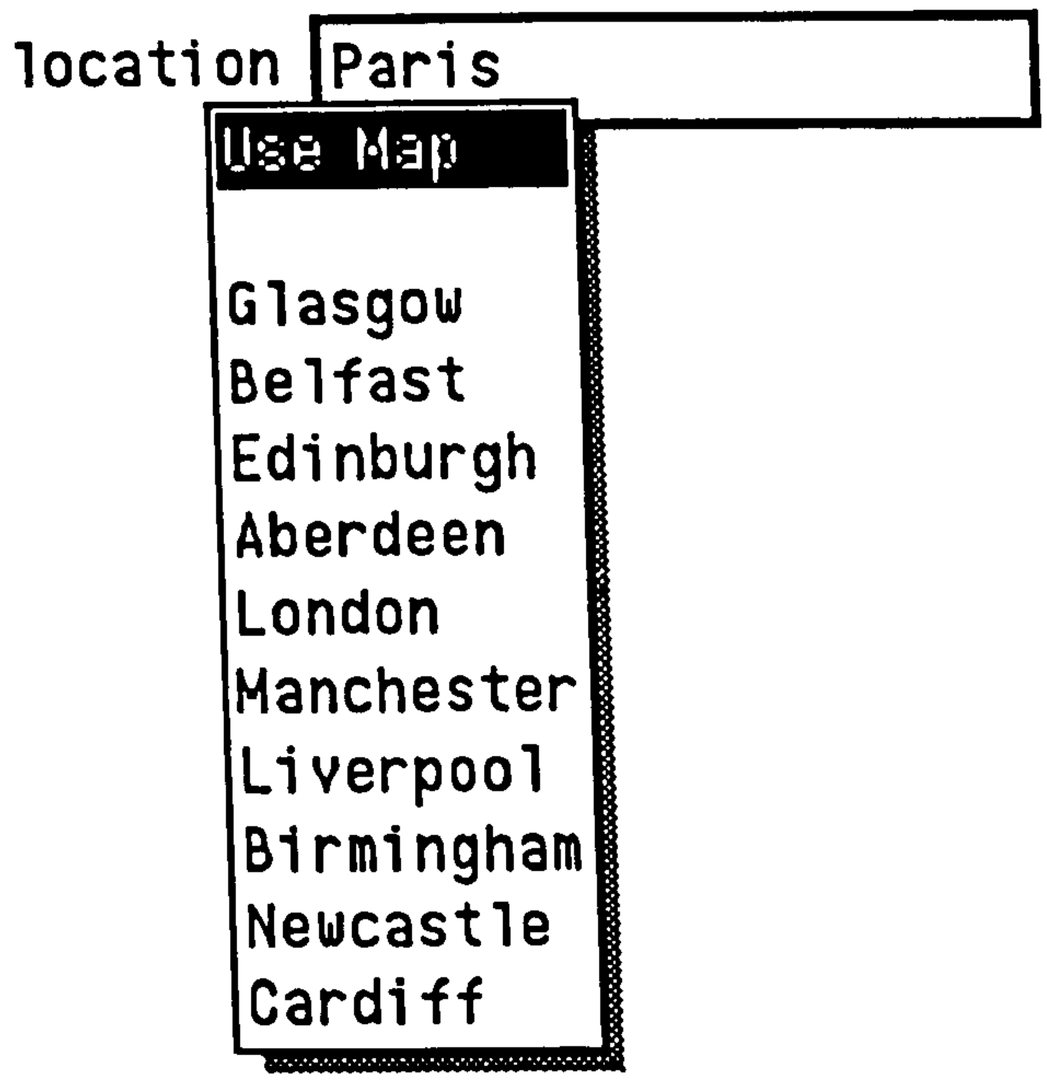


Figure 5.9.5.1. Free response field with defaults

5.9.6. DETERMINING THE USER'S LEVEL OF EXPERIENCE

```
demote_userlevel():
        _user_level  =  novice
        to_kb(user_level,_user_level).
```

The user model sets the user_level which determines which feedback predicates are used in the knowledge base and also suggests a particular style of dialogue.

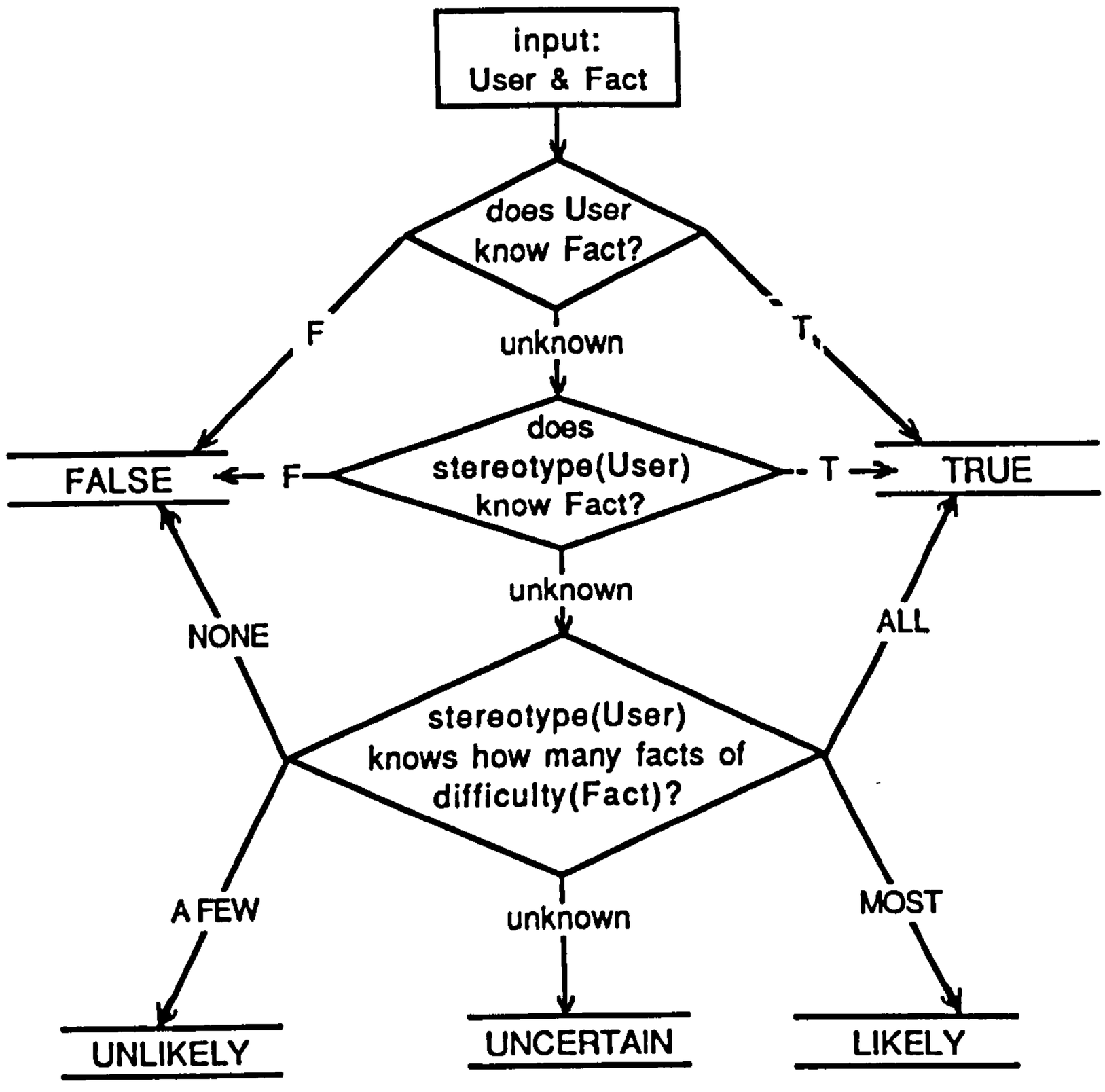


Figure 5.9.6.1. Algorithm for determining whether User knows Fact [CHIN 89].

The algorithm [CHIN 89], illustrated above, may be of use for determining the user's level of experience. It is also of use for establishing what concept should be included in a particular conceptualisation (interface template).

5.10. EMPLOYING APPLICATION PROGRAMS TO SOLVE PROBLEMS

In addition to the issues related to human computer interaction the knowledge handlers must employ more domain specific knowledge to solve specific tasks and must therefore utilise deep model knowledge systems (see chapter 2.6.8). There are a number of issues related to the integration of application software which are tackled in greater depth in chapter 7). However the following section provides a brief introduction to the use of application software for problem solving.

Application programs are used to solve tasks that cannot otherwise be tackled by the surface level knowledge systems of a particular domain. In general surface level knowledge deals with elements of knowledge that arise from human laws

[LANSDOWN 86], while deep level models deal with laws of nature (physical, mathematical models).

A necessary distinction must be made between the application program that the IFe is interfaced to and those applications which are utilised for automated acquisition of information. These will be referred to as; the target *application* and *casual task applications* or utilities such as perspective image generation programs, geometrical modelers, histogram / pie chart presentation programs, database management systems, etc.

A necessary process when constructing a front end is to identify sub-tasks within the domain and identify methods or incidental task application programs which are useful in solving these problems.

5.10.1. TASK SPECIFICATION

Once identified, task application programs must be integrated within the IFe system in such a manner as to preserve the high-level neutral language of the domain knowledge source. This is achieved by encapsulating application program specific information within a task-solution script so that the methods within an application program may be utilised in abstract terms. For example the generation of a perspective image of a geometric model requires detailed mathematical techniques. Rather than expect the knowledge engineer to code an application module to solve this particular problem, an existing software package is employed. Knowledge of how such a package should be used is isolated within a script. For the purpose of generating a perspective image the ABACUS viewer program is used and the task-solution script "perspective" written around it:

```
                # Resource category: perspective_image
                # Design tool: viewer
                # location: /package/abacus/bin
                # instantiation constraints: viewer_model
rm -f /tmp/viewer.pic.
                # If model exists then generate image otherwise abort
if [ -f $1 ]
then
                viewer > /dev/null 2>/dev/null <<.
                -1
                $1
                O
                /tmp/viewer.pic
                B
                .
                echo        "perspective_image    $1        /tmp/viewer.pic        complete"
                .
else
                echo        "perspective image    $1        model_not_found"
fi
```

Figure 5.10.1.1. Perspective image script

Therefore, rather than coding application related information within the surface level knowledge bases a predicate of the form:

generate(perspective_image, _Model_name)

would post a task request to the application handler which in turn would invoke the script perspective. This is only a general illustration. The issues are discussed further in chapter 7.

Owing to the multi-tasking nature of the environment, the task may be solved while the user continues with other issues provided that these issues do not rely upon the results of active task application programs.

Results are fed back to the surface level system in a similarly abstract form. In the case of the perspective image a message of the form:

perspective_image model_name image_name complete

is returned and must be interpreted by a corresponding predicate in the surface model of the form:

```
perspective_image(model,_Image, complete):-
        tell_user(picture, _Image).
```

which would feedback the generated image to an appropriate field on the current form set.

It is important to note that data formats must be compatible. In the case of the viewer image a field interpreter dedicated to displaying viewer images (taking full advantage of inhouse software) was coded within the forms package in order to minimise response time. A totally generic approach was investigated by coding a rasteriser between the vector image format produced by viewer and the portable bitmap representation of the forms package. However, as perspective images of complex models may contain anything up to and over a thousand vectors per image the speed penalty incurred by effectively drawing the image twice (once by viewer) was too great. In some instances the physical properties of the display device or region may be required by a resource in order to correctly format images. Although it is currently possible to achieve this using a simple properties request the approach of X and NeWS is perhaps a little more general whereby, in addition to widget

properties being accessible, parametric graphical objects are passed. These are automatically adjusted to suite the display area.

The example illustrated is a simple one and made easier by the non-graphical (terminal based) interface of ABACUS software. A number of applications require decision to be made by the user, based upon a current state solution, before a final solution can be formulated. A secondary mechanism is provide to interact directly with an application (discussed further in chapter 7):

```
modify(perspective_image      eyepoint 25.3, 10.0, 1.7)
```

Results from task solution application programs may also be posted directly to the data area to be retrieved later by the target application.

5.11. SUPPORTING DIGRESSION

The oportunistic nature of the system closely follows the dynamic fluid nature of human communication; with mechanisms that enable concepts to be addressed out of context. By adopting the protocols suggested the IFe allows the user to digress into other related or perhaps unrelated areas, either to confirm results or explore other possibilities. With respect to design procedures the protocols for concept communication enable the designer to follow less rigid courses (top-down or bottom up approaches) towards solution synthesis and mix modes within the same problem space. This obviously results in an extremely flexible environment.

5.12. AN EXAMPLE

The techniques and mechanisms described have been used in the formulation of a front-end for building performance prediction, described in chapter 6.

6. THE APPLICATION OF THE IFE TO BUILDING PERFORMANCE ASSESMENT AND PREDICTION

6. THE APPLICATION OF THE IFE TO BUILDING PERFORMANCE ASSESSMENT AND PREDICTION

The objectives of the IFe where to develop an environment capable of acting as an expert consultant in the field of computer aided building performance modelling to assist a designer, by offering intelligent defaults, on line assistance and guidance, in the problem description stage. By encoding the conceptual mapping between the user's domain and that of a performance prediction package, by formatting it's data requirements, the IFe is capable of interfacing complex modelling systems between a broad spectrum of users. In the field of building performance assessment and prediction the following three stereotypes where targeted [IFE 89]:

6.1. DESIGNER

The designer, dealing with the early tentative stages of design, characterised by high level, abstract concepts, is envisaged as operating with partial design solutions and incomplete data. From a design tool's viewpoint this information is likely to be fuzzy requiring parameterised defaults. The needs of this user category may also be satisfied by offering a general sense of the performance of a number of alternative design solutions (highlighting problematic areas), useful for guiding the decision making process.

6.2. ENGINEER

Once an overall design solution has been generated, a wealth of specific, detailed information is available. Although complementary to the above designer stereotype, objectives are more clearly defined, the task of this category of user will be to establish a more precise building performance prediction influencing the choice of building components. Traditional design tools match these requirements well. However, the role of the IFe in this instance is to isolate the user from the obtuse operational complexity of the specific design tools.

6.3. MODELLER

The modeller is envisaged as being a proficient user requiring almost direct access to the design methodologies encoded within the application. The only help required would be straightforward assistance with data preparation; providing access to standard databases and offering sensible default values to minimise data input.

6.4. EXPERIENCE LEVEL

In addition to the three types of user a further sub classification is made, placing each user type into the categories of expert and novice. The terms are used in the relation to the operation of complex simulation tools and does not refer to the user's ability in their own field where they may be expert. These two extremes may be bridged by beginner and intermediate [CHIN 89] classes. Novice users require comprehensive assistance and guidance facilitated by continuous (verbose) feedback, while expert users require a different type of (terse) feedback.

6.5. A CONCEPTUAL MODEL FOR BUILDING DESCRIPTION

Although three user stereotypes where identified the constraints of time permitted the development of a single user conceptualisation only. The following screen images represent a typical interaction session following an engineering stereotype.

6.6. MASTER FORM

In order to establish a context for discussion all users are presented with a master form, the template description for which is given in Appendix C. The following screen images[1] illustrate a typical dialogue session with a user.

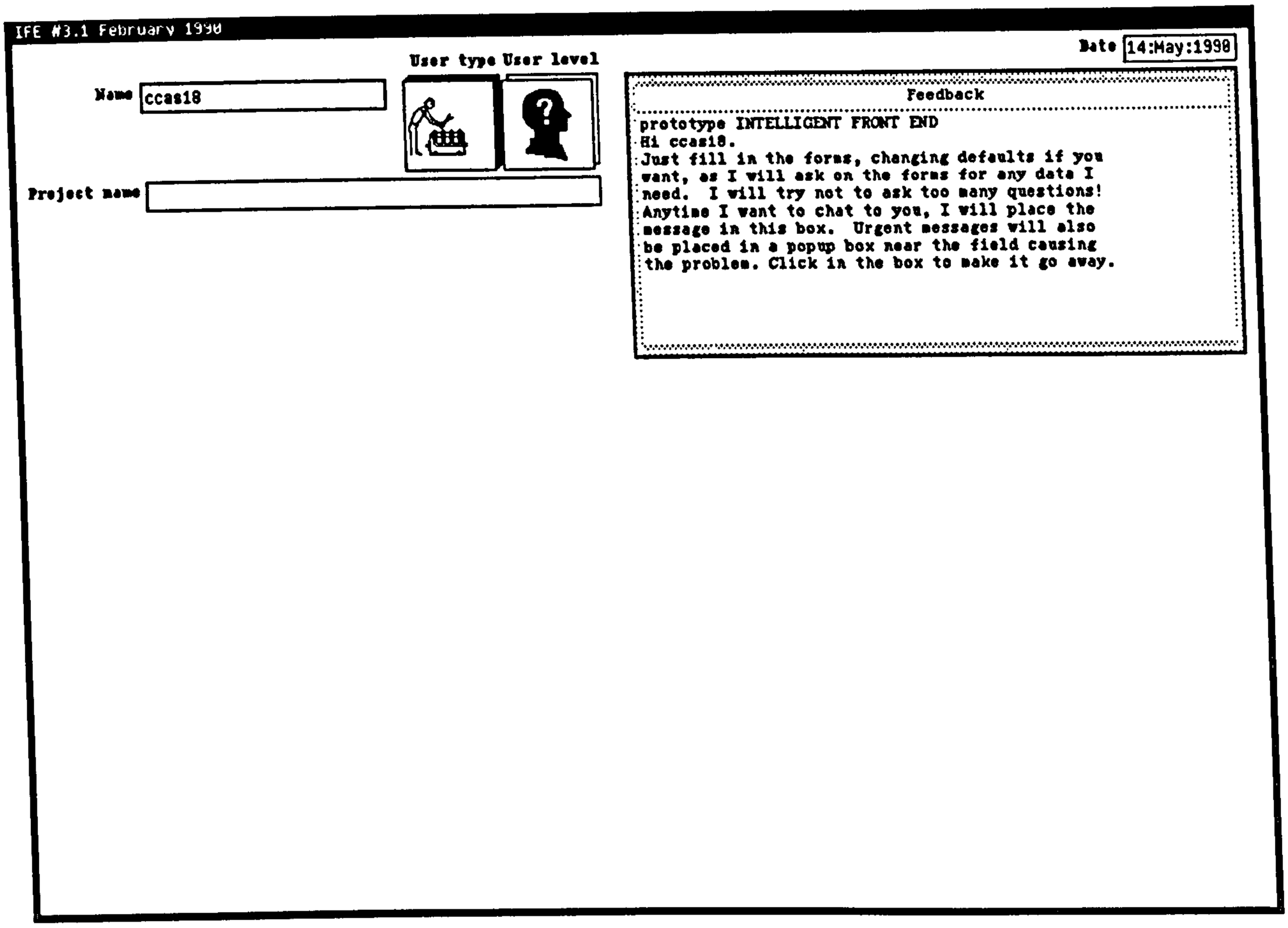


Figure 6.6.1. Master form: initial settings.

The master form, figure 6.6.1, consists of a number of concepts specifically related to the user; in particular: the users name, the currently supposed stereotype and the the users level of experience. Note that although the concept of user_type and user_level are handled by the knowledge bases, the user's perception should be one of occupation and level of assistance since asking the user to provide a self-critical evaluation of his/her abilities may result in inaccurate responses.

---

1 The screen images displayed are sun rasters which are produced at a higher resolution than is possible to manipulate with a Macintosh. In order to accommodate the images within the format of this thesis and maintain a degree of clarity the Sun desktop has been removed using a paint package eliminating the need to photo-reduce the images.

The current date is displayed at the top right of the master form together with a large feedback area. All general messages (in the form of assistance and warnings) are directed to this area. This provides the user with a familiar response region. Information regarding specific concepts may be displayed along side those areas by means of a pop-up window.

On initailisation the user model interrogates the user's environment to determine the user's name. The user's level and type may be inferred from previous records produced by the user model, illustrated below:

```
user_level(damian, expert).   /* defaults */
user_level(joe, expert).
user_level(james, expert).
user_level(_User, novice).    /* everyone else is a novice */
user_type(damian, modeller).
user_type(joe, engineer).
user_type(james, architect).
user_type(_User, engineer).  /* everyone else is an engineer */
```

In this instance the IFe has picked up the user name ccas18. No record of such a user exists so a novice engineer is assumed, figure 6.6.1.

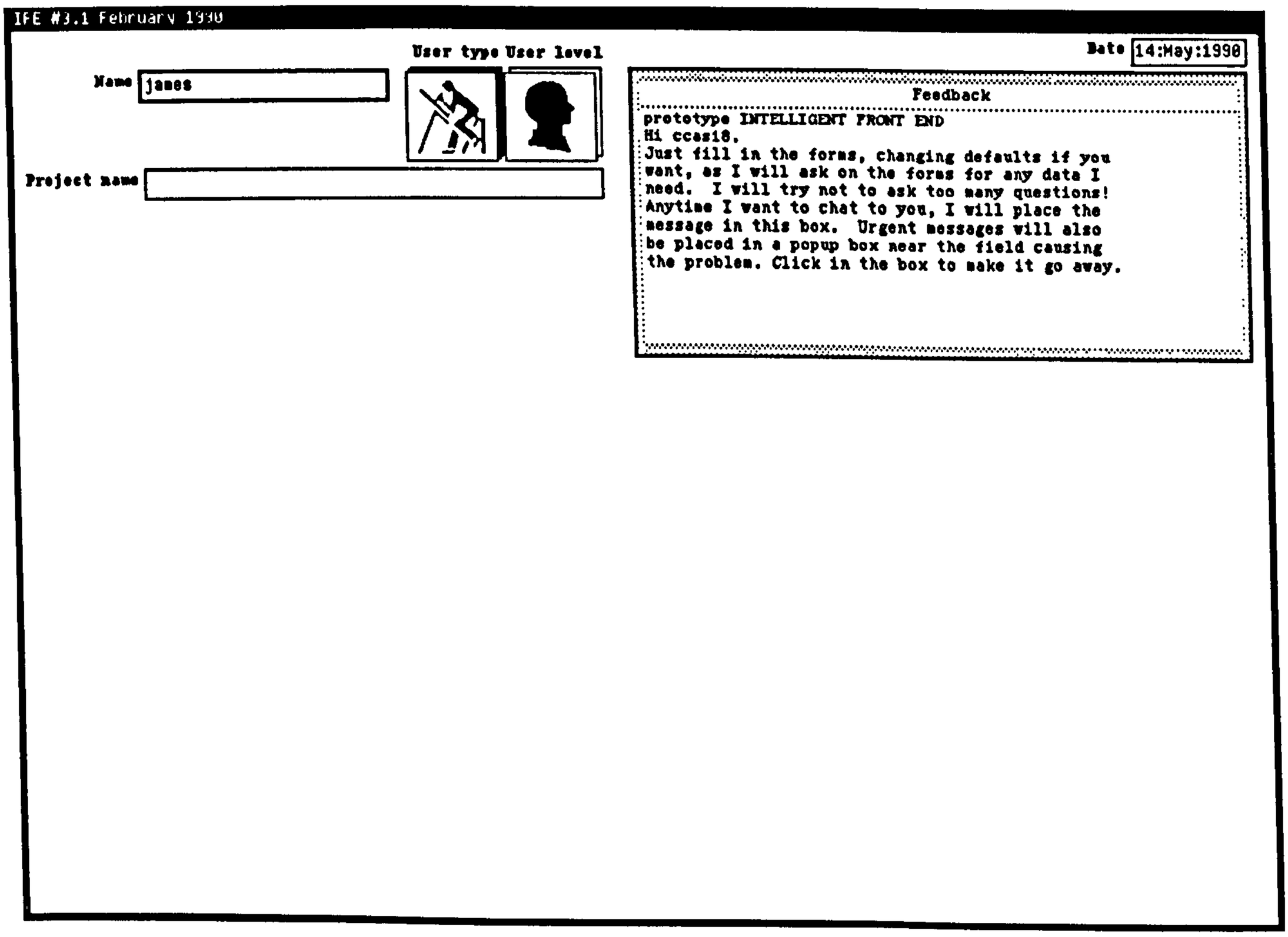


Figure 6.6.2. Master form: Name field changed. The corresponding user type and level updated accordingly.

At any time the user is free to change the values set by the knowledge bases as illustrated in screen images 6.6.2 and 6.6.3.

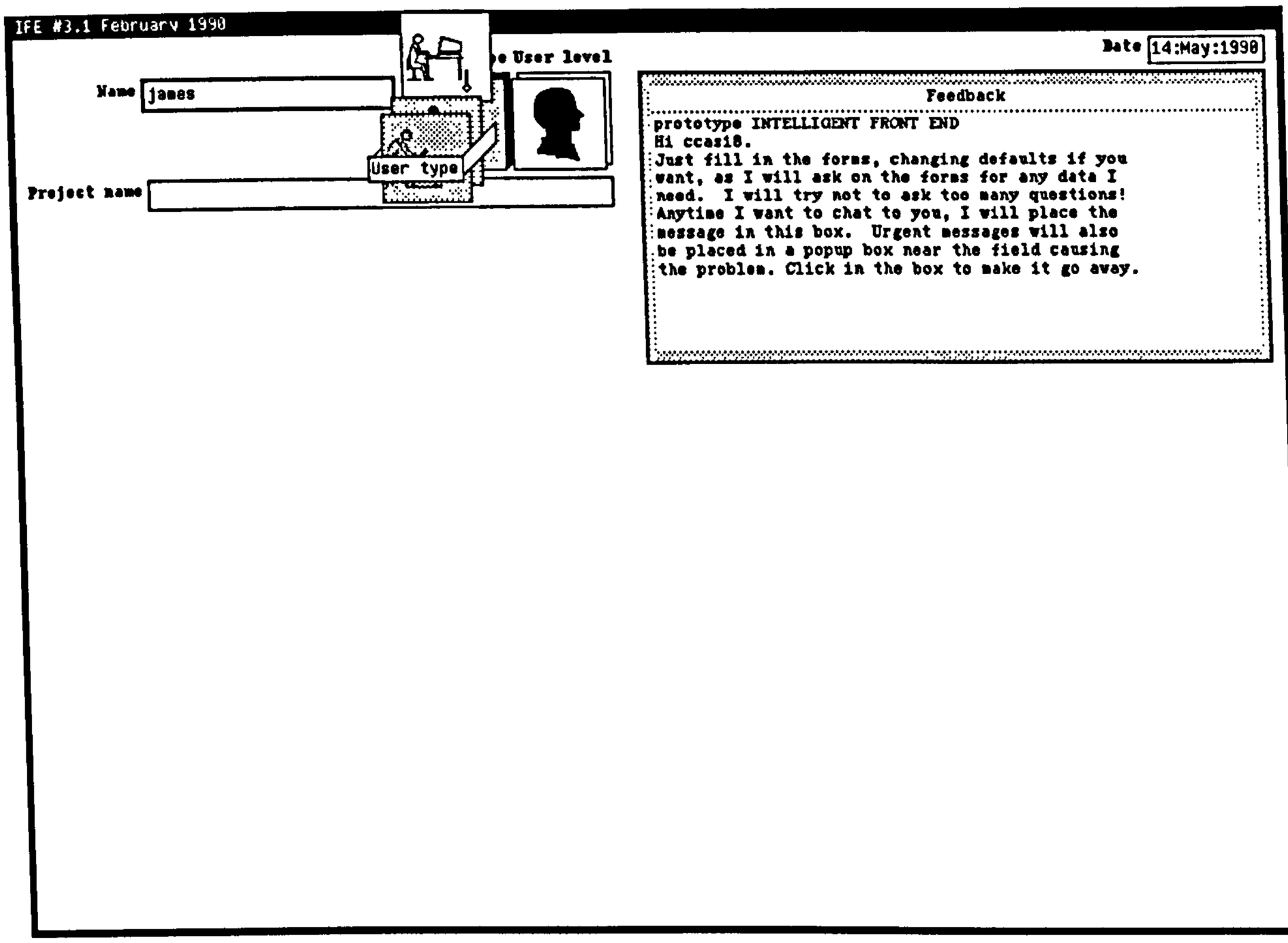


Figure 6.6.3. Switching between conceptual models.

Screen image (Figure) 6.6.2 shows the results of changing the value in the name field. Both user type and level are updated to architect and expert. Figure 6.2.3, illustrates how, using a cascading pop-up field, the user is able to switch between conceptual models. For the purpose of this example the user level is demoted to novice to illustrate the feedback mechanism (feedback for expert users is rather terse).

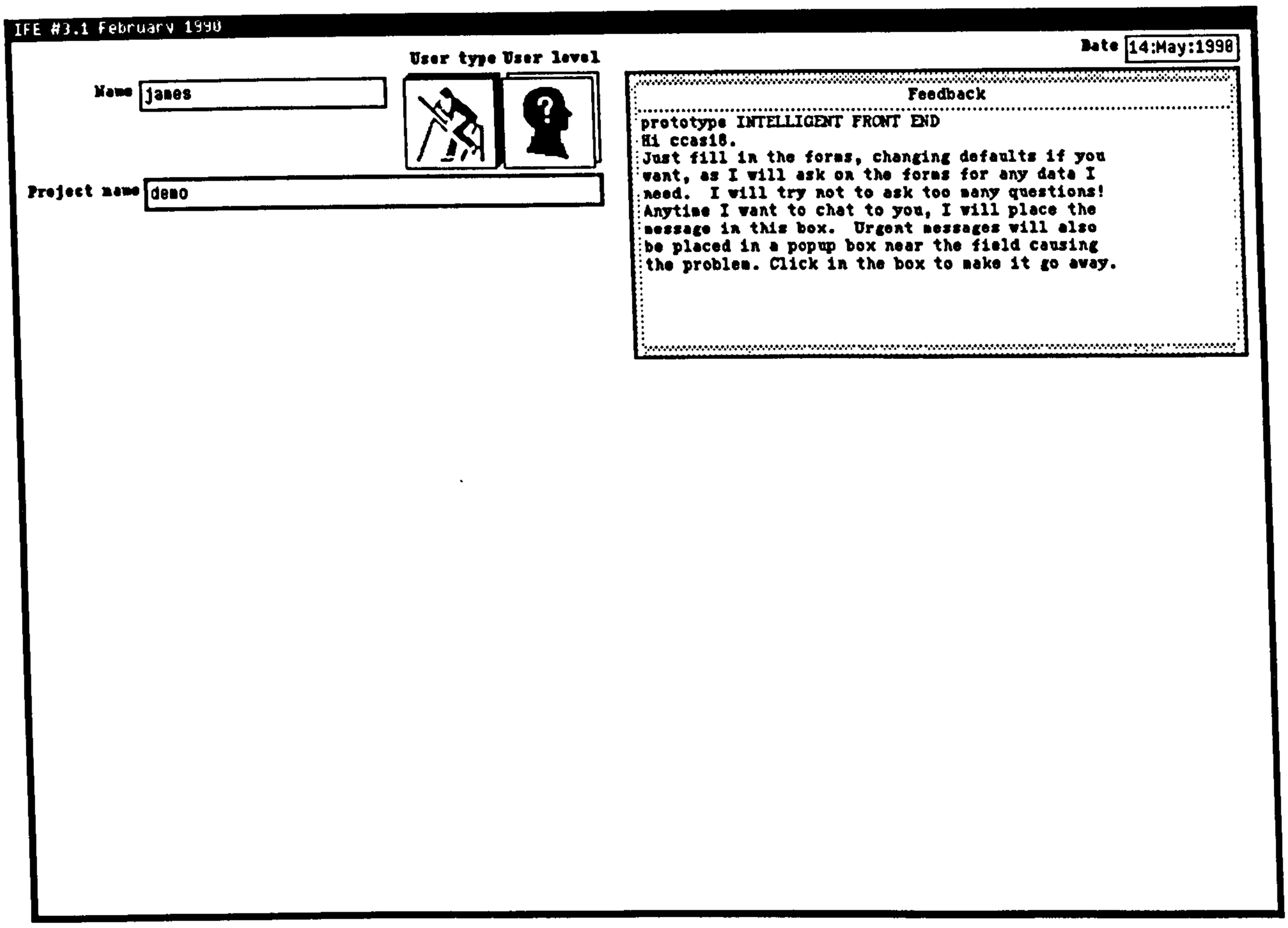


Figure 6.6.4. Entering the project name.

The only domain specific concept presented to the user, at this point in the dialogue, is that of a project identification name. This is used by the data handler to either create a new project area or restore a previous model, figure 6.6.4. The predicate proj_exists is used for this purpose.

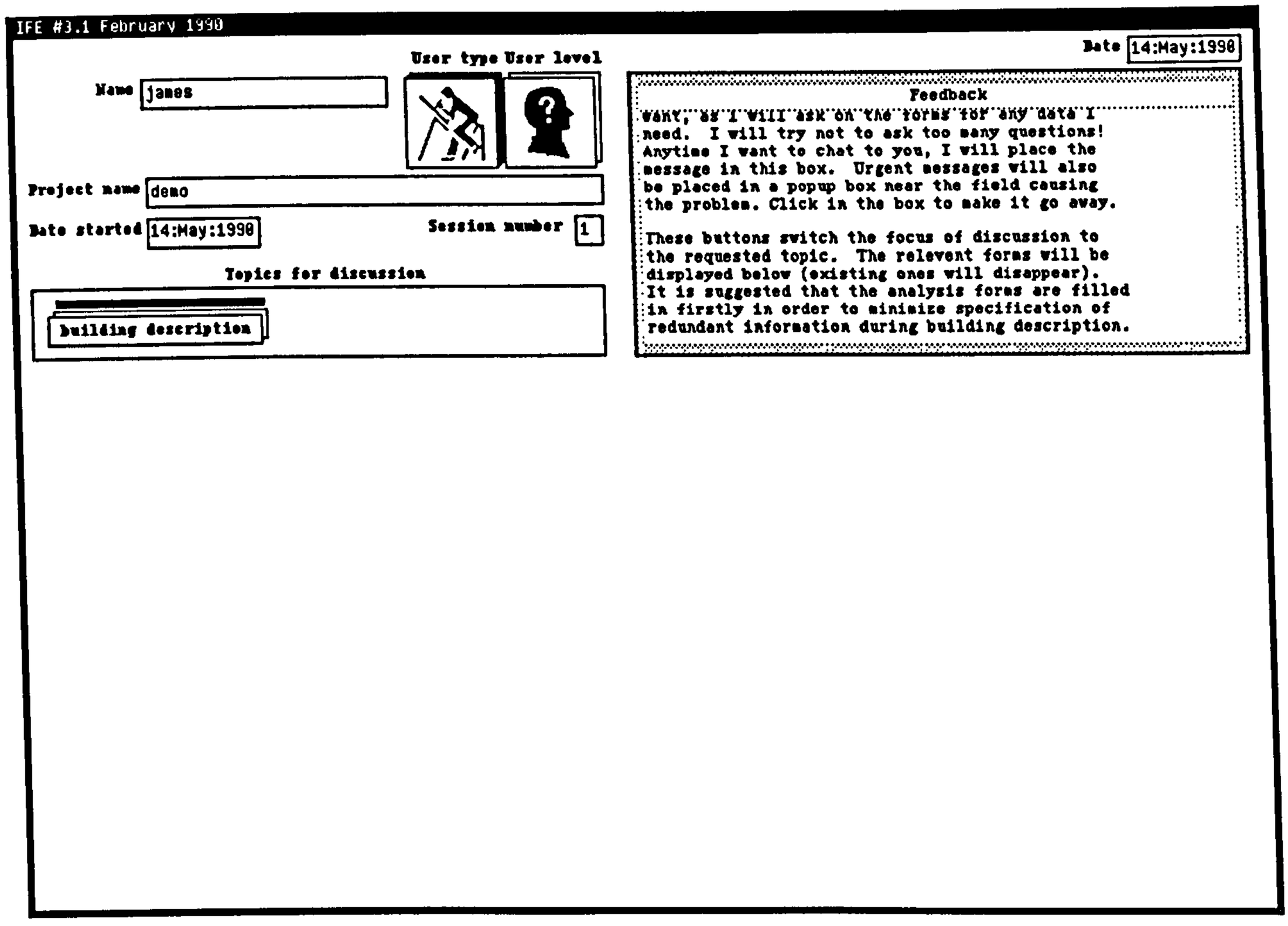


Figure 6.6.5. Topics for discussion.

Once entered and checked a secondary set of concepts is presented, figure 6.6.5. These include general concepts about the current project (date started and dialogue session; useful for retrieving previous sessions) together with dialogue options. Currently two dialogue options are provided:

i. analysis which is only presented if there is a sufficiently complete product description and

ii. a building description button.

In this instance it is assumed that the description is incomplete (the project does not exist) and that the relevant focus for discussion is that of the building description. Although this option is provided in the form of a button, it is suggested that the sub category of building description be selected automatically in situations of incomplete product descriptions.

As the user selects sub topics, both knowledge bases and proforma templates relating to those domains of discourse are dynamically loaded by the focus_concept predicate, chapter 5.6.2.

6.7. BUILDING DESCRIPTION - CONTEXTUAL INFORMATION

A basic contextual description of the building is addressed in terms of its location, function, and environment.

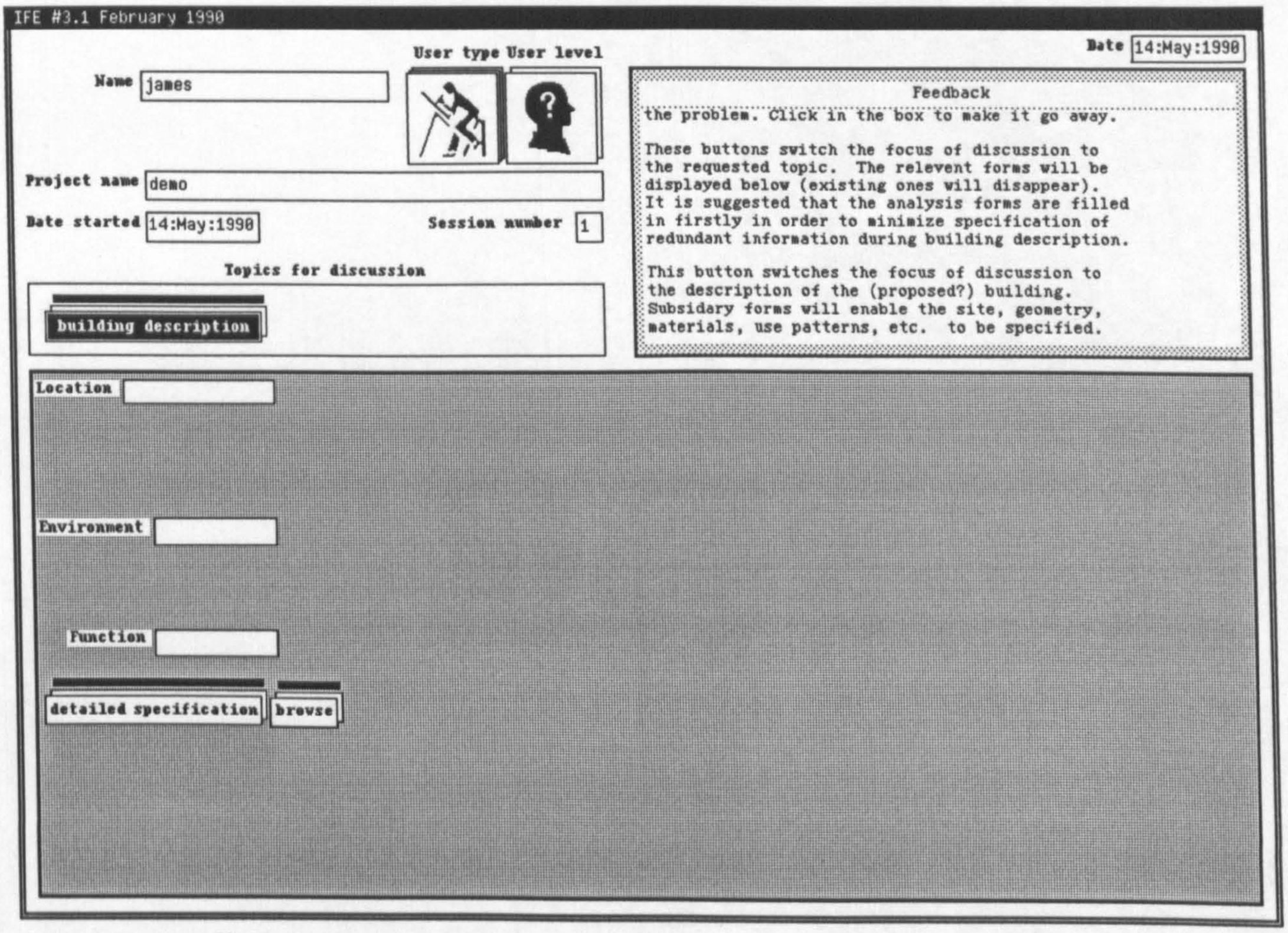


Figure 6.7.1.1. Site location default concept menu.

As illustrated in figure 6.7.1.1, the building description form contains generic contextual information such as the building's location, environment and function together with mechanisms for providing more specific detailed descriptions. Each will now be taken in turn.

6.7.2. SITE LOCATION - AN EXAMPLE OF RE-DESCRIPTION AND RE-PHRASING

The site location is required in order to determine the content of the default values offered to the user. In particular alternative sites, and standard climate files used during simulation.

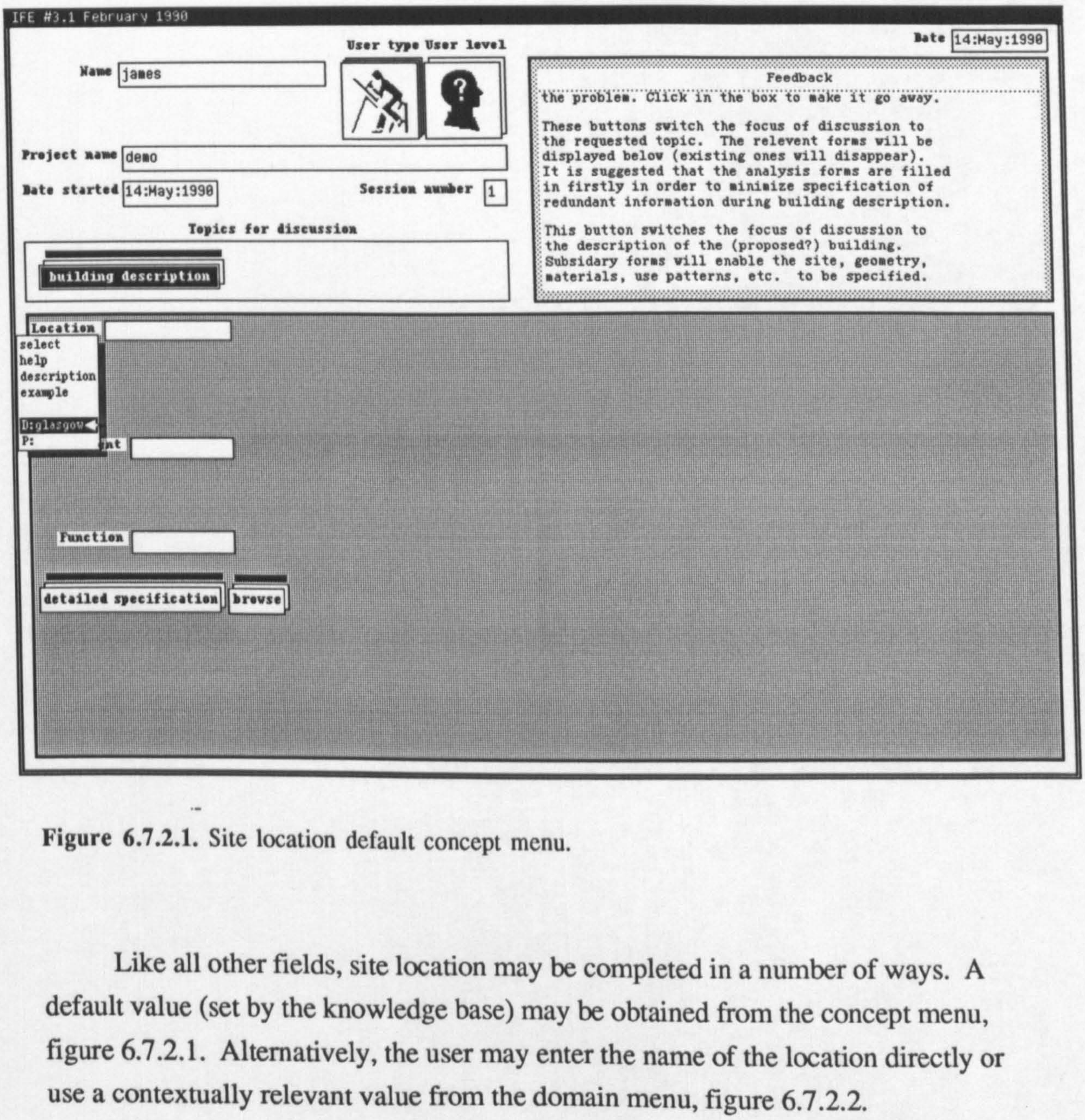


Figure 6.7.2.1. Site location default concept menu.

Like all other fields, site location may be completed in a number of ways. A default value (set by the knowledge base) may be obtained from the concept menu, figure 6.7.2.1. Alternatively, the user may enter the name of the location directly or use a contextually relevant value from the domain menu, figure 6.7.2.2.

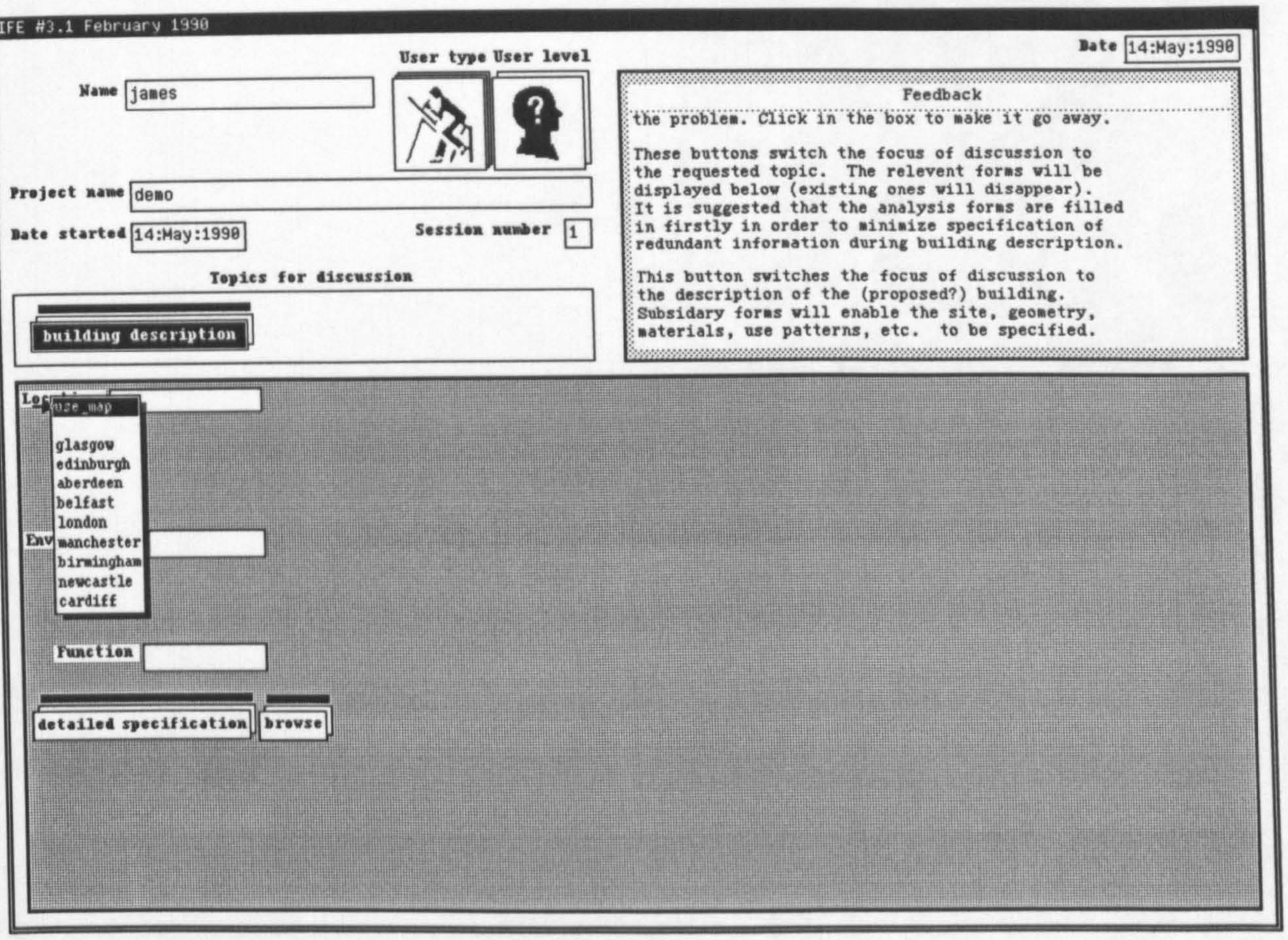


Figure 6.7.2.2. Location contextually relevant domain defaults.

This menu, initialised by the knowledge base, Appendix D (bld_spec), contains a special value; "user map", which provides a mechanism for the user to ask for another input method. In this case a separate resource (or new dialogue) is stated. The process is a simple map program, displaying an bitmaped image of a particular part of the world, figure 6.7.2.3. The user may select a city by moving the cross-hair cursor over the desired location and pressing a mouse button. The coordinates are passed out as:

user_said location x y.

This is interpreted by the location predicate, Appendix D.

At the same instant that the map program is invoked, a more explicit description of site location, in terms of its two elemental components: latitude and longitude, is displayed. This is perhaps how an expert user would specify the site location of the building rather than giving a general location name.

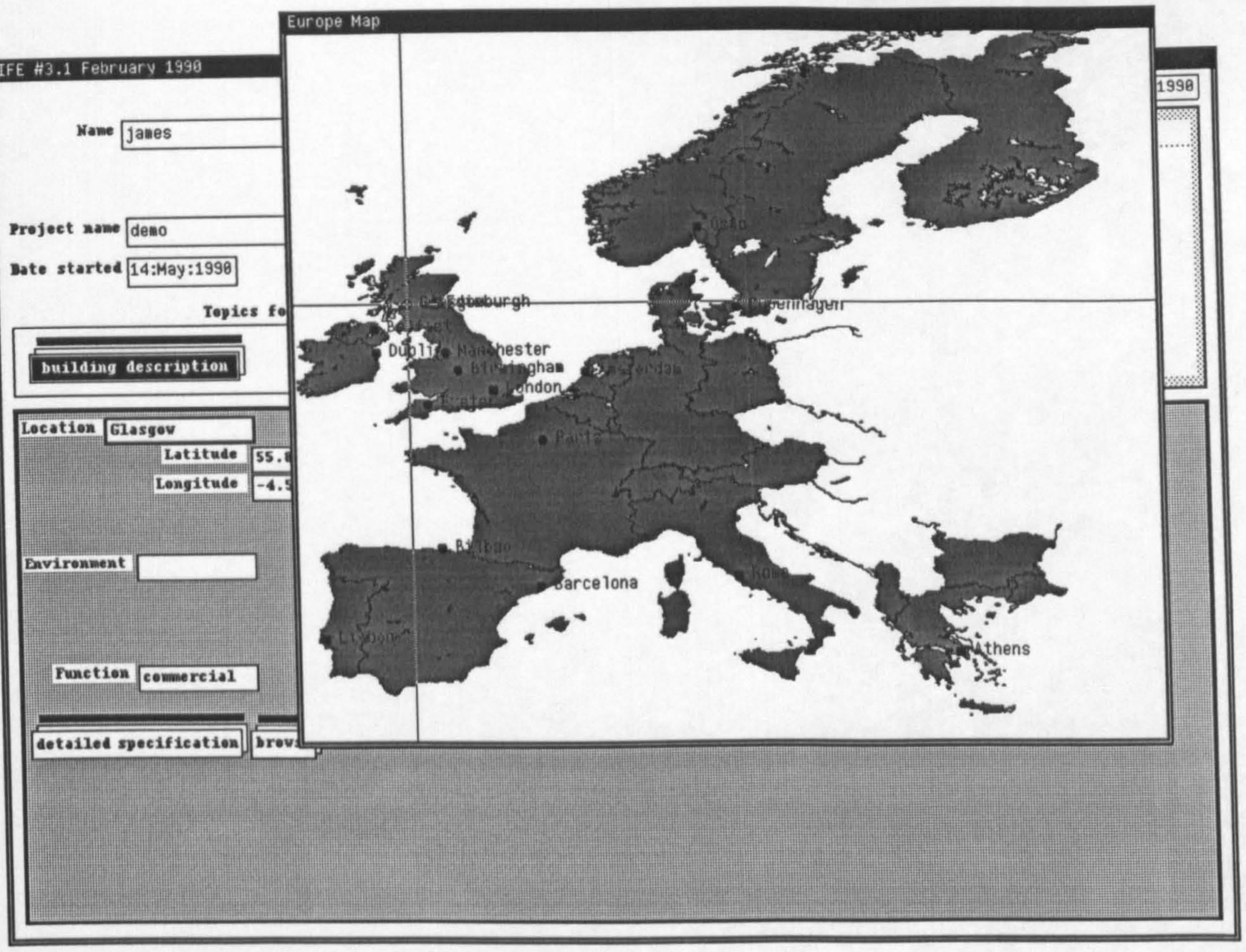


Figure 6.7.2.3. Location re-phrashed (graphically) and re-described.

When the "use_map" option is selected the map is displayed directly over the concept of location. This is done to obscure the normal description of the concept and force the user to address it's value by making a positive selection. Although it is possible for the user to move the window out of the way and enter the location name or coordinate value, the map program remains visible until a selection is made. This is perhaps an undesirable mode locked operation. It is currently only possible to interact with the map program directly. A more complete and consistent form of interaction would enable the user to move the cross-hair cursor and monitor changes in the latitude, longitude and location name fields, figure 6.7.2.3. Likewise the user should be able to enter partial information, such as the latitude and the cursor should snap to the location.

If an unknown location is entered or selected from the map the user is asked to provide a complete description (ie name or latitude and longitude, whichever is required). The knowledge base then stores this new piece of knowledge in a user defaults file. This is only done if the user is thought (by the user model) to be sufficiently knowledgeable.

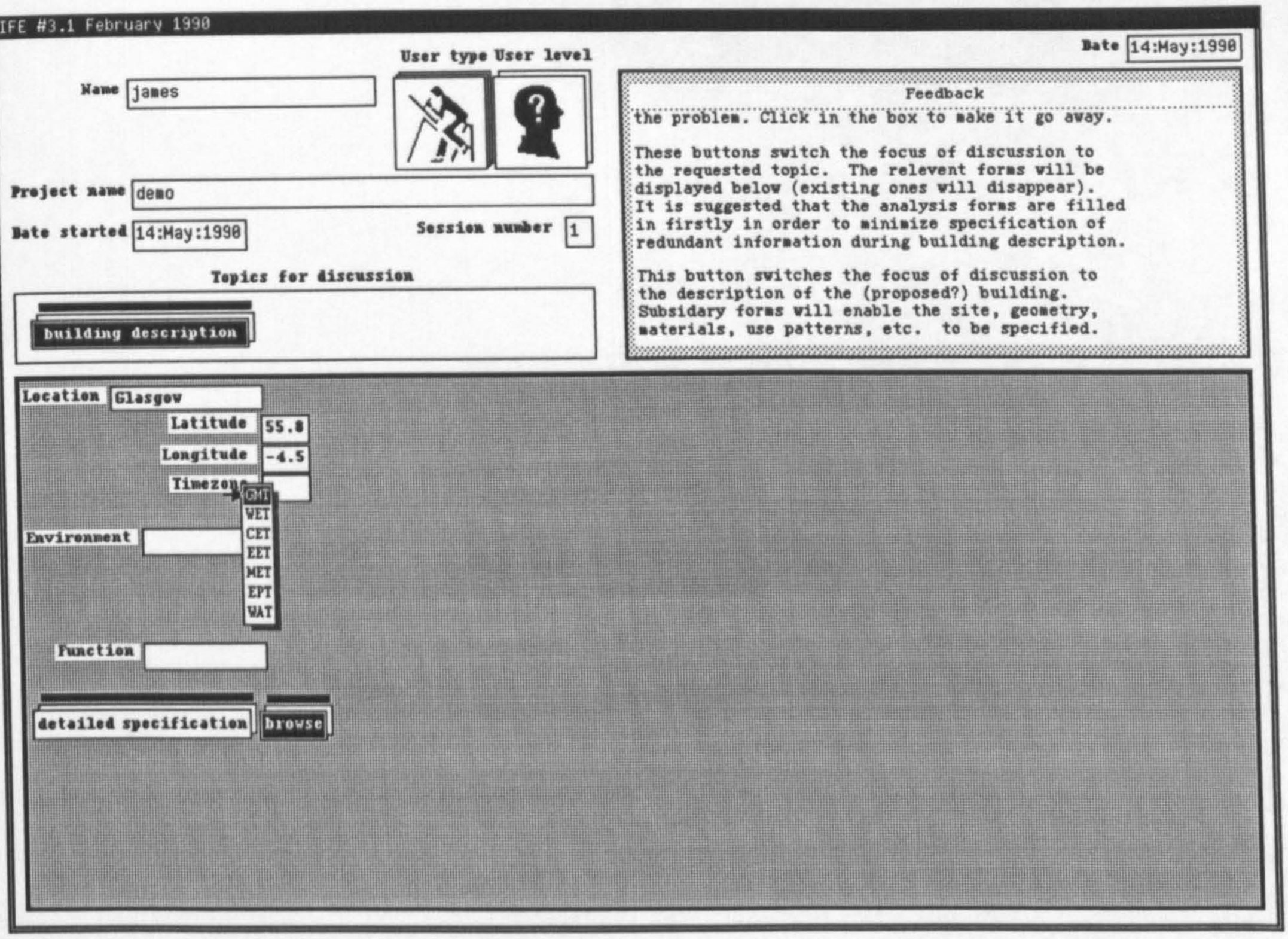


Figure 6.7.2.4. Time zone.

The actual choice of location names (major cities in the UK), provided on the domain menu and the countries displayed by the map program are determined by a further piece of information (time zone) not normally displayed.

```
bld_spec(initialize) :- %% now addressing building specification cpts
        getenv('IFE_LOC',_Loc),                    %% init location menu, depending on
                                                   %% users location
        ( _Loc = uk,
            offer_usr(location, 'use_map
         glasgow edinburgh aberdeen belfast london manchester birmingham newcastle cardif
        f' )
        ;
          _Loc = eur,
            offer_usr(location,  'london paris bonn rome madrid'  )
        ),

        feedback(bld_spec_sel).
```

The time zone, figure 6.7.2.4, is set by the system clock and the site variable IFE_ENVIRONMENT which is set (to EUROPE) at installation. The corresponding piece of prolog (MacRandal) is listed above.

If the time zone cannot be established the concept is presented to the user. The map program would use a world map and capital cities set on the domain menu.

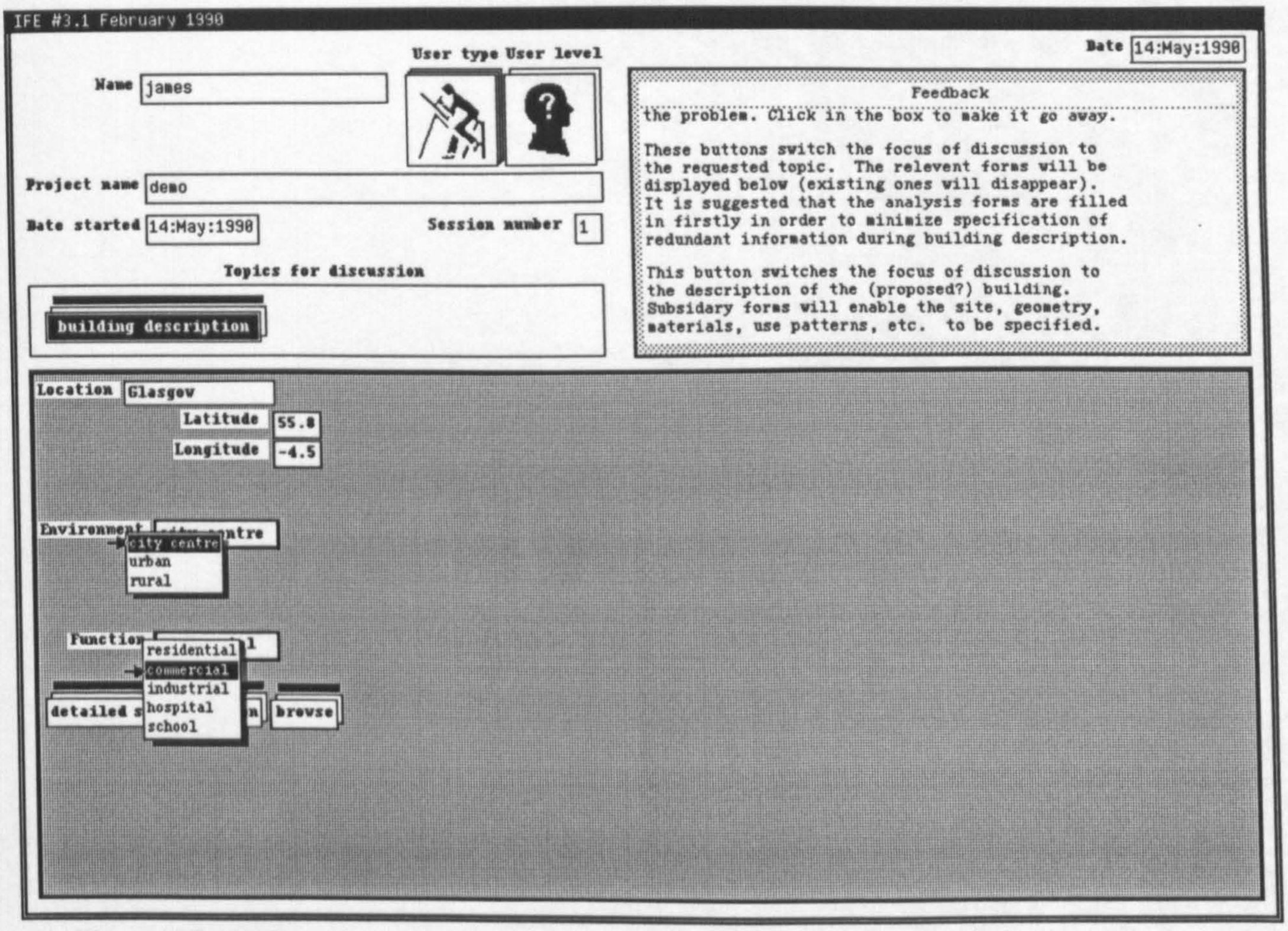


Figure 6.7.2.5. Domain menus for building environment and function.

Both the building's environment and function may be completed from their respective domain menus, figure 6.7.2.5, or in the case of an expert user the site exposure and ground reflectivity may be entered directly, figure 6.7.2.6.

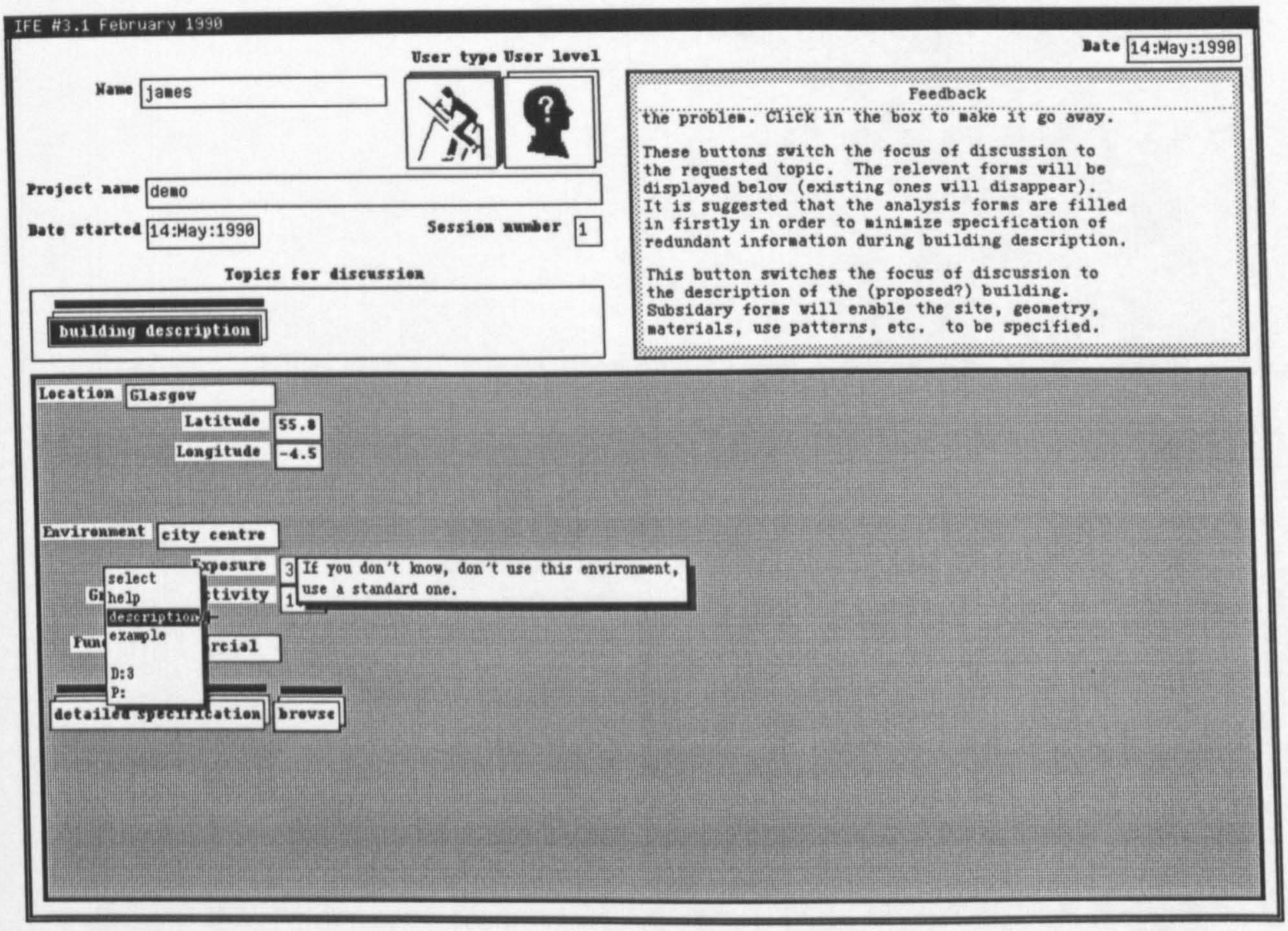


Figure 6.7.2.6. Expert description of building environment in terms of site exposure index and ground reflectivity.

The values set for the building's environment function category, determine which climate data files and occupancy regimes are used by the simulation package.

Although concepts are presented in a procedural manner, it is important to note that the order in which information is volunteered by the user is irrelevant as far as the knowledge bases are concerned. Therefore the user is free to describe any aspect of the building in any order.

6.8. SPECIFIC PROJECT RELATED INFORMATION

In order to perform an energy simulation for a particular building a more detailed description of the building is required. The relevant building description issues related to energy analysis and simulation are accessible from a sub-form activated by selecting the detailed specification option, figure 6.8.1.

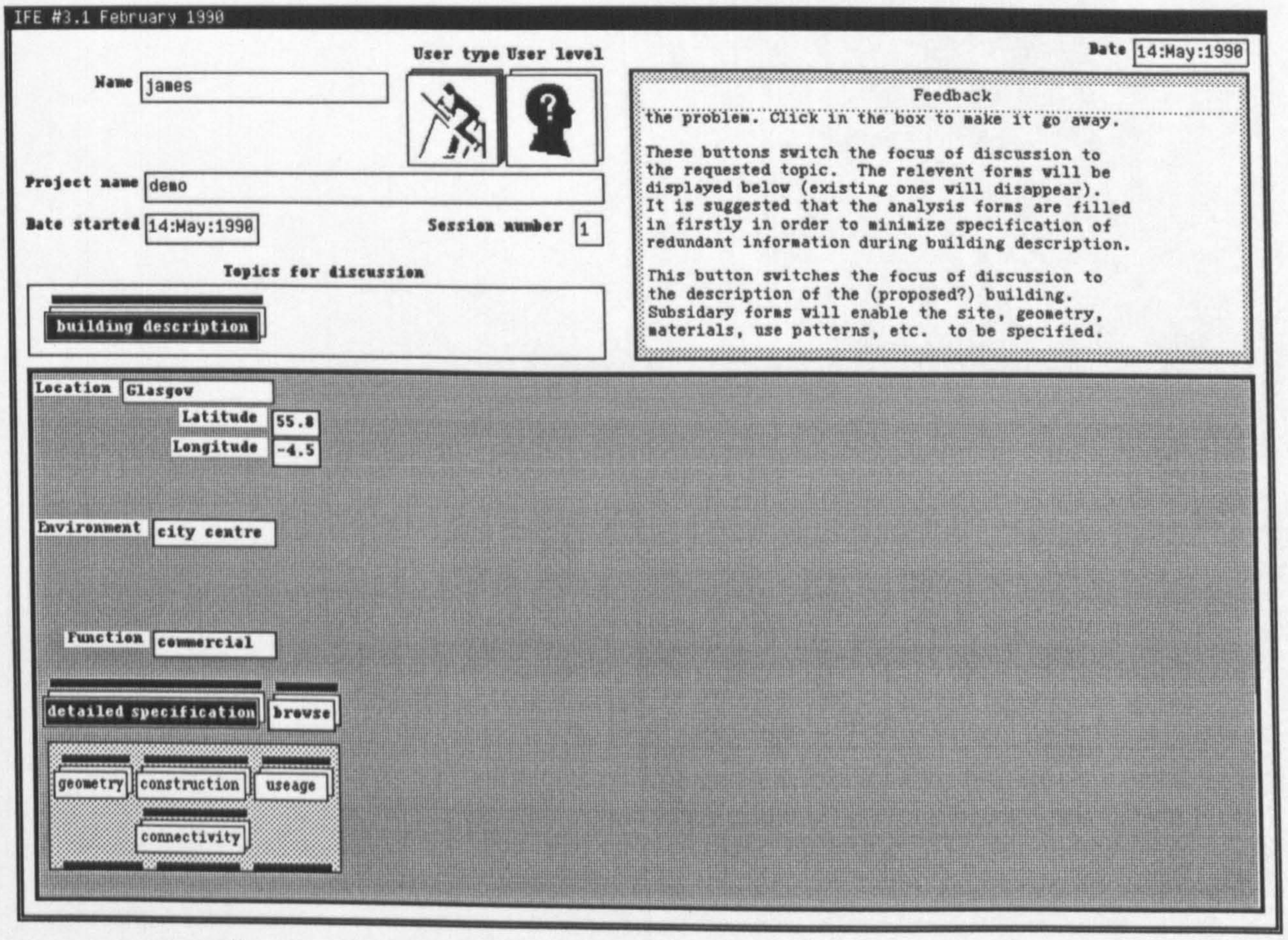


Figure 6.8.1. Detailed building specification.

This building specification focii form, illustrated in its entirety, in figure 6.8.2, provides access to detailed specification forms.

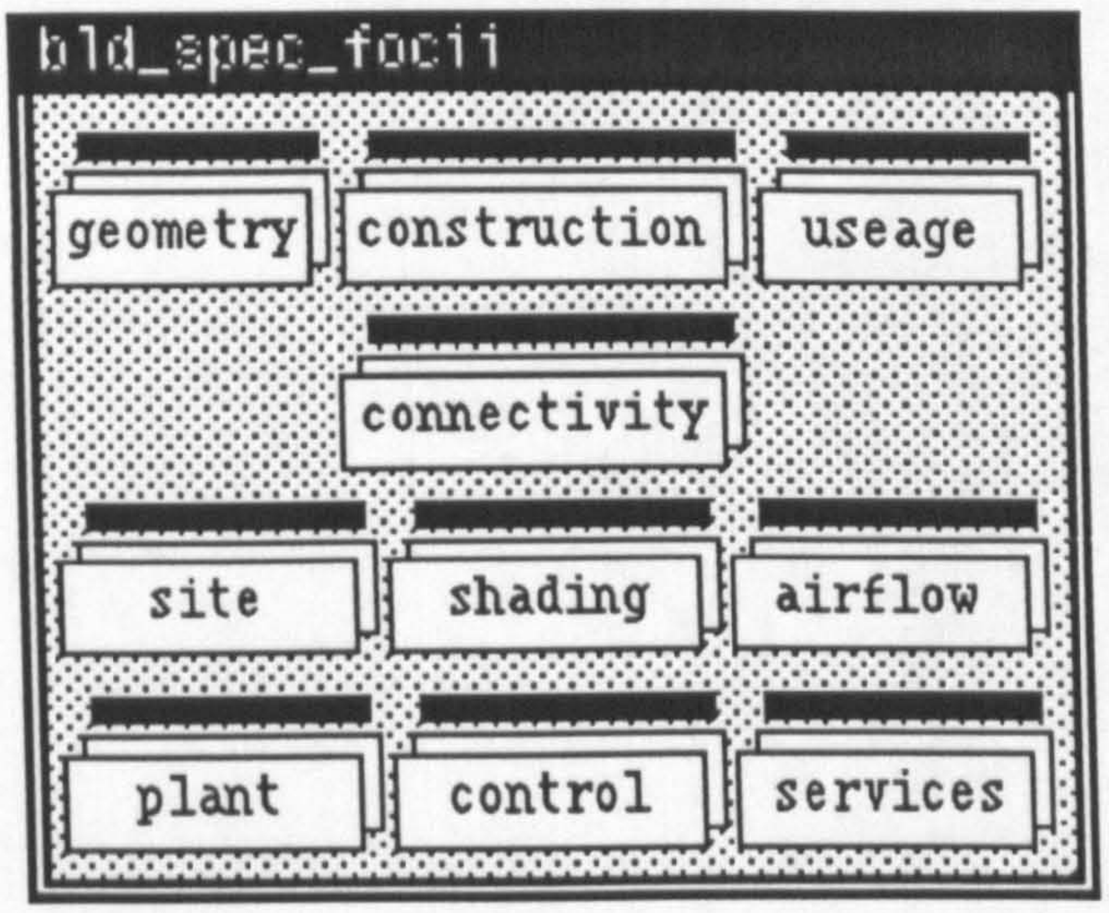


Figure 6.8.2. Building specification focii form.

Fundamental to most building descriptions are: a geometrical description, materials specification, and zone connectivity. The other topics presented are of relevance to building simulation.

Although a comprehensive set of mechanisms are suggested, it has only been possible to provide conceptual schemes for geometry and construction. Further research and funding is required to explore the other issues.

Owing to the highly modular structure of the system it is very easy to introduce these and other knowledge bases.

6.8.1. GEOMETRY SPECIFICATION

In response to the idiosyncratic nature of communication a number of alternative methods of inputting geometrical data have been provided, figure 6.8.1.1:

Import,

Draw,

Form fill

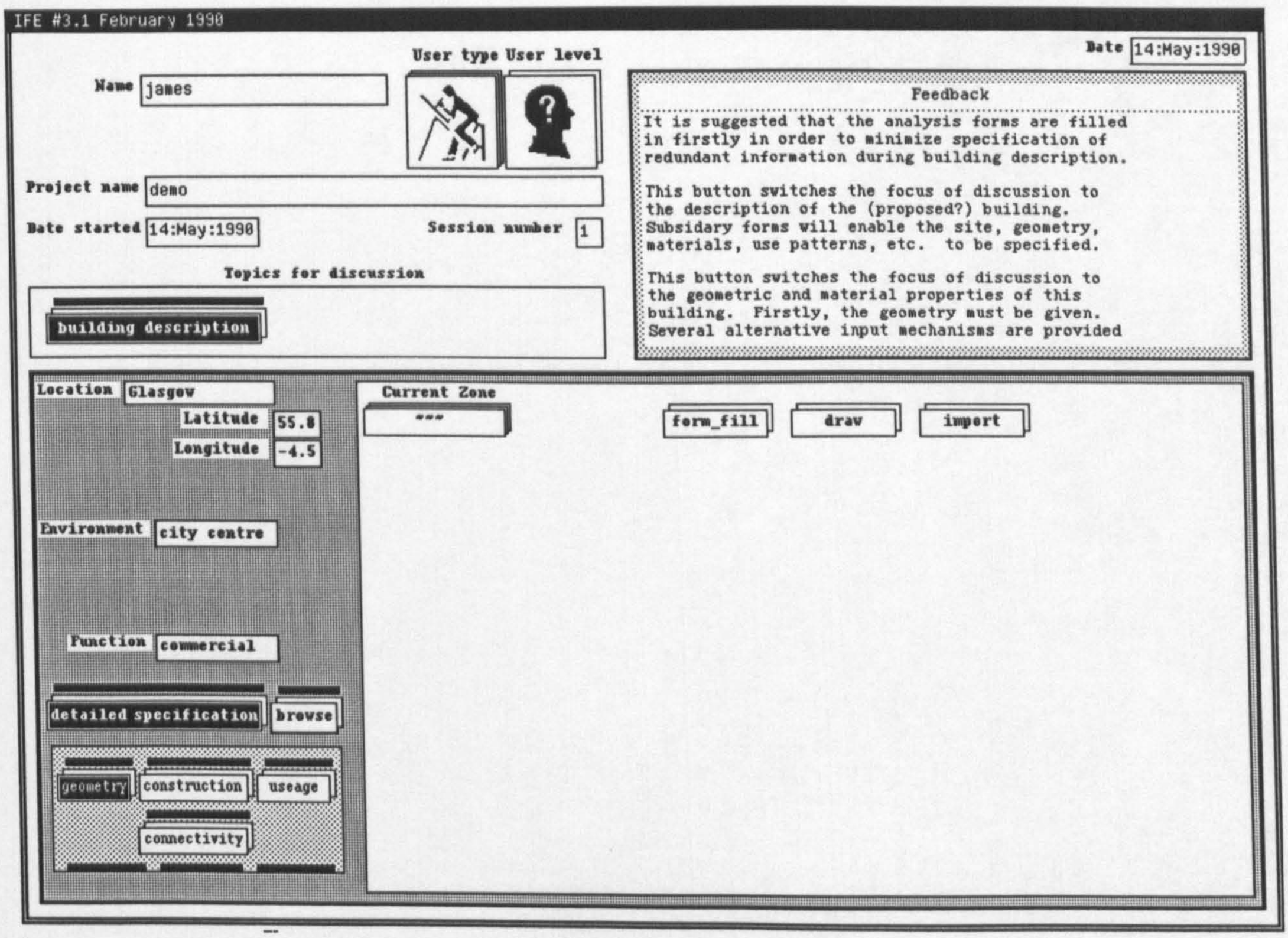


Figure 6.8.1.1. Geometry description (root) form.

Regardless of which method is employed by the user, all data is converter and stored in a neutral format. This facilitates cross migration of the product model and enables the user to interact with it in any manner. This system also enables the user to supplement one method of geometry input with another.

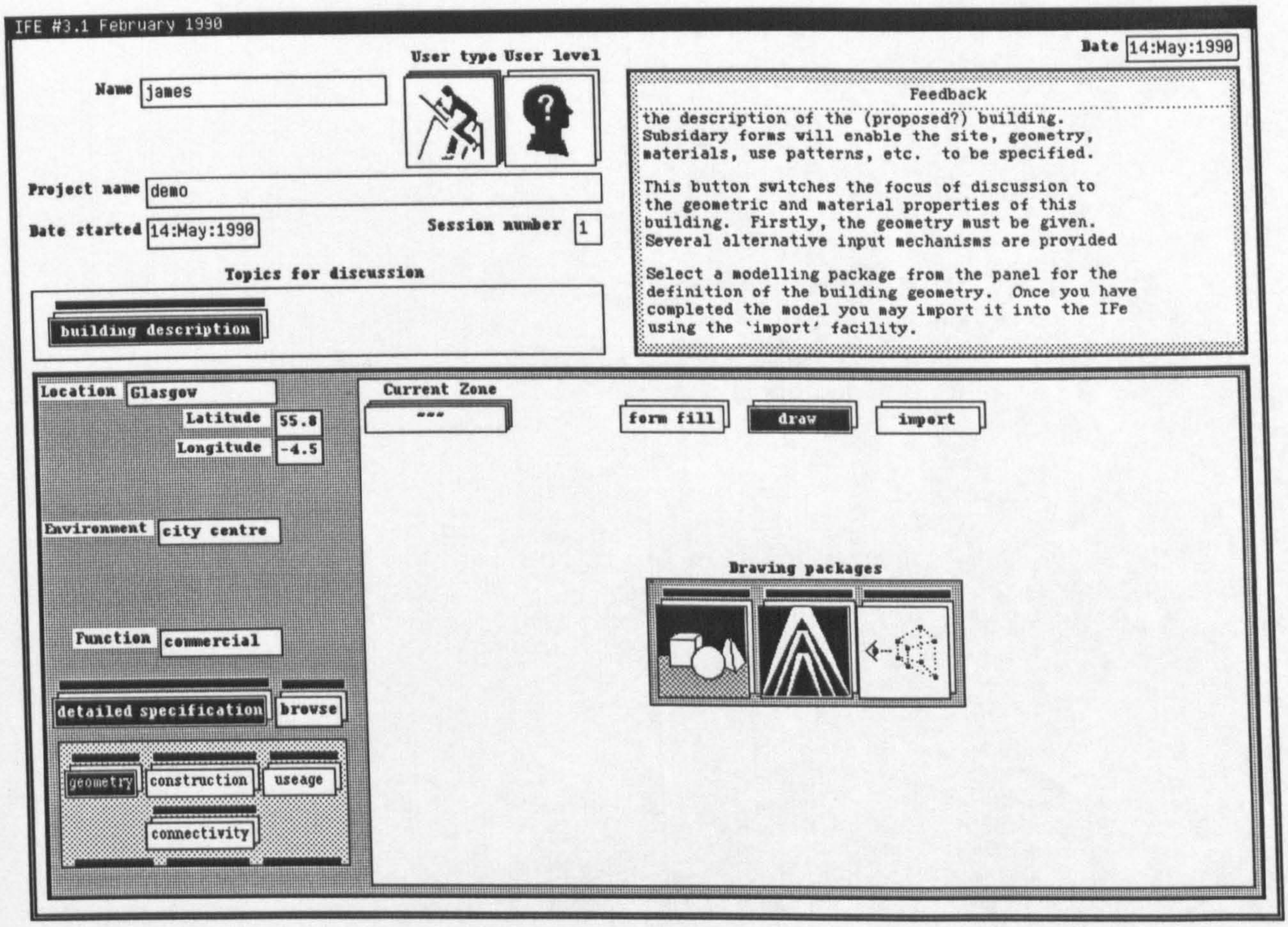


Figure 6.8.1.2. CAD file form.

The draw option presents a number of alternative packages for entering geometry, figure 6.8.1.2. This enables the user to define a building using a familiar system such as autocad or VIM (Viewer Input Management).

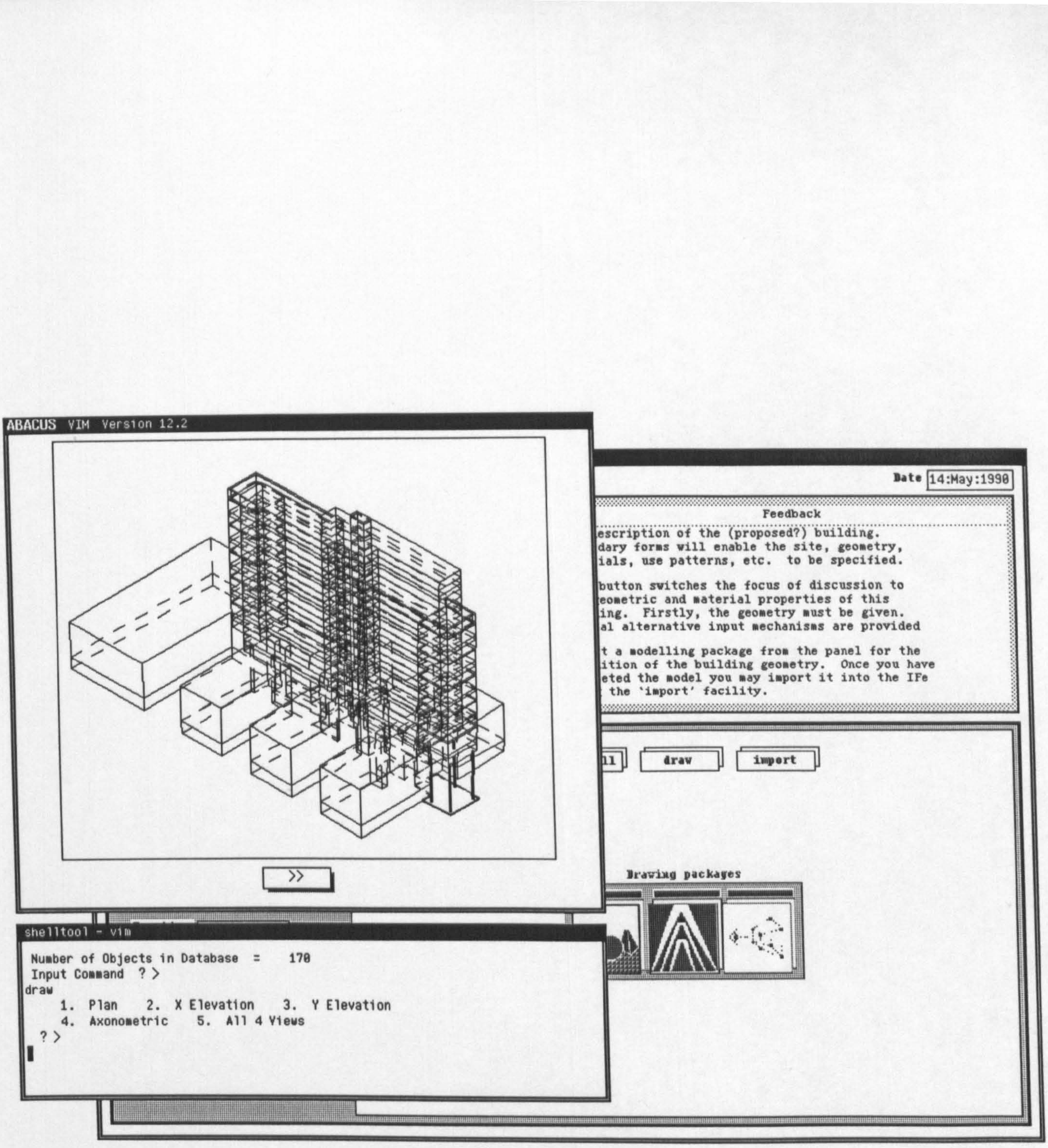


Figure 6.8.1.3. Utilising a 3rd party CAD package for the definition of building geometry.

Once a package has been selected from the scrolling form, the panel is de-focused and the draw option de-selected, figure 6.8.1.3.

As it is often not possible to integrate such packages directly into the IFe owing to the availability of source code, an *import* mechanism is provided, figure 6.8.1.4.

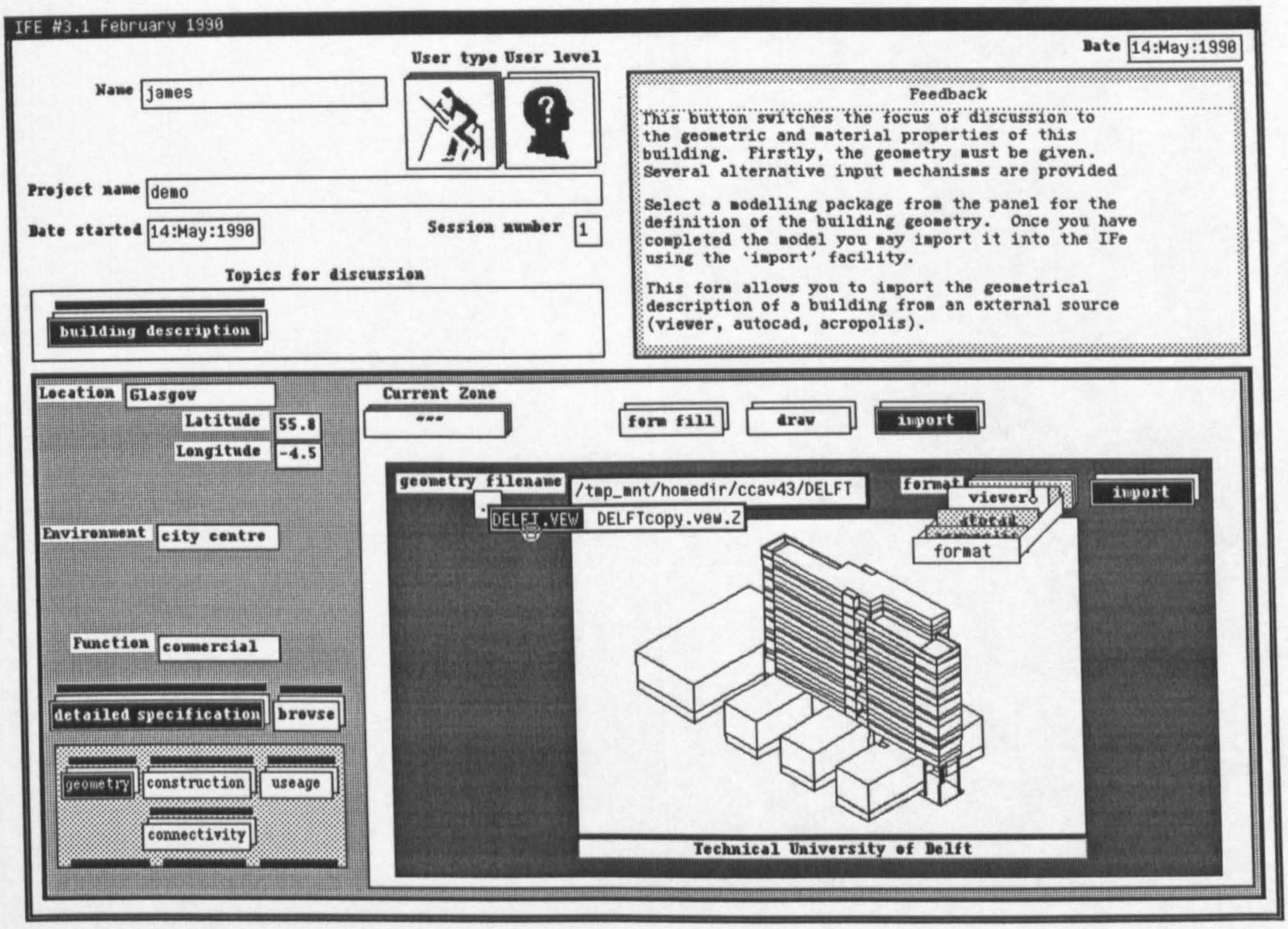


Figure 6.8.1.4. Geometry import form.

This form asks for the name of an existing geometrical model file (produced by another package). In addition to the name of the model file (which may be obtained using a file scanning field) the user is required to enter the geometry representation format. Currently only the ABACUS viewer definition is supported although a dxf to viewer conversion filter (Charles Chen, ABACUS) accommodates most of the primitives generated by Autocad[1]. It is envisage that the use of "magic numbers' in the headers of these files will be used in later versions to automatically select the conversion filter.

Once converted the in-house ABACUS viewer program is invoked as an incidental resource (using the solution plan perspective_image) to provide a perspective image of the model. This is useful as a visual selection aid. Once confirmed the model may be imported using the "import" button, figure 6.8.1.4.

---

1 A View to DXF has also been coded [RUTHERFORD 90].

perspective image of the model. This is useful as a visual selection aid. Once confirmed the model may be imported using the "import" button, figure 6.8.1.4.

The form fill option provides a number of basic editing facilities for creating and modifying geometry imported from other systems. Three basic representations are supported:

i) RECtilinear, figure .6.8.1.5,

ii) REGular extrusions, figure 6.8.1.6, and

iii) GENeral topographical and topological descriptions, figure 6.8.1.7.

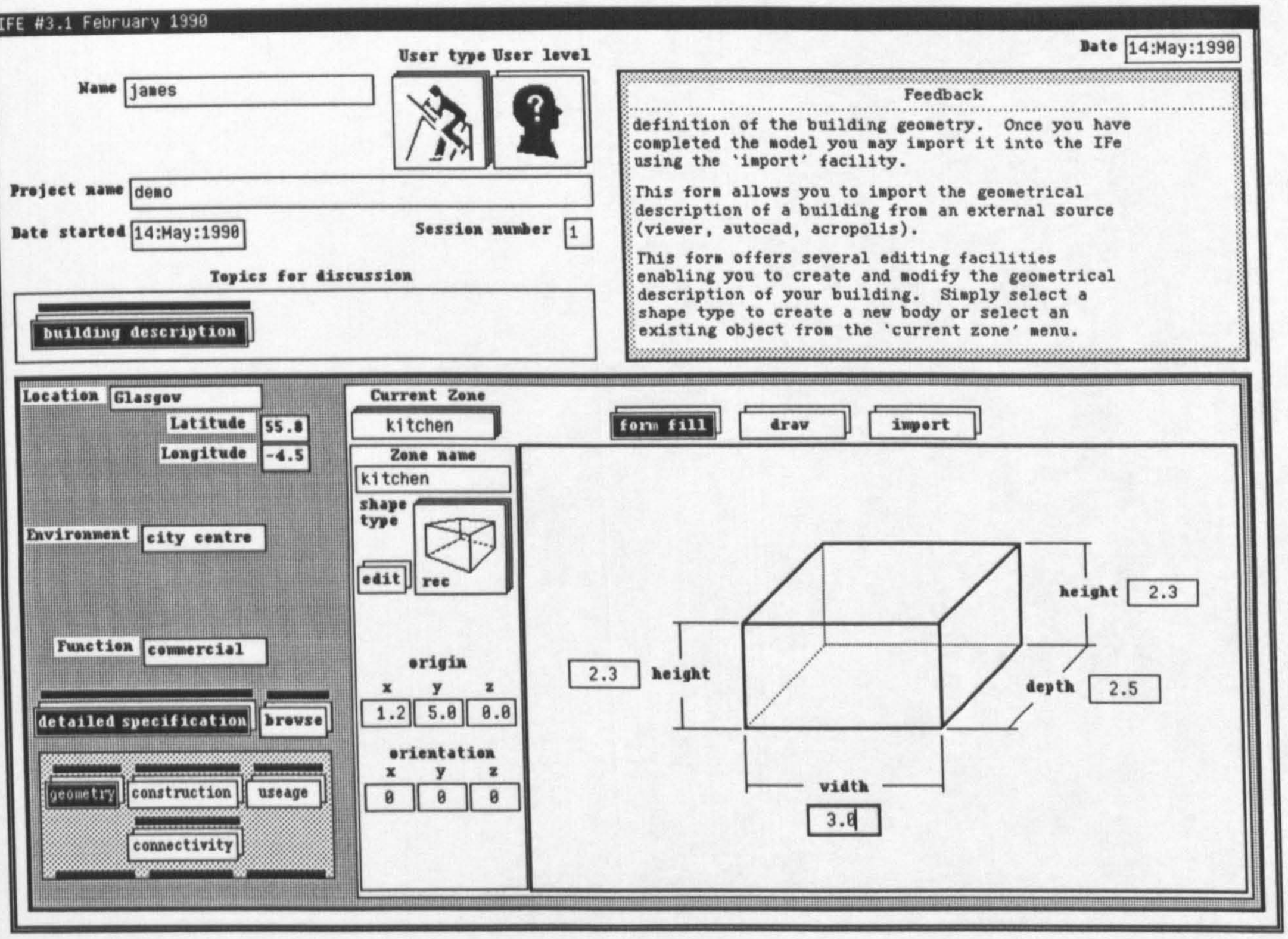


Figure 6.8.1.5. REC instance body.

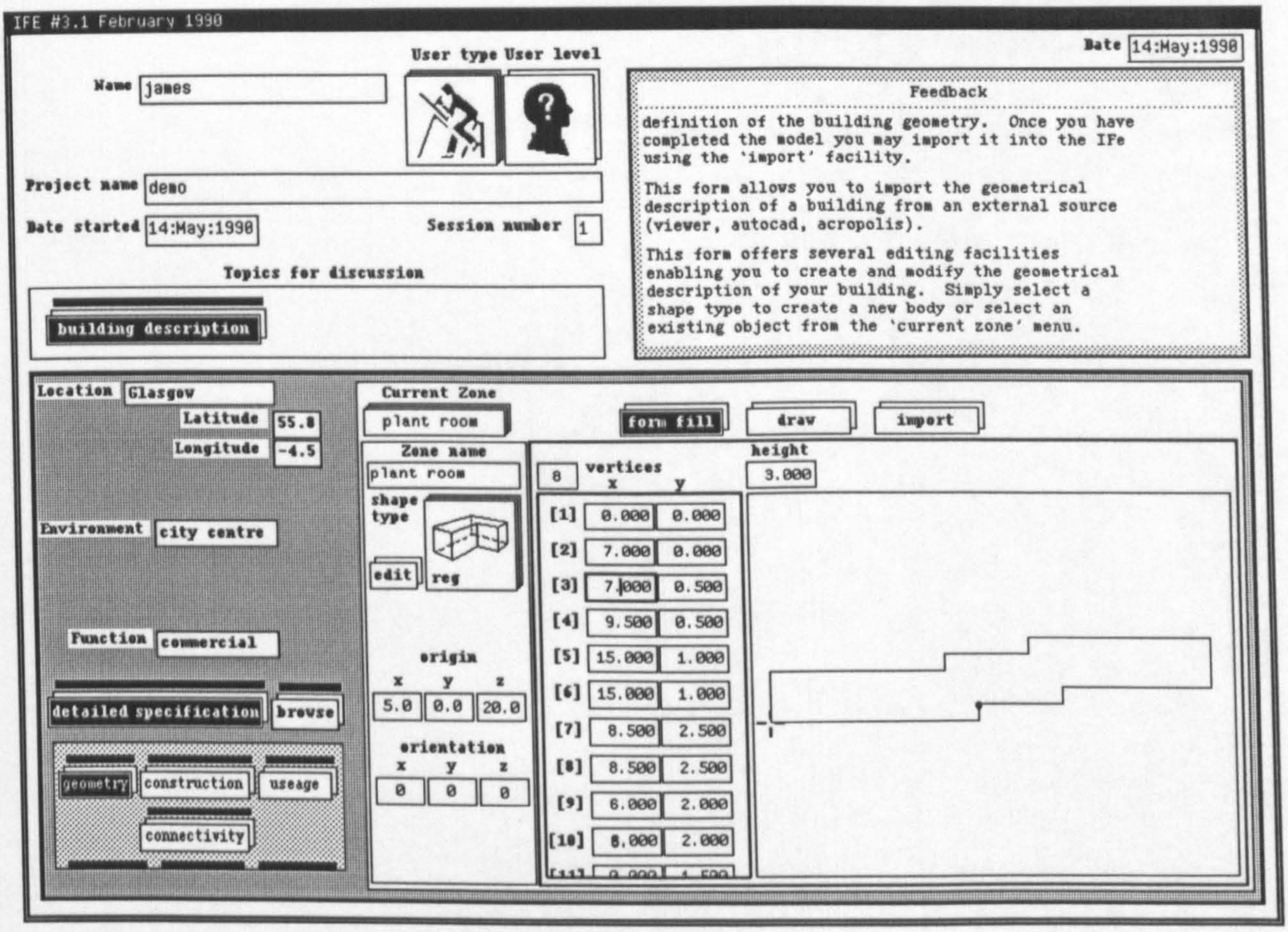


Figure 6.8.1.6. REG instance body.

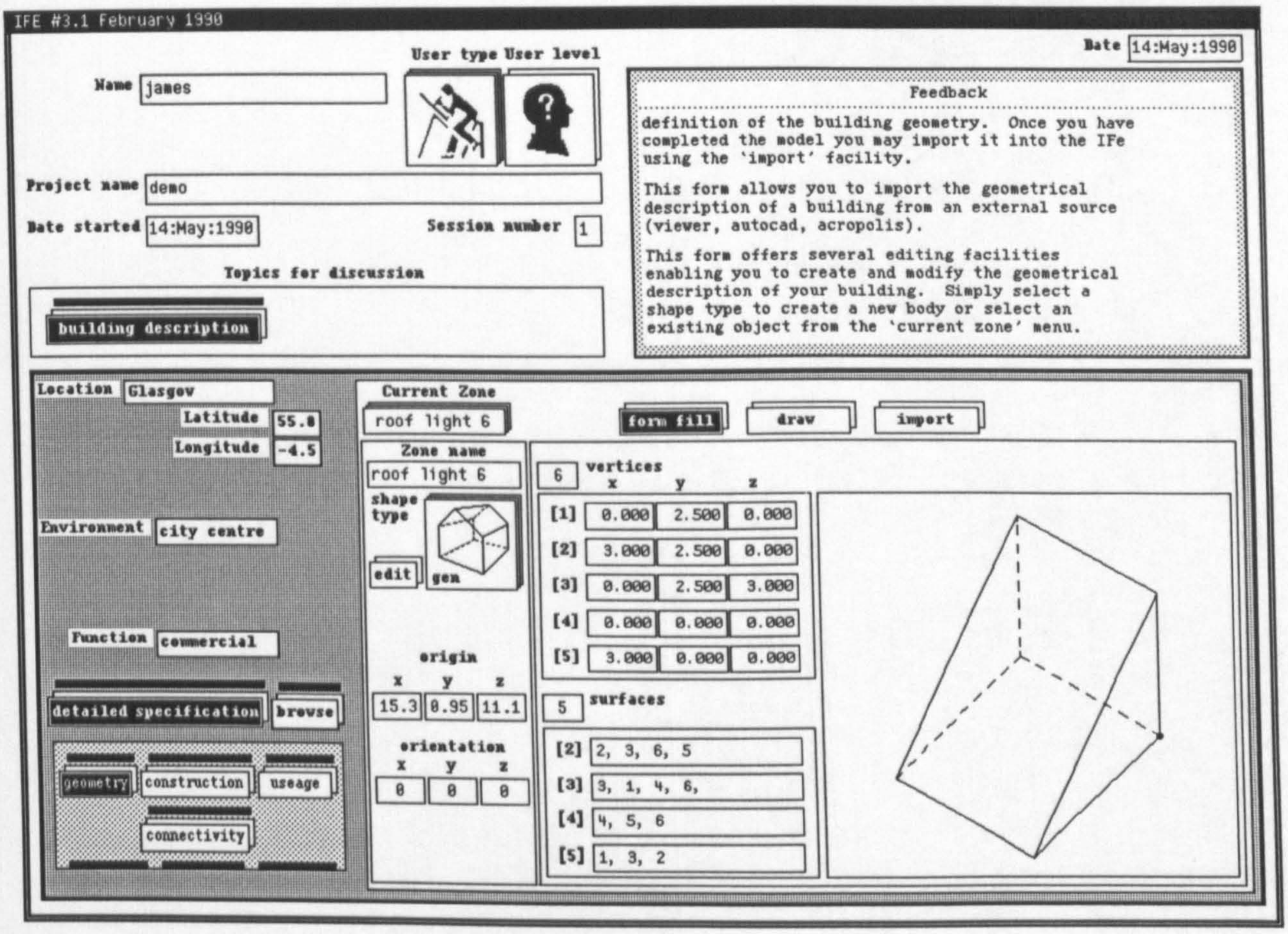


Figure 6.8.1.7. GEN body instance form.

The form fill form is divided into two areas:

i) generic geometrical description, include origin and orientation together with the zone name and instance type.

ii) instance attributes: one of the above REC, REG or GEN descriptions.

Existing zones may be retrieved using the cascading pop-up field "current zone" or simply by typing the zone name into the "zone name" field and pressing the edit button. This button serves two purposes, to:

i) force the user to de-select the zone name field and

ii) retrieve existing zones. If the zone does not exist an new instance is created and the current instance form is cleared.

By not selecting the edit button directly after changing the zone name is taken as a request to change the name of the current zone.

It is important to note that a body of type REC may be converted to either a REG, or GEN by selecting the the corresponding value from the shape type pop-up. Likewise it is possible for a REG body to be converted to a GEN. Once converted it is not possible to convert back to the original shape type since the body may have been deformed by the addition or modification of individual vertices.

6.8.2. CONSTRUCTION DEFINITION

Another fundamental issue requiring attention is the definition or specification of the thermo-physical properties of building components. This is achieved by selecting the construction option from the build spec focii form, figure 6.8.2.1.

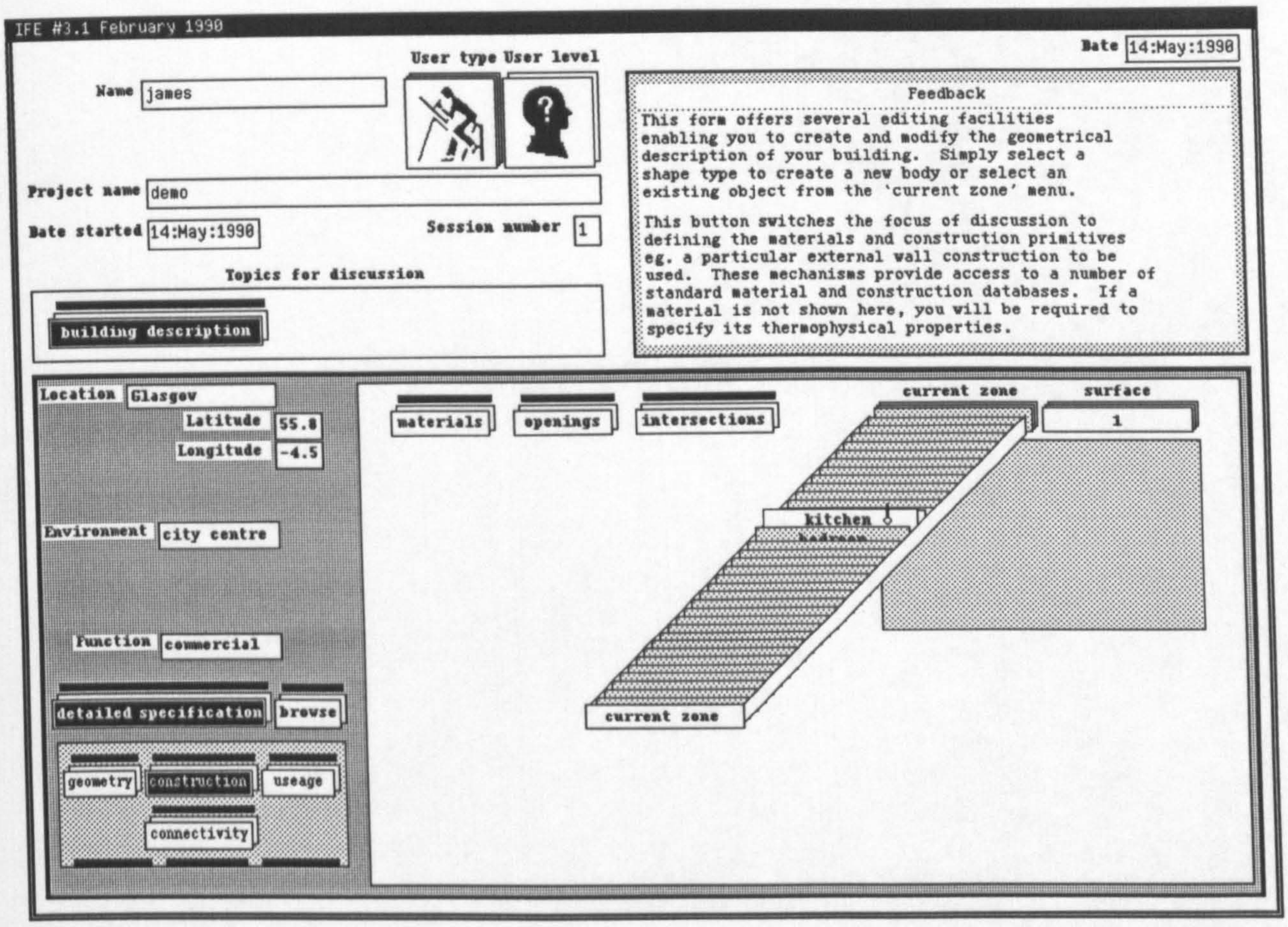


Figure 6.8.2.1. Building construction form.

The construction form contains several fields, figure 6.8.2.1: the materials, openings and intersections focus buttons, together with the current zone and surface pop-ups and a graphics field used to provide a visual representation of the current zone and surface.

6.8.3. MATERIAL SPECIFICATION.

When selected the material specification form offers the user a number of pre-defined construction types (determined by the building function category). These are presented in a schematic form on a cascading pop-up, figure 6.8.3.1.

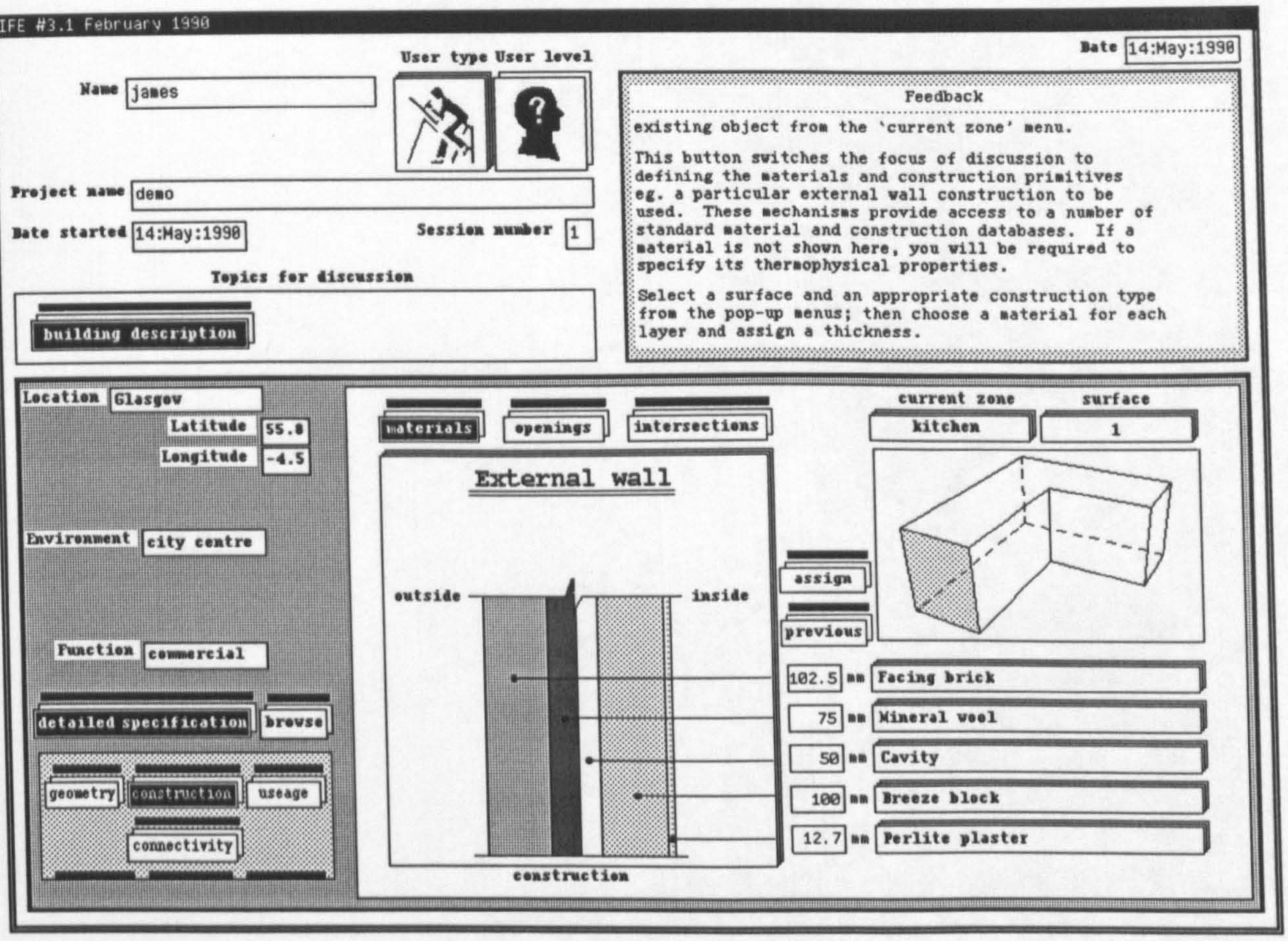


Figure 6.8.3.1. Multi-layered construction and material specification form.

Along with the graphical representation is a series of attribute fields relating to individual layers of the multi-layered construction. Each layer may have a number of alternative material types, obtained by selecting an item from the corresponding layer fields. The materials for each layer are obtained from standard material database, in particular ESP's constrdb file. The thickness of each layer may be set by the user, although for particular building elements such as standard bricks, thicknesses are defaulted.

Although proforma templates have been defined for the construction definition, corresponding knowledge bases have yet to be coded. The following screen images illustrate anticipated user dialogue and system responses. Feedback is also displayed in the general feedback window.

Figure 6.8.3.1, illustrates a vertical surface of a particular zone being assigned an external multi-layered construction type. The following image, figure 6.8.3.2, illustrates the user browsing through a series of construction types. It is important to note that "a plan to" modify event must be sent to the knowledge base which would respond by obscuring the values in the zone surface field, the graphical representation of the zone and the multi-layered construction attributes. Thus highlighting the inconsistency between current data and the user's actions.

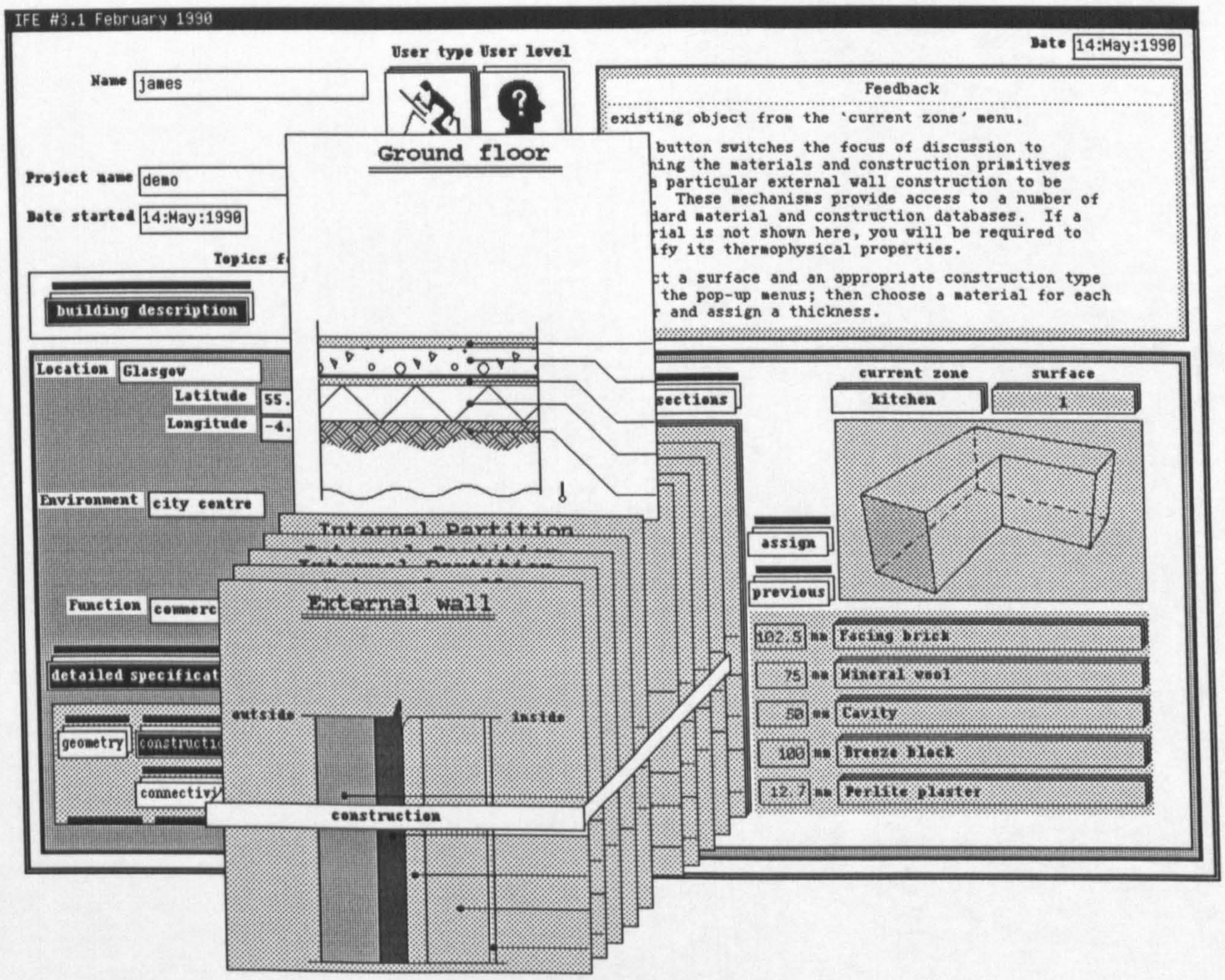


Figure 6.8.3.2. Browsing through multi-layered construction types.

Figure 6.8.3.3, shows that the user has selected a ground floor construction which is incompatible with the previous surface selection. This is highlighted in the feedback window.

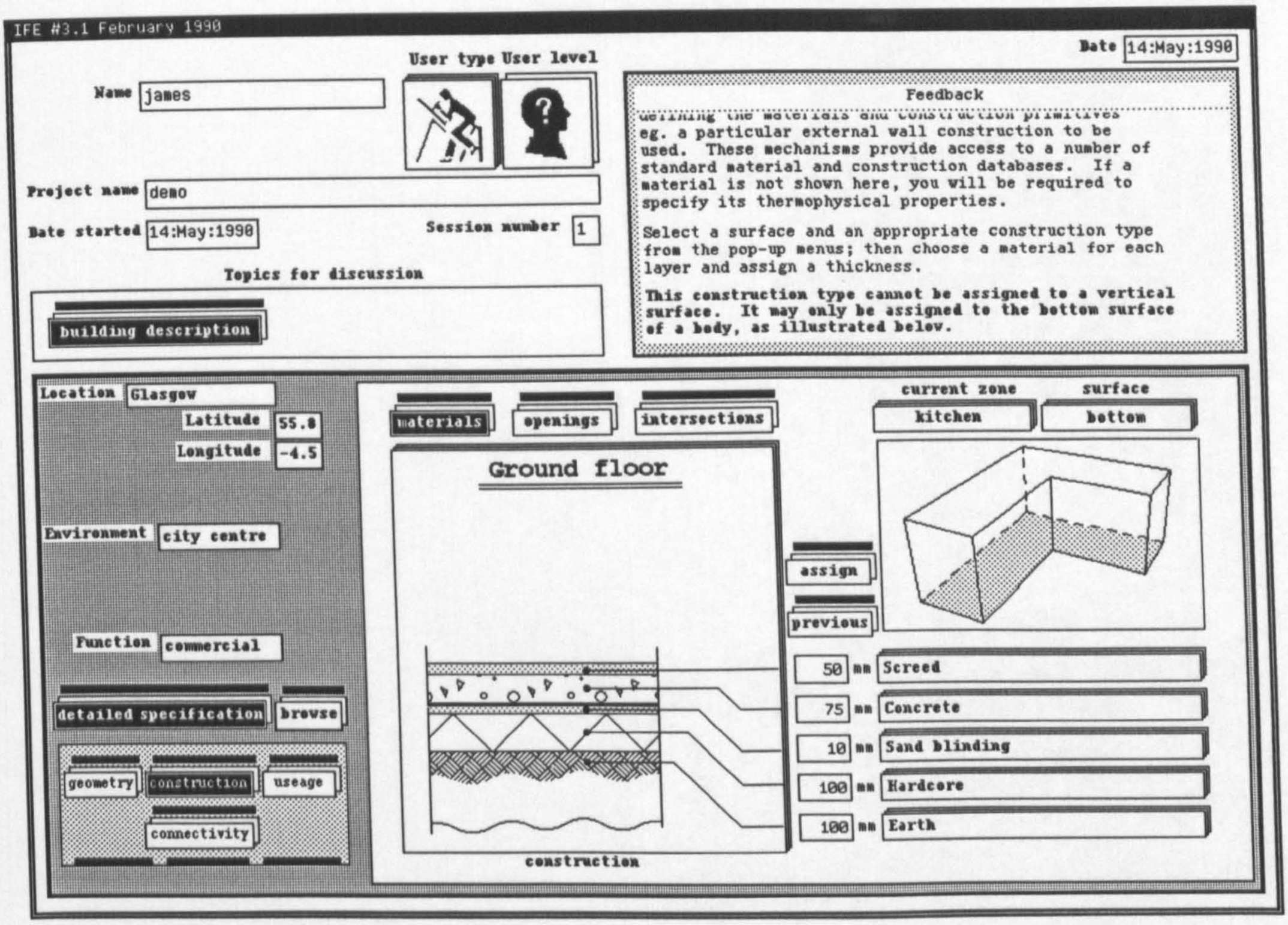


Figure 6.8.3.3. Automatic selection of surface in accordance with the construction type.

The knowledge base must contain information regarding valid construction contexts. In this instance the ground floor constructions may only be assigned to the bottom surface of a zone, which is automatically selected by the knowledge base, figure 6.8.3.3.

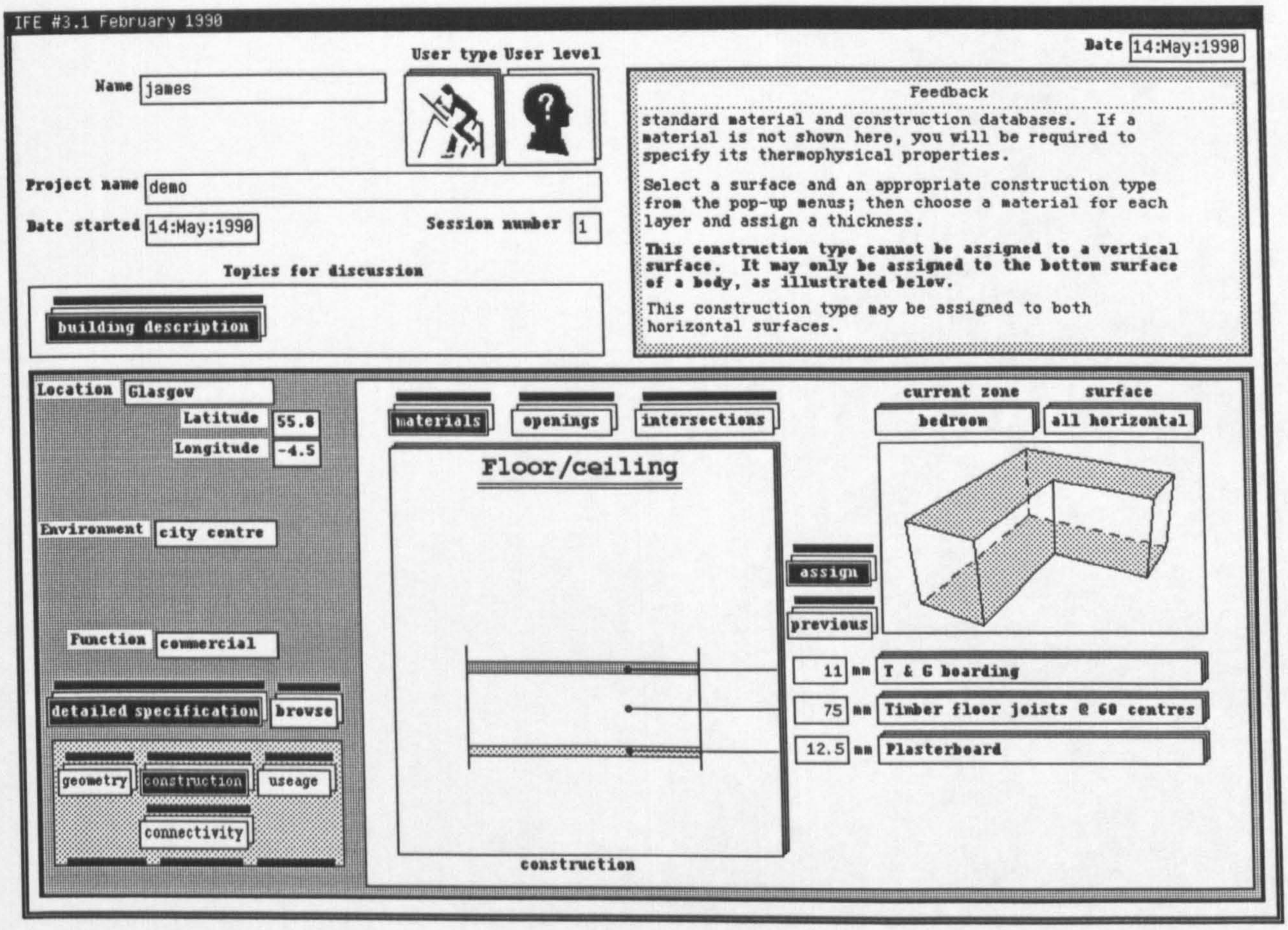


Figure 6.8.3.4. Assigning a multi-layered construction type to several surfaces.

Figure 6.8.3.4, illustrates the ability to assign a construction type (in this case an intermediate floor construction) to more than one surface at a time. This obviously increases efficiency.

6.8.5. OPENINGS IN MULTI-LAYERED CONSTRUCTIONS

This form offers two simple mechanisms for defining openings in multi-layered constructions. The methods follow those currently implemented in ESP, although the mechanisms presented here, figure 6.8.5.1 and 6.8.5.2, provide more flexible editing facilities. In addition to the opening type (either window or door, later cracks), the state of the opening is also required by ESP. This is entered using the state button which toggles between open and closed. It is envisaged that standard off the shelf units, as well as custom designed windows and doors be accommodated. Unit types my be obtained from standard databases using the unit type pop-up.

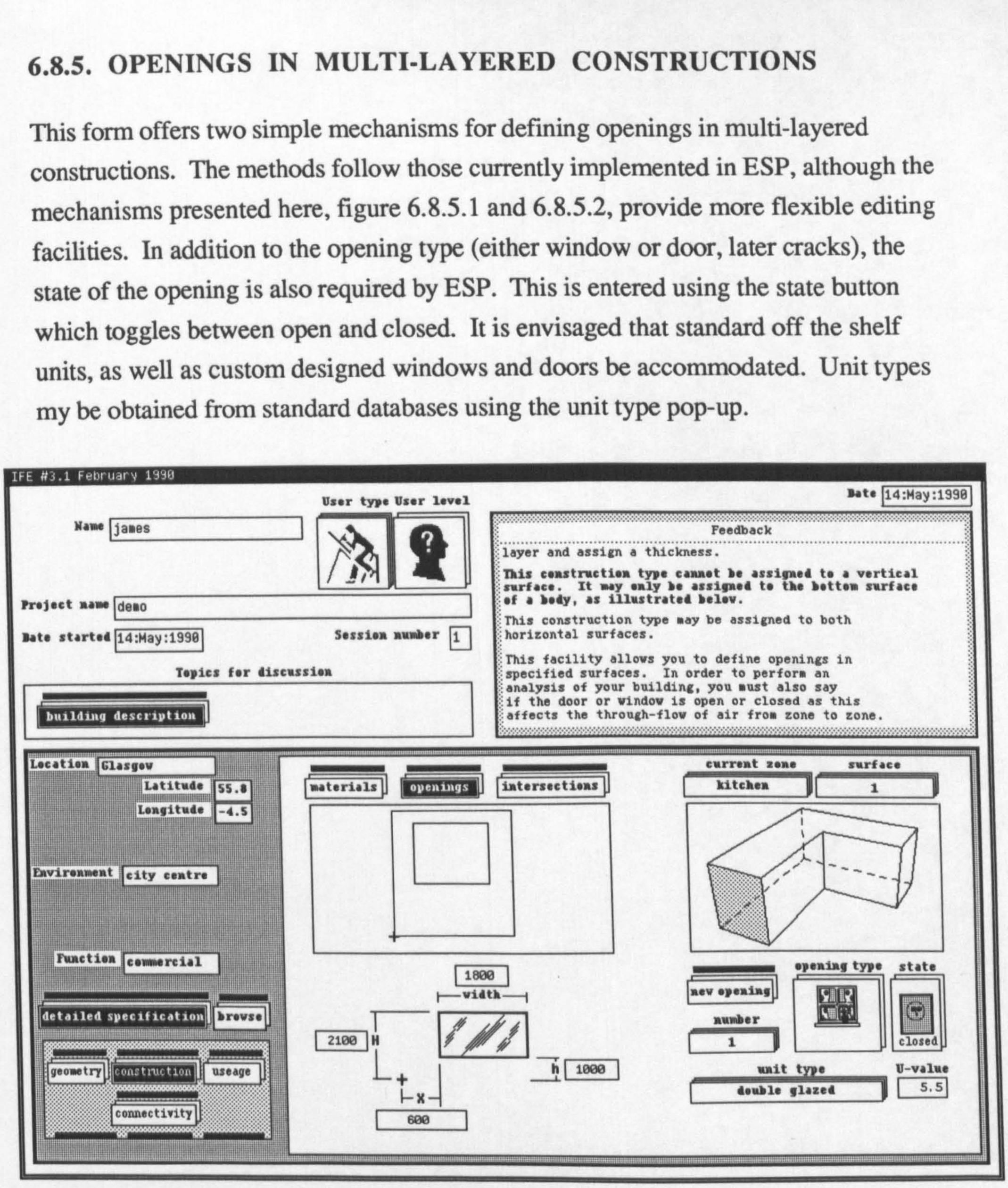


Figure 6.8.5.1. Window opening specification form.

As in the case of material specification a surface must be specified together with an opening number. This is achieved by selecting the "new opening" button for each opening in the selected surface. Existing openings may be retrieved for editing (deleting) using the "number" field.

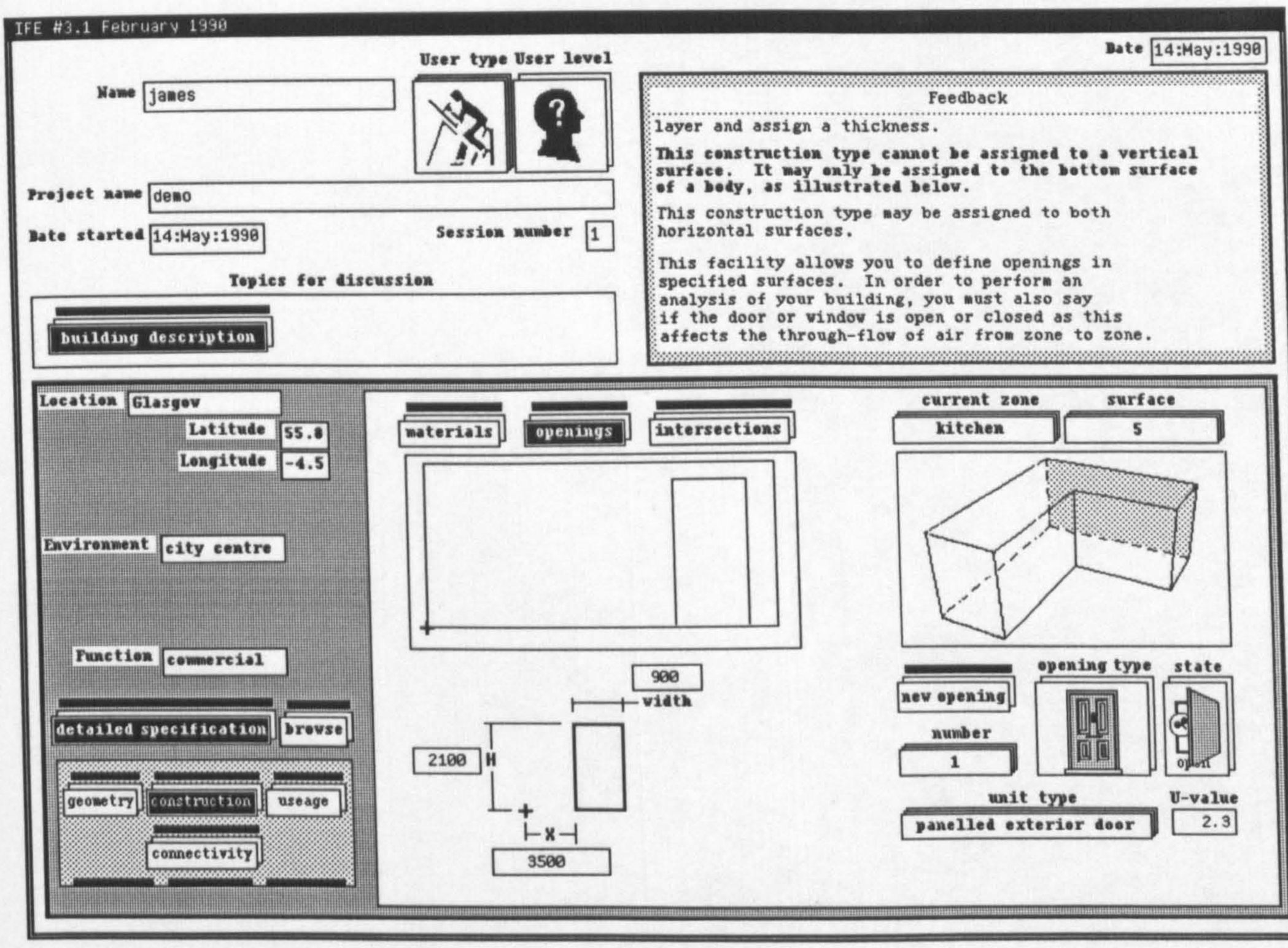


Figure 6.8.5.2. Door opening specification form.

For both window and door openings a simple attribution form is displayed, allowing the user to define the lintel and cill heights (windows only) together with the width of the opening. The offset from the surface datum (the left most corner) must also be entered. When standard units are selected the width may be automatically entered. A graphical representation of the current surface and opening is also provided. It is envisage that subsequent versions of the forms package will enable the user to directly manipulate this graphical entity.

6.8.4. INTERSECTIONS.

Again with incomplete knowledge handlers, the intersections form is intended to allow the user, or designer, to specify boundary conditions between multi-layered constructions.

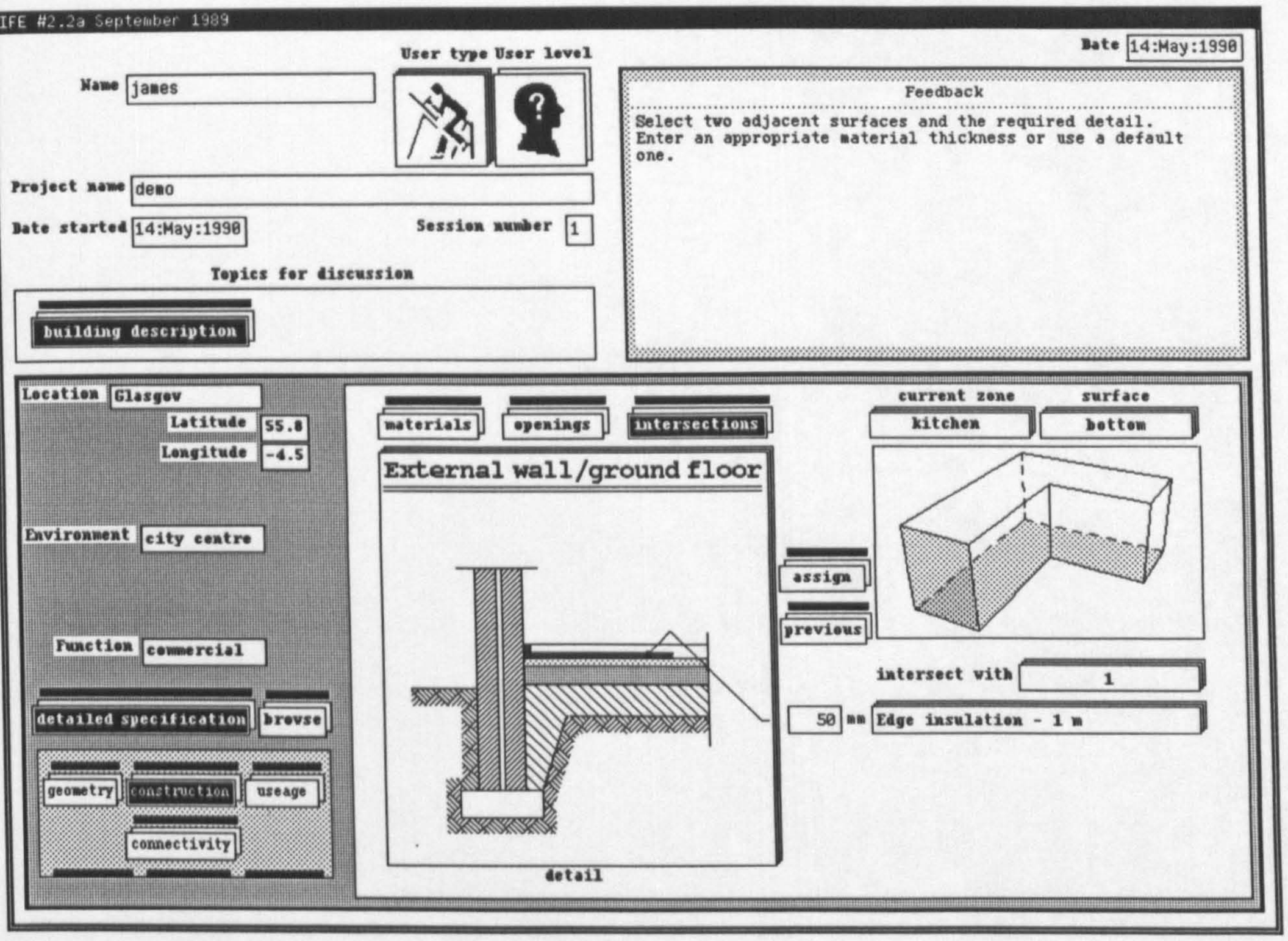


Figure 6.8.4.1. Boundary conditions.

For example, figure 6.8.4.1, illustrates the user specifying an edge insulation strip at the external wall boundary.

6.9. SUPPORTING WHAT IF - BROWSING AND RETRIEVING EXISTING MODELS

The ESP system is used to predict the energy performance of fully attributed buildings and is therefore used to check completed design solutions. However, a flexible design tool would allow the designer to run tentative design solutions in order to investigate various ideas. In order to accommodate this function a browsing mechanism has been provided.

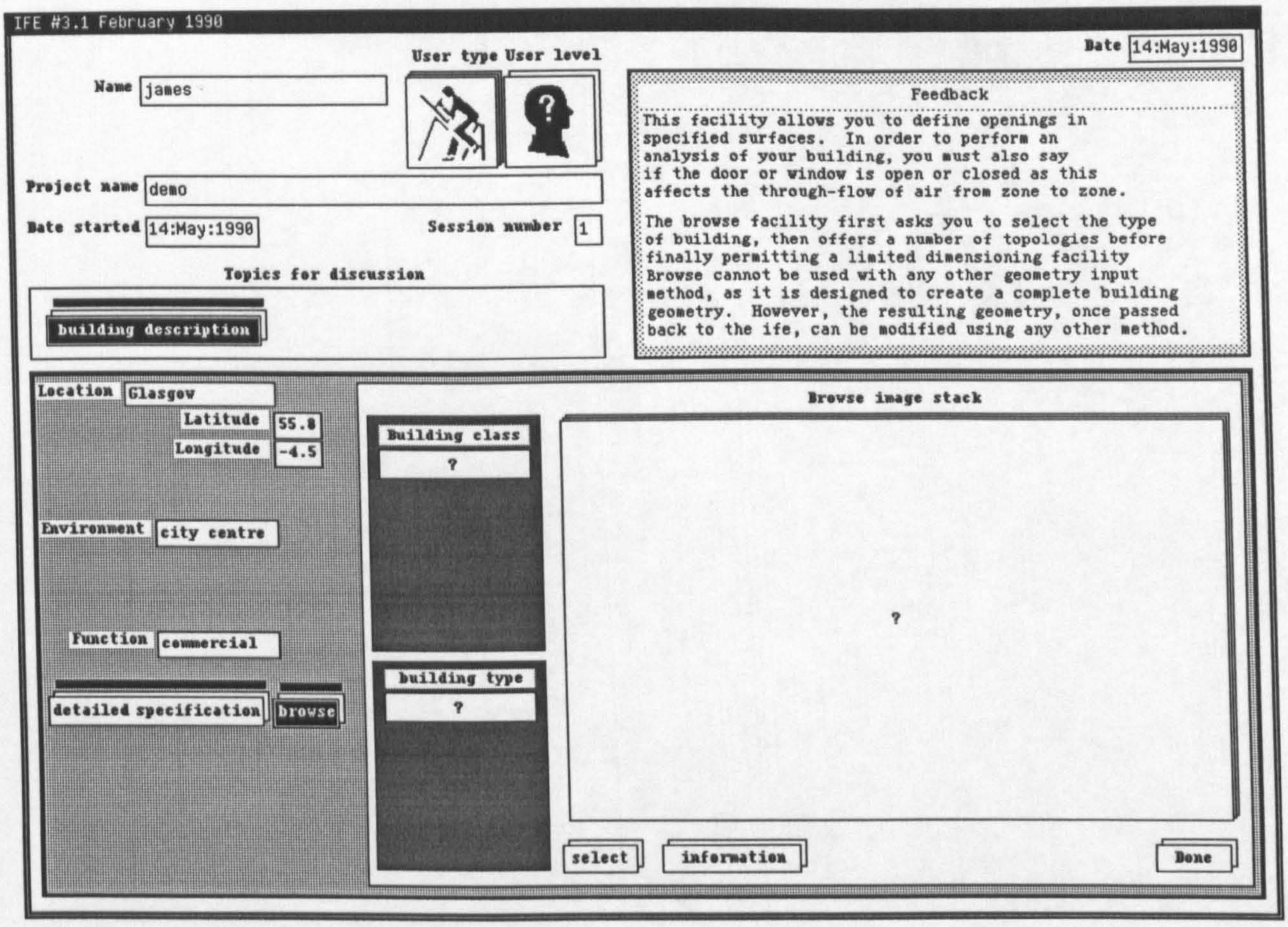


Figure 6.9.1. Browse facility; initial state.

The browse program (Rutherford) (invoked by the application handler at the request of the knowledge base) scans a predefined directory (~ife/lib/uc/uc/buildings) which contains a set of sub-directories corresponding to the current user conceptualisation. Within each of these directories any number of subdirectories, containing any number of exrep[1] images, may be placed.

---

1 Exrep image files may be created directly using the PixEd program (J. Rutherford, ABACUS and M. Martin, RAL) or generated from a source bitmap - such as a sun raster, using the bitmap conversion filter conv (Martin). Additional filters (rasterizers) have been incorporated within conv (Rutherford) to create bitmaped images of ABACUS viewer plot files.

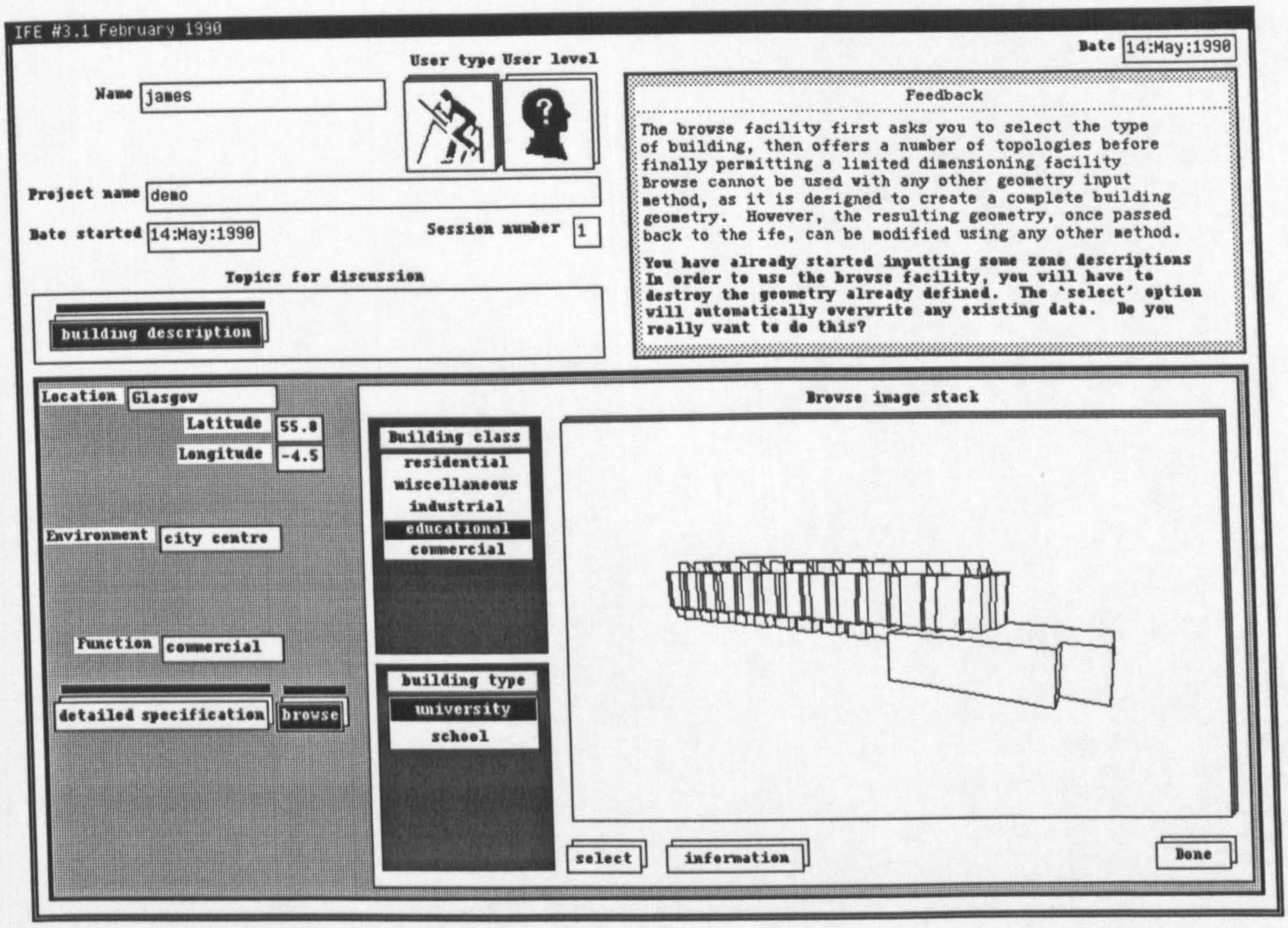


Figure 6.9.2. Browse offering building class and type. Note that the user has already made a selection and the corresponding images have been loaded.

The browse program interacts with the knowledge base which in turn interacts with the user, by means of two menus, figure 6.9.1. Simply by selecting a building class the browse utility returns the names of sub-classes of buildings which, when selected returns the names of images files. These images are then displayed on a cascading pop-up which the user may ripple through, figure 6.9.2. and 6.9.3.

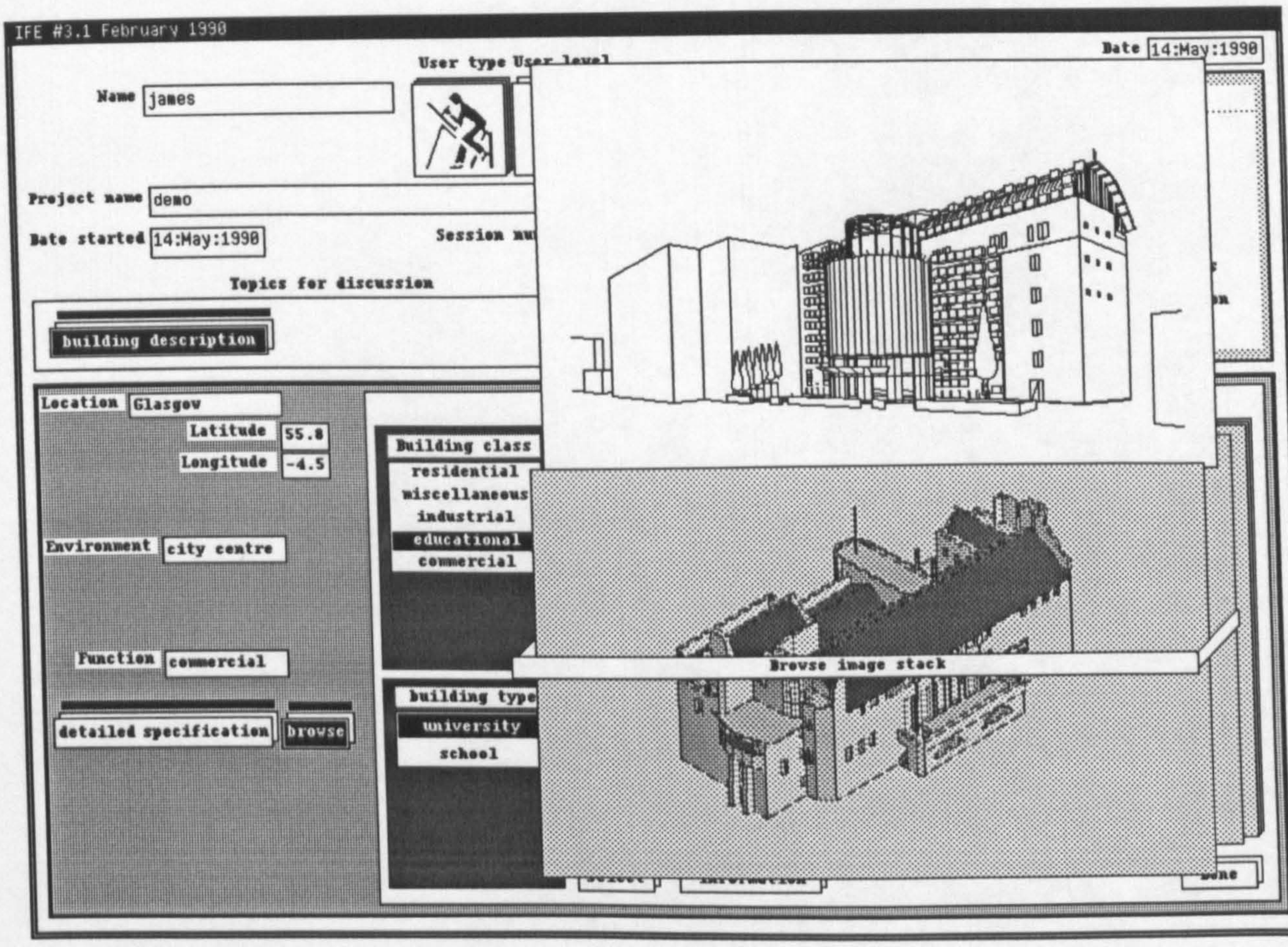


Figure 6.9.3. Interactive selection of design exemplars.

The browse facility therefore allows, for example, complete design exemplars to be offered to an IFe user. Entire models may be defined without the need to explicitly enter topographical and topological information. This facility will also enable related attribute data (constructional information, for example) to be stored and retrieved. The examples illustrated are based upon this assumption.

When the browse facility is selected a warning, indicating that the browse program is intended to define complete building descriptions and therefore overwrites existing information, is displayed in the feedback field.

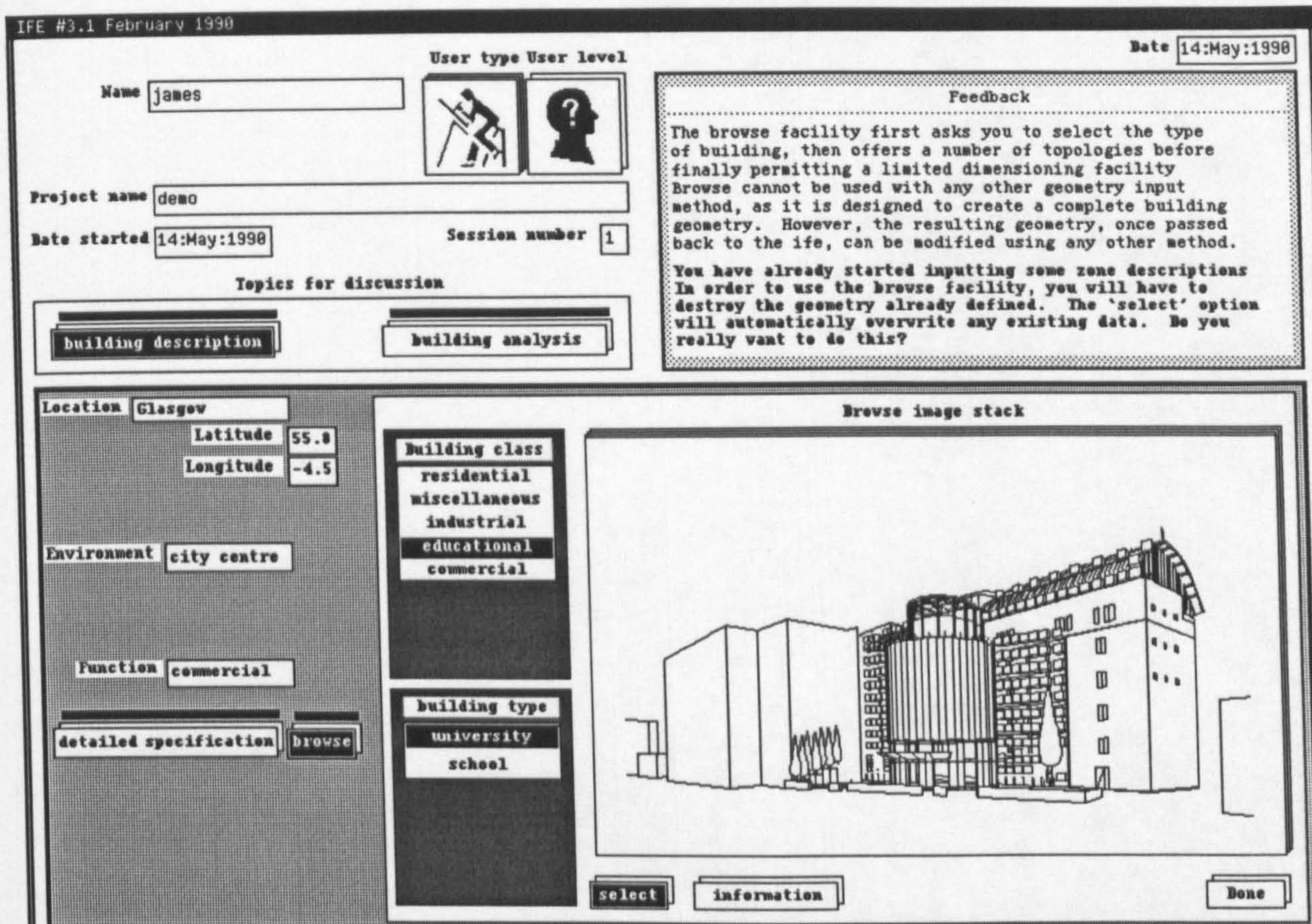


Figure 6.9.4. Design exemplar selected.

Once building data has been entered using this mechanism, it may be modified using any of the techniques already described. Currently the user has to indicate that data input is complete however this could be inferred by the knowledge handler. As soon as sufficient descriptive information has been entered the available topics for discussion would then be extended; including the option to analyse the building, figure 6.9.4.

6.10. ANALYSIS - PERFORMANCE METHODOLOGIES.

Once the building description is complete, by what ever means, the designer may now begin to investigate the performance of the building using a number of pre-defined assessment methodologies.

Each user conceptualisation contains appropriate performance methodologies, figure 6.10.1, which may be categorised into (Clarke):

i) mono-functional methodologies, and

ii) multi-functional methodologies.

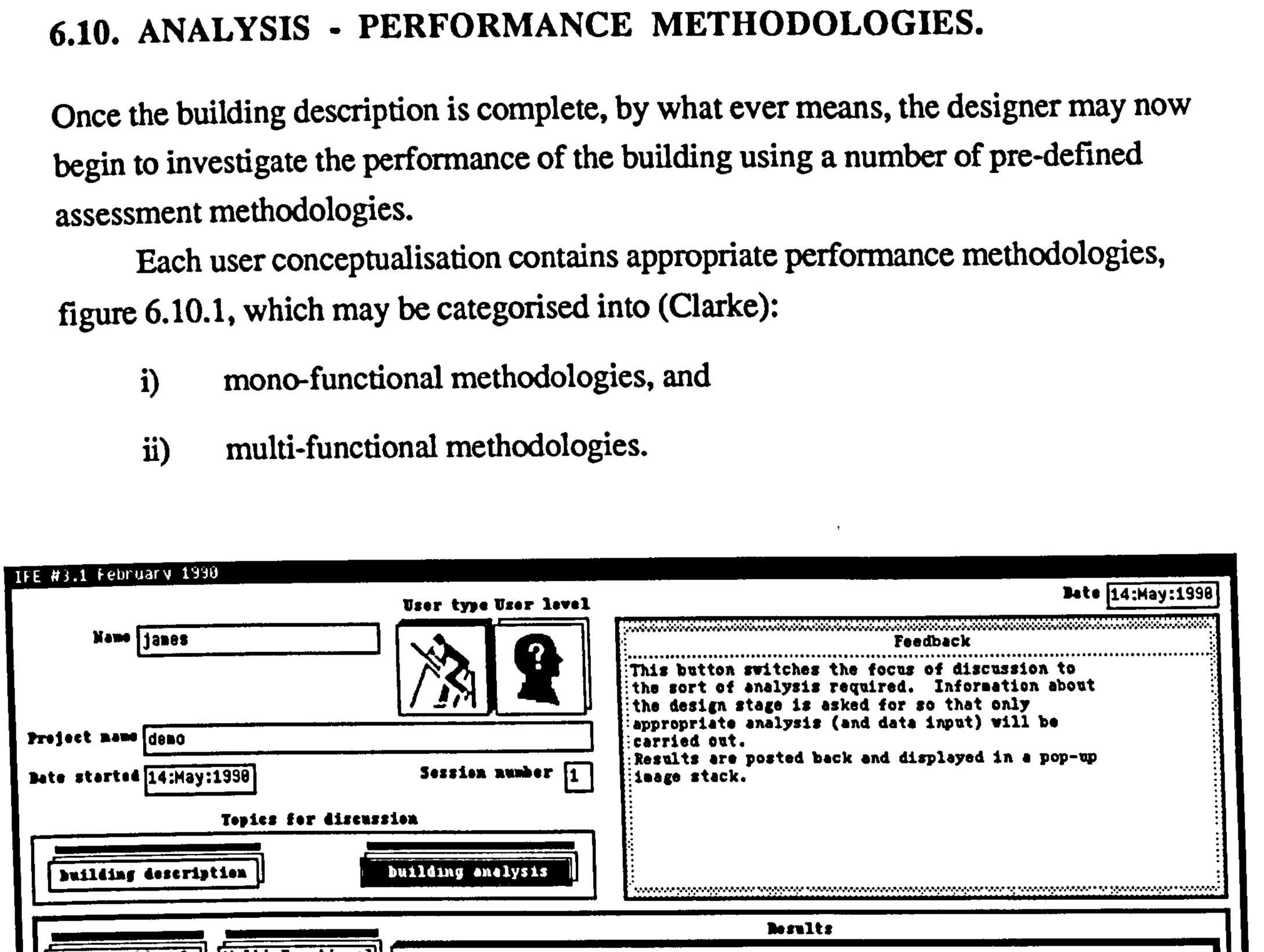


Figure 6.10.1. Performance methodologies

These methodologies essentially contain an expert's encoded knowledge of how to run the target application program (ESP) for a particular goal. By a simple selection mechanism a user is able to employ the knowledge and experience of a whole range of experts. In this instance the Unix Bourne Shell is used as a pseudo expert system

[CLARKE][IFE 89] shell; executing parameterised shell scripts which interrogate the product description area on the blackboard via the data handler.

6.10.1. DEFINING CONSTRAINTS

Operational and simulation constraints may be defined by the user by interacting with a number of fields. The analysis form currently allows the user to set the start and end times for the simulation using two date fields, figure 6.10.1.1.

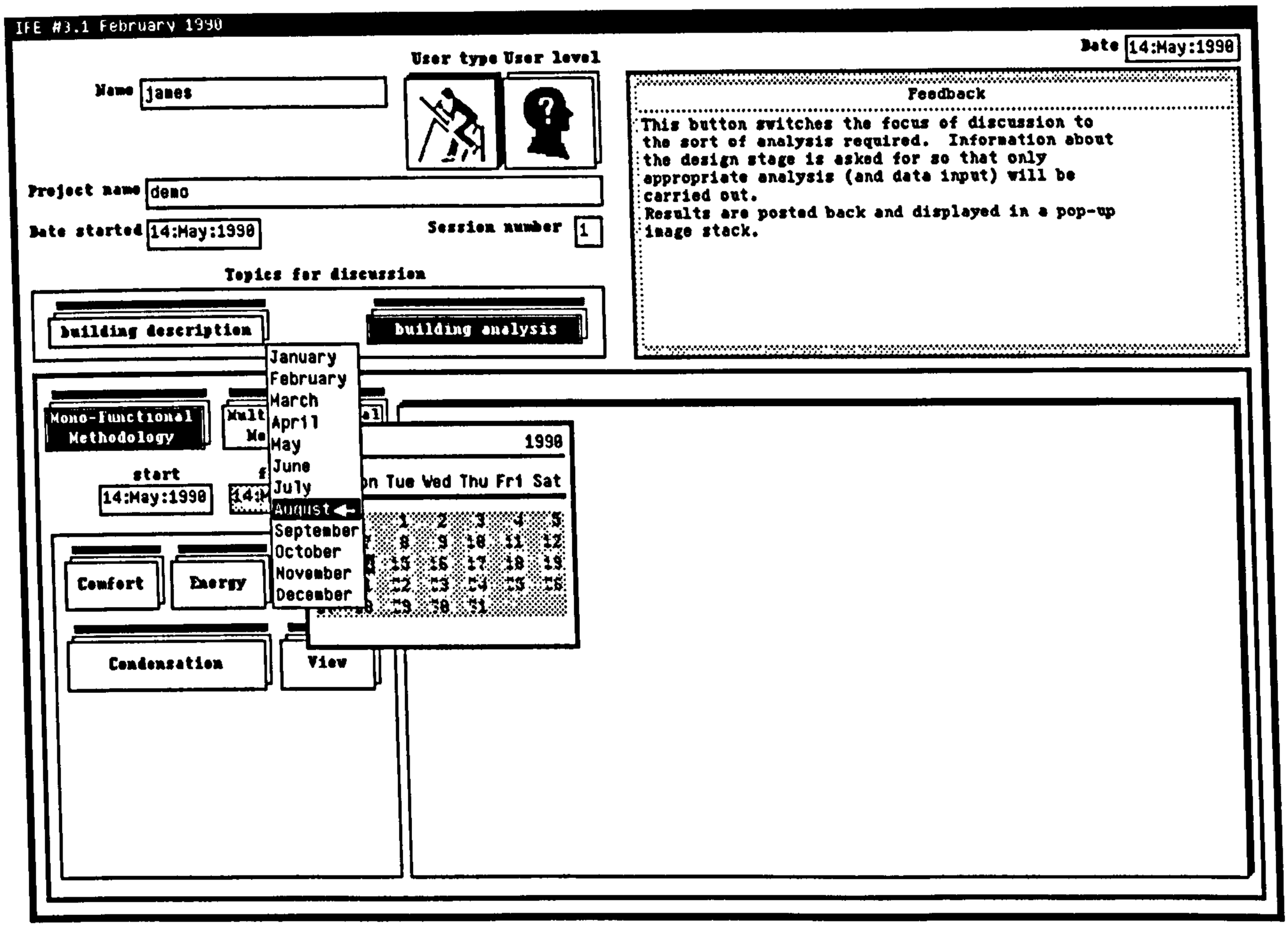


Figure 6.10.1.1. Defining simulation constraints.

Other constraints such as time increments may be set explicitly or inferred from the current stage of design; a more coarse time increment used at the earlier, tentative design stages.

Behind each button lies a script of the form illustrated below, figure 6.10.1.2 [IFE 89].

```
if test "X${IFe_HOME}" = "X"
then
        echo "Please set up 'IFe_HOME' shell"
        echo "variable and `export' it."
    exit
fi
clear
echo "ESP Script 1: - Heating Plant Sizing."

                # Set up defaults and process command line options.
results=${IFe_HOME}/lib/tmp/results
building=${IFe_HOME}/lib/tmp/building
climate=${IFe_HOME}/lib/tmp/climate
control=${IFe_HOME}/lib/tmp/control
start="9 1"
finish="15 1"
timesteps=2
if test $# -ne 0
then
  for i do
    case "$i" in
       -r) results=$2; shift; shift;;
       -b) building=$2; shift; shift;;
       -c) climate=$2; shift; shift;;
       -o) control=$2; shift; shift;;
       -s) start=$2; shift; shift;;
       -f) finish=$2; shift; shift;;
       -t) timesteps=$2; shift; shift;;
       --) shift;;
       -*) echo "Unknown option: $i"
           echo -n "Usage <FN_KEY> -c climate"
           echo " -b building -r results -s start"
           echo " -f finish -t timesteps -o control"
           exit 2;;
    esac
  done
fi
echo
echo "            Files used   : ${building}"
echo "                           ${control}"
echo "                           ${climate}"
echo "            Files created: none"
echo
echo " wait ....."
sim  >/dev/null <<~   # run ESPsim, redirect i/o
-6                          # line X
${climate}
1
${building}
y
1
3
${results}
${start}
${finish}
${timesteps}
0                 # the no save option
s
y
${control}
y
o
n
-
f                           # line Y
~
```

Figure 6.10.1.2. Heating plant sizing script [Clarke][IFE 89].

Figure 6.10.1.2 represents a very simple Script [CLARKE][IFE 89]:

Firstly, the variable IFe_HOME is looked for. This defines the home directory of the IFe system where the Scripts are located. Then the Unix program `clear' is invoked to clear the window in which the Script will run. All run time variables are then defaulted before being assigned on the basis of the command line options as established by the Appraisal Handler. Summary information is then output before the ESP simulation is requested by issuing the command `sim'. Since the script will have no user interaction, ESPsim's standard output and input are redirected. In this case standard output is discarded (put to /dev/null) while standard input is taken from the Script itself (<<) until the tilde (~) character is encountered. The ESP simulation is now performed against the driver commands of of lines X through Y. These commands are identical to those that would be entered if the system was being operated interactively. The -6 informs ESPsim that it will be operating in Script mode and so, internally, it reassigns its output channels to enable text and graphics outputs to be captured separately.

In this Script no results library is created. Instead ESPsim's summary output feature is invoked so that the final result passed back to the Application Handler is a file containing the following information,figure 6.10.1.3:

ESP Script 1: - Heating Plant Sizing.

Climate file : /usr/ife/lib/tmp/climate
Configuration file : /usr/ife/lib/tmp/building
Configuration description : ESP standard test

Control file name : /usr/ife/lib/tmp/control

Building save option : No results saved
No. of warning messages : 0

Simulation period : 1 day: from 9, 1

Start-up period : 1 day
Building time-step : 1 / hr
Number of zones : 5
Zone-time increments : 240
Building results file size : Not applicable
Simulation time : 120 secs

Result:

Period simulated from day 9 of month 1 to day 9 of month 1

Zone	Mx Air tmp (\deC)	Mn Air tmp (\deC)	Max heat (KW)	Max cool (KW)	Heating (KWhrs)	Cooling (KWhrs)
1	16.00 @ 8.50- 9- 1	6.58 @ 6.50- 9- 1	0.908 @ 8.50- 9- 1	0. @ 0.50- 9- 1	9.9	0.
2	20.00 @ 8.50- 9- 1	7.58 @ 6.50- 9- 1	1.805 @ 8.50- 9- 1	0. @ 0.50- 9- 1	19.6	0.
3	20.09 @17.50- 9- 1	8.08 @ 6.50- 9- 1	0.891 @10.50- 9- 1	0. @ 0.50- 9- 1	10.4	0.
4	16.71 @23.50- 9- 1	6.65 @ 9.50- 9- 1	1.777 @22.50- 9- 1	0. @ 0.50- 9- 1	3.6	0.
5	-0.14 @13.50- 9- 1	-4.22 @ 8.50- 9- 1	0. @ 0.50- 9- 1	0. @ 0.50- 9- 1	0.	0.

All zones :

Max. Temp. = 20.1 in Zone 3 on day 9 of month 1 at 17.50 hrs
Min. Temp. = -4.2 in Zone 5 on day 9 of month 1 at 8.50 hrs
Max. Heat. = 1.8 in Zone 2 on day 9 of month 1 at 8.50 hrs
Max. Cool. = 0. in Zone 1 on day 9 of month 1 at 0.50 hrs

Total heating requirements = 43.45 (KWhrs)
Total cooling requirements = 0. (KWhrs)

Figure 6.10.1.3. Results output from plant sizing script.

Much more comprehensive performance assessment Scripts are illustrated in the IFE final report [IFE 89].

Product data is extracted from the blackboard using a simple query language. For each target application a data definition script is required.

6.10.2. RESULTS, INTERPRETATION, FORMATTING AND FEEDBACK

Results from client applications are interpreted, according to the assessment methodologies employed and the user's conceptual vocabulary by a series of generic tools. Each tool is capable of formatting data in particular ways such as pie diagrams, bar charts, graphs. Rather than expect the forms package to provide these facilities, each of the interpretation modules produces a bitmaped image which is transmitted to the appropriate display field, although grpahical objects would be more appropriate as it would not be necessary for the resource to know the physical constraints of the display region.

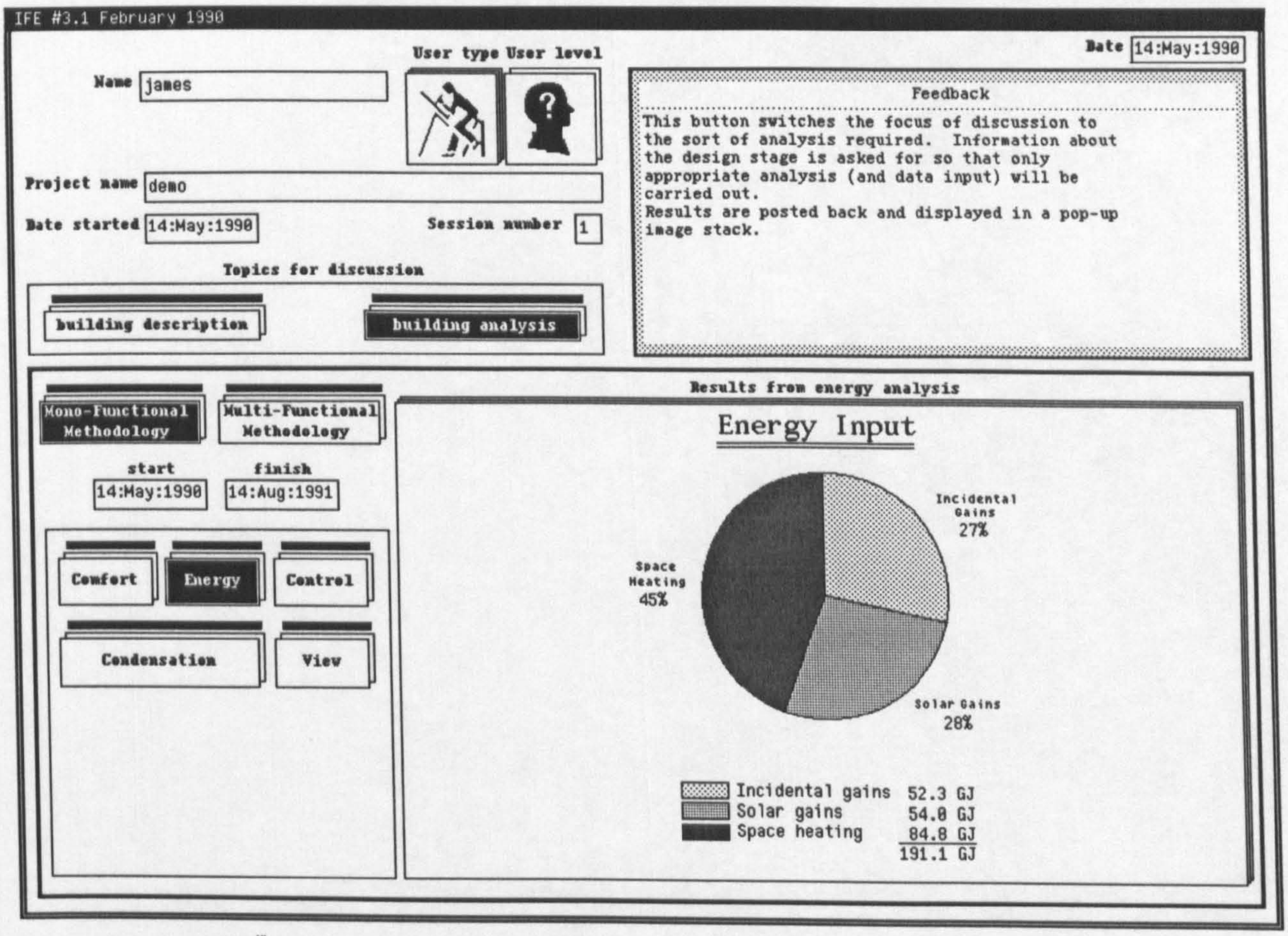


Figure 6.10.2.1. Results feedback from energy simulation.

Results are formatted and presented to the user, usually in a graphical form. A typical example of the output is illustrated in figure 6.10.2.1.

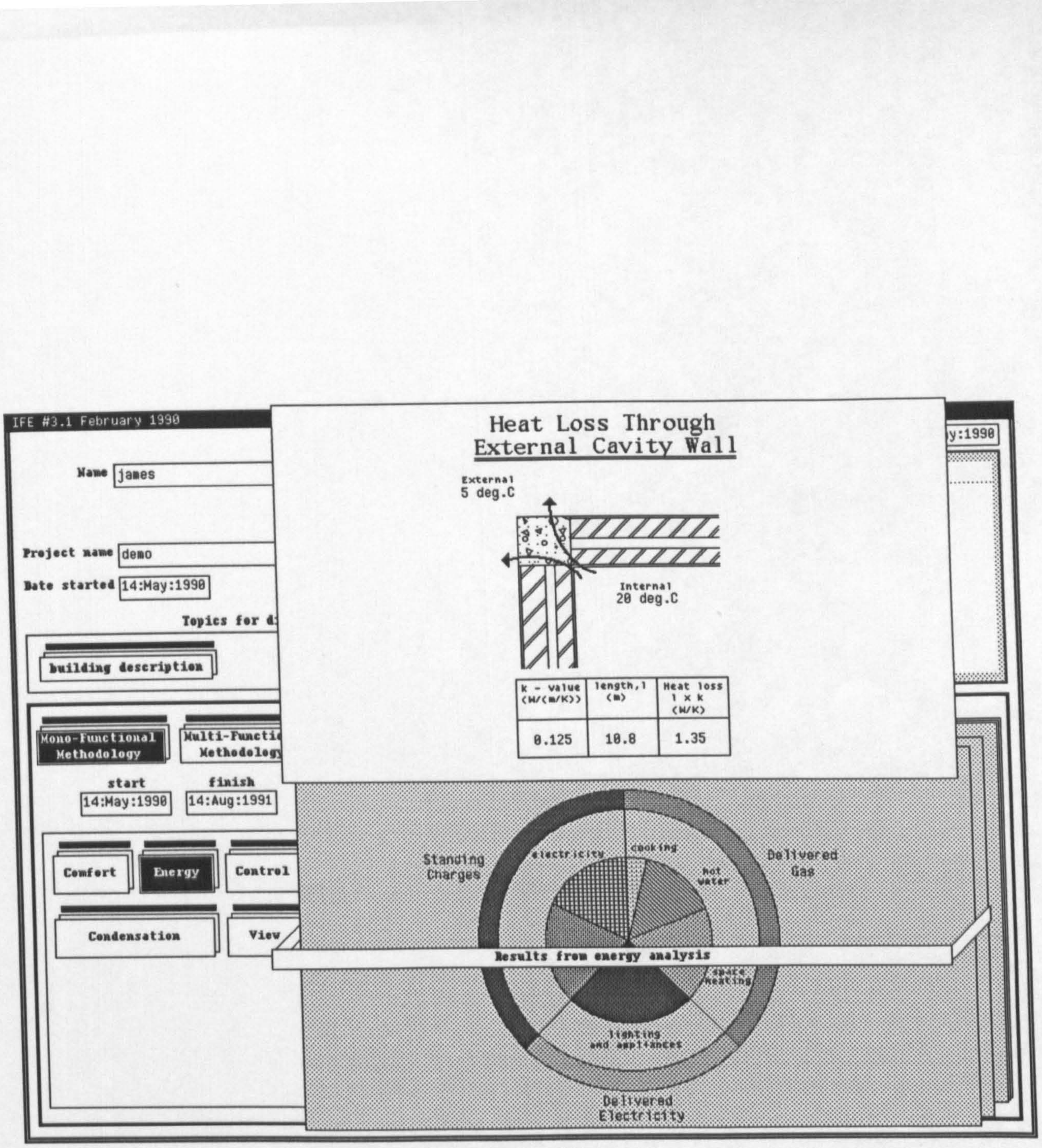


Figure 6.10.2.1. Rippling through simulation results; highlighting problematic areas.. Options for hard-copy will be available to the user.

Results are displayed in a cascading pop-up consisting of pie diagrams and tables. By employing other resources, results may be interpreted and suggestions for improvements, figure 6.10.2.1, and graphical representations of problematic areas, figure 6.10.2.2, presented.

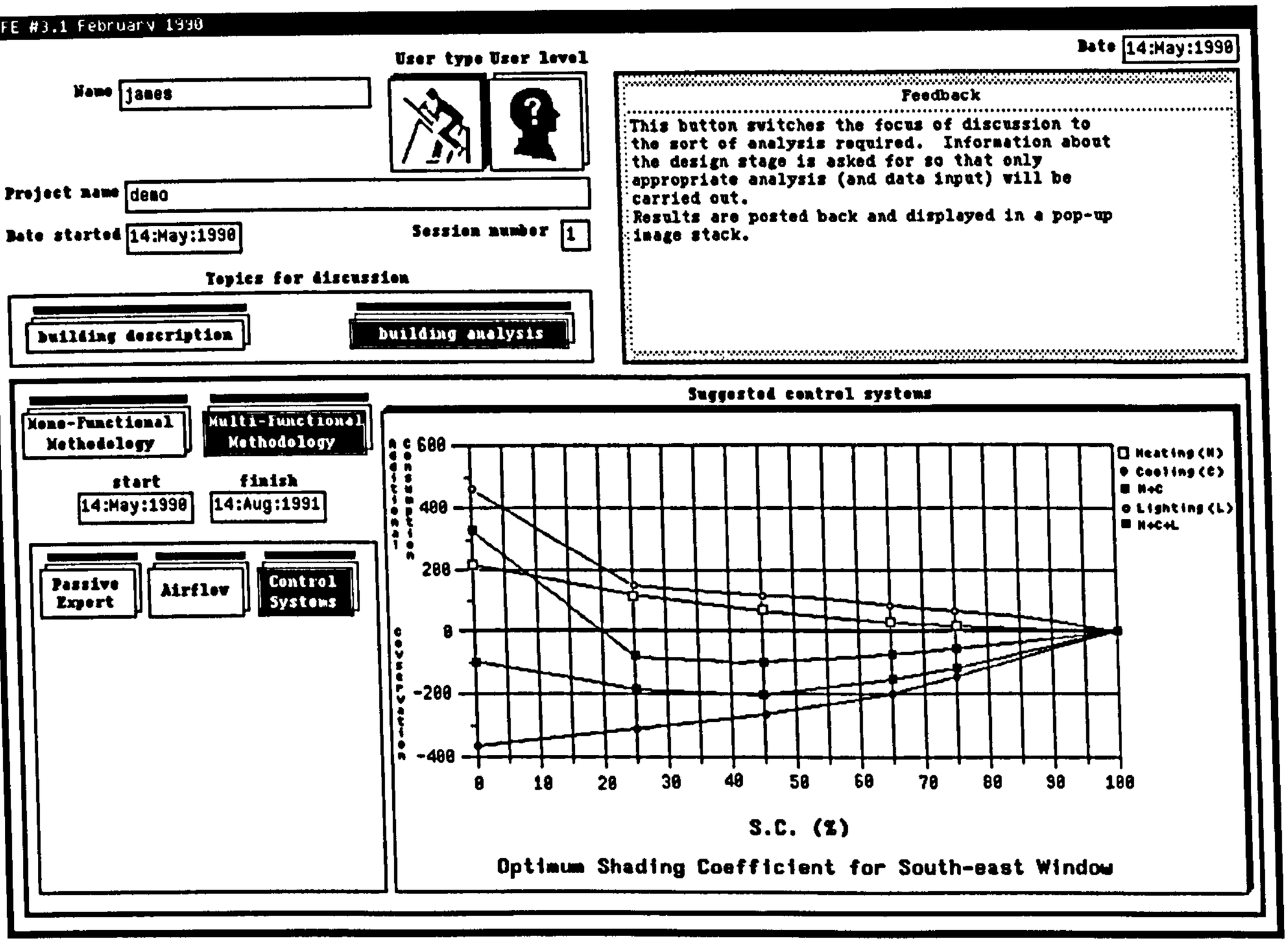


Figure 6.10.2.2. MUlti-functional methodologies; suggested control systems.

An integral part of any CAD package is a geometrical description of the product. Based around the dynamic structure identified in chapter 3, one of the resources used to relay simulation results back to the user is an interactive multi-representational 3D object visualisation and manipulation program, described in Appendix E. This program, operating on a Silicon Graphics Iris workstation, displays the building model as either a 3D wireline or rendered perspective, figure 6.10.2.3.

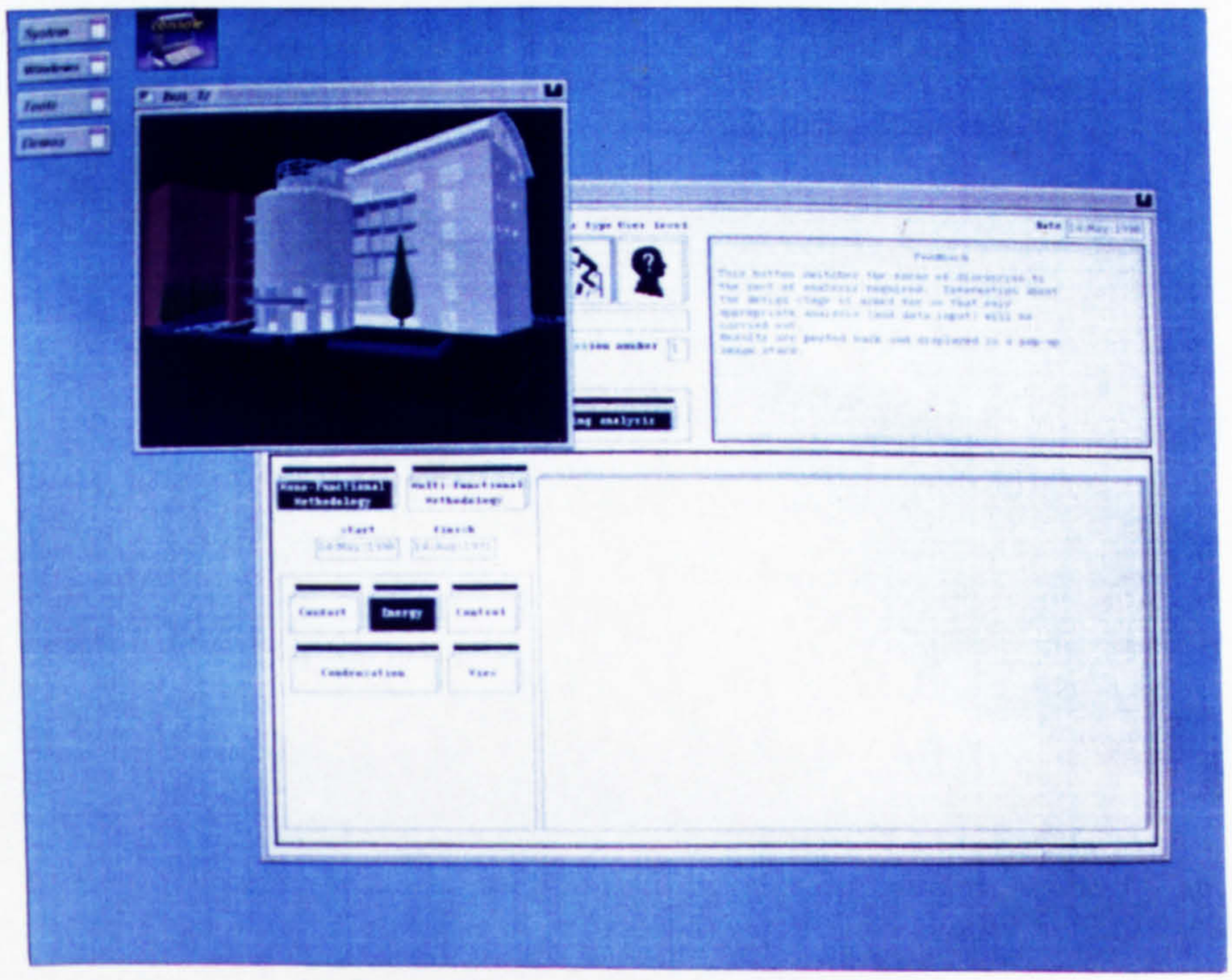


Figure 6.10.2.3. Interactive 3D display interface.

A domain template interface is created (in this case for the interpretation of thermal information) which interprets utterances generated by the knowledge base and maps them to program specific commands.

For example

Zone 1:overheating:10

maps to: Zone 1:colour: 200 0 0

resulting in a colour change of the graphical representation of Zone 1, figure 6.10.2.4.

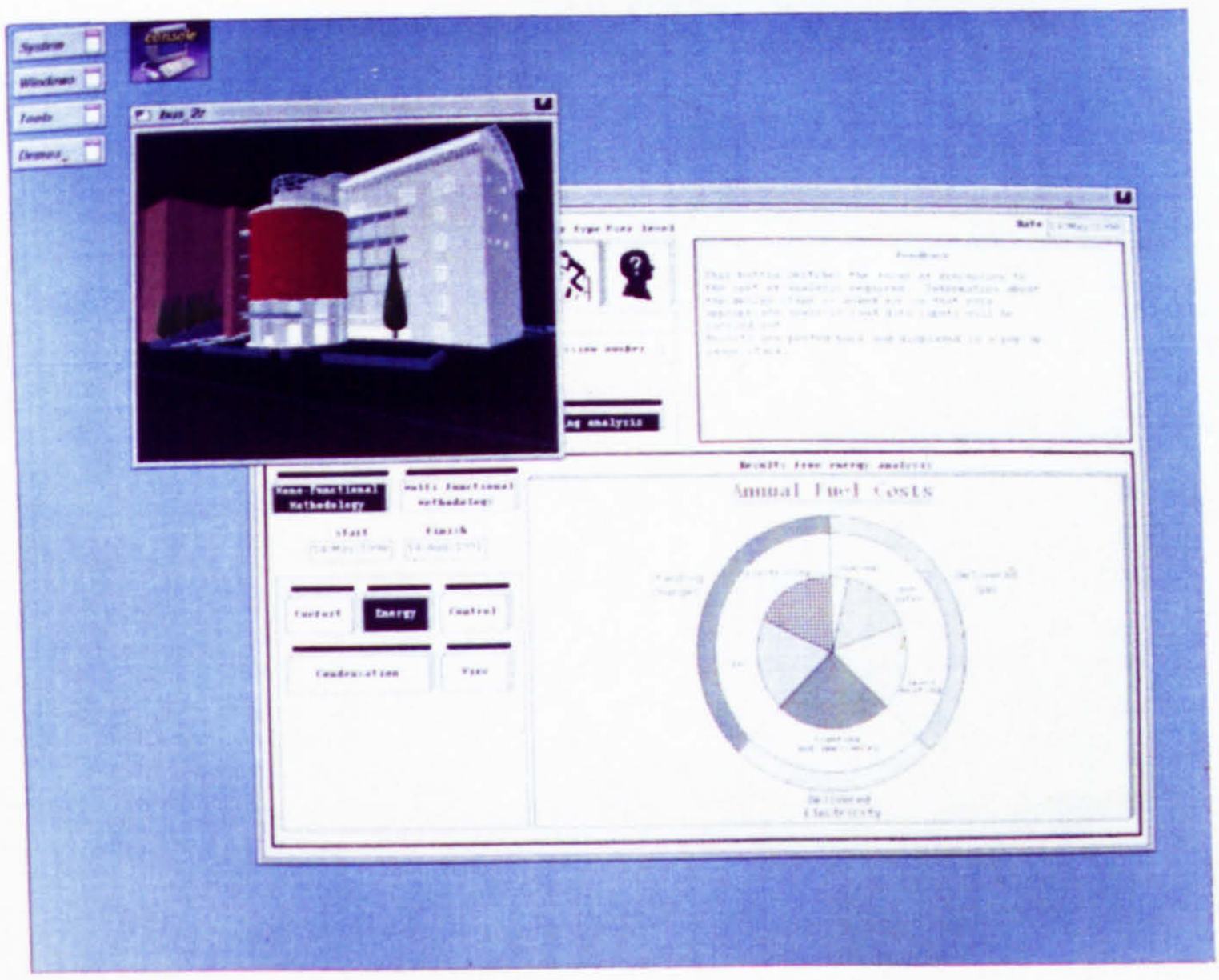


Figure 6.10.2.4. Monitoring time variant data.

This program is a current focus of effort and will eventually enable the user to interact directly with geometrical representations of concepts, formatting and transmitting utterances.

The example provided illustrates how many different application programs may be utilized in a useful and easy manner.

Constructing a user interface using the modules of the IFe may be broken down into three tasks:

- encoding domain specific knowledge
- defining in terms of form sets, a user interface
- task analysis and delegation.

All three points have been covered sufficiently to illustrate the concept of intelligent design assistance. However task analysis and solution synthesis requires additional attention. The issues related to the integration of existing applications (deep models) in this multi-level knowledge system (after Hart 82) are discussed in the following chapter together with result interpretation and presentation techniques.

The following chapter will discus the issues for task analysis and intelligent automatic selection of application programs.

7. INTELLIGENT DESIGN ASSISTANCE

7. INTELLIGENT DESIGN ASSISTANCE

Communication is only one aspect of an intelligent design assistant. Another equally important aspect is assistance with operating application programs and interpreting results. Although the ELAS (Expert Log Analysis System) for carrying out well-log analysis [HART 82, WEISS 82] comes close to the structure outlined in this thesis. There are currently no forms of intelligent design assistance in the construction industry [IBM].

7.1. TRADITIONAL ASSISTANCE

Traditional AI applications have been restricted to consultation systems questioning the user until enough information has been accumulated for the model to provide an interpretation.

As with CAD packages and expert systems in general, the source of the information from the user is often the interpreted results of other packages, filtered through the user. From the user's point of view, accumulating this data, reformatting it and re-entering the results is a time consuming and error prone activity, all of which detract from the real issue involved design. In the same way in which the user interacts with an expert system or other CAD package it is perfectly feasible for applications (algorithms etc) to provide information directly to the domain model, resulting in a distributed problem solving environment.

Although the concept of distributed problem solving is not a new one, E-Mail, for instance has given rise to shared authoring with sub-tasks being assigned to individual computer users often separated by many miles, individual application programs would act as knowledge resources providing contextually relevant expert information on behalf of the user/designer.

The structure of the IFe is an ideal mechanism for achieving these aims. A more consistent interpretation is, however, required.

7.1.1. INTEGRATING EXISTING SOFTWARE - (DEEP MODELS OF KNOWLEDGE)

There are two forms of integration: physical integration and conceptual integration of deep sources of knowledge with surface models.

To physically introduce an application program to the IFe requires a rudimentary communications link. All modules are linked to a parent application by means of a UNIX pipe. The more technical aspects of integration are illustrated in APPENDIX A. However, the knowledge module may be seen as having a single input (stdin) and a single output (stdout). Simply by connecting the output of both processes to the input of the other an inter-process communications link is established. As in the case of the blackboard and dialogue handlers, the application handler can support several processes at any one time. The maximum number is actually eight owing to the limit of sixteen file descriptors (fds) that may be opened for read and write at any one time (two fds per process). Owing to the dynamic, incidental nature of requests for applications during interaction eight processes is more than sufficient.

One of the major flaws of the current implementation of the IFe is the use of low level application knowledge with in the knowledge bases which should deal entirely in abstract terms. For instance the request to the dialogue handler for a map of Europe is coded as:

```
new_dialog(ife_map_prog,"~ife/bin/map","-s -o -e").
```

The capabilities of other IFe modules are presumed and therefore the only reason for dividing the various processes and operations into the modules that have been identified serve only to improve maintenance and to support parallel processing. The IFe has potential for being a truly general design assistant embodying generic surface knowledge together with deep models of knowledge.

It is important that the domain (or surface) knowledge deals only in functional requests and must not deal in application specific instructions. This ensures total generality enabling the integration of any application program.

The application handler and the method of integration enables existing applications programs to be used by the IFe in a very high level manner. For instance the request for a map of Europe would be coded as:

```
present(map,Europe).
```

How the request is interpreted is important. The current implementation of the application handler [RUTHERFORD] maintains a client list of all the available domain specific applications and there capabilities. The application handler matches the task

request against these capabilities and invokes the appropriate application. In order to achieve this it was necessary to wrap existing programs in a shell. For instance the ABACUS program viewer generates perspective images from a topological and topgraphical description of a model. The normal method of invoking viewer is illustrated below:

```
$ viewer
ABACUS VIEWER VERSION XXXX
terminal type >
9
geometry file >
model.vew
:
>>
```

Figure 7.1.1.1. ABACUS viewer terminal interaction.

At this point the application pauses, waiting or the space bar to be pressed before displaying a menu containing options for setting various viewing parameters and generating images in a range of formats and sizes.

The great advantage with ABACUS software is the ability to use application programs in non-graphical display environments. This stems from the need to generate images of large models requiring batch submission on VAX Mainframes.

UNIX shell scripts provide similar functionality and, in the case of viewer, the following script "perspective" was written encapsulating the functionality of the application program with a shell:

```
            # Resource category: perspective_image
            # Design tool: viewer
            # location: /package/abacus/bin
            # instantiation constraints: viewer_model
rm -f /tmp/viewer.pic.
            # If model exists then generate image otherwise abort
if [ -f $1 ]
then
            viewer > /dev/null 2>/dev/null <<.
            -1
            $1
            O
            /tmp/viewer.pic
            B
            .
            echo    "perspective_image    $1    /tmp/viewer.pic    complete"
            .
else
            echo    "perspective image    $1    model_not_found"
fi
```

Figure 7.1.1.2. Perspective image script

The script performs all the tedious invocation of the application obtaining the name of the model from the scripts command line arguments and redirecting all error messages and output from the program to /dev/null. The corresponding request for a perspective image is:

generate(perspective_image, model_name).

The application handler simply matches the task against the application category of each client and forks the process. The perspective script issues the message:

perspective_image model_name complete
or perspective_image model_name model_not_found

which is passed back to the knowledge handler and dealt with accordingly (see interpreting results).

The combination of application and shell or interpreter is referred to as a knowledge resource, a conceptual view illustrated in figures 7.1.2.1. and 7.4.1

7.1.2 RESOURCE INTEGRATION USING PROTOCOL INTERPRETERS

Each resource within the design environment (figure 7.4.1) consists of the application itself together with an interpreter (figure 7.1.2.1). The interpreter is responsible for translating application data and control comands into system data and vice versa. Once created, the interpreter enables existing software packages to be integrated within a coherent architecture.

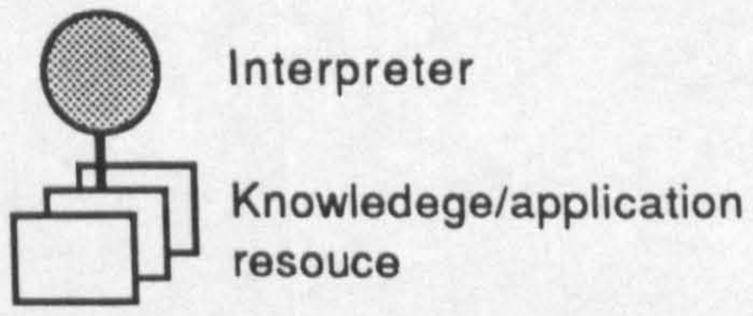


Figure 7.1.2.1. Integration of existing software package using protocol interpreters

Such an approach enables design resources (applications/expertise) to be encoded and validated in isolation and then inserted into the system with little or no modification. The system can therefore evolve in step with technological advances without incurring expensive maintenance costs, bringing new design methodologies and techniques to the design profession more quickly than otherwise possible.

Perhaps the only major problem with this approach is the doubling of the number of required system processes. Where source code for a particular resource is

available a reasonable solution would be to integrate the interpreter within the application and invoke it with an appropriate command line argument. Many of the in-house applications have been designed around a multi-platform I/O library enabling applications to run on a variety of terminal types. Simply by adding another "terminal" emulater; resource (consultant) interpreter, integration could be achieved without any additional overheads.

With this integrated approach a fundamental problem which must be overcome is one of redirecting the standard I/O function calls. The easiest method is to create the following function pointers:

```
int (*Read)();      /* -> input environment   */
int (*Write)();     /* -> output environment */
```

Figure 7.1.2.2. I/O redirection pointers.

Depending upon the mode of operation these functions would point to either the standard I/O functions or those relevant to the selected I/O environment. Figure 7.1.2.3, below, provides a simple switching procedure.

```
void set_io_environment(mode) int mode;
{
        switch(mode)
        {
                case CONSULTANT:
                        Read = ConsultantRead;
                        Write = ConsultantWrite;
                        break;
                case TEKTRONIX:
                        Read = TekRead;
                        Write = TekWrite;
                        break;
                default:
                        Read = read;
                        Write = write;
                        break;
        }
}
```

Figure 7.1.2.3. I/O redirection

An appropriate binding should be provided for other languages such as FORTRAN.

7.1.3. A GENERIC RESOURCE DESCRIPTION

The actual process of integrating applications may be generalised and the client description required by the application handler quantified by the following generic attributes (also identified by [HAMALAINEN 88]):

i) a *unique identifier*: used to reference the knowledge resource in the process table and provide tangible symbolic representation for the designer.

ii) *resource category*: categorises and identifies the domain of the knowledge resource. This is used either by the knowledge base or scheduler to determine an appropriate resource to satisfy a particular goal.

iii) *description*: a textual description of the methodologies employed by the resource. This enables the designer to choose a particular design methodology over another.

iv) *properties*: a description of the conceptual vocabulary of the knowledge resource. These words are used to fine tune the model after initialisation. The list of inputs and outputs is only required as a design specification for the interpreter.

v) *instantiation constraints*: these are required to initialise the resource and are also used in the decision making process.

vi) *resource name*: the actual name of the resource or application

vii) *resource location*: where the resource is located within the directory structure of the host processor.

The current method embodies an expert users (iv) knowledge of how to operate the application (v), encoded within the shell script, while the location of the resource is held in a much more subtle form namely; the PATH environment variable.

7.1.4. AUTOMATED TASK ANALYSIS

Rather than encoding scripts to deal with specific tasks by identifying the actual processes to utilise an application it becomes possible to automate the selection and invocation of knowledge resources. The actual stages may be quantified in the following terms [KAEMMERER 86]:

- *problem analysis*: The current task is evaluated against the capabilities of available resources and decomposed into sub problems if necessary.
- *solution planning*: from detailed knowledge and experience application programs appropriate for solving the task, either in part or whole, are brought together in a general problem solving strategy. Many strategies may be possible, depending upon the resolution of the problem decomposition and available data.
- *plan evaluation*: From the range of solution strategies the most appropriate, meeting any imposed constraints (such as execution time and accuracy), is selected.
- *plan instantiation*: for the selected plan, data required for each stage is gathered, referred to as value acquisition
- *instance execution*: once completed the plan may be executed
- *results gathering and interpretation*: On completion results are gathered and interpreted.

7.1.5. PROBLEM ANALYSIS AND SOLUTION SYNTHESIS

The selection strategy described is suitable only for problems which may be solved by a single invocation of one resource. However if a particular problem cannot be resolved by the available resources a different selection strategy must be employed.

The unresolved problem must be analysed and decomposed into discrete manageable sub-problems. In order to achieve this, low level knowledge of the functional capabilities of each resource is required. Each sub task is matched against the functional components of all the available resources. A number of different solutions may be generated and therefore each must be evaluated against some pre-defined constraint.

Once an appropriate solution has been selected an executable sequence must be prepared. This involves extracting relevant data from the product model and

formatting it for each resource in the chain. Dialogue frames are completed by extracting data from the product model. Results from each resource must also be formatted and filtered to subsequent operations or other participating resources.

Experience has shown that one person implements interface templates, knowledge bases and resources for a particular sub-domain. As all possible task problems are anticipated by the developer the problem analysis and solution synthesis process is performed manually as part of the user conceptualisation preparation. Solutions to pre-defined problems are synthesized and encoded in the form of appraisal methodology scripts and stored along side individual user conceptualisations. Scripts are chosen either by the resource handler or directly by the designer as illustrated in the example, figure 6.10.1.1 (IFE demo). Each appraisal methodology represents an experts choice of application sequences. By encoding low level knowledge of individual resources (explicit data requirements) together with appropriate inferences for problem analysis and solution synthesis, it is feasible to automate the selection process. Kaemmerer [KAEMMERER 86], provides a convenient summary of such a selection process which is illustrated bellow in figure 7.1.5.1.

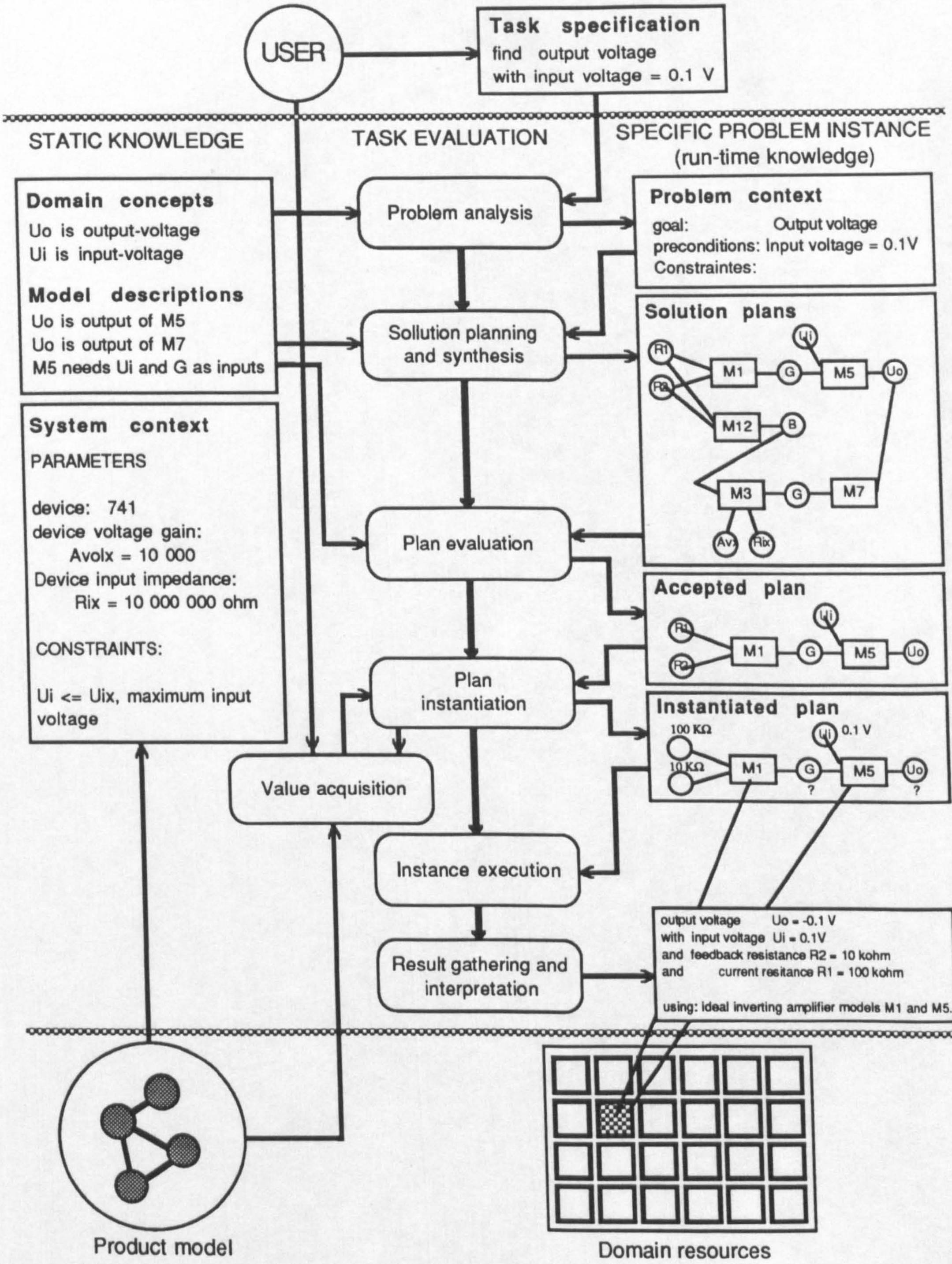


Figure 7.1.5.1. Graph-oriented knowledge representation and unification technique for automatically selecting and invoking software functions [KAEMMERER 86].

[KAEMMERER 86] conveniently summarises the process, Figure 7.1.5.1 placing it in the context of the design of a amplifier design. The solution plan is not confined to methods contained within a single application. Intermediate results from methods of one application may be piped through an appropriate conversion filter to those of another.

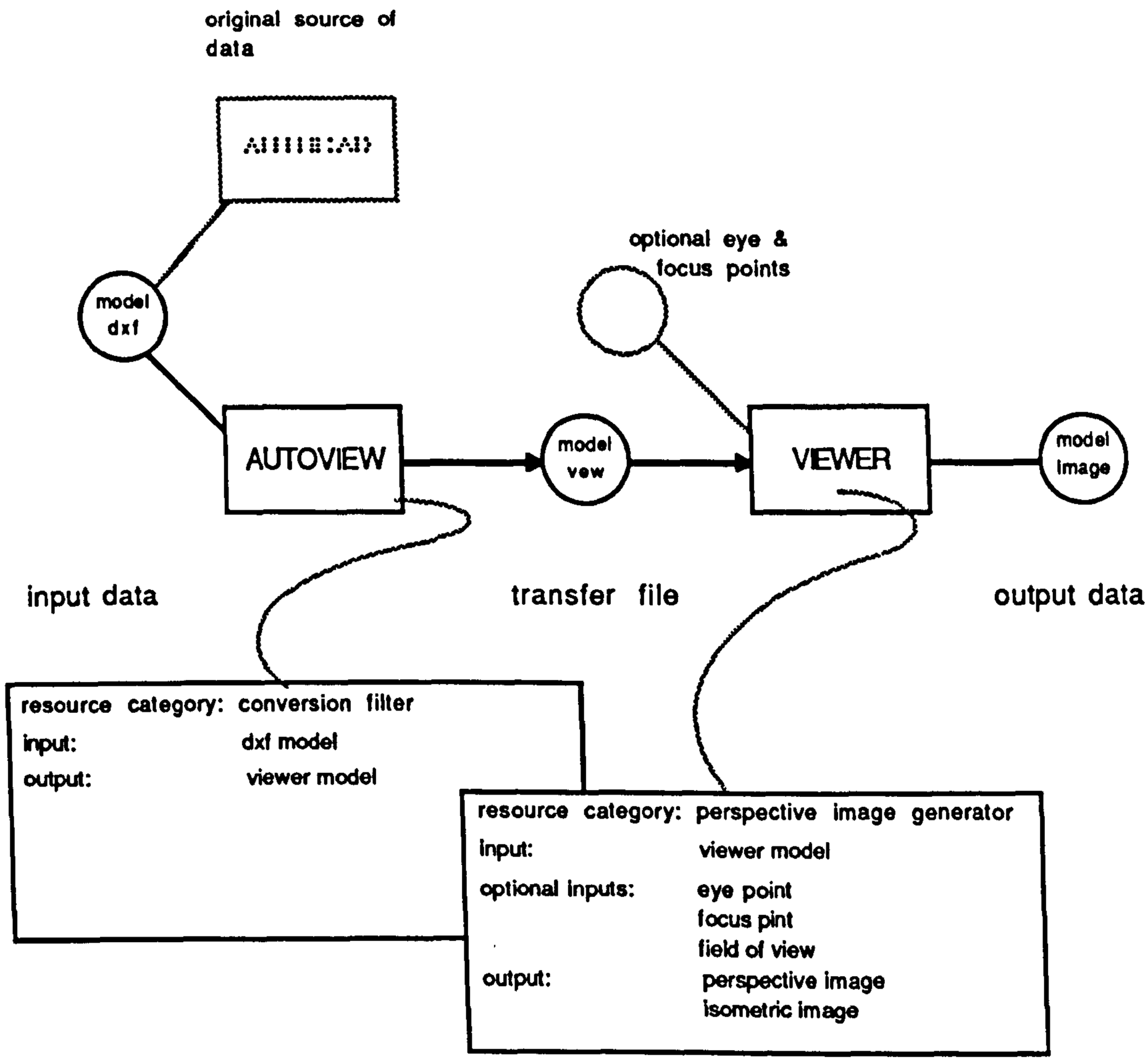


Figure 7.1.5.2. Solution plan for the task "generate perspective image of model from a dxf representation" of a 3D model.

Figure 7.1.5.2, above, illustrates the solution plan that would be compiled in order to generate a perspective image from a dxf representation of model (created using autocad), which is being imported into the neutral (ABACUS) format used by the IFe.

The resource handler firstly scans through its client list looking for a resource of the "perspective image" category. Once found the input requirements of this resource would be matched against the available data. If there is a conflict between these two data sets the resource handler backward chains through it's client list matching the outputs of each resource with the data requirements of the original task resource,

repeating the process until a match is found. In the case, illustrated above, the "autoview" program, written by Charles Chen of Computer Science, converts a DXF model to a viewer model file. The transfer file, figure 7.1.5.2, would normally be a temporary scratch file which would be deleted once the task is completed. In the case of the import option (Section 6.8.1 4) this file would be the original task request and the perspective image generated as a completely separate task.

In addition to storing input and output requirements of each available resource, information regarding the input and output formats is also required. For example the above *autoview* program, instead of creating a file, may write the converted data to the standard output stream, which would require either a pipe between resources (if the second was capable of reading model data from standard input, or a redirection to a file. The filter may also require data to entered from the standard input stream in which case the original source data would have to be CATed. Also whether references to source files may be passed as command line arguments or are required as answers to program prompts. Some of the possible invocation sequences are illustrated below:

filter model.dxf model.vew; **perspective_image** model.vew model.image

cat model.dxf | **filter** | **perspective_image** > model.image

filter < model.dxf > model.vew; **perspective_image** model.vew model.image

Figure 7.1.5.3. Possible invocation sequences for the generation of a perspective image from a DXF source file. Processes and programs are shown in bold type; | = unix pipe, <> = input and output redirection, ; = command termination.

The MICON Sythesizer Version 1 (M1) [BIRMINGHAN 87] is a knowledge based system written in OPS\83 [PSTI 86] used to produce a complete small computer design from a set of abstract requirements [BIRMINGHAM 89]. The system, figure 7.1.5.4, works in conjunction with an automated knowledge aquisition tool, CGEN (Code Generator) which aquires knowledge of how to build and when to use various computer structures (legal configurations of hardware: interconections between a micro-processor and memory array, for example). The operational knowledge required by the system is obtained from domain experts. The end-user is able to design computer systems by specifying abstract functional requirements and operational constraints.

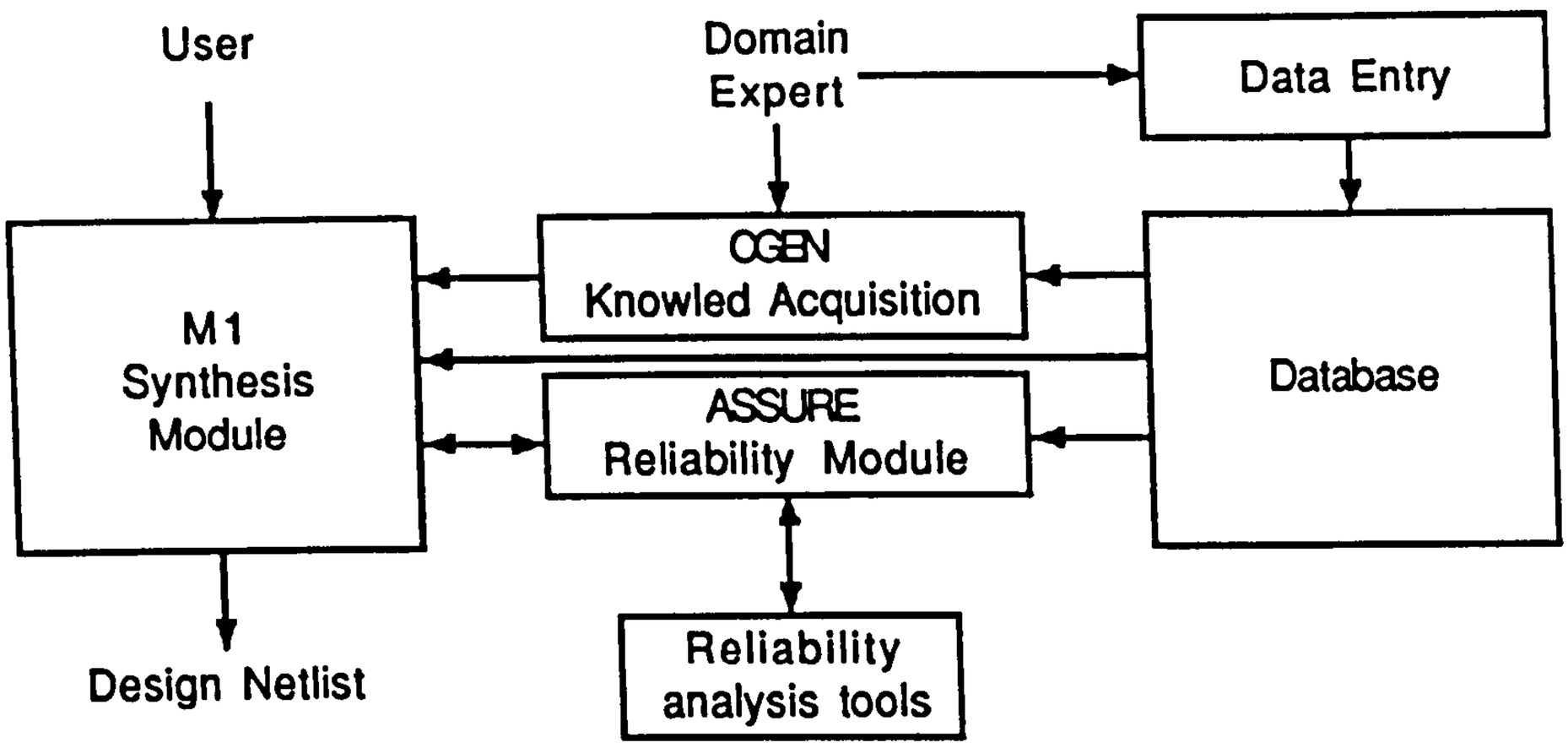


Figure 7.1.5.4. The MICON System for designing digital computer boards (from [GUPTA 90]).

The MICON system automatically assembles and executes solution graphs which may be be manipulated at various levels of granularity. Such an approcah may be readilly adopted in future developments of the design assistant, although this automated approach is only really valid if resource modules are being constantly interchanged; simplifying the integration process by avoiding the coding of shell scripts which is also one of the methods currently employed.

7.1.6. EVALUATING SOLUTION PLANS

Each solution strategy may be evaluated against a pre-defined set of constraints. Constraints such as execution time and accuracy may be moderated by operational context. For instance a wire line perspective image or simple guiding U-value calculations may be acceptable during preliminary design stages while, towards the final presentation and evaluation stages, a fully rendered image and a complete energy simulation may be more appropriate in terms of computational time. A number of applications may embody similar methods and there is also a need to satisfy constraints for a range of competitive resources.

7.1.7. SELECTING AND SCHEDULING COMPETITIVE RESOURCES

With time it is envisaged that a number of functionally similar resources will exist within a particular design environment; each offering differing services; rule of thumb to full simulation.

The resource handler compiles a list of potential clients within a particular domain. As a request for a resource category is received by the resource handler a secondary list of those resources able to respond is compiled. Each resource is then invited to bid for the task. Bids are based upon instantiation conditions. The example bellow (figure 7.1.7.1) illustrates the bidding process for the generation of a perspective image.

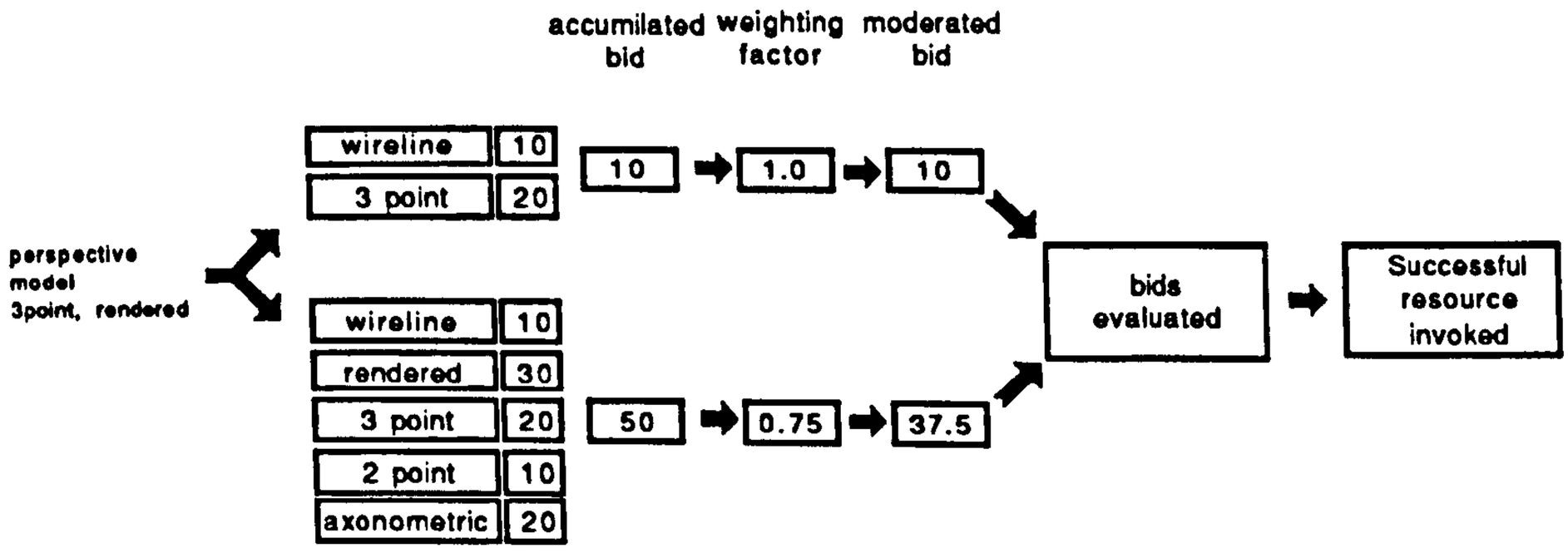


Figure 7.1.7.1. Selecting competitive resources

A request for a perspective with a number of constraints is made. Each of the prospective clients submits a bid; a summation of the methods rating for each constraint moderated by a weighting factor which is determined by the size of the model and the accuracy of the methodology employed. The bids are submitted and evaluated by the resource handler. The successful resource is invoked.

Scheduling and transactional protocols and queues are usually integrated within the general problem solving functions of the blackboard. As will become apparent, this functionality has been separated from the blackboard architecture.

7.2. RESOURCE MANAGER - SOLUTION SYNTHESIS ENGINE FOR AUTOMATED KNOWLEDGE ACQUISITION

A general problem with the UIMS philosophy is one of total encapsulation. Applications are hidden from view by an interface layer. For most applications this is not a problem. However, if a system is to be used as a design tool there is often no

method of manipulating the applications on a low level basis. The user often has little direct control of the applications involved or may not even be aware what applications are being employed.

The actual processes of inter client communication dictate that resources should be known in order that the end-user is aware of what appraisal methodologies are being employed, allowing the designer to interrupt packages during execution if necessary. This is often an advantage when the design solution changes during analysis or when it is realised that the package is working with an incomplete or inaccurate description of the model. Rather than waiting for execution to complete before re-defining the data the user would be able to interrupt the package just as in normal conversation with a design consultant.

How interaction proceeds after the interruption depends upon how well event handling is managed by the application. Ideally the application should return to a stable state and await instructions from the user, thus allowing small changes to be made to the model as opposed to aborting execution altogether and re-loading the entire model from scratch.

The Resource manager in the IFe does not currently allow interrupts from the user to filter through to target or incidental resources, although applications may be told to abort. This instruction is only acknowledged when the application is in a listening state.

The image below illustrates a refined resource handler capable of displaying active resources and allowing the user to interrupt them directly. This facility would perhaps only be presented to expert users, rather than novice users who would be too engrossed in the input of data to be unaware of the need for such a facility.

7.2.1. ENCAPSULATION

One of the main problems with traditional interface design principles, identified in chapter 2, is one of encapsulation. While, by providing a transparent operational layer between the user and target resources, the needs of the novice user are satisfied, there is a need to provide expert users with almost direct access to the design methodologies encoded within an application, while maintaining a degree of isolation between the user and the obtuse operational complexity of the specific design tools.

7.2.2. CUSTOMIZING SOLUTION PLANS

The problems associated with encapsulation would be resolved by providing a tool to directly manipulate or customise solution plans. Such a tool would provide an end-user with direct access to the solution graph generated by the resource handler or an expert end-user, perhaps in a graphical form, figure 7.2.2.1.

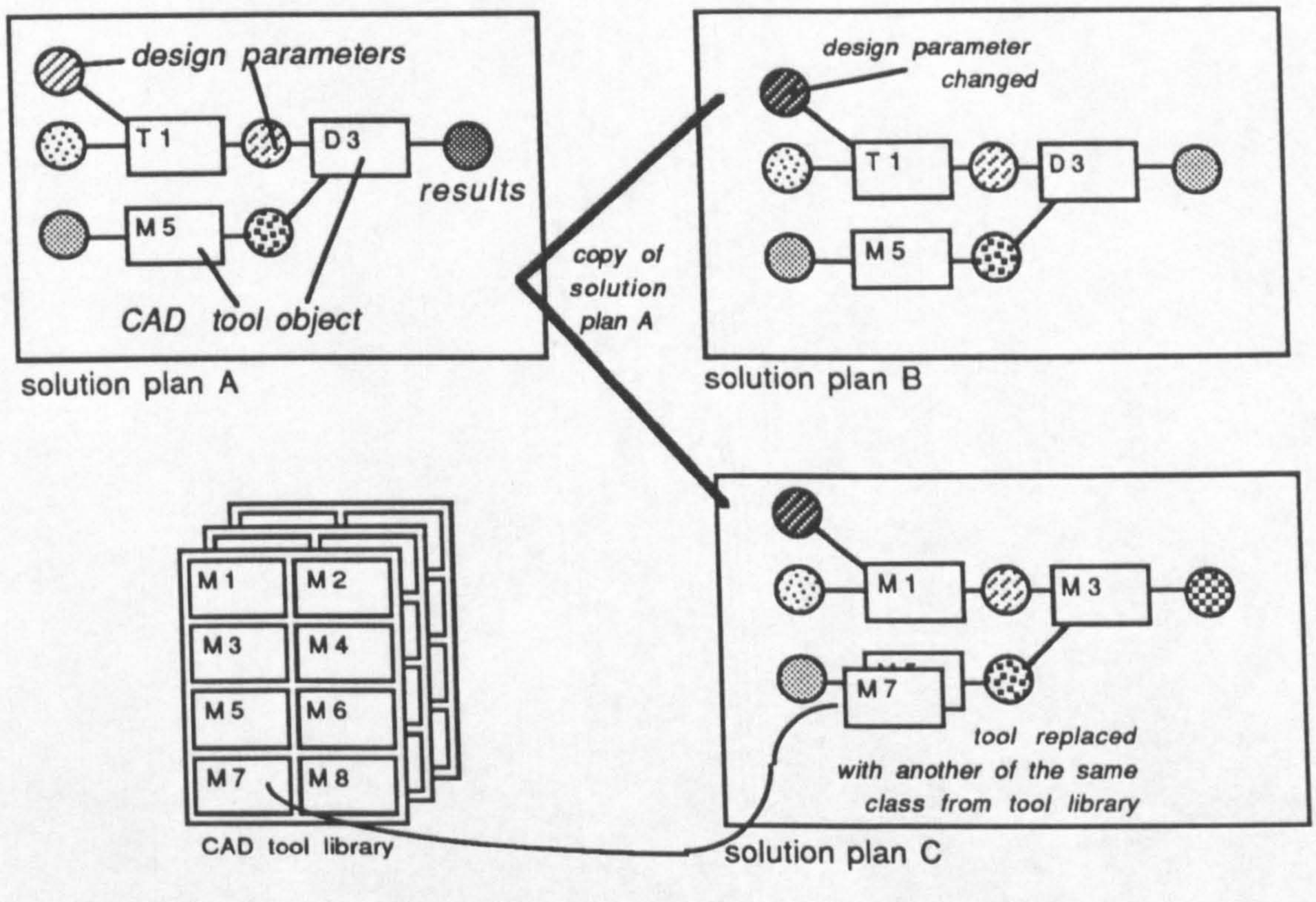


Figure 7.2.2.1. Customising solution plans.

The customisation tool would enable existing solution graphs to be copied, solution plan A and B, figure 7.2.2.1, and provide simple mechanisms to enable the user to change design parameters (such as the percentage glazing of a wall), solution plan B, and to replace design methods with others of the same category, solution plan C. As design practice is continually changing this is a useful facility.

Modified solution plans may therefore by activated (or run) in order to test tentative or alternative design solutions. Although providing the end user with direct access to the functionality of a series of design tools, the approach still maintains a general user-tool abstraction interface. The above approach may also be applied to objects (graphical for instance) with relationships defined between them.

7.2.3. REVISION CONTROL AND TRACKING

The benefits of solution plan manipulation (customisation) will enable a design environment to respond to shifts in design standards and regulations. Encoded methods may be replaced with an up to date module of the same resource category, illustrated by solution plan C, figure 7.2.2.1.

As solution plans directly effect the outcome of the design solution, it is important, when allowing end-users to modify solution plans directly, to keep records of those changes. The system should force the user to provide reasons for replacing design methods in order to provide documentary evidence of the decision making processes.

An access hierarchy should also be established preventing chaotic manipulation of solutions by all participating users.

7.3. ABSTRACT REFERENCING

The amount of data generated in CAD systems can be enormous. Passing large quantities of data through UNIX pipes is slow and would seriously impair the performance of the system and the designer (this is a serious problem with the X message passing mechanisms). To overcome this fundamental problem, abstract references to data sets are passed instead. These references are usually filenames of bitmaped images or large databases held somewhere on the system environment. Shared memory may also be employed but is a facility restricted to the host platform. This allows resources (adopting idiosyncratic schema) to extract only the data relevant to their needs and also enables the designer to manipulate this data symbolically. The demands imposed upon the blackboard are also reduced and provides a convenient fail safe recovery mechanism if the system should crash.

7.3.1. FORMATTING RESULTS.

Results from resources should be translated into general statements such as "perspective completed_image of_model" or annual_heating costs. Such statements are posted onto the blackboard, interpreted by the knowledge bases and presented to the user in a manner suited to their user type and level of experience. The example below illustrates how the annual heating costs, predicted by an energy analysis resource are interpreted. Re-phrasing results in this manner is particularly important in situations involving numerosity judgement.

```
energy(_Gains):-
        format_results(energy_input, _Gains).

where; _Gains contains a list of values:
        Incidental gains, 52.3 GJ, Solar gains, %$.0 GJ,...

format_results(energy_input,_gains, architect, novice):-
        generate(energy_input,pie_chart,"Energy Input",_Gains).
```

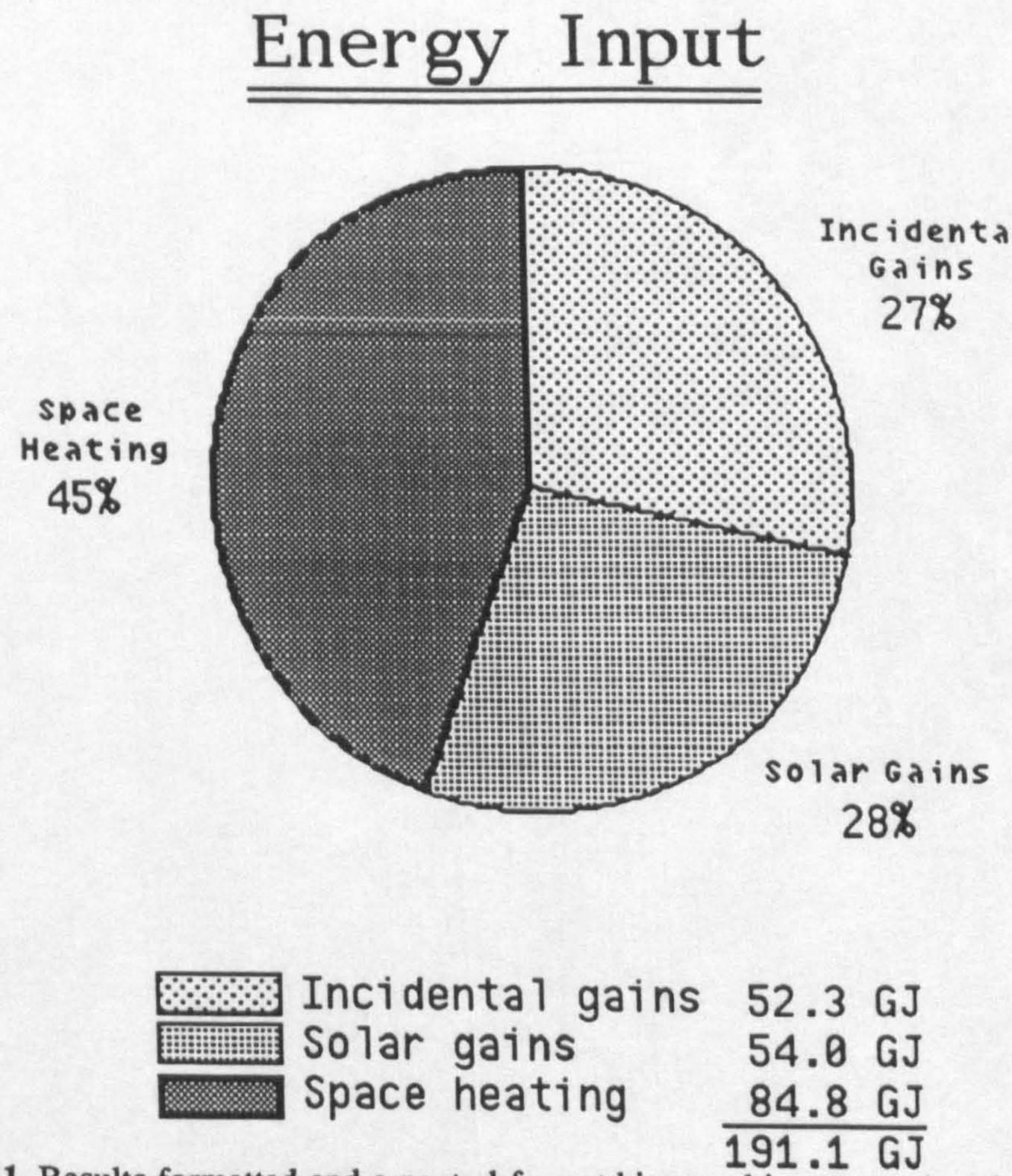


Figure 7.3.1.1. Results formatted and a neutral format bitmaped image produced, the results are passed back as an abstract reference to the image file.

where; pie_chart is the requested style of output,
"Energy Input" is the title for the chart.

```
format_results(energy_input,_Gains, engineer, expert):-
        generate(energy_input,table,"Energy Input",_Gains).

enery_input(_Gains,_Interpreted_results):-
        tell_user(analysis_results, _Interpreted_Results).
```

7.3.2. CONSTRUCTIVE CRITICISM.

The example above illustrates a relatively straight forward method of providing different presentation styles. It is also possible to provide subjective interpretation of the results, for example:

- The atrium is over heating. I suggest that you consider increasing the number of air changes per hour to 3.

Or

- This infringes building regulation xyz. In order to comply with this regulation you must increase the width of the stair by 0.5m and use self closing fire doors to all exits.

Data may also be interpreted in other ways indicating potential problem sources, as illustrated by Figure 6.10.2.1.

7.4. CONCEPTUAL OVERVIEW OF AN INTEGRATED DESIGN ENVIRONMENT

Rather than returning to the diagram of the IFe the computer based design environment may be viewed in a more recognisable form (figure 2.1.1). Figure 7.4.1, provides a conceptual overview of the design environment. Design resources are arranged hierarchically to illustrate the types of resources necessary for each stage of the design process, although the order in which individual resources may be utilized depends entirely upon the designer.

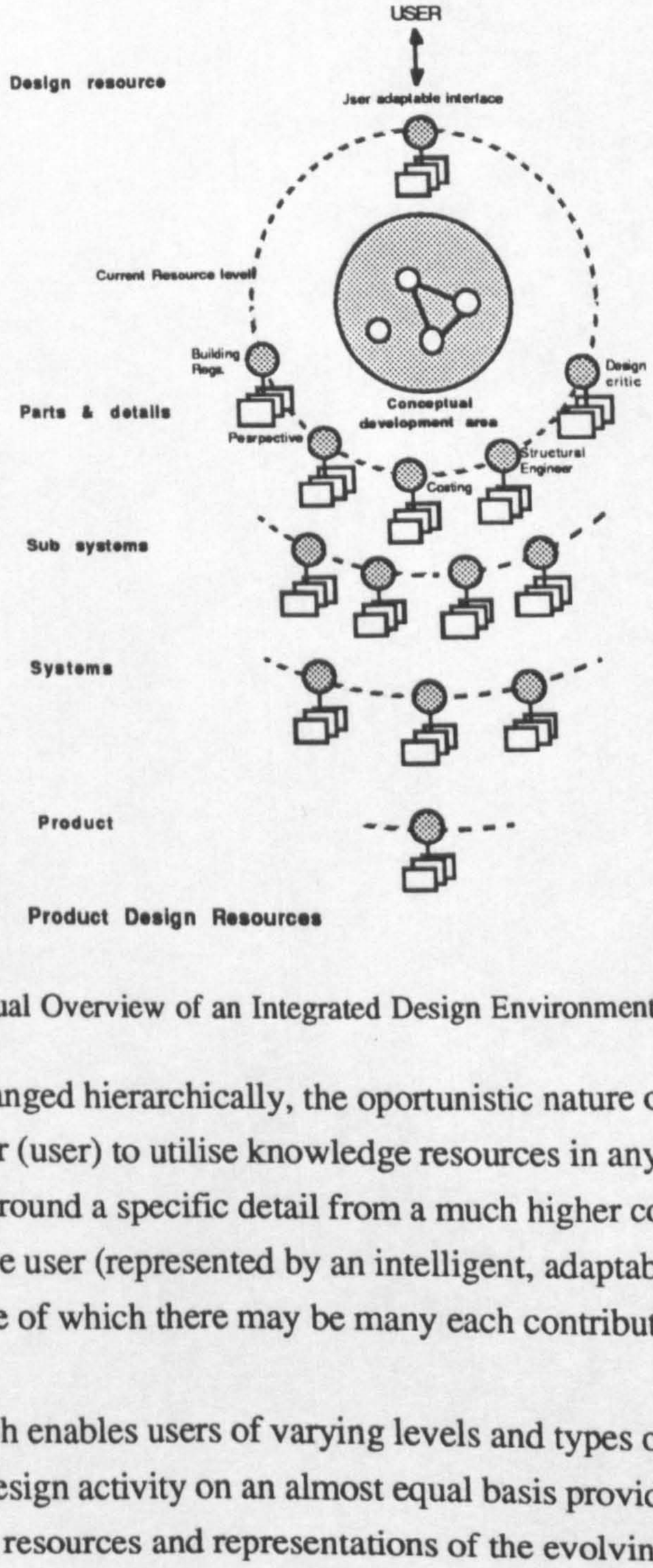


Figure 7.4.1. Conceptual Overview of an Integrated Design Environment [RUTHERFORD 89]

Although arranged hierarchically, the oportunistic nature of the overall system enables the designer (user) to utilise knowledge resources in any order; generate a complete solution around a specific detail from a much higher conceptual level. Within this environment the user (represented by an intelligent, adaptable front end) is viewed as a design resource of which there may be many each contributing to the entire design solution.

This approach enables users of varying levels and types of expertise to participate in the design activity on an almost equal basis providing access to a whole host of knowledge resources and representations of the evolving design concept, each

utilizing a knowledge structure to suit a particular aspect of the design decision-making activity: a kind of multi-lingual design environment. Issues such as access privileges must also be considered.

A fundamental issue, other than inter resource communication, is data management. Product information is currently held on the blackboard as a series of Tuples [IFE 89] (concept value) which are time stamped together with the source of the information (user, application, inference). MacRandal has provided a number of mechanisms for extracting information from the model (see the known predicate chapter 5).

Although recent developments such as the ISO standard STEP (STandard for Exchange of Product data)/PDES(Product Data Exchange Standard (American counterpart)) (and GARM [STEP 88] focus on the more relevant issues of the interchange of non spatial product models or complete descriptions of artifacts, the IFe must deal with many types of knowledge, this simple method of storage and retrieval is ideally suited to an integrated environment such as this allowing each resource to create their own schema.

There is however a need for a more structured product model if the IFe is to be utilised as a design environment.

7.5. MODEL DESCRIPTION

Product models are defined symbolically using objects. An object consists of a class identifier, a unique instance label, together with a number of attributes (figure 7.5.1).

Each object also carries with it a generic object manager responsible for the creation and retrieval of attribute slots. Within the object data is stored as attribute/value pairs (figure 7.5.1). Resources can request attributes by name or ask for a complete update on the current object description. Resources acting as attributers may also replace attribute values or create new slots. The object is therefore a generic, dynamically configurable data cell.

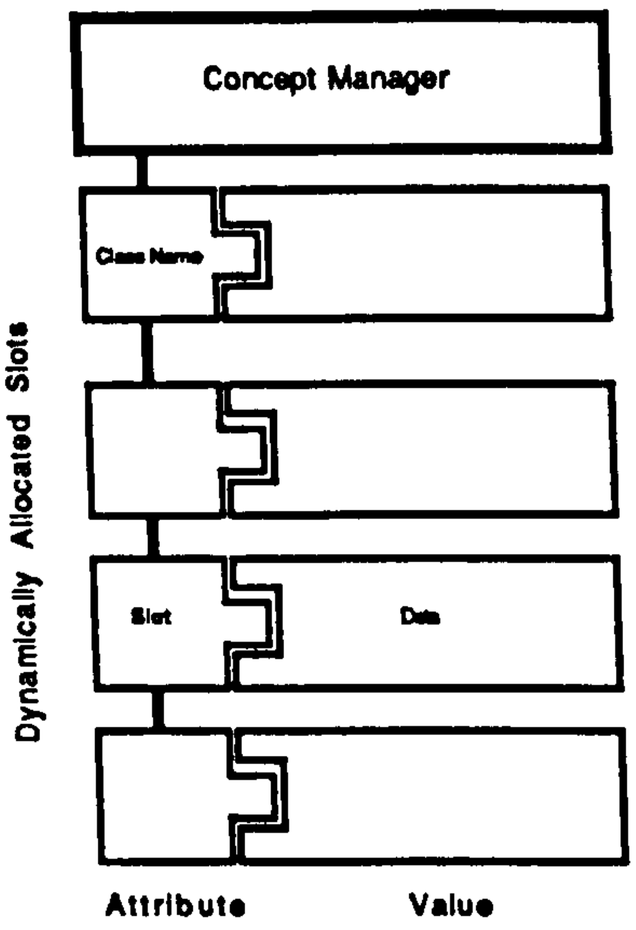


Figure 7.5.1. Generic Objects: Dynamic memory allocation

The same generic communications protocol used by all other resource interpreters may be extended to include object class names:

object name:method:data

or [class name]:method:data

where method is interpreted by the object manager or other resources. Where class name is used all objects within the referenced class respond to the method.

Using these basic (generic) objects together with object interpreters multiple interpretations of the same data are possible (figure 7.5.2).

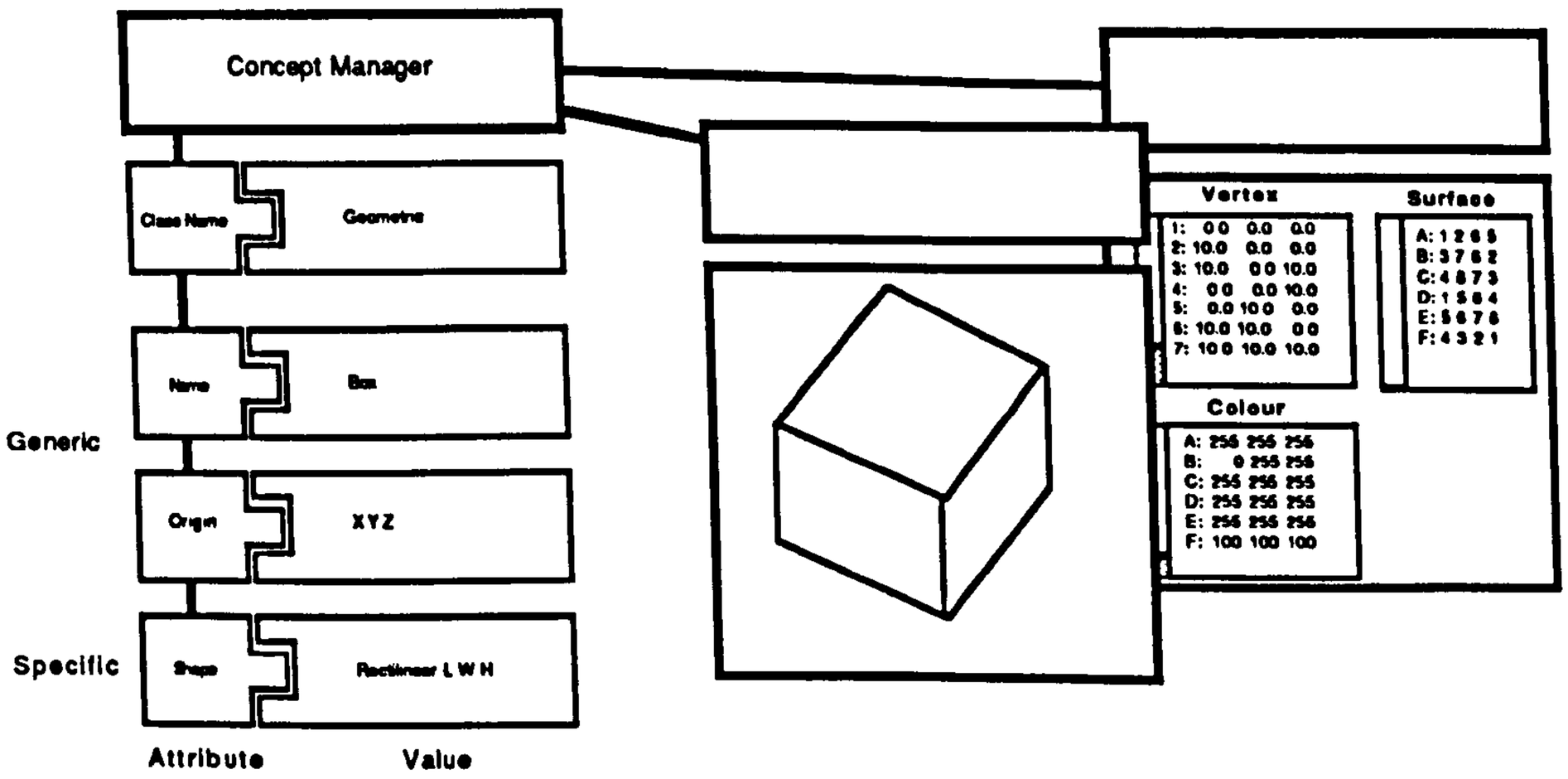


Figure 7.5.2. Concept Interpreters

This approach may also enable the end-user to communicate in several different languages (a music score, for example, may be interpreted as a built form and vice versa using synonyms (or shape grammars) for common concepts in each context).

This form of storage would enable a designer to dynamically build a conceptual (hierarchical) framework for a particular design solution by defining data cells (representing concepts and meta-concepts) and defining relationships between object attributes, figure 7.5.3

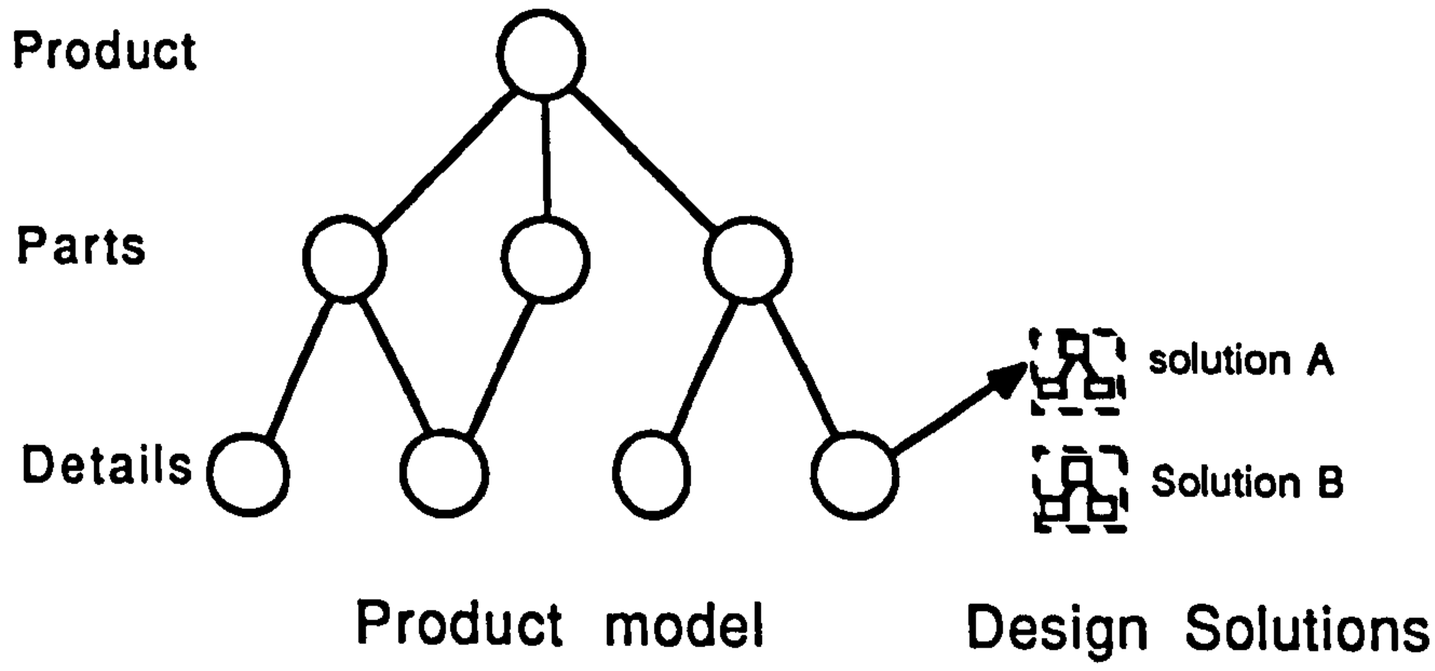


Figure 7.5.3. Abstraction hierarchy

This would enable numerous design solutions to be defined and tested within a complete model. The blackboard would have to be able to merge or fuse multiple solutions.

In order to accommodate the different schemas of each design tool, product information should be as comprehensive as possible. However, one should avoid redundant information which would result in consistency problems. Standards such as IGES and DXF rely heavily upon the interchange of 2 and 3D graphical information. Non spatial descriptions would normally be held along side such descriptions. Information which may have a graphical representation (held within the same file) must be updated when either the graphical or non spatial data is manipulated (who is responsible for consistency checking?). It is suggested that a product model should contain parametric descriptions of entities enabling a single definition to be interpreted (at run time) by different systems, figures 3.7.1.2.1 and 7.5.2, and thus reducing the need for consistency checking.

7.5.1. OBJECT RELATIONSHIPS

In some instances, in particular those related to geometry the product model alone is insufficient for the total description of a model. The dimensional control of buildings and components has always been an important issue; ensuring that elements fit correctly and are located accurately. For example the position of a geometric body may be defined in several ways; absolutely, in world coordinates, or relative to other objects or may be defined using a proportional system, figure 7.5.1.1..

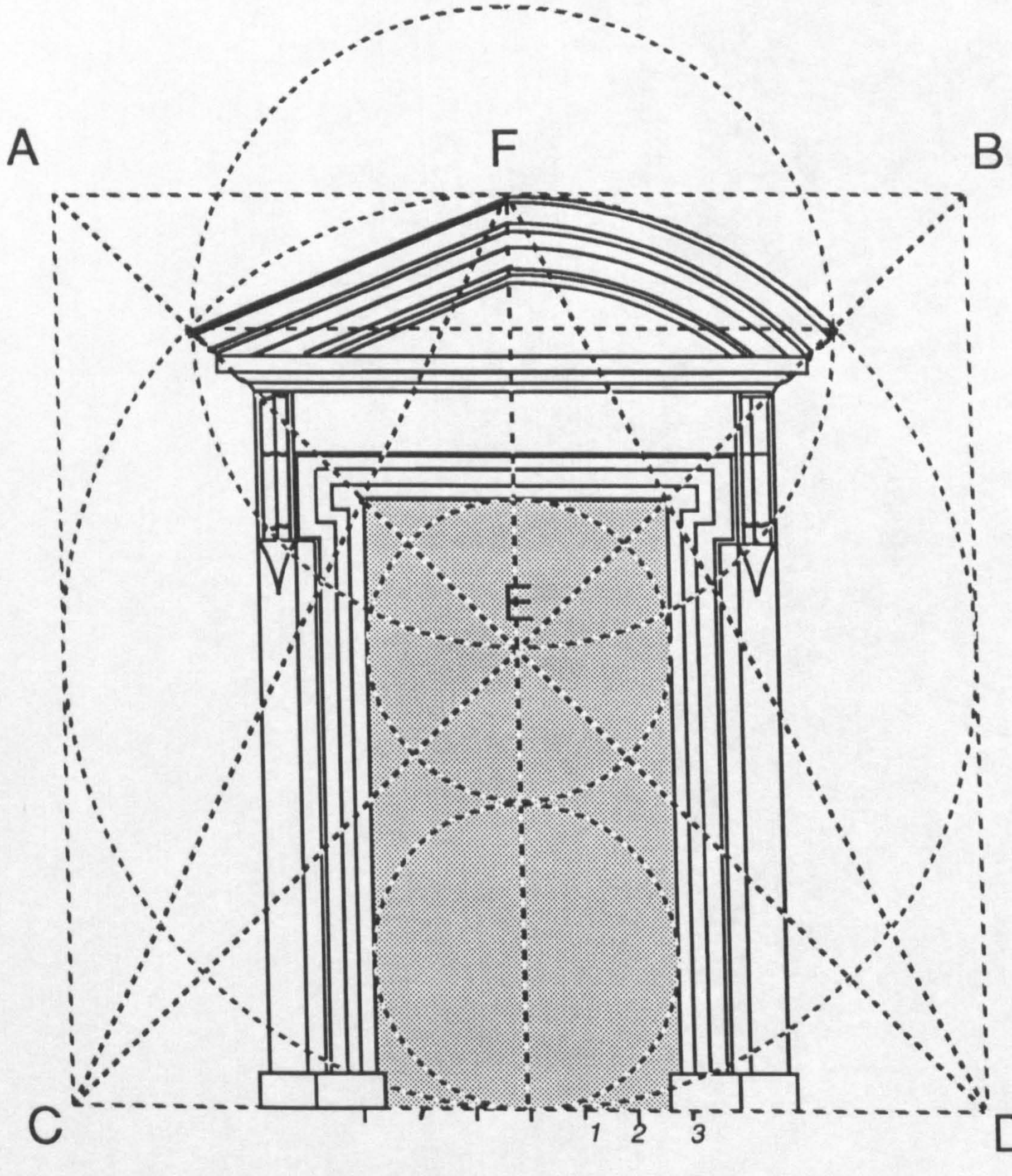


Figure 7.5.1.1. Georgian Doorway, Dimensions fixed by a proportional geometric system (from An introduction to Dimensional Co-ordination, HMSO, 1978)

A network describing the 'physical' relationships between such objects is, therefore, also needed if the model is to be more than just an elaborate reference to catalogued parts.

Example:

A wash hand basin fixed to a wall.
If the wall moves the wash hand basin with fixtures should also move together with services (plumbing).
If the wash hand basin moves the wall should stay put.

So in addition to the physical relationships between objects, dominance and submissive characteristics should also be held.

Currently, a simple hierarchical network is used within the IFe to maintain the product model together with references to object instances. In order to produce a comprehensive representation of an 'organic' product such as a building, more powerful abstraction mechanisms found in the General AEC Reference Model (GARM) (a subset of the ISO STEP) for instance would be required. The emerging standard (recently rejected because the documentation presented to ISI exceeded the maximum one thousand pages) facilitates multiple solutions by the interchange of Technical Solutions to Functional Units (or specifications), figure 7.5.1.2.

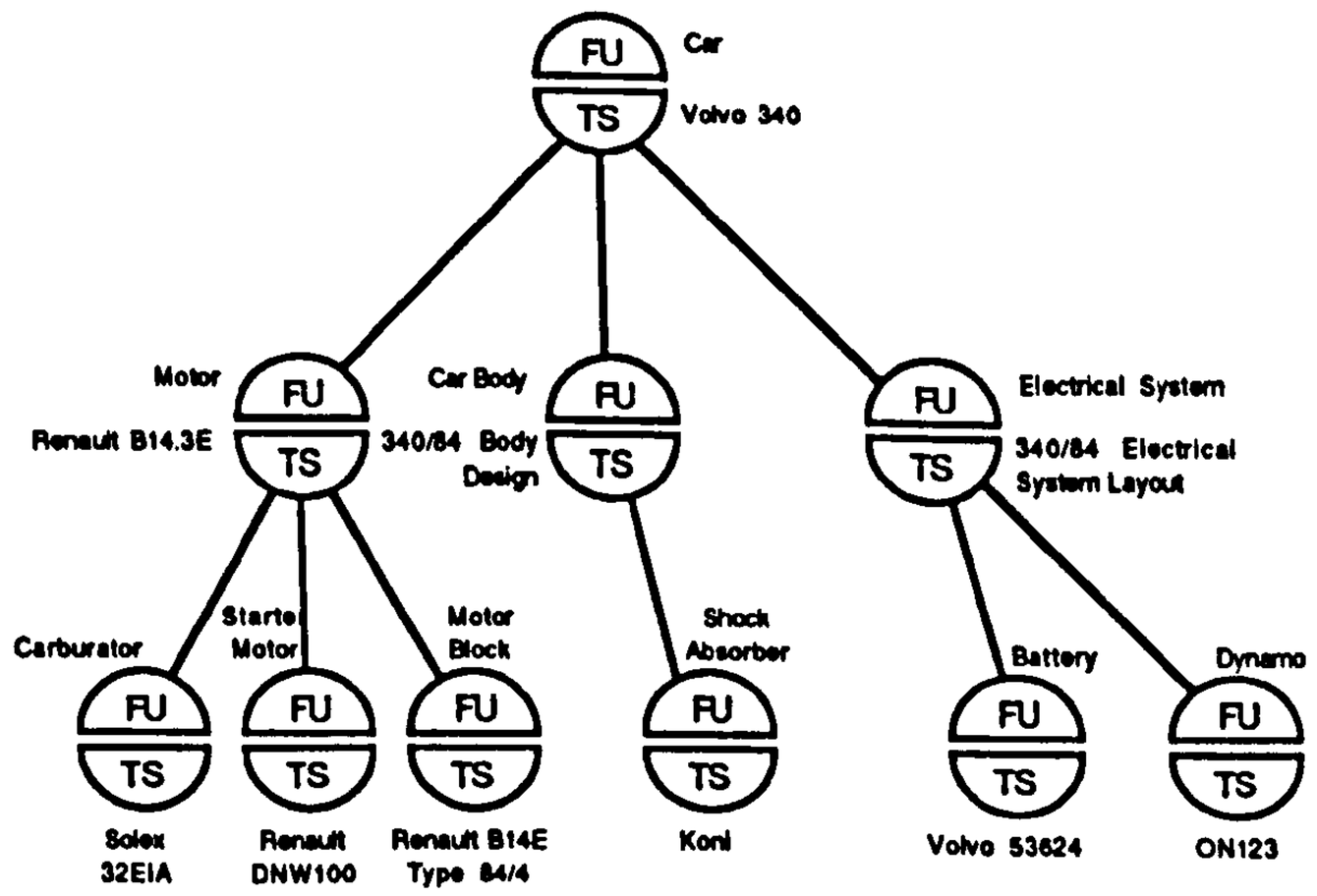


Figure 7.5.1.2. Example of the decomposition of a product (car) by means of Functional Units and Technical Solutions.

A Functional Unit (FU) and a Technical solution (TS) are represented as semi-circles. A car is a Functional Unit (it has a function and it can be defined by means of a set of requirements). A Volvo 340 is a possible Technical Solution which may

satisfy the requirements. The Volvo itself contains certain Functional Units, which may have been produced by other manufactures. A complete product model is therefore regarded as an aggregation (composition) of components with certain functions, which can be fulfilled by various (interchangeable) technical solutions. (from [STEP 88] page 18).

The STEP model supports both top down, functionally oriented, and bottom up, technically oriented, design procedures, figure 7.5.1.3, by separating a design solution into its conceptual and detailed parts.

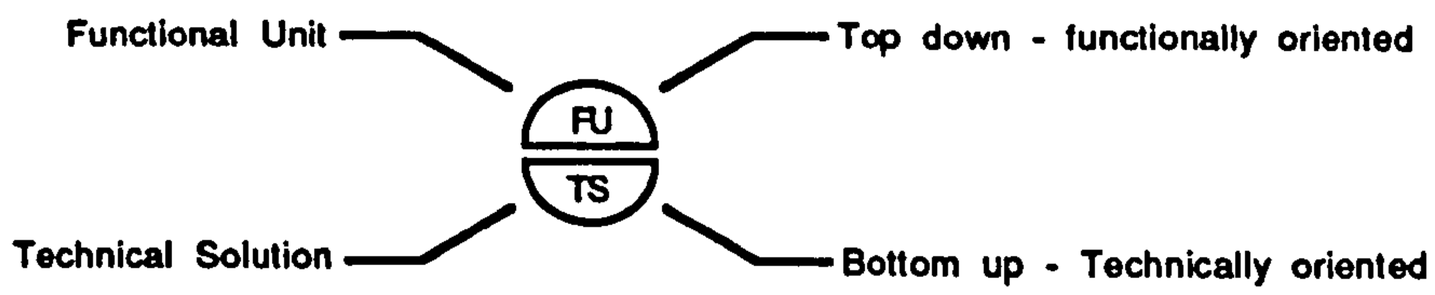


Figure 7.5.1.3. Interchangeable design solutions.

This enables several design solutions (generated in isolation) to be integrated within a complete product description. The ability to interchange technical solutions, also enables a product to respond to temporal shifts in design standards and practice.

Relationships between nodes are made using interfaces illustrated bellow in figure 7.5.1.4.

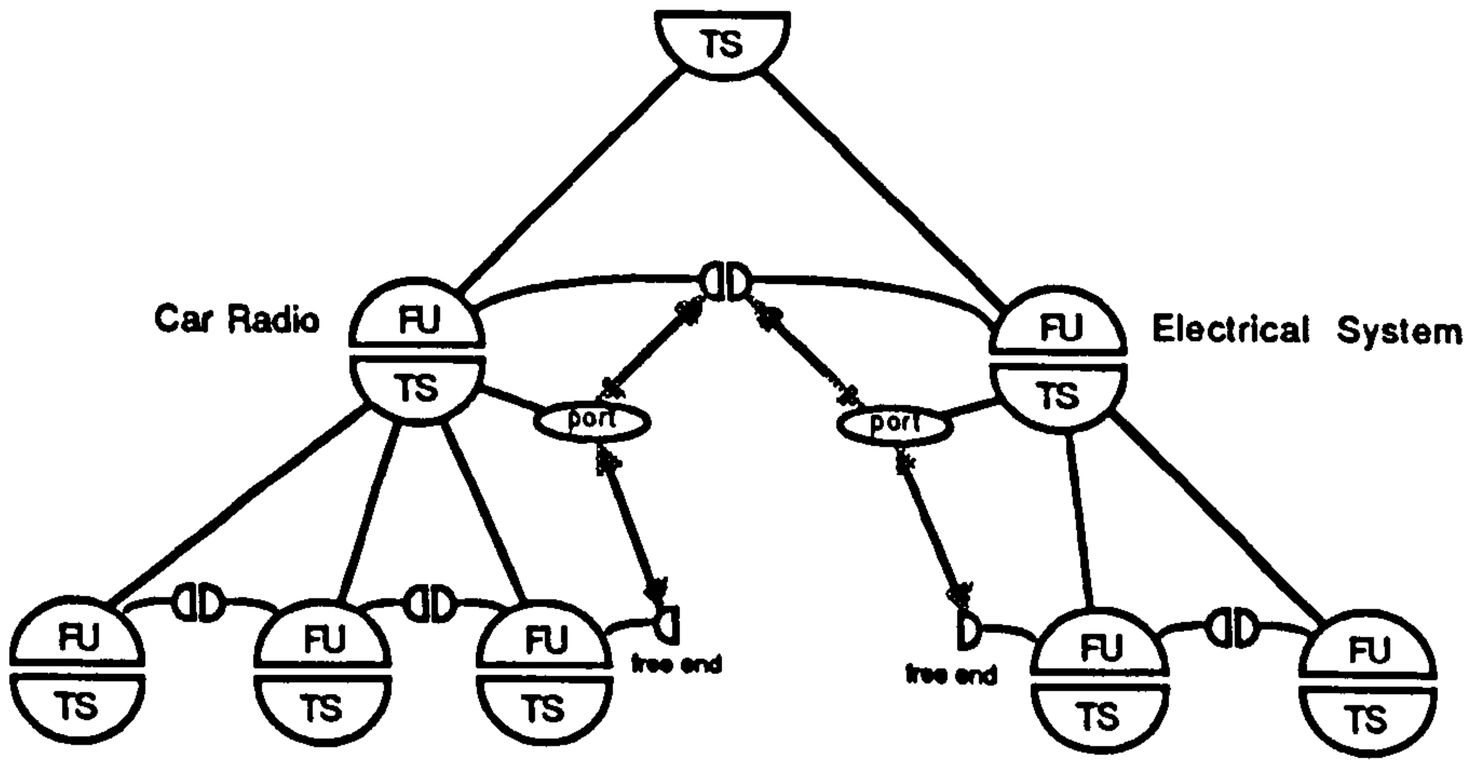


Figure 7.5.1.4. Functional Units which belong to different Technical Solutions can be connected via Free Ends, ports of Technical Solutions, and connected Ends on a higher level. For example, a car radio can be connected with the electrical system of the car on the higher level. How the ports of the radio and the electrical system are connected with internal components is defined by the decomposition of a Port in Free Ends. (from [STEP 88] page 21).

Relator, developed at the CAD Centre, Strathclyde University, works on a similar basis; enabling the end-user to construct abstraction hierarchies. This is achieved by a process of inheritance as in the case of the IFe.

The RATAS model [BJÖRK 89] developed specifically for building product definition also offers class abstraction hirarchies and generic attribute inheritance mechanisms. Bjork has identified five abstraction levels appropriate for describing building functionality:

Building layer	contains attributed data about the site location, climate, functions, size and projected cost of the building.
Systems layer	describing the systems that together constitute the building.
Subsystems layer	or functional group specifies identifies entities such as floor or lecture theatre.
Parts layer	contains references to tangible physical objects used to construct the subsystems layer. This layer must contain locational information.
Details layer	describing the physical relationships (intersections) between components in the parts layer.

Direct similarities may be drawn between these layers and those currently used to describe a building using the IFe.

Product models in STEP are defined in a language called Express. Although no attempts have been made to accommodate the emerging STEP standard within the IFe, since STEP/PDES will not be available as an International for several years [THOMAS 89], STEP will be compatible with other standards (such as a restricted form of IGES) by setting up a series of application protocols [MAANEN 89]. RAL have developed the RDST (ReaD STep file) program which will read any STEP physical file, performing syntax and type checking of parameters. RDST is internally driven by a parser trapping all errors. It is anticipated that the RDST and RAL's Express compiler will be linked and will eventually enable reader/writer programs to interpret any standard through express to a STEP file. According to a recent RAL newsletter (Van Maanen and Mike Mead), software for creating a STEP physical file is under development.

7.5.2. DATA MANAGEMENT

A great deal of research has been undertaken, under the Esprit project and the ISO standard STEP, to develop standards for data representation. Owing to the diverse nature of the types of data required for the integration of CAD packages and user modelling it would not be possible to adopt any of these formats. However, rather than expecting the blackboard to handle project data, an interface to a relational database such as ORACLE (which would be more reliable), could be developed as a resource employing one of the emerging standards. This could readily be achieved, without any modification to the system, simply by introducing a product model client, leaving the blackboard to pass messages between the various resources.

Therefore, instead of holding product information and the relationships between objects, on the blackboard which would overload it's functionality, the product model on the blackboard would simply be an access port to a database management system utilising an emerging standard such as STEP.

The Unix environment also provides many tools and utilities for managing and manipulating data. These utilities may be used as resources for the general maintenance of database files. Differential control of product files may be achieved by performing a unix "diff" between two design solution files, for example, and extracting the differences from the most recent file.

7.6. OBJECT MONITORING

The knowledge resources are chosen to deal with particular key issues of the design of a particular product assisting in the design by providing expert knowledge during the design session.

It is envisaged that the chosen knowledge resources would play an active role in the design process, constantly monitoring changes to the model.

Each resource may, through the blackboard, express an interest in a particular class of objects (geometry, stairs, for instance), and therefore would be notified of any class events; such as creation, modification & deletion.

For example it is envisaged that a geometry analyser would play a fundamental role in any CAD system; testing for object collisions and performing Boolean operations (union, intersection, difference) on groups of geometric bodies.

A building regulations resource may actively monitor stairs within a building, extracting location, size, floor to floor and other measurements from the product model together with usage information from the designer; establishing from tables the

optimum width, number and size of risers, specifying hand rails if necessary. The resource would be set into one of a number of modes, active (adjust object attributes to suit regulations), informative (inform the designer of regulatory infringements).

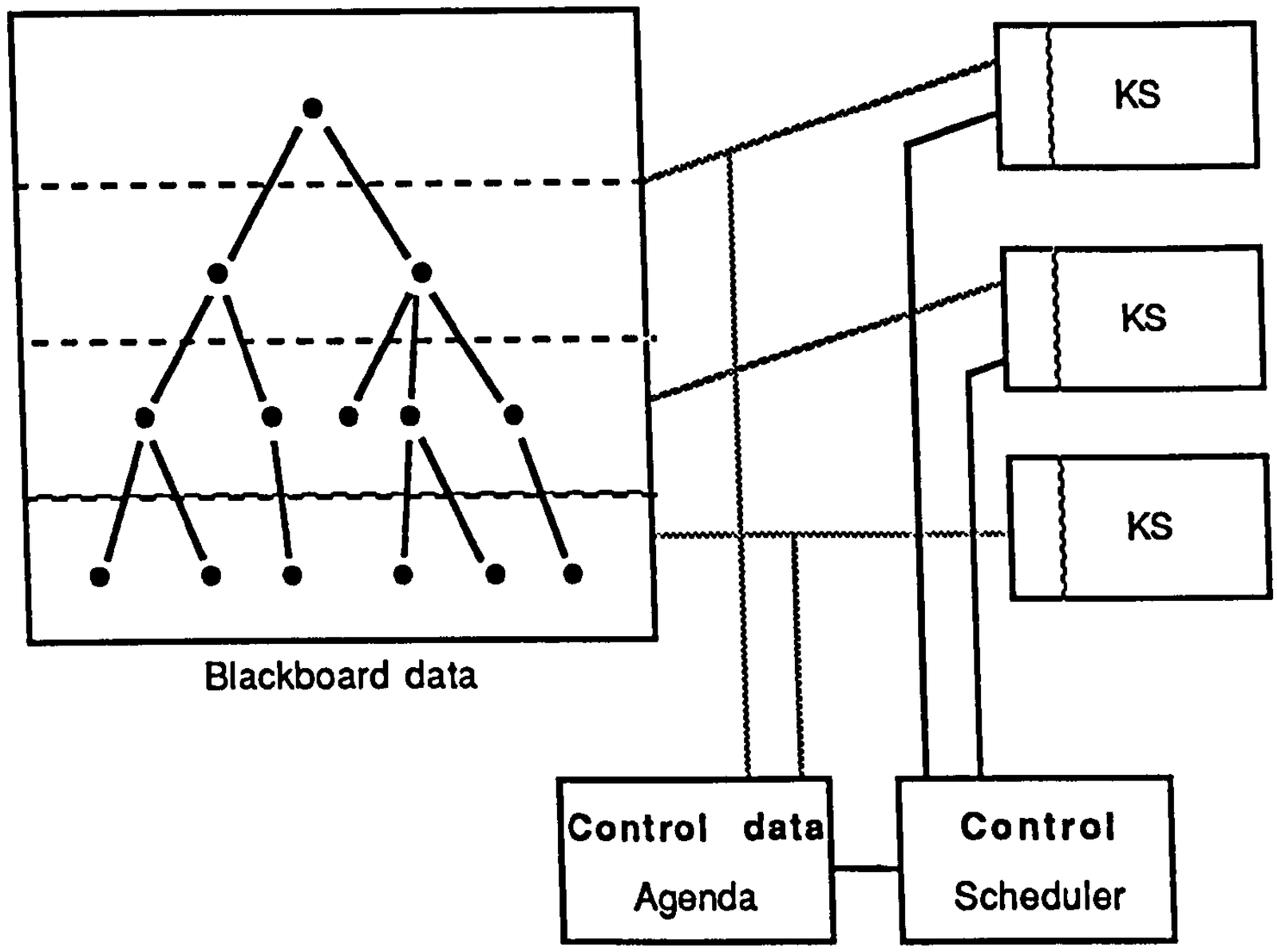


Figure 7.6.1. Opportunistic knowledge sources (KS) monitoring areas of a product model (from [ENGLEMORE 88]). A consistency enforcer is also required to ensure that, when a data entry is modified that it is internally consistent with other related information.

It has also been predicted that many resources may, simultaneously, monitor the same class of object, resulting in counter productive conflicts. In order to reduce the likelihood of 'bottle-necks' it has been recognised (in the RATAS model) that only in exceptional circumstances will the designer require to work on the entire product model but would work at distinct disciplinary levels (conceptual development, or detail development). This enables the resources to be grouped into distinct, sub disciplinary bands, (figure 7.6.1). Only those resources in a particular band (representing a particular stage in the design process) will be available (or active) thus reducing the number of clients competing for attention. Unlike traditional blackboards, the ife blackboard dispatches messages only to those clients expressing an interest in a particular area. The need to reduce competition arises when two or more resources are actively monitoring the same family of objects. It may be counter productive, even dangerous, to allow more than one resource to 'interact' with a object owing to the possible conflict of interests. Therefore, in addition to limiting the number of active resources monitoring the product model, those monitoring the same object class have to bid for attention. The resource manager deciding, on a simple numerical rating

scale, which resource/s to notify, as determined by an appraisals knowledge base. This knowledge base would decide which resource/s to direct events to, in what order and may even, under certain circumstances, ignore particular resources.

Many knowledge resources will be design tools in their own right and, since, each may communicate using the same generic protocol, the designer may interact directly with any resource in the system to produce 'valid' objects (design solutions) which may be placed on the product area.

The IFe as a whole would enable one designer to interact with all of the specialised knowledge resources (consultants) through a consistent user-interface which would map the designers conceptual level of comprehension to each individual resource, providing 'intelligent' defaulting and on-line assistance.

7.6.1. ACTIVE RESOURCES.

The current range of applications have been used as incidental problem solving workhorses or procedural data processors (data in, results out), run to completion. Some applications may require information that can only be established after some initial processing and therefore must, directly or indirectly communicate with the user. Resources may also monitor the activities of the user and must remain constantly active. Such resources may express an interest in particular classes of events, objects or concepts. Instead of relying on the blackboard to notify each resource of particular class activity, all events within a particular area are reported to the resource handler, and unless specific resources are specified the data passed to each active resource which filters out the events of interest.

7.6.2. CONCEPT FILTERING

Figure 7.6.2.1, below, illustrates the internal workings of a typical protocol interpreter. The category of concepts received by the resource is determined by which blackboard areas are subscribed to and are further refined by explicit discrimination. All events occurring in the subscribed blackboard areas are reported to the resource. However the resource may only be interested in a particular class of concepts; geometry, user events, requests for assistance.

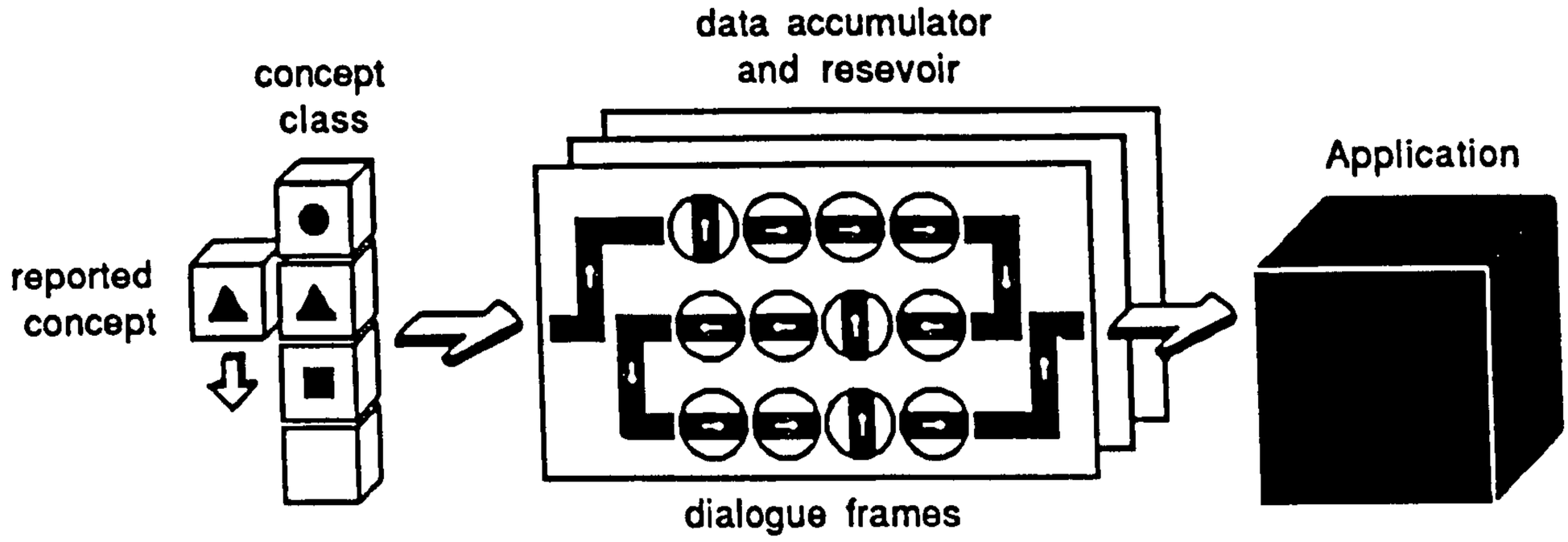


Figure 7.6.2.1. Selective concept interpreter and data accumulation for *question and answer* dialogues. *A vertical arrow indicates an empty slot.*

In order to selectively extract particular concepts the concept class is checked against an internal list, figure 7.6.2.1. Desired concepts are then passed to an interpreter which performs data translation and inferencing. Owing to the question and answer nature of existing applications dialogue frames are required. Each frame deals with a particular set of concepts pertinent to the interfaced application. Once a particular frame is complete a dialogue with the application is initiated by the interpreter and the data held on the dialogue frame is transmitted. Once a frame is complete any modification to a particular slot is immediately transmitted to the application, unless the frame is cleared after the initial dialogue in which case each of the slots must be attributed again from other resources. This enables default values for particular slots of a frame to be set.

Figure 7.6.2.2, below illustrates a dialogue frame for the ABACUS perspective program Viewer. When run in a non-interactive mode dialogues are initiated by the user by typing a character. A procedural question and answer session follows. The frame bellow illustrates the dialogue frame for the INPUTALL command.

Initiator	I	
Eye point	x y z	- - -
Focus point	x y z	- - -
Mid point	x y z	0 0 0
Angle of view	av	55
line type	lt	1

Figure 7.6.2.2. Viewer dialogue frame - Input all

The mid point, angle of view, and the line type are defaulted, while both the eye and focus points (cleared after each call) must be specified before the application is notified.

Bridges suggests that an example of a design resource that monitors the user's activities may express an interest in the column class of object for instance. If a single column, aligned to a pre-defined grid, is moved from being flush to the grid line to being centred, the resource may shift all objects within the same class to the intersection of grid lines (figure 4.8.7.2). Any other relationships, defined in the product model, would have to be updated and may result in further resource activity.

7.6.3. UTILISING DISTRIBUTED RESOURCES.

The system infra structure relies heavily upon the multiprocessor environment of the hardware platform. While this enables processes to be executed in parallel, the exchange of information in the form of messages and results can often grind the system to a halt. This is a noticeable problem with the current IFe, with two nip (prolog) processes running in addition to the forms package, dialogue handler, and data handlers. Current research provides two extreme solutions to this problem. The first [DENNIS 80] is to arrange for the blackboard to exchange small packets of information between processes operating on each of the nodes of a networked system on a regular basis. The other extreme is to operate an autonomous process [ENSOR 88] on each processing node. In this situation information is passed less

frequently with the local node handling the common operations [LESSER & CORKHILL 78].

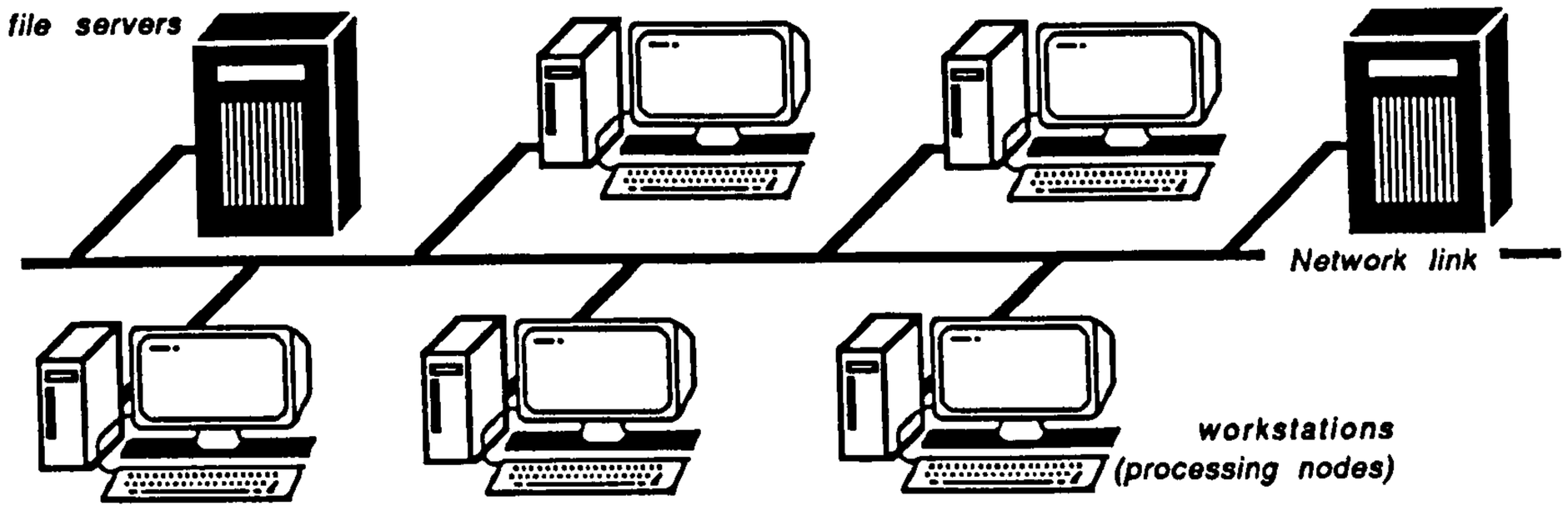


Figure 7.6.3.1. Networked processing nodes;.*Communication TCP/IP Ethernet.*

Such a facility is only possible on networked systems, figure 7.6.3.1. With a network file system it is possible to access centrally managed databases. This is achieved by mounting a partition of the storage device on each node, figure 7.6.3.2., creating a transparent file structure (although this method is problematic when a processor is down).

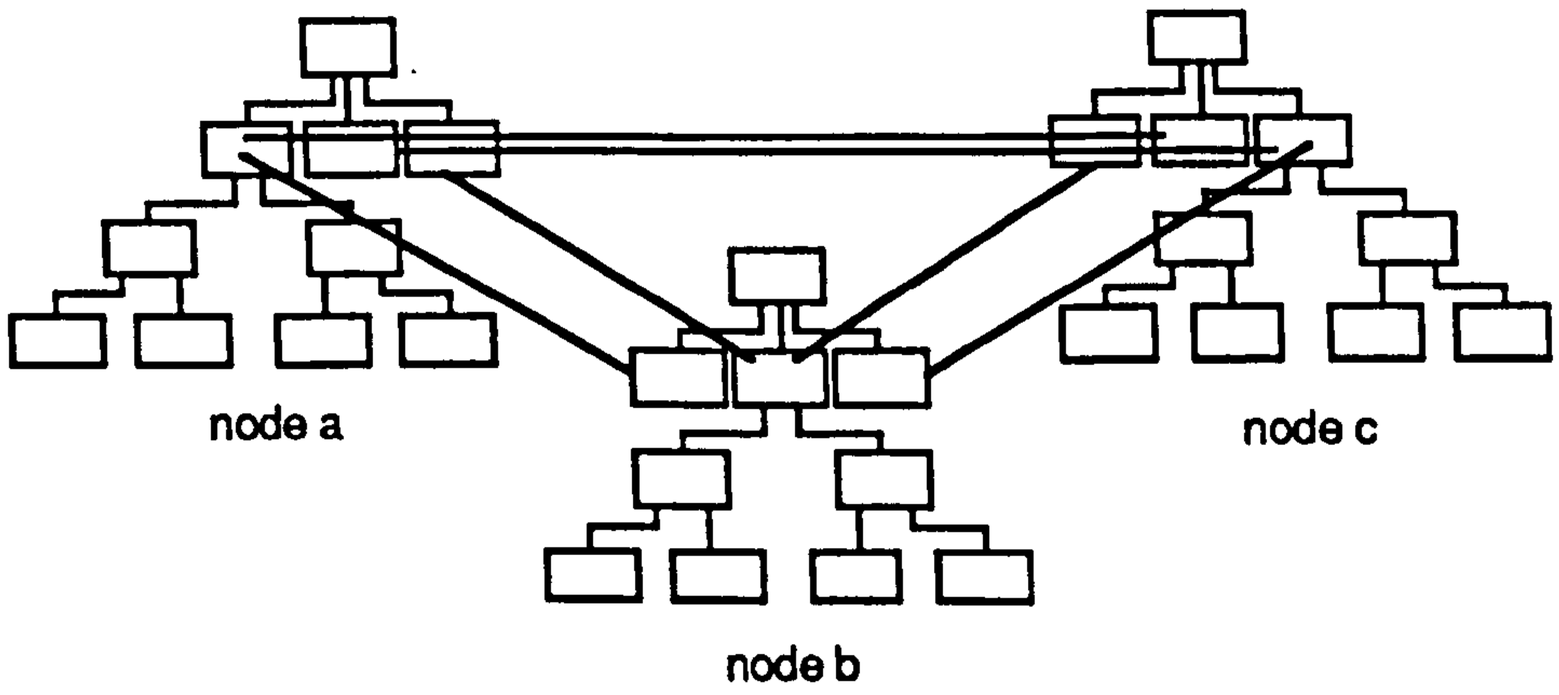


Figure 7.6.3.2. Networked File System. Disk partitions on each processing node may be mounted and accessed transparently.

7.6.4. VIRTUAL DESIGN WORKSTATION

Rather than embedding the transactional functionality within the blackboard, the manner in which the application handler has been implemented, enables instances of the application handler to be placed on satellite processing nodes. Requests may therefore be passed to each of theses nodes, either to alleviate the computational burden on the host processor or to take advantage of specific hardware characteristics of a particular workstation.

A special resource (etherlink) has been implemented for passing messages between processing nodes. On intialisation the application requires the hostname of the target workstation. A the the same time a remote command to start up another etherlink process is made and a connection is made (it is envisaged that local and remote NFS deamons would be more efficient). Etherlink has the potential to fork a single process. In this case it is another resource handler. The result, illustrated in figure 7.6.4.1, is an integrated distributed system. Since the control of passing messages is centrally managed by the knowledge resources, the integrity of the data passed between the asynchronous clients is maintained.

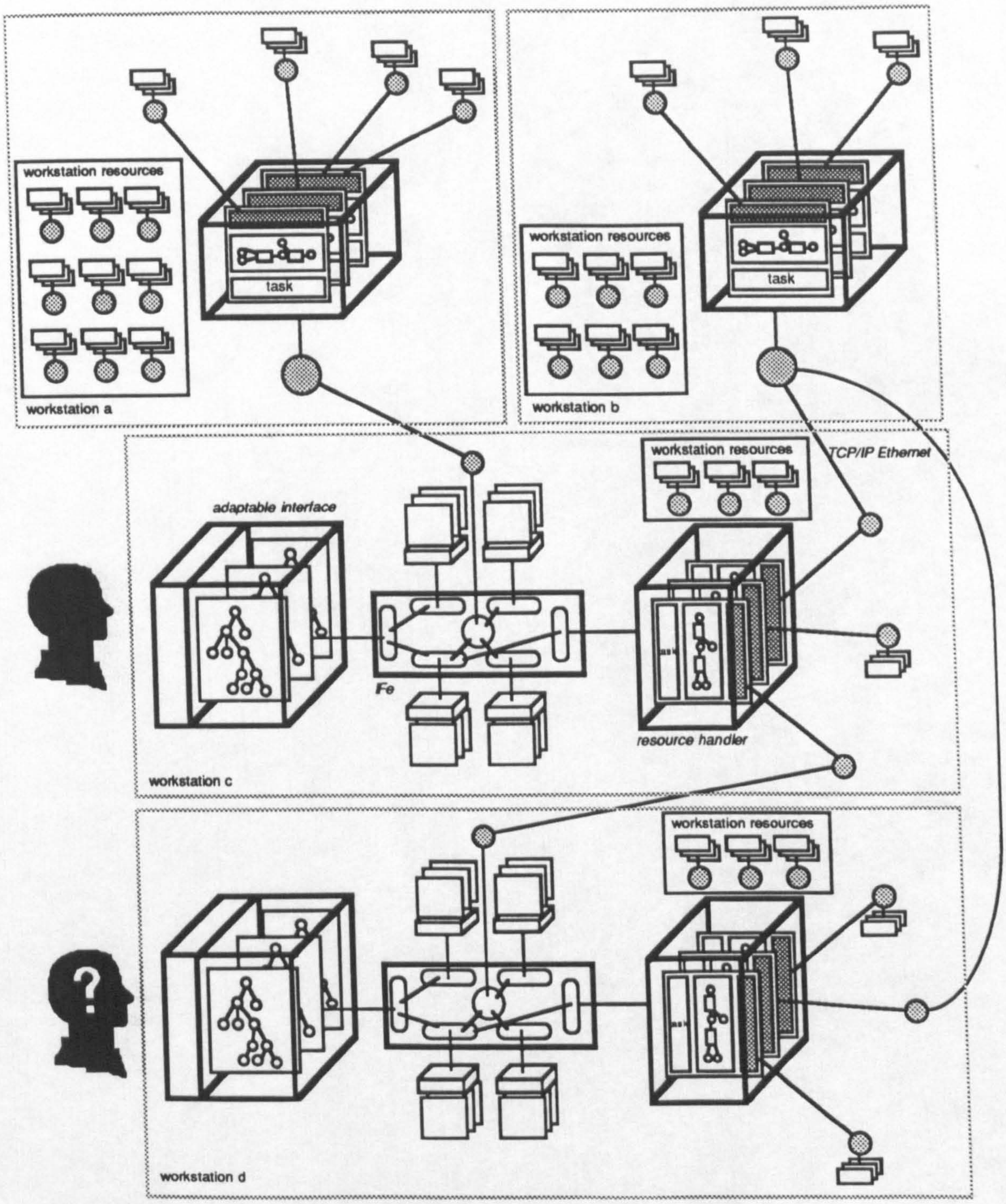


Figure 7.6.4.1. Distributed concurrent design environment (virtual design workstation).

Satellite resource handlers are placed on each of the processing nodes. Task requests are passed to individual resource handlers and solution plans (graphs) generated and invoked providing a transparent processing environment. Tasks may also be assigned to individual designers with different levels of experience (workstation d) enabling such users to make useful contributions to an emerging design solution, figure 7.6.4.1.

Solution graphs may be submitted to specific workstations reducing the burden upon the host machine. Although the form of distributed processing is intended to overcome the processing inadequacies of the host workstation, with time such an approach will inevitably increase message traffic and therefore degrade system response times. The SWORD system (developed at CMU, for such a concurrent processing environment) is able predict critical times of network congestion and direct task requests to less congested processing nodes. This is achieved by modelling human (operating) characteristics.

7.6.5. SECURITY

A typical problem with any database management system is one of security. With a distributed system such as the one suggested the issue of security is further highlighted, particularly when processing tasks are assigned to processing nodes which may not be centrally administered.

Processes active over the network are able to modify the blackboard. If local and remote demons are used for relaying messages then theoretically anybody could connect to the design environment, providing an opportunity for mischief.

The Unix operating system, often critised for inadequate security if not administered correctly, does provide file protection mechanisms. It is suggested that large volumes of data should not be transmitted across the network but saved to a file and abstract references to the physical file (with appropriate protections set) transmitted. This has two additional and obvious benefits:

- reduced network traffic.
- fail-safe data management: if the system should crash it is less likely that results and valuable data would be lost.

7.6.6. HANDLING MULTIPLE DESIGN DOMAINS

Although modules within the IFe are generic and may be utilised in a number of different situations, the current configuration of the IFe tailored to a single domain. To enable the designer to access parallel design domains the organisation and manipulation of the databases would require modification. The diagram bellow illustrates the current file structure. Simple modifications are illustrated in figure 7.6.6.1.

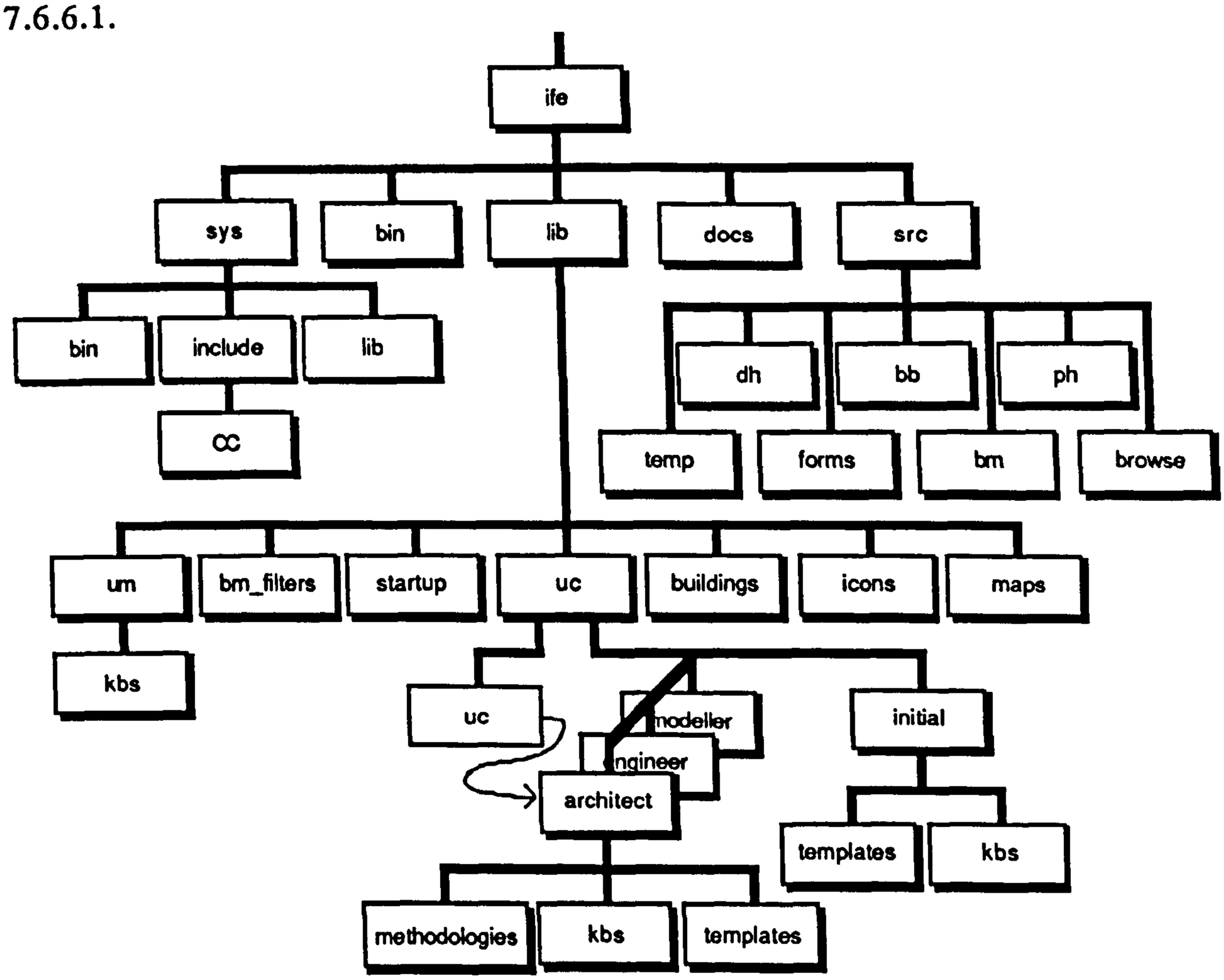


Figure 7.6.6.1. Current IFe file structure.

Critical to the operation of the IFe is the IFe file structure in particular the file ~ife/lib/uc/uc (user conceptualisation). This file is symbolically linked to one of the pre-defined stereotypes illustrated in figure 7.6.6.1. The user, knowledge, application and dialogue handlers access knowledge bases, appraisal methodologies and proforma templates through this generic link. This ensures that non of the IFe modules is aware of a particular user conceptualisation. Therefore to add a new conceptualisation only requires the creation of a new directory containing the relevant knowledge base,

proforma templates and appraisal methodologies pertinent to that particular conceptualisation.

Currently the IFe only supports one domain, namely; energy modelling. All the applications pertinent to this domain are stored in the ~ife/bin directory together with the generic system modules. From the point of maintenance this is perhaps undesirable. A more suitable directory structure is illustrated bellow in figure 7.6.6.2., where only the system modules are stored in the ~ida/resources directory. A design domain directory is also added to facilitated other knowledge domains; each containing a resource directory together with a library containing files for individual packages held in the resource directory. Just as for the IFe file structure individual domain directories contain user stereotypes. In addition to the knowledge bases, and template files, a dictionary directory is included. This replaces the ~ife/lib/icons directory and contains alternative interpretations of concepts; foreign translations, bitmaped images, etc.

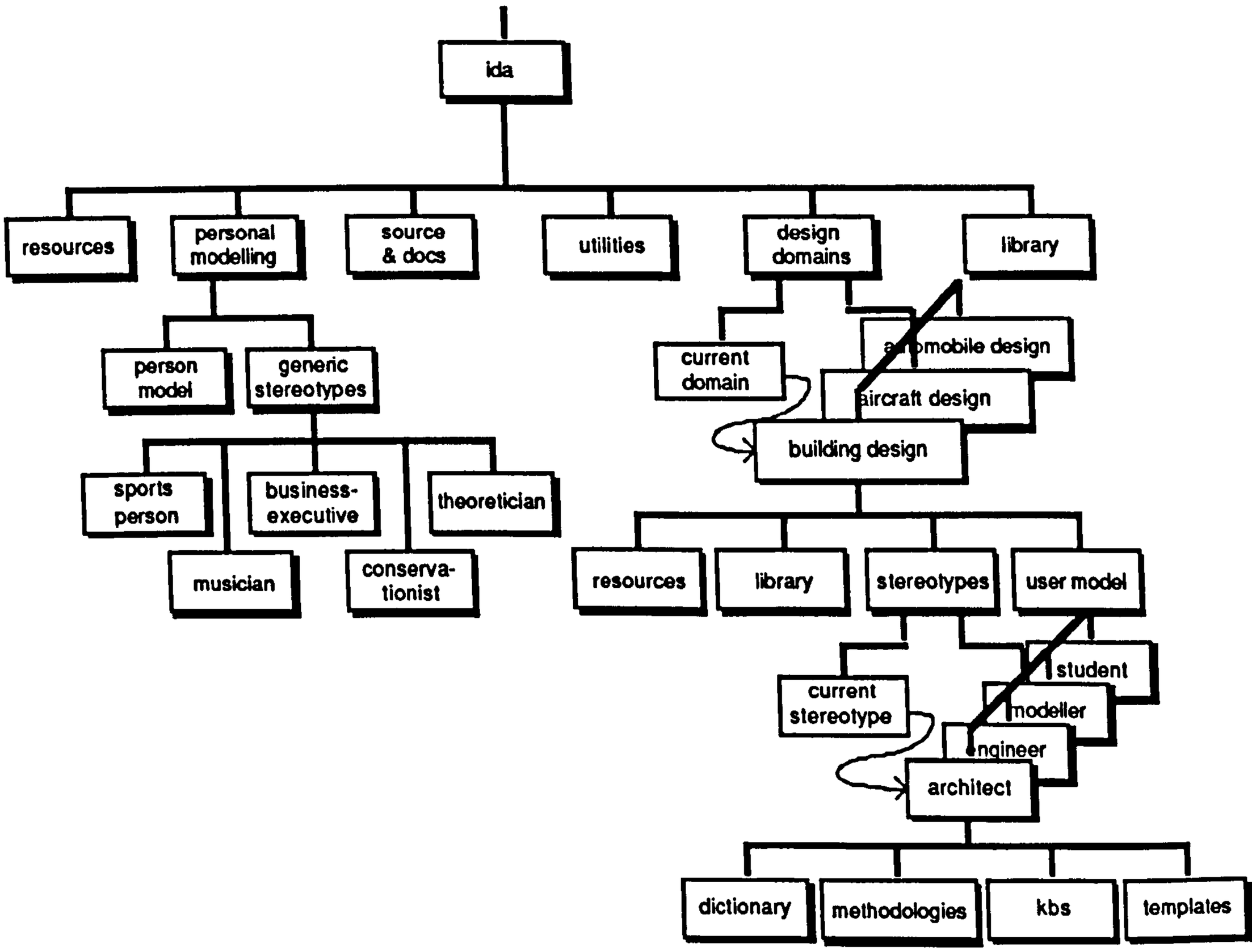


Figure 7.6.6.2. File structure for an intelligent design assistant. *Not all of the directories have been expanded for clarity.*

Just as the *change_cpt* script changes the current user conceptualisation pointer so a similar *change_dpt* would be responsible for switching between design domains.

In addition to the domain specific user model a secondary person model should also be included to refine the dialogue. Rich [RICH 89], provides detailed methods for implementing user models of this kind.

8. CONCLUSION AND FUTURE DEVELOPMENTS

8. CONCLUSION AND FUTURE DEVELOPMENTS

A critical issue in both natural and computer design process is one of communication. One of the main objectives set out in this thesis, namely; to communicate with a range of user types, has been satisfied by the construction of a graphical (lexivisual) user interface (forms, chapter 4) based upon the communication of conceptual models (outlined in chapter 3). The forms package, with a range of interchangeable concept interpreters, coupled to the IFe user handler, provides mechanisms to dynamically interchange templates and facilitates dynamic re-phrasing and redescription, enabling the system to support several user stereotypes (although currently limited to two extreme levels within a single stereotype).

By employing multiple levels of abstraction, using synonyms (chapter 4.9.3.9) for example, a truly adaptable dialogue between computer and user is possible.

Users may not be proficient in all aspects of building specification. The dynamic loading of domain knowledge and interaction templates enables different levels of conceptualisation to exist within a single dialogue. The focus mechanism (chapter 5) also enables the environment developer to modify both knowledge bases and proforma templates without actually having to suspend interaction as in the case of some UIMS, chapter 2.

The identification of a generic communications protocol, chapter 4.8, and the development of a high level formalised neutral language, chapter 5, enables many different interaction devices to be dynamically interchanged or run in parallel (the map program, chapter 6, for instance); each communicating, by means of formatted natural language utterances, chapter 3,4&5, with a central (orchestration) knowledge base.

By encoding knowledge of the user's task, expectations, and problem domain and physically separating this from the user interface and applications layers, as is currently implementated, a truly modular system has evolved whereby contributions from many individuals may be accommodated within a single system. The approach, although fragmented, is completely modular and therefore infinitely extendible. Since a central knowledge is responsible for orchestrating the dialogue consistency is ensured.

This also ensures that, regardless of the interaction device(s) employed to communicate with the user, a consistent and contextually relevant level of assistance and intelligent defaulting is offered.

The use of application protocol interpreters, chapter 7, enables new and existing software to be integrated within a consistent environment quickly and easily; facilitating the automatic acquisition of data and knowledge. Simply by enshrouding an existing application package within a parameterised shell script and creating a corresponding resource description, a design environment may be extended or customised without disrupting any of the other modules within the system.

The generic description of application resources detailed in chapter 7, facilitates a high level (abstract) categorisation of incidental design tools and methods enabling knowledge bases to be constructed with out reference to the obtuse operational procedures of the computing environment. In addition to ensuring that the encoded domain and task knowledge maintains a high level of generality and portability, this method will also hopefully encourage designers to customise their own environment and hence extend the range of available conceputalisations.

The key to the operational success of the IFe is the blackboard, acting as a communications centre. By separating the transactional functionality of traditional blackboards from the notification mechanisms, currently implemented, multiple resource handlers may be placed upon several, networked processing nodes, allowing tasks to be assigned to specific hardware devices in an abstract manner, chapter 7. As well as employing application packages, tasks may be assigned to individual users by initialising another IFe process with an appropriate conceptual template. Thus users of varying levels and types of expertise may make useful contributions to emerging design solutions.

Although the system described exists in a protypical form, in order to become a useful design environment a substantial amount of research and implementation is still required; particularly at the user-interaction level.

It is envisaged that interaction tools such as the interactive 3D manipulation package, described in Appendix E, will play a useful role.

Since all resources and interaction modules adhere to a standard communications protocol all are plug compatible. Therefore interfaces such as that illustrated bellow, figure 81, (similar in nature to the ConMan interface [HAEBERLI 88], but enabling the the system to respond to a user's utterances, providing appropriate contextually relevant defaults and assistance) may be constructed quickly and easily by the user in response to individual problems.

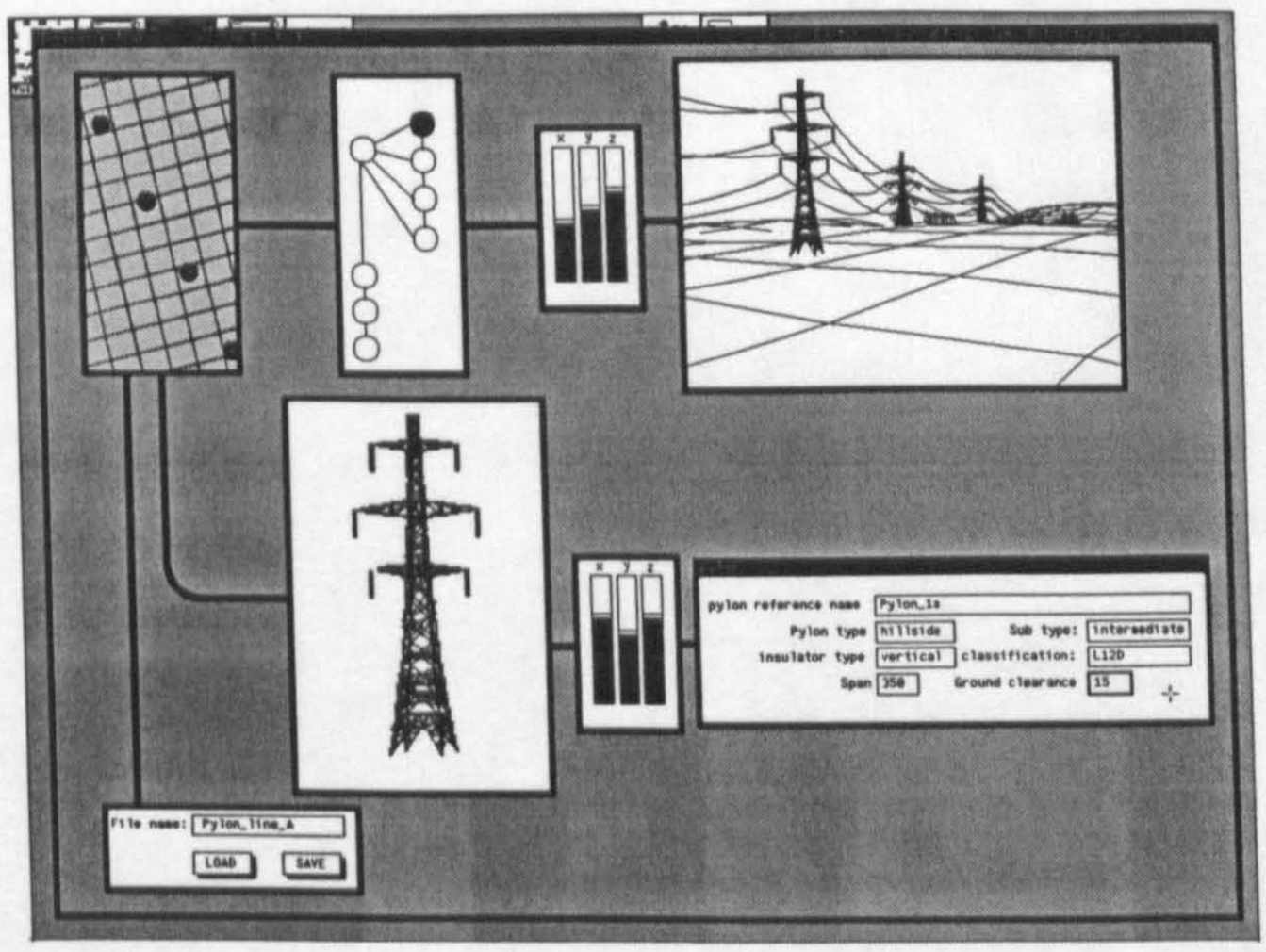


Figure 8.1. Plug compatible interaction tools.

The recent advances in network transparent window managers such as X and NeWS should make the integration of distributed resources easier bringing the power of dedicated graphics workstations to the designers desktop.

Since the WW graphics library, developed by Mark Martin at the Informatics Division of RAL is no longer supported it is anticipated that the X graphics library will now be used. Although, the current release of WW is implemented on top of the X window manager it is not possible to exploit the network transparency offered by X. Therefore in order to exploit the power of workstations such as Silcon Graphics Iris on a distributed level, it will be necessary either re-code the forms package or adopt a hypermedia package such as Intermedia [YANKELOVICH 88].

In any situation involving the utilisation of knowledge resources direct access to individual resources is necessary to:

- establish what methodologies are being employed
- monitor progress
- intervene by interrupting

The current implementation of the IFe is suffers slightly from encapsulation. In order to satisfy the requirements of expert users, namely the ability to directly access the functionality of appraisal packages, a solution plan interface suggested in chapter 7 should be implemented. This would also isolate the end-user from inconsistencies arising from the substitution of design tools (as a result of shifts in standards and technological advances) by enabling application programs to be manipulated at a functional level rather than an operational one.

The automation of solution graphs (plans), although offering no great benefit over the current implementation of shell scripts, would make the system internally more consistent and easier to maintain. Further research is required to determine an appropriate compilation mechanism.

A mechanism, similar to that utilised by the TAILOR system, chapter 5, for automating the customisation of descriptions would greatly enhance the rephrasing mechanisms. It is suggested that such functionality should be encoded as a discrete resource, annexed to the user handler, accepting requests for alternative descriptions (verbose or terse) of a given concept to suite various user types.

It also suggested, that rather than the knowledge handler formatting messages to the user, that this should be done by the user handler. The knowledge base should contain information specific to the domain and should request the communication of concepts to the user. Such requests would then be processed by the user handler which would then decide upon an appropriate presentation mechanism. Such an approach would enable more design oriented knowledge (shape grammars, regulations, design critics and the like) to play an active part in the specification of building models. Such knowledge would exist as discrete resources, monitoring activity on a central product model.

The overall system enables the designer to work within a flexible abstraction hierarchy, building by defining a conceptual model a complete description of a product. In to this basic framework object instances are inserted to produce a fully attributed entity. Issues relating to product definition and description have only be touched upon. Extensive research in this area has already been carried out by several collaborating institutions. It is anticipated that aspects of the General AEC Reference Model

(GARM), a sub set of the emerging ISO standard STEP/PDES, will significantly influence subsequent implementations.

Knowledge resources may consist of constraints, paradigms, algorithms and simulation procedures. The demonstrator project will focus on developing interpreters which will allow the system to deploy the following resources:

i) a geometry analyser (e.g. GAM) which performs Boolean operations on groups of geometric bodies and takes off appropriate quantities.

ii) visualisation facilities (such as VIEWER, VISTA) which will allow the evolving design to be viewed in varying degrees of detail, from bound box to full coloured, textured and shaded image.

ii) environmental appraisal, including thermal, lighting and acoustic performance; the range of available applications packages is large and a number of these, covering the range from 'rule of thumb' to rigourous simulation, will be implemented.

iv) structural analysis, ranging from the concept generator proposed by Rafiq and MacLeod [RAFIQ 88] to the detailed RC design systems offered by McDonnel Douglas.

v) cost estimation and construction planning based on for example, the work of the Civil Engineering Department at the University of Dundee and the Computer Science Department at the University of Strathclyde [BUCHANAN 89].

It is suggested that the approach proposed is appropriate to the current SERC ITA theme "Design Assistance (Intelligent)", offering two very important advantages in a collaborative research programme:

i) Existing application packages can readily be accommodated as knowledge resources,

ii) the R&D effort can be partitioned into the modules identified above with each being the focus of effort for a research term.

In order to extend the principles outlined in this thesis funding is being sought from the SERC's IT group. If successful a demonstrator Intelligent Design Assistant will be produced at ABACUS in collaboration with the CAD Centre and the Department of Civil Engineering at the University of Strathclyde. Current investigations have already attracted some interest, in particular from IBM UK, BDP, AUTODESK, and MacDonnel Douglas, who, as industrial partners, will actively participate in future developments.

APPENDIX A

Inform Communications Library

A. INFORM COMMUNICATIONS LIBRARY

In addition to being an integral part of an interactive dialogue system, the forms package may also be used as a means of controling application programs directly. In order to facilitate this additional role an interface (inform) has been developed to enable the rapid development of graphical user-interfaces. In contrast to the UIMS approach the high level neutral control mechanism within the library provides the developer with the most flexible development tools. Inform is a library of both low and high level functions for message handling between two application programs. The intention is for a parent process to fork and exec the forms package. Both parent and child processes communicate via Unix pipes. Owing to the many different input streams required by for the implementation of such an interface events are handled by a synchronous I/O multiplexor.

A.1. SYNCHRONOUS I/O MULTIPLEXING

One of the main difficulties encountered was processing the four event streams:

window events

mouse events

keyboard events

pipe read events

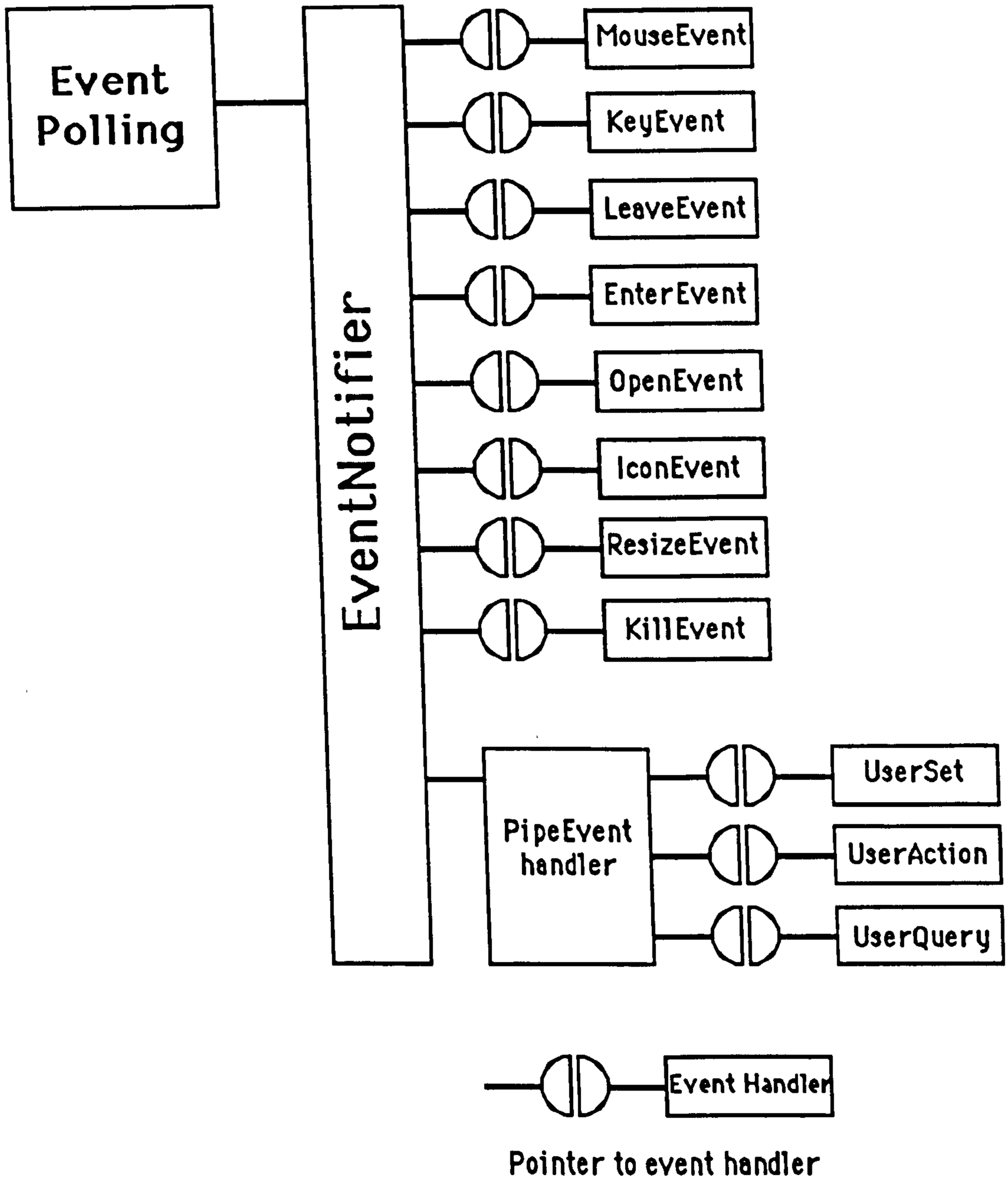


Figure A.1.1. Schematic view of multiplexor.

The first three streams where handled by the graphics library. The problem was to add the standard input event stream. The developer of WW (Martin) has anticipated this need by allowing application developers to replace the input stream handler ipwait. This function is called to test the event queue in one of two modes:

request wait for next event

sample return regardless of the state of the event queue.

Events are returned in the d_event field of the dd structure. The currently supported events are:

```
IPOTHER         - mouse events
IPKEY           - keyboard events
IPLEAVE         - window leave events
IPENTER         - window enter events
IPSEEN          - window open events
IPNOTSEEN       - window close events
IPSIZE          -window resize events
IPWANTKILL      - window kill (hangup) events
```

In reality ipwait points to the ipxwait function which does the real event handling, thus enabling the developer to create a new ipwait function and access the basic multiplexer. The function ipfwait, figure A.1.2. bellow, supliments the above events with a pipe read handler.

```
ipfwait()
{
        fd_set  rd, wrt, ex;
        int  n = 0;
        int  event = IPNOEVENT;
        static struct timeval nowait = {0L, 0L};
        jwindow* jwp;
        int wfd;
        ex = *(fd_set*)dd->d_selectmore;     /** add fd for window **/
        rd = *(fd_set*)dd->d_selectmore;
        FD_SET(inform.chanel[READ],&ex);
        FD_SET(inform.chanel[READ],&rd);
        FD_ZERO(&wrt);
        /** Shouldn't use ww junk structures to get window fd **/
        jwp = jwin(ddwin);
        wfd = (int)window_get(jwp->jw_sunwindow,WIN_FD);
        n = select (32, &rd, &wrt, &ex,(struct timeval *)NULL);
        if(FD_ISSET(inform.chanel[READ],&rd))
        {
                n = read(inform.chanel[READ],inform.buf,INFORMBUFSIZE);
                switch (n)
                {
                        case -1:    CloseDown("Can't read pipe");       break;
                        case  0:    CloseDown("Closing pipe");                  break;
                        case  1:    event = IPPIPE;                             break;
                        default:    inform.buf[n] = '\0';
                                    event = IPPIPE;                             break;
                }
        }else if(FD_ISSET(wfd,(fd_set*)dd->d_selectmore))
        {
                ipset (IPSAMPLE);
                ipxwait ();
                event = dd->d_event;
        }
        dd->d_event = event;
}
```

Figure A.1.2. Event polling - ipfwait(), (inform.c)

Select() examines the I/O descriptor sets whose addresses are passed in readfds, writefds, and exeptfds to see if some of their descriptors are ready for reading, ready for writing, or have an exceptional condition pending [SUN]. The window file descriptor and the inform read channel are added to both the read and write descriptors using the FD_SET macro. Select is then called with a non-NULL pointer to a zero-valued timeval structure ((struct timeval*)NULL) to effect a poll. The read file descriptors are then tested for pipe events and window events as illustrated by the extracted code. The two input sources are merged in a simple event handler illustrated schematically in figure A.1.1. and A.3.1.

A.1.1. PIPE READ EVENT

If the read channel is flagged as ready an attempt to read it is made and dd->d_event is set to IPPIPE.

Initially single characters where read from the queue and appended to the buffer. The reason for this was to prevent user events from the window from being blocked by long reads. It also enabled the buffer to be dynamically extended. This approach resulted in a number of unforeseen side effects.

Events from the window stream where not being cleared and had the effect of being echoed several times. Character repeating was undesirable.

Having tracked down the cause of event feedback, individual character reads where substituted by a single block read. In addition to resolving event feedback it had also increased the speed and efficiency of the routine. The only disadvantage to this method was the difficulty in dynamically extending the buffer. A fixed size buffer of 512 characters was chosen as this is the maximum a unix pipe could reliably handler handle without being blocked. This imposed restrictions on the size of data that could be transmitted and data was often lost as a result. The buffer size was increased to an arbitrary 1000 characters with no loss of data. An attempt will be made to dynamically extend the buffer.

Carriage return '\n' terminates a data packet. However an attempt is made to read INFORMBUFSIZE characters so the input buffer may contain several parcels of data.

If the standard input channel does not have a pending read event the window file discriptor is tested. If the channels associated with window report events pending ipxwait is called and sets dd->d_event accordingly.

A.2. SYSTEM COMPATIBILITY

As part of the system V operating system the select function was changed; file descriptors where initially specified by setting integer bit fields and subsequently changed to an fd_set. To comply with both versions the FD_ macros have been defined for operating systems below OS V.

```
#ifndef FD_SETSIZE
/* cannot typedef int fd_set; as on Sun fd_set is a struct already */
# define fd_set int
# define FD_SETSIZE        32
# define FD_CLR(fd,in)     *(in) &= ~(1<<fd)
# define FD_SET(fd,in)     *(in) |= (1<<fd)
# define FD_ISSET(fd,in)   (*(in)&(1<<fd))
#endif FD_SETSIZE
```

Figure A.2.1. FDSET - Extract from ww library (M.M. Martin RAL).

A.3. EVENT NOTIFIER

Events are intercepted by a low level event polling routine (ipfwait) and dispatched to the appropriate event handler. The events handled by the interface are shown bellow. As each event is taken from the queue it is processed. All event handlers apart from the pipe event handler are user definable and initially defaulted to noop(); (no operation). The notifier below as well as handling events from a UNIX pipe also handles window events. This enables the application to employ its own graphics display as well as utilising that of the forms package. It should be emphasized that the forms package also employs the functions within the inform library for event handling and notification.

The routine loops indefinitely. Each time round the I/O streams are interrogate by the re-defined ipwait routine. A simple switch statement is employed to dispatch events to appropriate event handlers, figure A.3.1.

```
InForm()
{
        dd->d_ipwait=ipfwait;    /** re-define synchronous I/O multiplexer **/
        ipset(IPON);                        /** activate input to window **/
        ipset (IPSAMPLE);                   /** set input sampling **/

        while(1)
        {
                ipwait();
                switch(dd->d_event)
                {
                   case IPOTHER:                inform.MouseEvent();   break;
                   case IPKEY:                  inform.KeyEvent();                 break;
                   case IPLEAVE:                inform.LeaveEvent();   break;
                   case IPENTER:                inform.EnterEvent();   break;
                   case IPSEEN:                 inform.OpenEvent();    break;
                   case IPSIZE:                 inform.ResizeEvent();  break;
                   case IPNOTSEEN:              inform.IconEvent();    break;
                   case IPWANTKILL:             inform.KillEvent();                break;
                   case IPPIPE:                 ProcessPipeEvent();    break;
                }
        }
}
```

Figure A.3.1. Main event notifier (inform.c)

The inform structure, figure A.3.2. contains pointers to event handling routines (initially defaulted to noop(), figure A.3.3) which may be defined by the application developer.

```
typedef struct _events{
        /*      Event handlers:                                         */
        int     (*MouseEvent)();
        int     (*KeyEvent)();
        int     (*LeaveEvent)();
        int     (*EnterEvent)();
        int     (*IconEvent)();
        int     (*OpenEvent)();
        int     (*ResizeEvent)();
        int     (*KillEvent)();
        int     (*PipeEvent)();
        int     (*PipeError)();
        int     (*UserSet)();
        int     (*UserQuery)();
        int     (*UserAction)();
        int     chanel[2];              /* pipe read & write    */
        char    buf[INFORMBUFSIZE];             /* pipe buffer                  */
        char    c;                              /* last character typed */
        /* buffer parsed into fieldname, action, parameters, event      */
        char    *fieldname;                     /* field giving event   */
        char    *user_action;                   /* event                */
        char    *user_param;                    /* parameters                   */
        int     user_event;                     /* event type                   */
}Dialogue;
Dialogue inform;
```

Figure A.3.2. Inform structure.

```
noop()    /* do nothing */
{
#ifdef DEBUG
        fprintf(stderr,"Forms:noop\n");
#endif DEBUG
}
```

Figure A.3.3. Noop - blank routine.

Each event type will be described in detail in a following section with example handlers.

A.4. INFORM DATA STRUCTURE

The important event from the point of view of connecting to the forms package is the PIPE event.

A.5. PIPE EVENT

This signifies that a message from the standard input channel has been received. Events from the forms package are transmitted using the following protocol:

concept<separator>event<separator>value <terminator>

This will be referred to as data packet. Each data packet is terminated by a new line '\n'. The inform buffer may contain several data packets. The separators signify the type of event and are currently defined as:

```
# define USER_SET                ':'
# define USER_QUERY    '?'
# define USER_ACTION   '!'
```

Figure A.5.1. Event tokens

A.6. PROCESSING PIPE EVENTS

The event notifier dispatches pipe events to a fixed command parser, figure A.6.1. ProcessPipeEvent() extracts data packets from the inform buffer and processes them individually using ProcessCommand, figure A.6.2.

```
ProcessPipeEvent()
{
        extern char* strtok();
        char* string;

        string = strtok(inform.buf,"\n");
        while(string != NULL)
        {
            ProcessCommand(string);
            string = strtok(NULL,"\n");
        }
}
```

Figure A.6.1. Pipe event handler

```
ProcessCommand(string)char* string;
{
        char* concept= NULLPTR(char);
        char* method  = NULLPTR(char);
        char* param   = NULLPTR(char);
        char* c;

        concept= string;
        if((c=index(string,USER_SET))!=NULLPTR(char))
        {
            method=c+1;
            *c='\0';
            if((c=index(method,USER_SET))!=NULLPTR(char))
            {
                    param=c+1;
                    *c='\0';
                    UserSet(concept,method,param);
            }
         }else
         if((c=index(string,USER_QUERY))!=NULLPTR(char))
         {
            method=c+1;
            *c='\0';
            if((c=index(method,USER_QUERY))!=NULLPTR(char))
            {
                    param=c+1;
                    *c='\0';
                    UserQuery(concept,method,param);
            }
         }else
         if((c=index(string,USER_ACTION))!=NULLPTR(char))
         {
            method=c+1;
            *c='\0';
            if((c=index(method,USER_ACTION))!=NULLPTR(char))
            {
                    param=c+1;
                    *c='\0';
                    UserAction(concept,method,param);
            }
         }

}
```

Figure A.6.2. Pipe event parser

ProcessCommand() determines which of the three event types have occurred, splits the command string into its three components (concept,method, and value) and calls the appropriate event handler. The three default event handlers are described bellow but may be substituted by the application developer although it is envisaged that those suggested are sufficient to cope with most situations.

A.7. USER_SET EVENT HANDLER.

This is by far the most common event. When a field is de-selected, by selecting another, any changes made to the field's value are reported as:

concept:USER_SET:value

```
UserSet(concept,method,param)char* concept; char* method; char* param;
{
        inform.fieldname   = concept;
        inform.user_action = method;
        inform.user_param  = param;
        inform.user_event  = USER_SET;
        inform.UserSet();
}
```

Figure A.7.1. User set

The UserSet routine sets the appropriate fields in the inform structure and reports the event to the re-definable inform.UserSet routine. The simplest method of handling events is to provide a list of concepts and concept handlers; match the incoming concept name against entries on the list and call the associated routine. For this purpose the NotifyProc structure, figure A.7.2, has been defined.

```
typedef struct _NotifyProc{
        char *fieldname;
        int  (*notifyproc)();
}NotifyProc;
```

Figure A.7.2. Notify proc structure

A typical example of a notification array would be:

```
static NotifyProc Concepts[] = {
        "name",         SetName,
        "location",     SetLocation,
        0
};
```

The default inform.UserSet event handler is shown following listing, figure A.7.3.

```
/* Check to see if one of the functions has been fired
 * If not then pass to current node to deal with it
 */
user_set()
{
        char* s;
        char* f;
        register int i=0;

        while((f=UsrProc[i].fieldname)!=NULLPTR(char))
        {
            if((s=Tail(inform.fieldname))!=NULLPTR(char))
            {
                    if(strncmp(s,f,strlen(f))==NULLPTR(char))
                    {
                            if(UsrProc[i].notifyproc!=NULLPTR(char))
                                    UsrProc[i].notifyproc();
                            break;
                    }
            }
            i++;
        }
}
```

Figure A.7.3. User Set Event handler (default).

The fieldname is split into two component parts:

field address and fieldname

using the Tail function, below:

```
#include <string.h>

char * Tail(string) char *string;
{
        char *c;

        return(((c=strrchr(string,'/'))!=NULL)?c+1:string);
}
```

Figure A.7.4. Tail function.

The field name is compared against each entry in the NotifyProc array. Note that strncmp is used and that the NotifyProc[i].name is used to provide the length for the comparison. This enables partial matches to be made. For example:

Assume that an event has been reported by the field "field a" but only "field" is present in the NotifyProc array, figure A.7.5. The event is therefore reported to the SetField procedure. Although not immediately apparent this facility is very useful for re-phrasing.

If events for "field a" are to be handled explicitly but "field b",etc, events are to handled generically it is important to enter "field a" notifier before the generic "field" handler.

```
..[]={
        "field a",          Field_A_Handler,
        "field",  Generic_field_handler,
        0
};
```

Figure A.7.5. Example NotifyProc structure.

Similarly USER_QUERY and USER_ACTION events are handled by comparing entries in the QueryProc and ActionProc lists.

A.8. HANDLING WINDOW EVENTS

In addition to managing events from a parent/child process the inform library also handles events from the window manager. These include: keyboard events, mouse events and window manipulation events and are dealt with by the inform procedure (A.1).

A.9. MOUSE EVENTS

Mouse events are reported when a button or buttons are pressed. Button release events are not reported. The button status is held in dd->d_buttons and may be any ORed combination of:

ITEMBUTTON

MENUBUTTON

SHOWBUTTON

The position of the cursor is held in two integer fields:

dd->d_x , dd->d_y

The coordinate value is updated on subsequent calls to ipxwait().

A.10. KEYBOARD EVENTS

The last character typed is held in inform.key field. It is the responsibility of the developer to buffer any strings.

A.11. USER_ACTION EVENTS

This class of event descrides the user activity by responding and reporting the following mouse and window events. It is anticipated that this event category will be extended to include "plan to" and "not plan to" (after [SZEKELY 90]) in order to anticipate and respond appropriately to the user's intentions .

A.11.1. DESELECT

As soon as the cursor leaves the frame of the window the following event is reported:

/meta-concept@USER_ACTION@:deselect

A.11.2. SELECT

As the cursor wanders into a window the event is reported. The event handler within the forms package determines which window the cursor is in and thus determines the users current focus of attention which is then used to anticipate further selections (see *Directing input*, section 4.7.10.8.).

A.11.3. DEFOCUS

This event type indicates that the window has become iconic and therefore no longer the focus of the user's attention. The event is reported as:

concept@USER_ACTION@iconised

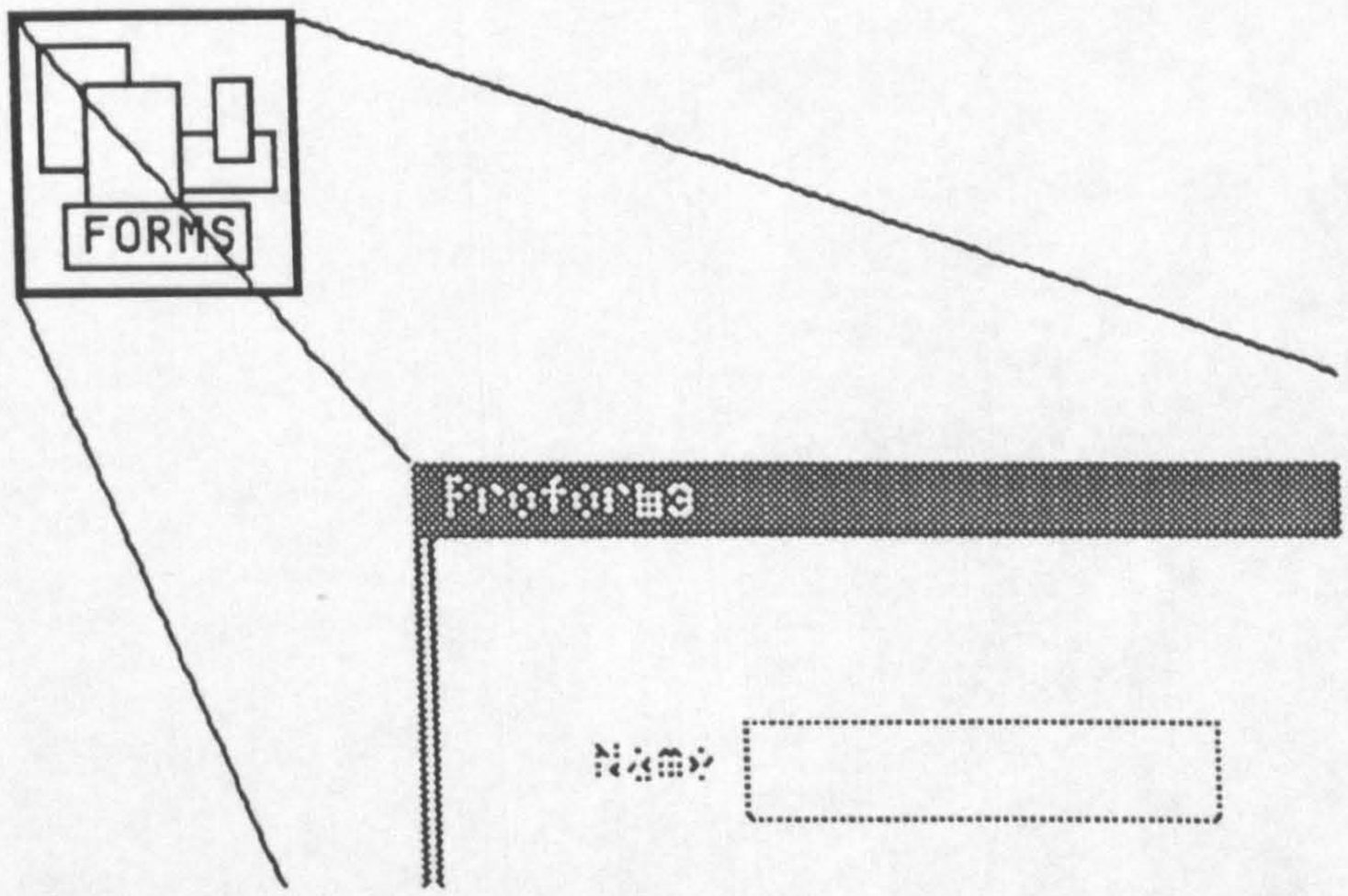


Figure A.11.3.1. Defocus

While in this closed state it is possible to write directly into the window icon and therefore provide feedback to the user in the form of animated image sequences or textual messages. Below is a sample routine for producing an animated sequence of images in the icon as it closes. Note that an interupt must be provided to enable the user to interupt the animation otherwise the sequence would have to be complete before the window may be opened.

ADD Animation routine

A.11.4. FOCUS

This indicates that the previously iconic window has been opened and is therefore is in the user's frame of attention. The event is reported as:

meta-concept@USER_ACTION@focused

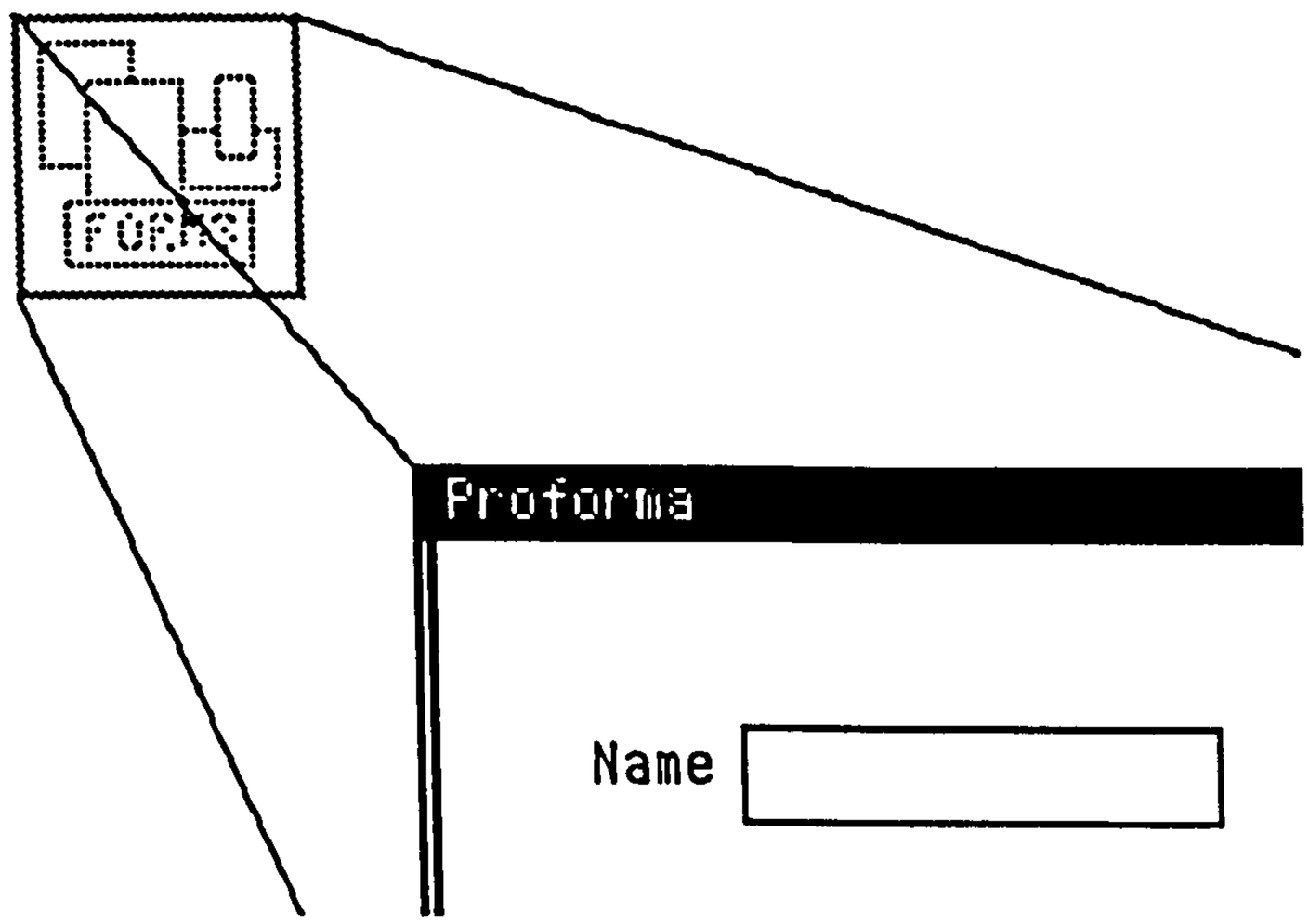


Figure A.11.4.1. Focus event - the window (desk) has been opened.

A.11.5. RESIZE EVENT

A window resize event may be interpreted as either a request for more information (a larger work space) or perhaps even an indication of saturation by a reduction of the window area. This event is reported as:

fieldname@USER_ACTION@re-size L H

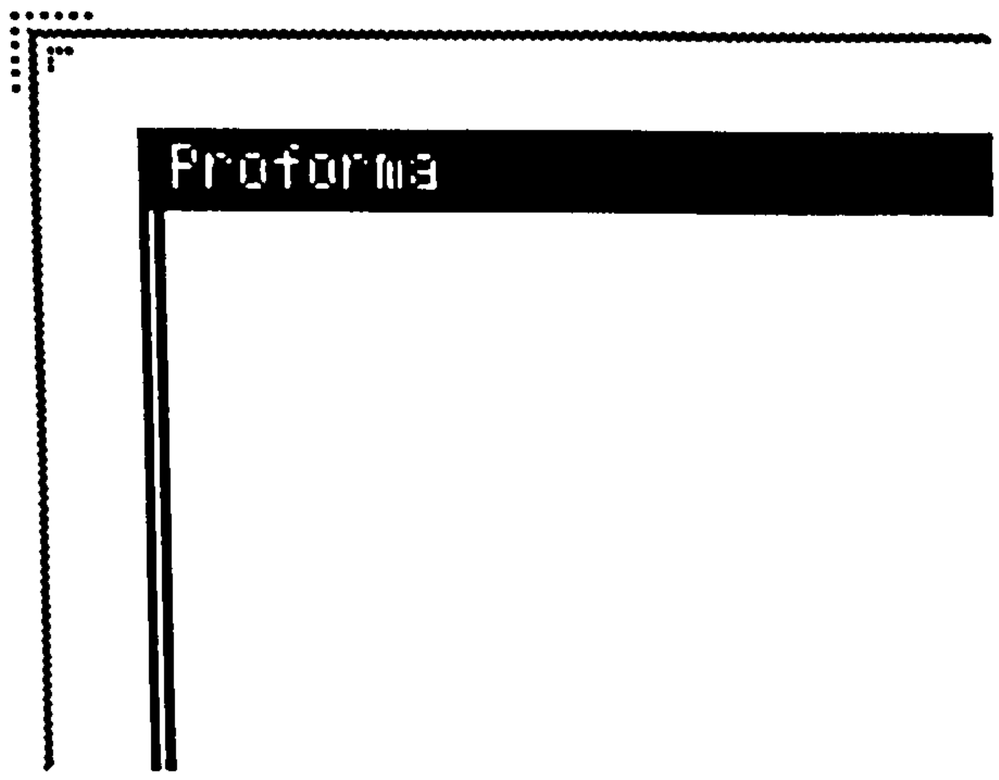


Figure A.11.5.. Resize event - the window has changed size.

The new window size way be obtained from ddwin->w_bm->bm_box. All window repairs must be handled.

A.11.6. KILL EVENT

In some situation the user may request that the current application be terminated. This should ideally be handled by the provision of "quit" button. However, window manager options often include a quit event, as indicated in figure A.11.6.1. In this situation the window manager should not have the power to terminate an application as this may be too catestrophic. Therefore this SIGnal event is trapped and reported as:

meta-concept@USER_ACTION@abort

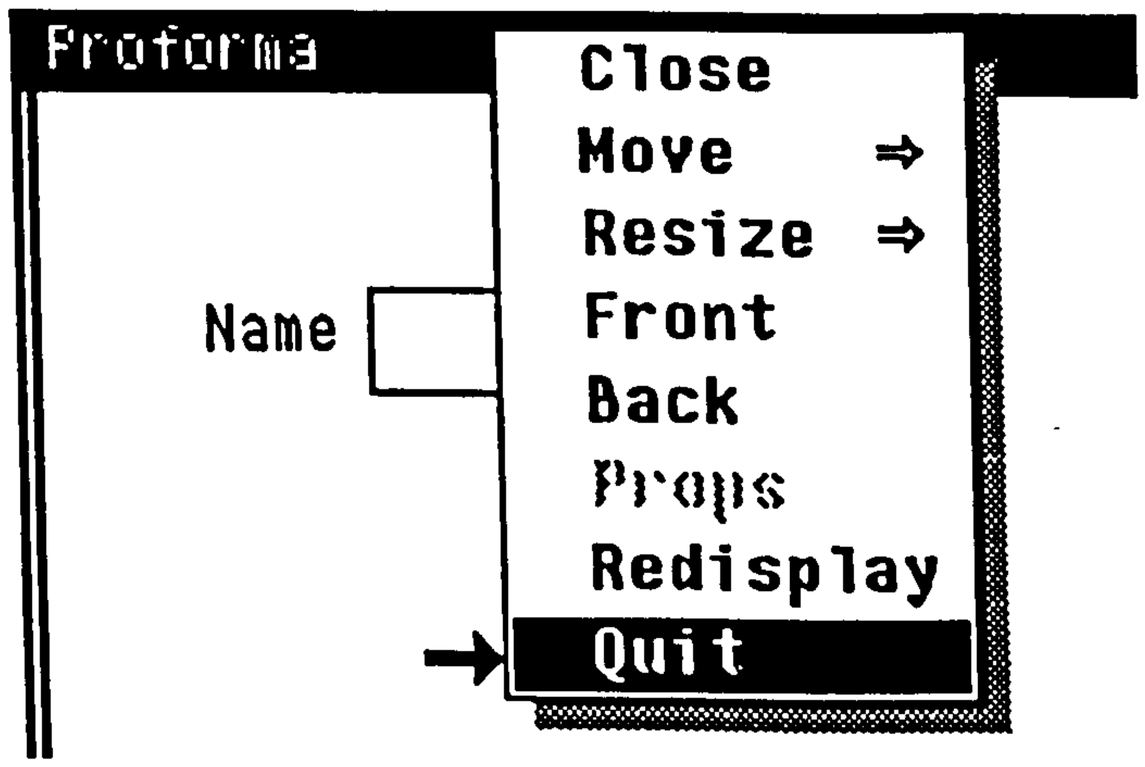


Figure A.11.6.1. The user wishes to terminate the dialogue.

This prevents the user destroying the window and thus terminating the dialogue. It enables the application to exit gracefully (confirmation dialogue boxes) and tidily. This was initially handled by the forms package sig_trap() routine but has since been incorporated into the ww library.

A.12. USER_QUERY EVENTS

These events are request for information, either help, descriptions or default and previous values. The request takes the form of the following formatted utterance:

/meta-concept/concept?USER_QUERY?X

where; X may be either: help,
description
example
default value
previous value

The value of X is obtained from the concept menu (section 4.7) and may be re-defined (extended) by the developer (section 4.9.2.).

A.13. TRANSMITTING MESSAGES TO THE FORMS PACKAGE

Message to the forms package follow the same communications protocol (chapter 4.8.4.) for incoming events:

<concept><operator><value>

A number of procedures (or macros) have been defined to facilitate message transmission based upon a single message handling routine; *ConFom()*.

A.13.1. CONFORM

The basic message passing procedure is ConForm (Control Forms):

ConForm(char* concept, char* operator, char* value);

ConForm() simply formats and write message to *stdout*.

A.13.2. ADDRESSING CONCEPTS

The means of addressing concepts is described fully in chapter 4.8.4). The following macros have been defined around ConFom resulting in a high level neutral language (MacRandal).

A.13.3. HIGH LEVEL DIALOGUE INTERFACE

A number of macros have been built around the ConForm procedure providing a high level dialogue interface. The vocabulary is is illustrated below by both the macro definition and the corresponding graphical event.

A.13.3.1. ASK USER

Present the user with a question.

```
#define ask_user(concept) \
                ConForm(concept,DISPLAY," ");
```

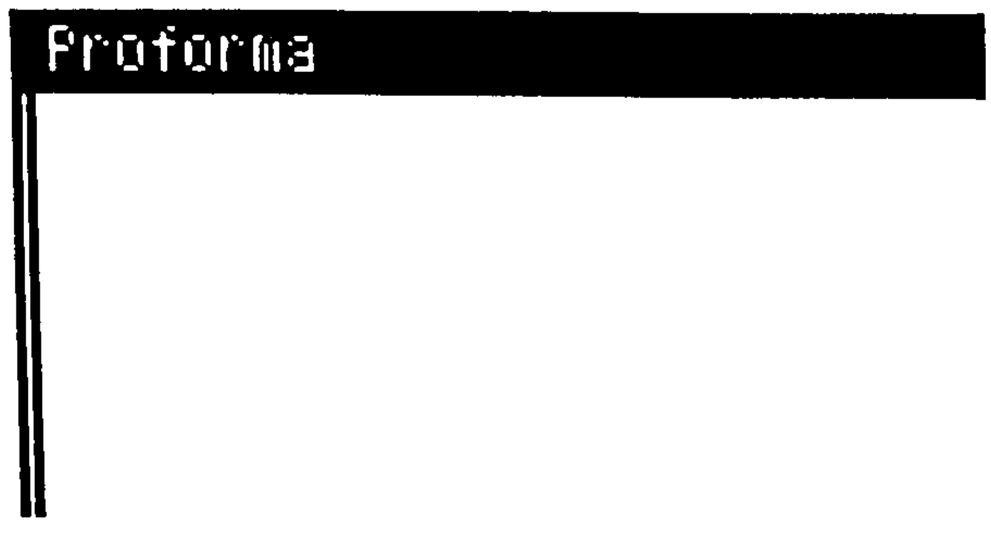


Figure A.13.3.1.1. Ask user - initial state

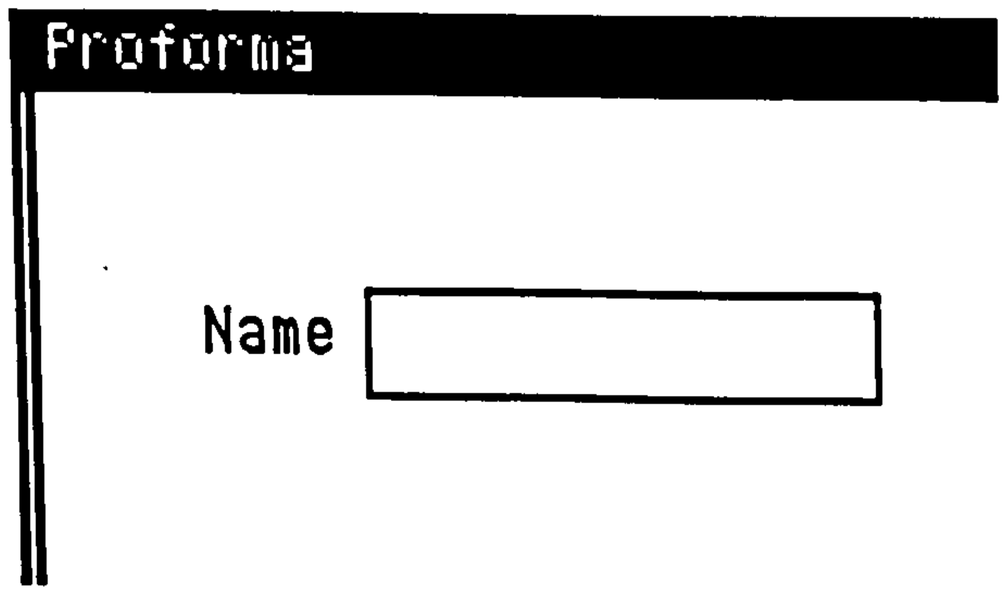


Figure A.13.3.1.2. ask_user("username");

A.13.3.2. UNASK USER

Remove a question.

```
#define unask_user(concept)\
                ConForm(concept,HIDE," ");
```

A.13.3.3. TELL USER

Tell the user the value of a concept.

```
#define tell_user(concept,value)\
                ConForm(concept,SET_CURRENT,value);
```

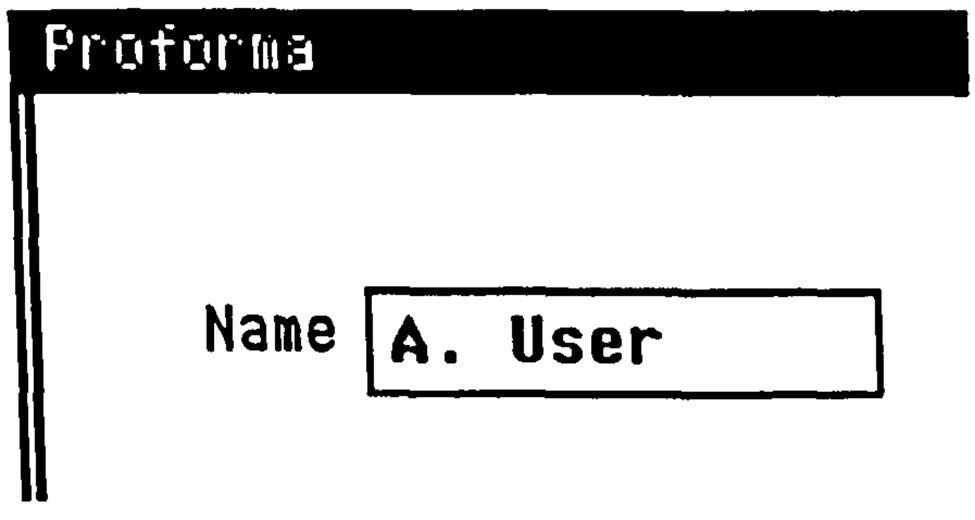


Figure A.13.3.3.1. tell_user("@username",getenv("LOGNAME"));

A.13.3.4. OFFER_USER

```
#define offer_user(concept,value)\
                ConForm(concept,SET_MENU,value);
```

Figure A.13.3.4.1. offer_user("materials","brick|block|concrete|timber");

A.13.3.5. SUGGEST_USER

```
#define suggest_user(concept,value)                    \
                ConForm(concept,SET_DEFAULT,value);
```

Note that the text is printed using the application font which in this case is bold.

A.13.3.6. FOCUS_USER

Set the focus for discussion to a particular meta-concept

A.13.3.7. DEFOCUS_USER

Converse of focus_user.

A.13.3.8. CURRENT FOCUS

```
#include "method_strings.h"

#define current_focus(type,meta_concept)                    \
                if(!strncmp(inform.user_param,"on"))        \
                        Focus(type,meta_concept);           \
                else if(!strncmp(inform.user_param,"off"))  \
                        Defocus(type);
```

This macro enables the user to change the focus of discussion. This is achieved by defining a button field with the name:

b_meta_concept

and a form with the name:

meta_concept.

The macro when used, determines the focus type whether to focus or de-focus the user towards or away from a meta_concept. The current meta_concept is defocused and the button used to focus it is set to "off".

A.13.3.9. FIELD DUPLICATION

A field duplication handler has been written to manage lists of concepts. For example it is often necessary to create lists of vertices. This is achieved by duplicating the physical and behavioural properties and characteristics of a base field. An incremental vertical and horizontal offset must be defined for the base field in order to establish the physical direction of growth. This is achieved by the additional field attributes:

horizontal offset <int>

vertical offset <int>

The means of duplicating a field (concept) is achieved using:

parent:duplicate:concept * N-times

This results in 'N' physical copies (appropriately offset) of the named concept which is adopted by the named parent. Not all fields may be *duplicatable*. Therefore in order to flag a field as an element of a notional array the suffix:

concept **[n]**

must be added to the concept name. This identifies specific list entries which may be manipulated in the normal manner. It must also be noted that only the physical properties of the field are duplicated, concept values are not unless the '!' (literal) flag is present:

parent:duplicate!:concept * N-times

in which case the new copy inherits current, previous, default and optional (menu) values.

A similar suffix may also be added to the field label to provide a visual guide. Both array concept and label suffixes are automatically incremented with additional calls. The depth of the array must, however, be maintained by the developer. A list (or duplicate) handler has been defined to manage the task, figure A.13.3.9.1.

```
/** Duplicate.c Copyright (C) James H. Rutherford. ABACUS 1988 **

#include <stdio.h>
#include "memory.h"

#ifndef NULLPTR
#define NULLPTR(x) ((x*)0)
#endif  NULLPTR


typedef struct _instance {
        char*     rootname;
        int       size;
        int       displayed;
        struct _instance* next;
}instance;

static struct {
         instance* root;
         instance* tail;
}duplicate_list;

instance* CreateDuplicate(name) char* name;
{
        instance* new = (instance*) cmalloc(sizeof(instance));

        if(new!=NULL)
        {
                new->rootname = stralloc(name);
                new->size = 0;
                duplicate_list_append(new);
        }
        return(new);
}

instance* CheckDuplicateList(name) char* name;
{
         instance *iptr = duplicate_list.root, *found = NULL;

        while(iptr != NULL)
        {
                if(!strcmp(iptr->rootname,name))
                {
                        found = iptr;
                        break;
                }
                iptr = iptr->next;
        }
        return(found);
}

duplicate_list_append(obj) instance* obj;
{
         if(obj == NULLPTR(instance)) return;
         if(duplicate_list.root == 0)
                 duplicate_list.root = obj;
         else
                 duplicate_list.tail->next = obj;
         duplicate_list.tail = obj;
}
```

Figure A.13.3.9.1 (a). Duplicate list handler

```
Append(name,num) char* name; int num;
{
        instance *iptr;
        char cmd[512];

        iptr=((iptr=CheckDuplicateList(name))==NULL)?CreateDuplicate(name):iptr;
        if(iptr==NULLPTR(instance))
        {
                fprintf(stderr,"Append::failed to create %s[%d]\n",name,num);
                return;
        }
        /*** perhaps redisplay field[0] to field[iptr->size] ***/
        sprintf(cmd,"@%s[%d]*%d\n",iptr->rootname,iptr->size,num);
        iptr->size+=num;
        Conform("/","duplicate",cmd);
}

Retract(name,num) char* name; int num;
{
        instance *iptr;
        char cmd[512];
        register int i=0;

        iptr=((iptr=CheckDuplicateList(name))==NULL)?CreateDuplicate(name):iptr;
        if(iptr==NULLPTR(instance))
        {
                fprintf(stderr,"Append::failed to create %s[%d]\n",name,num);
                return;
        }

        for(i=iptr->size; i>num; i--)
        {
                sprintf(cmd,"@%s[%d]",iptr->rootname,i);
                Conform(cmd,"hide"," ");
        };
}

Conform(field, method, param) char *field, *method, *param;
{
#ifdef DEBUG
        fprintf(stderr,"%s:%s:%s\n",field,method,param);
#endif DEBUG
        ConForm(field, method, param);
}
```

Figure A.13.3.9.1.(b). Duplicate list handler

```
Duplicate(name,num)
{
        instance *iptr;
        int diff = 0;
        register int i = 0;
        char cmd[512];

        iptr=((iptr=CheckDuplicateList(name))==NULL)?CreateDuplicate(name):iptr;
        if(iptr==NULLPTR(instance))
        {
                fprintf(stderr,"Duplicate::failed to create %s[%d]\n",name,num);   return;
        }
        if(num > iptr->size)          /** extend list **/
        {       /** redisplay existing fields **/
                if(iptr->size > 0)
                {
                   for(i=0; i<=iptr->size; i++)
                   {
                        sprintf(cmd,"@%s[%d]",iptr->rootname,i);
                        Conform(cmd,"display"," ");
                   }
                }
                /** create new fields **/
                diff = num - iptr->size;
                sprintf(cmd,"@%s[%d]*%d\n",iptr->rootname,iptr->size,diff);
                iptr->size+=diff;
                Conform("/","duplicate",cmd);
}else if(num < iptr->size)    /** close down list **/
{
                for(i=iptr->size; i>=num; i--)
                {
                        sprintf(cmd,"@%s[%d]",iptr->rootname,i);
                        Conform(cmd,"hide"," ");
                };
                for(i=0; i<=num; i++)
                {
                        sprintf(cmd,"@%s[%d]",iptr->rootname,i);
                        Conform(cmd,"display"," ");
                }
        }else if(num == iptr->size) /** redisplay **/
        {
                for(i=0; i<=iptr->size; i++)
                {
                        sprintf(cmd,"@%s[%d]",iptr->rootname,i);
                        Conform(cmd,"display"," ");
                };
        }
}
```

Figure A.13.3.9.1 (c). Dulicate list handler

An initial dilema regarding an appropriate location for this functioanlity was encountered; should the handler be inserted into the forms package or the dialogue handler? As a duplicate facility exists within the forms package exists it has been decided that the management of lists should be handled externally.

Therefore it is suggested that the overloaded operator call *duplicate* be added to the dialogue handler. A suitable protocol also needs to be established, the one suggested below may be adequate.

The duplication handler maintains a list of duplicated fields together with the total size of the list. Below are a few example calls to illustrate the routine.

Duplicate(x_coord,10);	creates 10 copies of the field x_coord[0]
Duplicate(x_coord,4);	hides x_coord[5] - x_coord[11]
Duplicate(x_coord,7);	redisplays x_coord[0] - x_coord[8]
Duplicate(x_coord,15);	creates an additional 5 copies
Duplicate(vertex/x_coord,15);	as above except narrow down the seacrh

Figure A.13.3.9.2. Field duplication protocol

The 'Duplicate' routine is obviously overloaded; creating and re-displaying fields. However, the routine was meant to act as a list handler and as such should make field duplication/creation and re-display opaque to the developer. The usual methods of hiding and setting fields may also be used, thus enhancing the facility.

NOTE: The base field x_coord[0] must already exist on the proforma and that the concept name argument to the Duplicate function does not include the array suffix. This is maintained within an internal (private) hash table ensuring that the duplication functionality is totally transparent. Prolog, used for knowledge encapsulation, also makes the use of square brackets difficult. Therefore, in order to cater for fieldnames being passed through prolog predicates the routine strips off square brackets.

A corresponding c utility needs to be written for the initial_kb to enable kbs to create duplicate lists. This should be trivial. The anticipated syntax is:

duplicate(fieldname,number_required)

Individual list members may be controlled in the usual manner using commands such as:

@element[n]:set current:value

A.14. EXAMPLE APPLICATIONS

The functionality of the forms package may be accessed by a simple procedure outlined below:

i) define NotifyProc lists for the three events (set, action, and query)

ii) specify a SetDefaults procedure.

iii) call Forms(argv).

The example application bellow illustrates the basic procedures.

```
#include "inform.h"

SetName()
{
     if(name!=NULLPTR(char))
        free(name);
     name = stralloc(inform.user_param);
};

SetAge()
{
     age = atoi(inform.user_param);
};

set_defaults()
{
     offer_user("name",getenv("LOGNAME");
};

inform.NotifyProc[] = {
     "name",        SetName,
     "age",         SetAge,
     0
};

inform.SetDefaults = set_defaults;

main(argc,argv)int argv; char** argv;
{
     Forms(argv);
};
```

Figure A.14.1. Simple program.

More complex application structures are possible by segmenting the calls as illustrated in the following application vpict. The application is a simple tool for browsing through a series of predefined depth cued images developed for the Central Electricty Generating Board (CEGB). These pre-processed images are displayed in a GKS workstation display window. Although a command line interpreter was included in the original program, for the purpose of retrieving images during demonstrations a graphical front end, using the forms package, was bolted on, figure A.14.2. Other programs such as the PixEd (Pixel Editor) have also been developed.

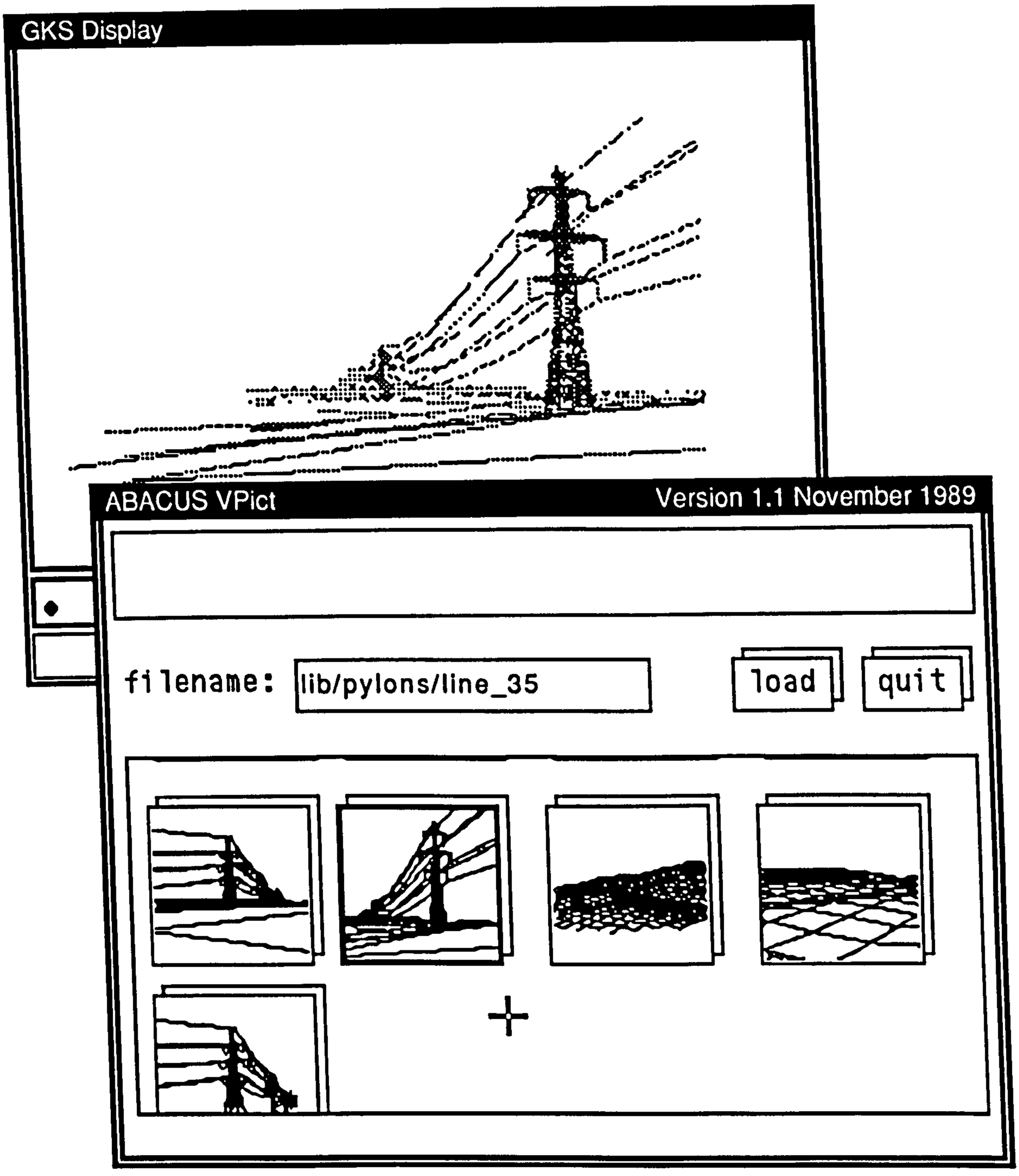


Figure A.14.2. Vpict; GKS display with forms interface.

The following code illustrates how a single graphical application may have three interfaces:

i) native
ii) forms
iii) resource - used for resource integration into larger systems

This is achieved by a series of #defines (compilation switches) in main() as illustrated in figure A.14.3.

```
/* vpict.c Copyright (c) James H. Rutherford ABACUS. 1989
        vpict        Displays colour depth cued pluto images
*/
#include <stdio.h>
#include "inform.h"
#include "memory.h"

#ifndef   NULLPTR
#define   NULLPTR           (x) ((x*)0)
#endif    NULLPTR

extern int ReadFile();
extern int CloseDownVpict();

char* filename;

SetFilename()
{
        if(filename!=NULLPTR(char)) free(filename);
        filename = stralloc(inform.user_param);
}

LoadFile() /** display if notified **/
{
        if(filename!=NULLPTR(char))    ReadFile(filename);
        else                           Feedback("Please enter filename");
};

DisplayFile() /** immediate display **/
{
          ReadFile(inform.user_param);
}

NotifyProc Procs[] = {
        "filename",        SetFilename,
        "display",         DisplayFile,
        "load",            LoadFile,
        "quit",            CloseDownVpict,
        0
};

int Defaults(){};

main(argc,argv)int argc; char** argv;    /** Pick up command line arguments **/
{
        OpenWindow();                    /** Open GKS workstation **/
#ifdef NATIVE
        WindowMainLoop();                /** Call native event handler **/
#else
#ifdef FORMS                             /** Forms interface **/
        Forms(argv,Procs,Defaults);      /** Initialise forms package **/
#else RESOURCE
        inform.chanel[READ] =0;          /** Read from stdin **/
        inform.chanel[WRITE]=1;          /** Write to stdout **/
        init_inform();                   /** Initialise system defaults **/
        init_events();                   /** Set default event handlers **/
        UsrProc=Procs;                   /** Set concept handlers **/
        Defaults();                      /** Initialise default values **/
        InForm();                        /** Poll events **/
#endif FORMS
#endif STANDALONE
};
```

Figure A.14.3. VPict. Note that the GKS code is ommited for clarity.

APPENDIX B

Proforma Templates

```
new desk
        name                IFE #3.1 February 1990
        origin              23 38 pixels
        size                915 655 pixels
        help                This is the IFE master form
        data font           /usr/lib/fonts/fixedwidthfonts/serif.b.10
        label font          /usr/lib/fonts/fixedwidthfonts/cour.b.10
        application font    /usr/lib/fonts/fixedwidthfonts/cour.r.10
end desk

        new form
                name                chat_area
                origin              453 30 pixels
                size                443 206 pixels
                shade               light grey
                data font/usr/lib/fonts/fixedwidthfonts/serif.r.10
                label font/usr/lib/fonts/fixedwidthfonts/serif.r.10
                application font    /usr/lib/fonts/fixedwidthfonts/serif.r.10
                border              2
                selection border    2
end form

                new field
                        name                chat_user
                        type                character
                        origin              6 28 pixels
                        size                428 169 pixels
                        label string        Feedback
                        label position      fit above
                        help                A text area for general chit-chat from the ife\n\
                                            description. This field is used by the ife to display\n\
                                            non urgent messages, suggestions and\n\
                                            session status info. It scrolls, so\n\
                                            previous messages can be recovered.
                        border              0
                        selection border    0
                end field

                parent form         /

        new field
                name                user_name
                type                alphanum + .
                origin              95 32 pixels
                size                179 20 pixels
                label string        Name
                label position      left
                selection border    1
                help                The users name (ie. yours)
                description         This field is used to maintain\n\
                                    a cronological record of the\n\
                                    people working on the project.
        end field
```

```
new field
        name            user_type
        type            popup
        origin          289 33 pixels
        size            64 64 pixels
        label string    User type
        label position  fit above
        selection border 0
        border          8
        menu            modeller\
                        architect\
                        engineer
        help            for info
        description     This button identifies the currently\n\
                        selected user type. This dictates the\n\
                        style and content of the forms to be\n\
                        displayed during this session. An\n\
                        attempt will be made to ask only for\n\
                        information liable to be available to\n\
                        this user, using the internal knowledge\n\
                        to provide values for the more technical\n\
                        energy computing aspects. All feedback\n\
                        given will also reflect this bias.
end field
new field
        name            user_level
        type            button
        origin          361 33 pixels
        size            65 65 pixels
        label string    User level
        label position  fit above
        selection border 0
        menu            expert\
                        novice
        help            for info
        description     This button identifies the currently\n\
                        selected user level. This dictates\n\
                        the level and style of help/guidance\n\
                        given during the interaction and also\n\
                        influences the content of some of the\n\
                        forms shown. An attempt will be made\n\
                        to ask only for information liable to\n\
                        be available to this user, using the\n\
                        internal knowledge to provide values\n\
                        for the more esoteric aspects.
end field
```

```
new field
        name            date
        type            date
        origin          818 5 pixels
        size            80 18 pixels
        label string    Date
        label position  left
        selection border 1
        help            Todays date (supplied by system)
        description     This field is used to timestamp\n\
                        any modifications made during this\n\
                        session.  It is used to maintain a\n\
                        cronological record of the people\n\
                        working on the project and the\n\
                        modifications they instigate.
end field
new field
        name            started
        type            date
        origin          97 135 pixels
        size            81 20 pixels
        label string    Date started
        label position  left
        selection border 1
        start           hidden
        help            Date of first session for this project
        description     This field gives the start date for\n\
                        the work on this project, ie. the date\n\
                        of the 1st ife session.
end field
new field
        name            session
        type            integer
        origin          411 135 pixels
        size            18 20 pixels
        label string    Session number
        label position  left
        selection border 1
        start           hidden
        help            Session number of this consultation
        description     This field is used to timestamp any\n\
                        modifications made during this session.\n\
                        It allows a cronological record of the\n\
                        people working on the project to be\n\
                        maintained.
end field
```

```
new field
        name            project
        type            file
        origin          97 105 pixels
        size            333 20 pixels
        label string    Project name
        label position  left
        selection border 1
        help            project name or identifier
        description     This field is used to identify the\n\
                        required project so the data can\n\
                        be retrieved later.  It is also used\n\
                        to maintain a cronological record of\n\
                        the people working on the project.
end field
new field
        name            m_focus
        type            label
        origin          149 160 pixels
        size            152 20 pixels
        start           hidden
        current value   Topics for Discussion
        help            push a button (below) to change topic of discussion.
        description     These buttons switch the focus of\n\
                        discussion to the requested topic.\n\
                        The relevent forms will be displayed\n\
                        below (existing ones may disappear).\n\
                        It is suggested that buttons are\n\
                        worked through from left to right to\n\
                        minimize specification of redundant\n\
                        information.
        selection border 0
        border          0
        label font      /usr/lib/fonts/fixedwidthfonts/serif.r.10
        application font /usr/lib/fonts/fixedwidthfonts/serif.r.10
        data font       /usr/lib/fonts/fixedwidthfonts/serif.r.10
end field
```

```
new form
        name            topics
        origin          10 186 pixels
        size            419 50 pixels
        label colour    black
        start           hidden
end form

        new field
                name            b_spec
                type            button
                origin          12 20 pixels
                size            156 20 pixels
                label position  fit above
                selection border 0
                menu            building description
end field

new field
                name            b_analysis
                type            button
                origin          246 20 pixels
                size            156 20 pixels
                label position  fit above
                selection border 0
                menu            building analysis
end field
```

```
new form
        name                bld_spec
        origin              12 249 pixels
        size                885 380 pixels
        shade               mid grey
        start               hidden
        help                This is the building specification top level form
        data font           /usr/lib/fonts/fixedwidthfonts/serif.b.10
        label font          /usr/lib/fonts/fixedwidthfonts/cour.b.10
        application font    /usr/lib/fonts/fixedwidthfonts/cour.r.10
        border              1
        selection border    2
end form

        new field
        name                location
        type                alpha
        origin              67 6 pixels
        size                109 16 pixels
        label string        Location
        label position      left
        selection border    1
        default value       glasgow
        menu                use_map\
                            \
                            glasgow\
                            belfast\
                            edinburgh\
                            aberdeen\
                            london\
                            manchester\
                            birmingham\
                            newcastle\
                            cardiff
        help                Geographical location of building (MAP -> point to a map).
        description         This data is used to determine site related\n\
                            information such as climate data and sun position.
end field

new field
        name                latitude
        type                real            +-+.NSEW
        origin              176 28 pixels
        size                33 18 pixels
        start               hidden
        label string        Latitude
        label position      left
        selection border    1
        help                Latitude, degrees North
        description         This field is used to calculate sun\n\
                            position during the simulation.  It\n\
                            is also used to help select an\n\
                            appropriate climate set to provide\n\
                            default boundary conditions.
end field
```

```
new field
        name                longitude
        type                real +-+.NSEW
        origin              176 50 pixels
        size                33 16 pixels
        start               hidden
        label string        Longitude
        label position      left
        selection border    1
        help                Longitude, degrees West
        description         This field is used to calculate sun\n\
                            position during the simulation.  It\n\
                            is also used to help select an\n\
                            appropriate climate set to provide\n\
                            default boundary conditions.
end field
new field
        name                timezone
        type                alphanum
        origin              176 71 pixels
        size                33 16 pixels
        label string        Timezone
        label position      left
        selection border    1
        current value       GMT
        default value       BST
        start               hidden
        menu                GMT\
                            WET\
                            CET\
                            EET\
                            MET\
                            EPT\
                            WAT
        help                Timezone of location
        description         This field is used to calculate sun\n\
                            position during the simulation.  It\n\
                            is also used to help select an\n\
                            appropriate climate set to provide\n\
                            default boundary conditions.
end field
new field
        name                environment
        type                alpha + _
        origin              88 108 pixels
        size                88 18 pixels
        label string        Environment
        label position      left
        selection border    1
        menu                city_centre\
                            urban\
                            rural
        help                Environment of building
        description         This field is used to estimate the\n\
                            effect that the building surroundings\n\
                            will have on its performance.
end field
```

```
new field
        name            exposure
        type            integer
        origin          177 133 pixels
        size            11 18 pixels
        start           hidden
        label string    Exposure
        label position  left
        selection border 1
        default value   3
        help            Site exposure index
        description     If you don't know, don't use this environment,\n\
                        use a standard one.
end field
new field
        name            grnd_rflct
        type            real
        origin          177 154 pixels
        size            32 18 pixels
        start           hidden
        label string    Ground reflectivity
        label position  left
        selection border 1
        default value   1.25
        help            Ground reflectivity index
        description     If you don't know, dont use this\n\
                        environment, use a standard one
end field
new field
        name            function
        type            alpha
        origin          87 190 pixels
        size            88 20 pixels
        label string    Function
        label position  left
        selection border 1
        menu            residential\
                        commercial\
                        industrial\
                        hospital\
                        school
        help            Function of the building.
        description     This field shows the generic function of\n\
                        the building.  This influences a range\n\
                        of building attributes, especially those\n\
                        specified in on the "occupation" forms.
end field
```

```
new field
        name                b_bld_spec_focii
        type                button
        origin              6 238 pixels
        size                158 20 pixels
        label position      fit above
        selection border    1
        menu                Detailed specification
end field
new field
        name                b_bld_browse
        type                button
        origin              170 239 pixels
        size                47 20 pixels
        label position      fit above
        selection border    1
        menu                Browse
end field
```

```
new form
        name                bld_spec_focii
        origin              9 270 pixels
        size                211 89 pixels
        start               hidden
        shade               light grey
        help                Push a button to change\n\
                            topic of discussion.
        description         These buttons switch the focus of\n\
                            discussion to the requested topic.  The\n\
                            relevent forms will be displayed below\n\
                            (existing ones will disappear).  It is\n\
                            suggested that they are worked through\n\
                            from left to right to minimize\n\
                            specification of redundant information.
        border              1
        selection border    0
        label colour        black
        data font           /usr/lib/fonts/fixedwidthfonts/serif.r.10
        application font    /usr/lib/fonts/fixedwidthfonts/serif.r.10
        label font          /usr/lib/fonts/fixedwidthfonts/serif.r.10
end form

        new field
                name                b_geometry
                type                button
                origin              4 21 pixels
                size                52 18 pixels
                label position      fit above
                selection border    1
                menu                geometry
                help                push when ready for building geometry input forms
                description         This button switches the focus of\n\
                                    discussion to the description of the\n\
                                    (proposed?) building's geometry.\n\
                                    Subsidary forms will enable the\n\
                                    geometry input method to be chosen\n\
                                    and will allow editing of existing data.
        end field
        new field
                name                b_construction
                type                button
                origin              64 21 pixels
                size                78 18 pixels
                label position      fit above
                selection border    1
                menu                construction
                help                push when ready for construction definition forms
                description         This button switches the focus of\n\
                                    discussion to the description of the\n\
                                    (proposed?) buildings construction\n\
                                    characteristics.  Subsidary forms\n\
                                    will enable the use of various default\n\
                                    patterns for the different aspects of\n\
                                    the buildings materials.  Editing of\n\
                                    existing data will be possible
        end field
```

```
new field
        name            b_useage
        type            button
        origin          150 21 pixels
        size            52 18 pixels
        label position  fit above
        selection border 1
        menu            useage
        help            push when ready for building useage forms
        description     This button switches the focus of\n\
                        discussion to the description of the\n\
                        (proposed?) buildings occupation\n\
                        characteristics.  Subsidary forms will\n\
                        enable the use of various default\n\
                        patterns for the different aspects of\n\
                        the buildings occupation.  Editing of\n\
                        existing data will be possible
end field

new field
        name            b_connectivity
        type            button
        origin          63 59 pixels
        size            78 18 pixels
        label position  fit above
        selection border 1
        menu            connectivity
        help            push when ready to specify zone connectivity
        description     This button switches the focus of\n\
                        discussion to describing the\n\
                        connectivity of the building and its\n\
                        HVAC system.
end field

new field
        name            b_site
        type            button
        origin          4 97 pixels
        size            60 17 pixels
        label position  fit above
        selection border 1
        menu            site
        help            push when ready for site description forms
        description     This button switches the focus of\n\
                        discussion to describing the site\n\
                        for the building.
end field
```

```
new field
        name                b_shading
        type                button
        origin              72 97 pixels
        size                63 18 pixels
        label position      fit above
        selection border    1
        menu                shading
        help                push when ready for shading forms
        description         This button switches the focus of\n\
                            discussion to the shading of the\n\
                            (proposed?) building.  The forms\n\
                            will enable details of surrounding\n\
                            buildings to be entered, as well as\n\
                            the shading mechanism and their\n\
                            operational strategy provided on\n\
                            the building.
end field
new field
        name                b_shading
        type                button
        origin              143 97 pixels
        size                59 18 pixels
        label position      fit above
        selection border    1
        menu                shading
        help                push when ready for shading forms
        description         This button switches the focus of\n\
                            discussion to the shading of the\n\
                            (proposed?) building.  The forms\n\
                            will enable details of surrounding\n\
                            buildings to be entered, as well as\n\
                            the shading mechanism and their\n\
                            operational strategy provided on\n\
                            the building.
end field
new field
        name                b_airflow
        type                button
        origin              143 97 pixels
        size                59 18 pixels
        label position      fit above
        selection border    1
        menu                airflow
        help                push when ready for air flow description forms
        description         This button switches the focus of\n\
                            discussion to an airflow analysis\n\
                            of the building.  The data required\n\
                            includes external pressure\n\
                            distribution as well as the leakage\n\
                            distribution of the building.
end field
```

```
new field
        name            b_plant
        type            button
        origin          4 137 pixels
        size            60 18 pixels
        label position  fit above
        selection border 1
        menu            plant
        help            push when ready for plant description forms
        description     This button switches the focus of\n\
                        discussion to a description of the\n\
                        (proposed?) plant. Subsidary forms\n\
                        will enable the components and\n\
                        placements to be specified. Also,\n\
                        subject to earlier input about\n\
                        analysis type and design stage, the\n\
                        plant control strategy / mechanism\n\
                        can can be given.
end field

new field
        name            b_control
        type            button
        origin          72 137 pixels
        size            63 18 pixels
        label position  fit above
        selection border 1
        menu            control
        help            push when ready for plant control forms
        description     This button switches the focus of\n\
                        discussion to a description of the\n\
                        control systems and operations to\n\
                        be used for the analysis. The\n\
                        information required depends on the\n\
                        type of analysis requested earlier.
end field

new field
        name            services
        type            button
        origin          143 137 pixels
        size            59 18 pixels
        label position  fit above
        selection border 1
        menu            services
end field
```

```
new form
        name                geometry
        origin              237 6 pixels
        size                642 367 pixels
        start               hidden
        shade               mid grey
        help                This is the geometry input top level form
        data font           /usr/lib/fonts/fixedwidthfonts/serif.b.10
        label font          /usr/lib/fonts/fixedwidthfonts/cour.b.10
        application font    /usr/lib/fonts/fixedwidthfonts/cour.r.10
        border              1
        selection border    0
end form

        new field
                name                b_g_form_fill
                type                button
                origin              192 20 pixels
                size                74 18 pixels
                label position      fit above
                selection border    1
                menu                form_fill
                help                Select form fill method of inputting zone geometries
                description         Geometry can be input one zone at a time according\n\
                                    to shape type.
        end field
        new field
                name                b_g_draw
                type                button
                origin              285 20 pixels
                size                74 18 pixels
                label position      fit above
                selection border    1
                menu                draw
                help                Select graphical method of inputting zone geometries
                description         Geometry can be defined using the ife's geometry\n\
                                    modeller.
        end field
        new field
                name                b_g_cad_file
                type                button
                origin              378 20 pixels
                size                74 18 pixels
                label position      fit above
                selection border    1
                menu                import
                help                Select file containing geometry
                description         Geometry available in a foreign format (known to the\n\
                                    ife) can be read in from a file prepared externally.
        end field
```

```
new field
        name                current_zone
        type                popup
        origin              5 20 pixels
        size                102 18 pixels
        label string        Current Zone
        label position      fit above
        selection border    1
        menu                ~~~ =0\
                            kitchen=1\
                            living=2\
                            dining=3\
                            bed1=4\
                            bed2=5
        help                The number of the zone currently displayed
        description         The current zone is the one shown below.  It\n\
                            can be changed to any other defined zone by\n\
                            selecting the zone name from the option menu\n\
                            (left mouse button on field label) or by using\n\
                            the right button in the field to toggle though\n\
                            all the options.
end field
```

```
new form
        name            g_draw
        origin          208 148 pixels
        size            233 90 pixels
        shade           white
        start           hidden
        label colour    black
        label position  fit above
        data font       /usr/lib/fonts/fixedwidthfonts/serif.b.10
        label font      /usr/lib/fonts/fixedwidthfonts/cour.b.10
        application font /usr/lib/fonts/fixedwidthfonts/cour.r.10
        border          1
        selection border 2
end form

new field
        name            drawing_package
        type            button
        origin          6 19 pixels
        size            64 64 pixels
        label position  fit above
        selection border 0
        menu            geom.ex=vim
end field

new field
        name            drawing_package
        type            button
        origin          81 19 pixels
        size            64 64 pixels
        label position  fit above
        selection border 0
        menu            acad.ex=acad
end field

new field
        name            drawing_package
        type            button
        origin          154 19 pixels
        size            64 64 pixels
        label position  fit above
        selection border 0
        menu            rove.ex=rove
end field
```

```
new form
        name                g_cad_file
        origin              20 58 pixels
        size                599 301 pixels
        shade               light grey
        start               hidden
        data font           /usr/lib/fonts/fixedwidthfonts/serif.b.10
        label font          /usr/lib/fonts/fixedwidthfonts/cour.b.10
        application font    /usr/lib/fonts/fixedwidthfonts/cour.r.10
        border              1
        selection border    0
end form

        new field
            name            geom_file
            type            file
            origin          135 11 pixels
            size            214 20 pixels
            label string    geometry filename:
            label position  left
            selection border 2
    end field

    new field
            name            geom_format
            type            popup
            origin          425 14 pixels
            size            73 20 pixels
            label string    format
            label position  left
            selection border 1
            menu            dxf\
                            viewer\
                            atocad\
                            acropolis
    end field

    new field
            name            import_file
            type            button
            origin          512 14 pixels
            size            74 18 pixels
            label position  fit above
            menu            import
            selection border 1
    end field

    new field
            name            geom_file_pic
            type            graphics
            origin          134 41 pixels
            size            368 230 pixels
            label string    perspective image
            label position  fit below
            selection border 1
    end field
```

```
new form
        name                    form_fill_geom
        origin                  3 44 pixels
        size                    120 318 pixels
        shade                   white
        selection border        1
end form

        new field
                name            zone_name
                type            character
                origin          4 15 pixels
                size            110 18 pixels
                label string    Zone name
                label position  fit above
                selection border 1
        end field

        new field
                name            shape_type
                type            popup
                origin          47 46 pixels
                size            64 64 pixels
                label string    Shape\ntype
                label position  left
                selection border 0
                border          8
                menu            rec\
                                reg\
                                gen
        end field

        new field
                name            edit_shape
                type            button
                origin          4 92 pixels
                size            33 18 pixels
                label position  fit above
                selection border 1
                menu            edit
        end field

        new field
                name            origin_label
                type            label
                origin          34 150 pixels
                size            49 16 pixels
                current value   origin
                selection border 0
                border          0
        end field

        new field
                name            x_origin
                type            real
                origin          4 184 pixels
                size            34 18 pixels
                label string    x
                label position  fit above
                selection border 1
        end field
```

```
new field
        name                y_origin
        type                real
        origin              42 184 pixels
        size                34 18 pixels
        label string        y
        label position      fit above
        selection border    1
end field
new field
        name                z_origin
        type                real
        origin              80 184 pixels
        size                34 18 pixels
        label string        z
        label position      fit above
        selection border    1
end field
new field
        name                orientation_label
        type                label
        origin              20 214 pixels
        size                82 16 pixels
        current value       orientation
        selection border    0
        border              0
end field
new field
        name                x_oriintation
        type                real
        origin              4 244 pixels
        size                34 18 pixels
        label string        x
        label position      fit above
        selection border    1
end field
new field
        name                y_orientation
        type                real
        origin              42 244 pixels
        size                34 18 pixels
        label string        y
        label position      fit above
        selection border    1
end field
new field
        name                z_orientation
        type                real
        origin              80 244 pixels
        size                34 18 pixels
        label string        z
        label position      fit above
        selection border    1
end field
```

```
new form
        name                rec_body
        origin              126 44 pixels
        size                511 318 pixels
        shade               white
        selection border    1
end form
        new field
                name                rec_image
                type                graphics
                origin              88 53 pixels
                size                323 202 pixels
                label position      fit above
                selection border    0
                border              0
                current value       image:rec_image.ex
        end field
        new field
                name                rec_height
                type                real
                origin              32 167 pixels
                size                49 18 pixels
                label string        Height
                label position      right
                selection border    1
        end field
        new field
                name                rec_width
                type                real
                origin              201 260 pixels
                size                49 18 pixels
                label string        Width
                label position      fit above
                selection border    1
        end field
        new field
                name                rec_depth
                type                real
                origin              404 167 pixels
                size                49 18 pixels
                label string        Depth
                label position      left
                selection border    1
        end field
```

```
new form
        name                reg_body
        origin              126 44 pixels
        size                511 318 pixels
        shade               white
        selection border    1
end form
        new field
                name                reg_vertices
                type                integer
                origin              5 15 pixels
                size                28 18 pixels
                label string        vertices
                label position      right
                selection border    1
        end field
        new field
                name                reg_height
                type                real
                origin              158 15 pixels
                size                49 18 pixels
                label string        height
                label position      fit above
                selection border    1
        end field
        new field
                name                reg_image
                type                graphics
                origin              158 38 pixels
                size                348 274 pixels
                label position      fit below
                selection border
        end field
        new form
                name                reg_plan
                origin              5 38 pixels
                size                147 274 pixels
        end form
        new field
                name                x_plan[1]
                type                real
                origin              34 7 pixels
                size                49 18 pixels
                label string        [1]
                label position      left
                selection border    1
        end field
        new field
                name                y_plan[1]
                type                real
                origin              86 7 pixels
                size                49 18 pixels
                label position      right
                selection border    1
        end field
```

```
new form
        name                gen_body
        origin              126 44 pixels
        size                511 318 pixels
        shade               white
        selection border    1
end form

        new field
                name                number_of_vertices
                type                integer
                origin              5 15 pixels
                size                27 18 pixels
                label string        vertices
                label position      right
                selection border    1
        end field

        new field
                name                number_of_surfaces
                type                integer
                origin              5 184 pixels
                size                28 18 pixels
                label string        surfaces
                label position      right
                selection border    1
        end field

        new field
                name                zone_display
                type                graphics
                origin              209 38 pixels
                size                297 274 pixels
                label position      fit above
                selection border    1
        end field

        new form
                name                vertices
                origin              5 38 pixels
                size                198 134 pixels
                shade               white
                selection border    1
        end form

                new field
                        name                x_vertex[1]
                        type                real
                        origin              34 7 pixels
                        size                49 18 pixels
                        label string        [1]
                        label position      left
                        selection border    1
                end field

                new field
                        name                y_vertex[1]
                        type                real
                        origin              86 7 pixels
                        size                49 18 pixels
                        label position      fit above
                        selection border    1
                end field
```

```
        new field
                name                z_vertex[1]
                type                real
                origin              137 7 pixels
                size                49 18 pixels
                label position      fit above
                selection border    1
        end field
        parent form                 @gen_body
new form
        name                surfaces
        origin              5 208 pixels
        size                197 104 pixels
        shade               white
        selection border    1
end form
new field
        name                surface[1]
        type                integer
        origin              34 7 pixels
        size                153 18 pixels
        label string        [1]
        label position      left
        selection border    1
end field
```

```
new form
        name                construction
        origin              237 6 pixels
        size                642 367 pixels
        shade               white
end form

        new field
                name                b_materials
                type                button
                origin              24 21 pixels
                size                71 17 pixels
                label position      fit above
                selection border    0
                menu                materials
                label colour        black
        end field

        new field
                name                b_openings
                type                button
                origin              114 21 pixels
                size                71 17 pixels
                label position      fit above
                selection border    0
                menu                openings
                label colour        black
        end field

        new field
                name                b_intersections
                type                button
                origin              204 21 pixels
                size                99 17 pixels
                label position      fit above
                selection border    0
                menu                intersections
                label colour        black
        end field

        new field
                name                current_zone
                type                popup
                origin              378 21 pixels
                size                113 17 pixels
                label string        current zone
                label position      fit above
                selection border    0
                border              8
        end field
```

```
new field
        name                surface
        type                popup
        origin              500 21 pixels
        size                107 17 pixels
        label string        surface
        label position      fit above
        selection border    0
        border              8
        menu                bottom\
                            top\
                            all horizontal\
                            all vertical\
                            1\
                            2\
                            3\
                            4
end field
```

```
new form
        name                materials
        origin              10 42 pixels
        size                616 323 pixels
        border              0
        selection border    0
end form

        new field
                name                construction_type
                type                popup
                origin              14 7 pixels
                size                279 294 pixels
                label string        construction
                label position      fit below
                selection border    0
                border              8
                menu                ground_floor\
                                    floor_ceiling\
                                    external_cavity_wall\
                                    internal_stud_partition
        end field
        new field
                name                construction_zone_image
                type                graphics
                origin              370 3 pixels
                size                232 137 pixels
                label position      fit below
                selection border    0
                border              1
        end field
        new field
                name                assign_construction
                type                button
                origin              301 89 pixels
                size                60 17 pixels
                label position      fit above
                menu                assign
                selection border    1
                border              1
        end field
        new field
                name                previous_construction
                type                button
                origin              302 127 pixels
                size                59 17 pixels
                label position      fit above
                menu                previous
                selection border    1
                border              1
        end field
        new form
                name                ground_floor_construction
                origin              301 149 pixels
                size                304 149 pixels
                border              0
                selection border    0
        end form
```

```
new field
        name                layer_1_thickness
        type                real
        origin              4 10 pixels
        size                37 18 pixels
        label string        mm
        label position      right
        selection border    1
        border              1
end field
new field
        name                layer_1_material
        type                popup
        origin              64 10 pixels
        size                233 17 pixels
        label position      right
        selection border    0
        border              8
end field
new field
        name                layer_2_thickness
        type                real
        origin              4 39 pixels
        size                37 18 pixels
        label string        mm
        label position      right
        selection border    1
        border              1
end field
new field
        name                layer_2_material
        type                popup
        origin              64 39 pixels
        size                233 17 pixels
        label position      right
        selection border    0
        border              8
end field
new field
        name                layer_3_thickness
        type                real
        origin              5 68 pixels
        size                36 18 pixels
        label string        mm
        label position      right
        selection border    1
        border              1
end field
new field
        name                layer_3_material
        type                popup
        origin              64 67 pixels
        size                233 17 pixels
        label position      right
        selection border    0
        border              8
end field
```

```
new field
        name            layer_4_thickness
        type            real
        origin          4 97 pixels
        size            37 18 pixels
        label string    mm
        label position  right
        selection border 1
        border          1
end field
new field
        name            layer_4_material
        type            popup
        origin          64 97 pixels
        size            233 18 pixels
        label position  right
        selection border 0
        border          8
end field
new field
        name            layer_5_thickness
        type            real
        origin          4 126 pixels
        size            37 18 pixels
        label string    mm
        label position  right
        selection border 1
        border          1
end field
new field
        name            layer_5_material
        type            popup
        origin          64 126 pixels
        size            233 18 pixels
        label position  right
        selection border 0
        border          8
end field
```

```
new form
        name                openings
        origin              10 42 pixels
        size                616 323 pixels
        border              0
        selection border    0
        data font           /usr/lib/fonts/fixedwidthfonts/serif.b.10
        label font          /usr/lib/fonts/fixedwidthfonts/cour.b.10
        application font    /usr/lib/fonts/fixedwidthfonts/cour.r.10
end form
        new field
                name                surface_opening_image
                type                graphics
                origin              18 3 pixels
                size                279 138 pixels
                label position      right
                selection border    0
                border              1
        end field
        new field
                name                opening_zone_image
                type                graphics
                origin              370 3 pixels
                size                232 137 pixels
                label position      left
                selection border    0
                border              1
        end field
        new field
                name                new_opening
                type                button
                origin              366 167 pixels
                size                78 20 pixels
                label position      fit above
                selection border    0
                border              1
                menu                new opening
        end field
        new field
                name                opening_type
                type                button
                origin              464 167 pixels
                size                78 65 pixels
                label string        opening type
                label position      fit above
                selection border    0
                border              1
                menu                window\
                                    door
        end field
```

```
new field
        name            opening_state
        type            button
        origin          554 167 pixels
        size            44 65 pixels
        label string    state
        label position  fit above
        selection border 0
        border          1
        menu            closed\
                        open
end field
new field
        name            opening_number
        type            popup
        origin          366 215 pixels
        size            78 16 pixels
        label string    number
        label position  fit above
        selection border 0
        border          8
        menu            ~~~
end field
new field
        name            unit_type
        type            popup
        origin          366 260 pixels
        size            176 17 pixels
        label string    unit type
        label position  fit above
        selection border 0
        border          8
        menu            single glazed\
                        double glazed\
                        triple glazed
end field
new field
        name            unit_uvalue
        type            real
        origin          558 257 pixels
        size            44 17 pixels
        label string    U-value
        label position  fit above
        selection border 1
        border          1
end field
```

```
new form
        name                window_opening
        origin              16 148 pixels
        size                284 158 pixels
        border              0
        selection border    0
end form
        new field
                name                window_image
                type                graphics
                origin              50 26 pixels
                size                177 107 pixels
                label position      left
                selection border    0
                border              0
                current value       image:window_opening_image
        end field
        new field
                name                window_width
                type                real
                origin              134 5 pixels
                size                48 17 pixels
                label string        width
                label position      fit below
                selection border    1
                border              1
        end field
        new field
                name                window_cill_height
                type                real
                origin              234 93 pixels
                size                47 17 pixels
                label string        h
                label position      left
                selection border    1
                border              1
        end field
        new field
                name                window_offset
                type                real
                origin              57 137 pixels
                size                83 17 pixels
                label string        X
                label position      above
                selection border    1
                border              1
        end field
```

```
        new field
                name            window_lintel_height
                type            real
                origin          2 65 pixels
                size            47 17 pixels
                label string    H
                label position  right
                selection border 1
                border          1
                default value   2100
        end field
        parent form             @openings
new form
        name                    door_opening
        origin                  16 148 pixels
        size                    284 158 pixels
        start                   hidden
        border                  0
        selection border        0
end form
        new field
                name            door_image
                type            graphics
                origin          52 26 pixels
                size            114 109 pixels
                label position  left
                selection border 0
                border          0
                current value   image:door_opening_image
        end field
        new field
                name            door_width
                type            real
                origin          161 5 pixels
                size            48 17 pixels
                label string    width
                label position  fit below
                selection border 1
                border          1
                default value   900
        end field
        new field
                name            door_offset
                type            real
                origin          57 137 pixels
                size            83 17 pixels
                label string    X
                label position  above
                selection border 1
                border          1
        end field
```

```
new field
        name            door_lintel_height
        type            real
        origin          2 65 pixels
        size            47 17 pixels
        label string    H
        label position  right
        selection border 1
        border          1
        default value   2100
end field
```

```
new form
        name            bld_browse
        origin          31.5 0.2
        size            95 32
        start           hidden
        shade           white
end form

        new form
                name            building_class_sub
                type            form
                origin          1 1.25
                size            20 12
                shade           dark grey
                selection border 0
                label position  fit left
                border          1
        end form

            new field
                        name            browse_class
                        type            menu
                        origin          2.5 1.5
                        size            15 1
                        label string    Building class
                        label position  fit above
                        menu            ??
                        selection border 0
                        border          1
            end field

            parent form                 @bld_browse

        new form
                name            building_sub
                type            form
                origin          1 11
                size            20 13
                shade           dark grey
                selection border 0
                label position  fit left
                border          1
        end form

            new field
                        name            browse_type
                        type            menu
                        origin          2.5 1.5
                        size            15 1
                        menu            ??^
                                        ??
                        label colour    white
                        label position  fit above
                        label string    building type
                        border          1
        end field

        parent form                     @bld_browse
```

```
new field
        name                browse_image
        type                popup
        origin              22 1.5
        size                68 27
        label string        Browse image stack
        label position      fit above
        menu                pic_a
        selection border    0
        border              10
end field

new field
        name                browse_done
        type                button
        origin              82 22
        size                6 1
        label position      fit right
        menu                Done=abort
        selection border    0
        border              1
end field

new field
        name                browse_select
        type                button
        origin              70 22
        size                7 1
        label position      fit right
        menu                select
        selection border    0
        border              1
end field

new field
        name                browse_info
        type                button
        origin              44 22
        size                7 1
        label position      fit right
        menu                info
        selection border0
        border              1
end field
```

```
new form
        name                analysis
        origin              1 12
        size                125 30
        start               hidden
        shade               light grey
        border              1
        selection border    2
end form

        new field
                name                anal_label
                type                label
                origin              25 1
                size                0 0
                label position      fit left
                label string        Analyses
                help                push a button (right) to change\n\
                                    topic of discussion.
                description         These buttons switch the focus of\n\
                                    discussion to the requested topic.  The\n\
                                    relevent forms will be displayed below\n\
                                    (existing ones will disappear).  It is\n\
                                    suggested that they are worked through\n\
                                    from left to right to minimize\n\
                                    specification of redundant information.
                selection border    0
                border              0
                data font           /usr/lib/fonts/fixedwidthfonts/cour.b.16
                label font          /usr/lib/fonts/fixedwidthfonts/cour.b.16
                application font    /usr/lib/fonts/fixedwidthfonts/cour.b.16
        end field
        new field
                name                b_mono_functional_methodology
                type                button
                style               key
                origin              0.7  1.2
                size                16 2
                menu                Mono-Functional\n  Methodology
                label position      fit above
                label colour        black
                start               hidden
                help                push when ready to discuss analysis requirements
                description         This button switches the focus of\n\
                                    discussion to a sort of analysis\n\
                                    required.  Information about the\n\
                                    design stage is solicited so that\n\
                                    only appropriate analysis (and data\n\
                                    input) will be carried out.
                data font           /usr/lib/fonts/fixedwidthfonts/cour.b.10
                label font          /usr/lib/fonts/fixedwidthfonts/cour.b.10
                application font    /usr/lib/fonts/fixedwidthfonts/cour.b.10
        end field
```

```
new field
        name            b_multi_functional_methodology
        type            button
        style           key
        origin          19 1.2
        size            16 2
        label position  fit above
        label colour    black
        menu            Multi-Functional\n  Methodology
        start           hidden
        help            push when ready for building description form
        description     This button switches the focus of\n\
                        discussion to a description of the\n\
                        (proposed?) building. Subsidary\n\
                        forms will enable the geometry\n\
                        and materials to be specified.
        data font       /usr/lib/fonts/fixedwidthfonts/cour.b.10
        label font      /usr/lib/fonts/fixedwidthfonts/cour.b.10
        application font /usr/lib/fonts/fixedwidthfonts/cour.b.10
end field

new form
        name            mono_functional_methodology
        origin          1 3
        size            35 25
        start           hidden
        shade           white
        label position  fit above
        label colour    black
        selection border 0
        border          1
 end form

new field
        name            comfort_analysis
        type            button
        origin          1 1
        size            9 2
        menu            Comfort
end field

new field
        name            energy_analysis
        type            button
        origin          12 1
        size            9 2
        menu            Energy
 end field

 new field
        name            control_analysis
        type            button
        origin          23 1
        size            9 2
        menu            Control
end field
```

```
new field
        name            view
        type            button
        origin          23 4
        size            9 2
        menu            View
end field
new field
        name            condensation_analysis
        type            button
        origin          1 4
        size            20 2
        menu            Condensation
    end field
    parent form         @analysis
new form
        name            multi_functional_methodology
        origin          1 3
        size            35 25
        start           hidden
        shade           white
        label position  fit above
        label colour    black
        selection border 0
        border          1
 end form
        new field
            name        passive_expert
            type        button
            origin      1 1
            size        9 2
            menu        Passive\n Expert
end field
new field
            name        airflow
            type        button
            origin      12 1
            size        9 2
            menu        Airflow
 end field
```

```
        new field
                name            control_systems
                type            button
                origin          23 1
                size            9 2
                menu            ControlNnSystems
        end field
        parent form             @analysis
new field
        name                    analysis_results
        type                    popup
        origin                  37 1.2
        size                    86 28
        label string            Results
        label position          fit above
        selection border        0
        border                  12
end field
```

APPENDIX C

Icons

C.1. Concepts represented graphically on the master form

novice[1]

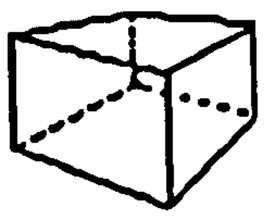
expert

architect_designer

engineer[2]

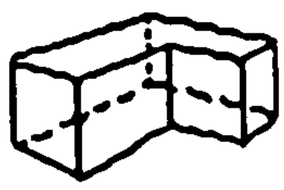
modeller[2]

C.2. Concepts represented graphically on the geometry forms



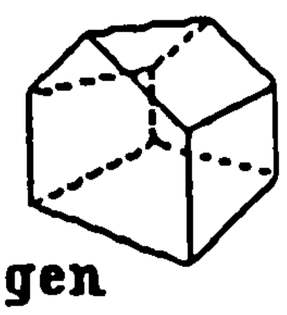
rec

rec_body



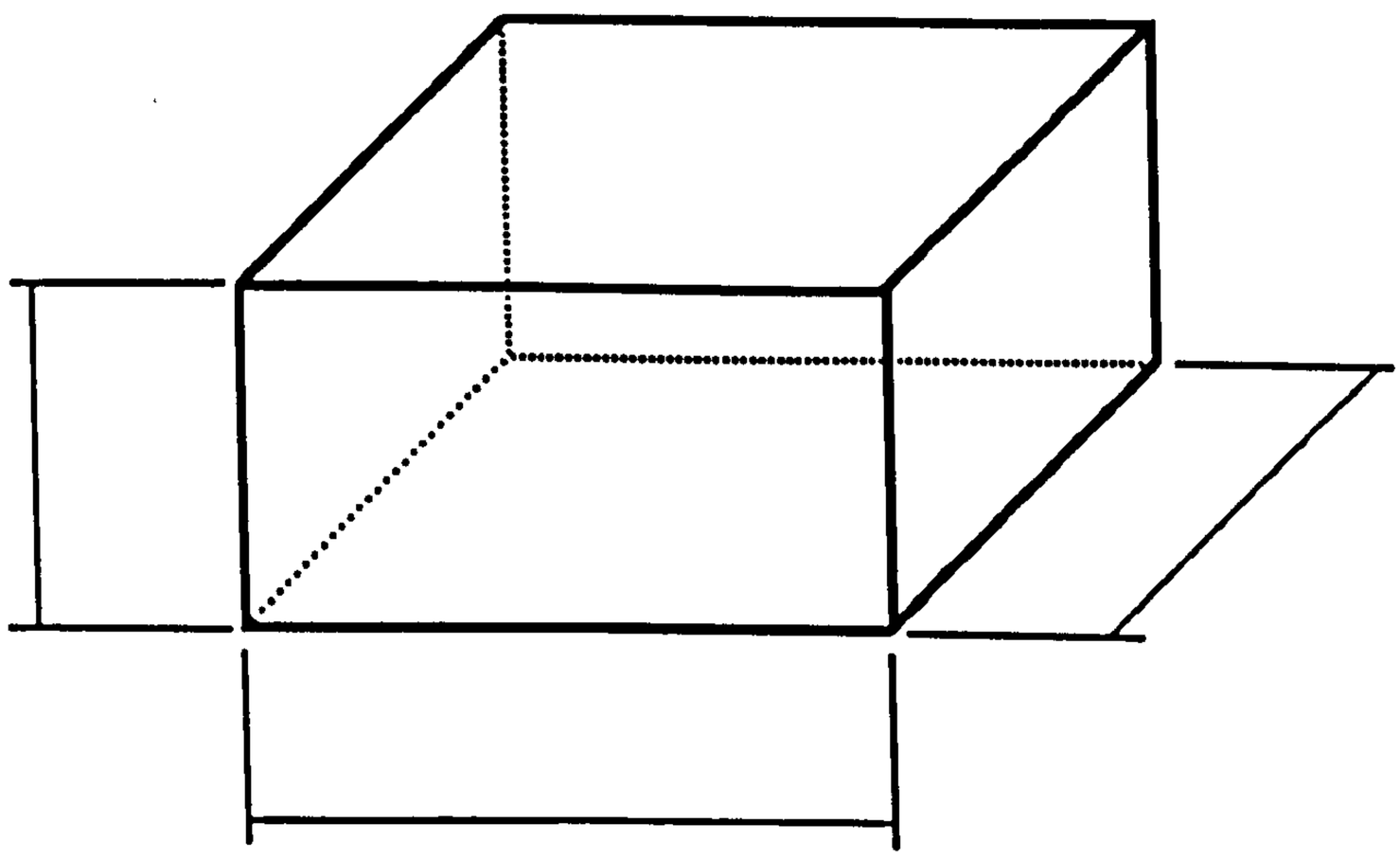
reg

reg_body



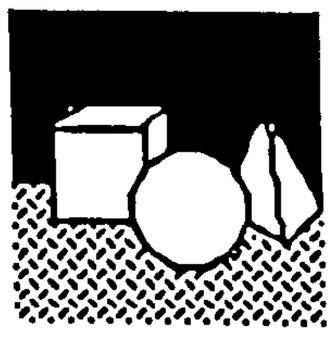
gen

gen_body

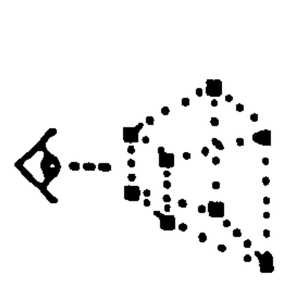


rec_body_attributes

autocad[3]



vim



iris_modeller[1]

---

1 Sun MicroSystems
2 MacRandal
3 AutoDesk

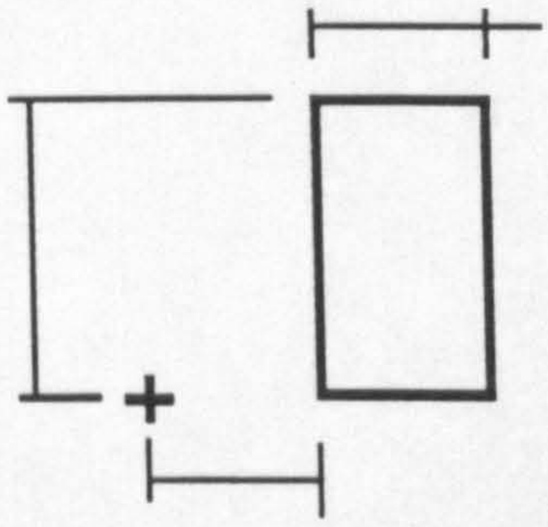
C.3. Concepts represented graphically on the construction openings form

door

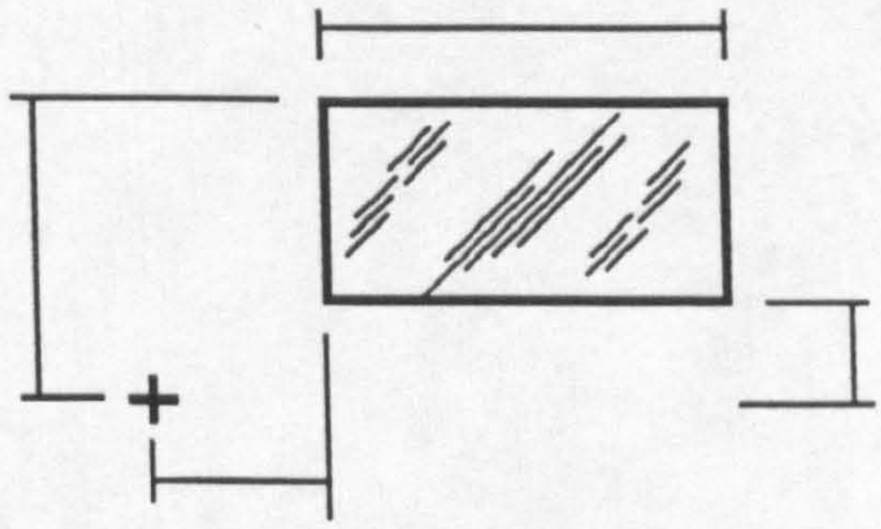
window

open

closed



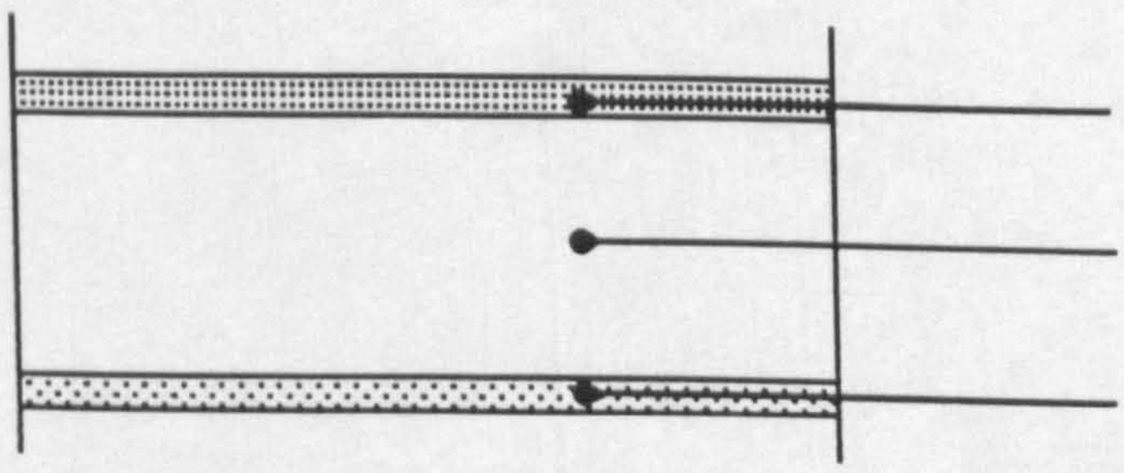
door_attributes



window_attributes

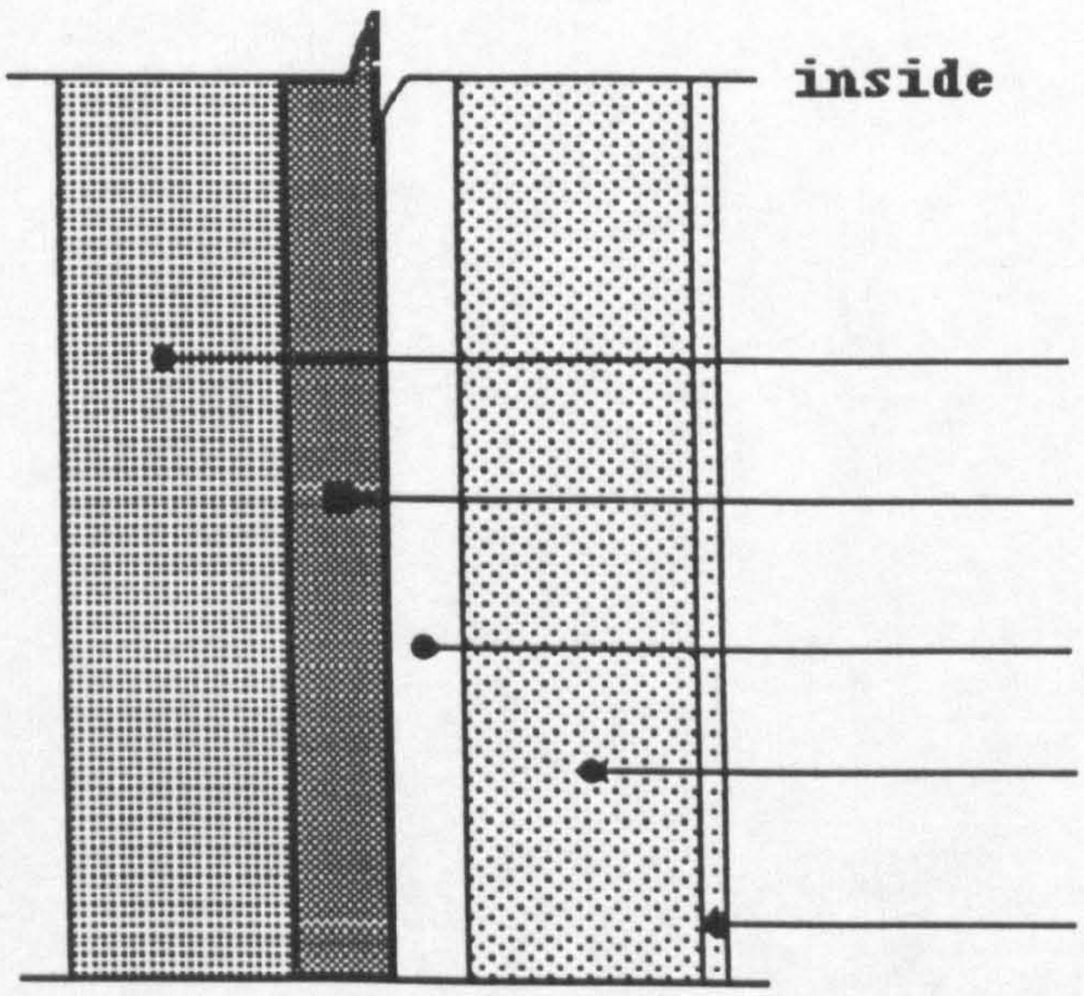
C.4. Concepts represented graphically on the construction materials form

Floor/ceiling



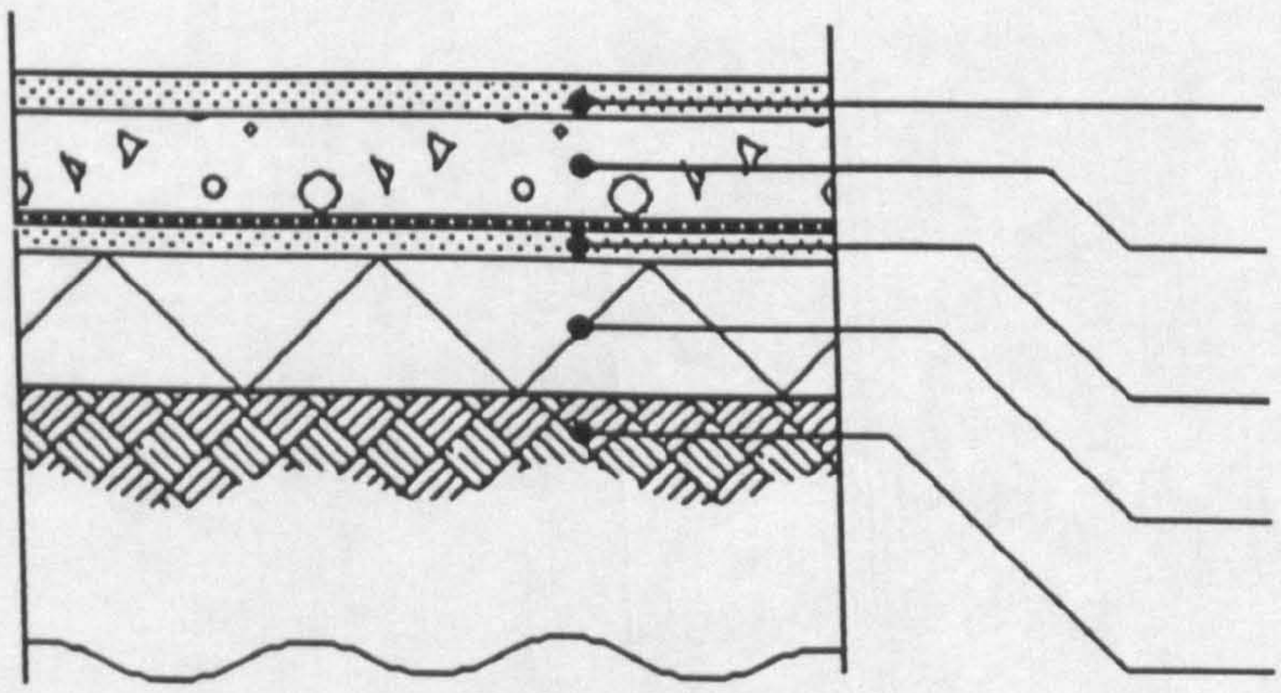
floor_ceiling_detail

External wall



external_cavity_wall_detail

Ground floor



ground_floor_detail

APPENDIX D

Knowledge Bases

The knowledge bases (complimentary to the proforma templates Appendix C) listed in this section have been generated by original authors of the IFe (mainly Damian Mac Randal, RAL) and are included for competeness.

```
startup:-                                                   /* read and execute clauses*/
        prompt(_,''),
        '$char_type'('$',__,2),
        to_bb(mk_area, u_cpt),                              /* ignored, startup problem*/
        to_bb(mk_area, u_mdl),
        to_bb(mk_area, user_dialog),
        to_bb(update_me, user_dialog, user_dialog),
        to_bb(update_me, application, application),
        to_bb(update_me, u_mdl, 'user_model          u_mdl_thinks'),
        to_bb(update_me, u_cpt, 'user_cpt   u_cpt_got'),
        get_date(_Date),
        kset(date, _Date),
        tell_usr(date, _Date),
        chat_usr([
                '   prototype INTELLIGENT FRONT END',
                ''
                 ]),
        cmd_loop.

cmd_loop:-
        repeat,                                             /* infinite loop */
                get_cmd,
                fail.

get_cmd:-
        write('>'), ttyflush,
        getterm(_Area, _Rest),                              /* ignore BB area */
        ( _Area == abort -> halt ; true ),
        ( _Area == quitrqst -> quitrqst ; true ),
        ( _Rest = [user_said|_Term1] ->
                _Term = _Term1
        ;
                ( _Rest = [application_said|_Term2] ->
                        _Term = _Term2
                ;
                        _Term = _Rest
                )
        ),
        ( _Rest == [_Cpt|help] -> _Term = true ; true ),
        ( _Rest == [_Cpt|description] -> _Term = true ; true ),
        !, _Pred=..._Term,                                  /* uchk */
        !, _Pred,
        !.

getterm(_Area, _Term):-
        getname(_Name1,_Chr1),
        name(_Area,_Name1),
        ( _Chr1 \== 10 ->
                getname(_Name2,_Chr2),
                name(_Pred_name,_Name2),
                getargs(_Chr2,_Args),
                _Term=[_Pred_name|_Args]
        ;
                _Term = []
        ).
```

```
getargs(_Chr,_Args):-
        ( _Chr\==10 ->
                getname(_Name,_Chr_3),
                convert(_Name,_Atom),
                getargs(_Chr_3, _Args_1),
                _Args=[_Atom|_Args_1]
        ;
                _Args=[]
        ).

getname(_Name,_Chr):-
        _Chr \== 10,
        get0(_Chr_1),
        ( _Chr_1\==9, _Chr_1\==10 ->
                getname(_Name_1,_Chr_2),
                _Chr=_Chr_2,
                ( _Chr_1==32 ->                         /* replace ' ' with '_' */
                        _Name=[95|_Name_1]
                ;
                        _Name=[_Chr_1|_Name_1]
                )
        ;
                _Name=[],
                _Chr=_Chr_1
        ).

getname(_Name, _Chr):-
        _Name=[].


convert([43|_String], _Var):-                   % handle '+' signed numbers
        convert(_String, _Var).
convert([45|_String], _Var):-                   % handle '-' signed numbers
        convert(_String, _Var2),
        _Var is - _Var2.

convert(_String, _Var):-
        name(_Atm, _String),
        ( number(_Atm) ->
                _Var = _Atm
        ; flt(_Atm, _Flt) ->
                _Var = _Flt
        ;
                _Var = _Atm
        ).
```

```
/* Dont listen to what the user says about his name / type / level -
 *        rely on the user model's opinion
 */
u_mdl_thinks(user_name, nokey, _User_name, _Who_set) :-
        ( abolish(user_name,1) ; true ),
        assert(user_name(_User_name)).

u_mdl_thinks(user_type, nokey, _User_type, _Who_set) :- %% new user_type
        ( called(_Cpt_set),                          %% redo existing cpt sets
                _Pred=..[_Cpt_set, refresh],
                _Pred,
                fail
        ;
                true                                 %% replaced
        ).

u_mdl_thinks(user_level, nokey, _User_level, _Who_set) :-
        ( intro(_User_level) ; abolish(user_level,1) ),
        assert(user_level(_User_level)). /*for speed in feedback selection*/

intro(expert) :-
        user_name(_User_name),
        chat_usr([
                ['Hi ',_User_name,'.'],
                'What is the problem this time?',
                '']),
        retract(intro(expert)).              % leave novice mesage in case needed
intro(novice) :-
        user_name(_User_name),
        chat_usr([
                ['Hi ',_User_name,'.'],
                'Just fill in the forms, changing defaults if you',
                'want, as I will ask on the forms for any data I',
                'need.  I will try not to ask too many questions!',
                'Anytime I want to chat to you, I will place the',
                'message in this box.  Urgent messages will also',
                'be placed in a popup box near the field causing',
                'the problem. Click in the box to make it go away.',
                '']),
        abolish(intro,1).

project(help).      %%%%% stupid form package
project(description).       %%%%% stupid form package
project( _Project) :-                        /* existing project */
        proj_exists(_Project, _Session, _Started, _Data, _LogFile),
        kset(session, _Session),
        kset(started, _Started),
        ask_usr(session, _Session),
        ask_usr(started, _Started),
[_Data],                     %%%%% retrieve existing master cpts!!!!!
( called(_Cpt_set),                  %% set up previously done cpt sets
        _Pred=..[_Cpt_set, refresh],
        _Pred,
        fail
;
        true                                 %% replaced
),
to_bb(load_log, _Log),
uset(project, _Project),
start_session.
```

```
project(_Project) :-                    /* new project */
        kset(session, 1),
        get_date(_Date),
        kset(started, _Date),
        ask_usr(session, 1),
        ask_usr(started, _Date),
        uset(project, _Project),
        start_session.
        tell_usr(user_type, architect).

start_session :-
        ask_usr(b_analysis),
        ask_usr(b_bld_spec),
        assert(user_level(novice)),          %%%%% demo fudge
        feedback(focus_enabled).

feedback(focus_enabled, novice) :-
        chat_usr([
                'These buttons switch the focus of discussion to',
                'the requested topic.  The relevent forms will be',
                'displayed below (existing ones will disappear).',
                'It is suggested that the analysis forms are filled',
                'in firstly in order to minimize specification of',
                'redundant information during building description.',
                 "]).

b_analysis(on) :-
        focus_concept(master, analysis),
        analysis(initialize).

b_bld_spec(on) :-
focus_concept(master, bld_spec),
bld_spec(initialize).
```

```
bld_spec(initialize) :- %% now addressing building specification cpts
        getenv('IFE_LOC',_Loc),          %% init location menu, depending on
                                         %% users location
        ( _Loc = uk,
                offer_usr(location, 'use_map
 glasgow edinburgh aberdeen belfast london manchester birmingham newcastle cardiff' )
        ;
          _Loc = eur,
                offer_usr(location, 'london paris bonn rome madrid' )
        ),
        feedback(bld_spec_sel).

 feedback(building_sel, novice) :-
        chat_usr([
                'This button switches the focus of discussion to',
                'the description of the (proposed?) building.',
                'Subsidary forms will enable the site, geometry,',
                'materials, use patterns, etc.  to be specified.',
                "]).


bld_spec(refresh) :-                %% tell user all known about this meta-cpt
        ( known(location, _Location) ->
                ask_usr(location, _Location)
        ;
                refresh(latitude),
                refresh(longitude)
        ),
        refresh(environment),
        refresh(function).

 location(use_map) :-                /* user selected map */
        new_dialogue(map,'map -e -o -s'),
        ask_usr(latitude),
        ask_usr(longitude).

location(_Location) :-              /* current location */
        known(location, _Location),
        uset(location, _Location).
```

```
location(_Location) :-                  /* known location */
        uset(location, _Location),
        position_of(_Location, _Latitude, _Longitude),
                /* known, check compatible with user set position, if any */
        (\+check_position(_Location, _Latitude, _Longitude) ->
                kset(latitude, _Latitude),    /*no user_supplied position*/
                kset(longitude, _Longitude),
                kset(timezone, 'GMT'),
                ( _Latitude < 0,
                        _Lat2 = - _Latitude,
                        name(_Lat2,_Lat2_str),
                        append(_Lat2_str, " S", _Lat2_str2)
                ;
                        name(_Latitude, _Lat2_str),
                        append(_Lat2_str, " N", _Lat2_str2)
                ),
                ask_usr(latitude, _Lat2_str2),
                ( _Longitude < 0,
                        _Lng2 = - _Longitude,
                        name(_Lng2,_Lng2_str),
                        append(_Lng2_str, " W", _Lng2_str2)
                ;
                        name(_Longitude, _Lng2_str),
                        append(_Lng2_str, " E", _Lng2_str2)
                ),
                ask_usr(longitude, _Lng2_str2)
        ;
                true
        ).

check_position(_Location, _New_latitude, _New_longitude) :-
        knownw(latitude, _Latitude, user_set),
        knownw(longitude, _Longitude, user_set),
        ( near(_Latitude, _Longitude, _New_latitude, _New_longitude, 1.5) ->
                true
        ;       feedback(loc_pos_clash),
                suggest_usr(location, _Location),
                suggest_usr(latitude, _New_latitude),
                suggest_usr(longitude, _New_longitude)
        ).

location(_Location) :-                  /* unknown location */
        knownw(latitude, _Latitude, user_set),  /* but pos already set */
        knownw(longitude, _Longitude, user_set).

location(_Location) :-                  /* unknown location */
        feedback(location_unknown),                     /* no position set yet */
        ask_usr(latitude),
        ask_usr(longitude).

feedback(loc_pos_clash, novice) :-
        chat_usr([
                'To the best of my knowledge, the given location is not',
                'at the latitude/longitude you specified earlier.',
                'If you want the locations real position, use the default',
                'values in the latitude/longitude boxes.',
                '']).
```

```
feedback(loc_pos_clash, expert) :-
        chat_usr([
                'Check latitude/longitude.',
                "]).

feedback(location_unknown, novice) :-
        chat_usr([
                'Sorry, I don"t know where that is.',
                'You"ll need to give me its position (ie. its latitude/',
                'longitude).  You should also verify the type of site set',
                'below (default: city centre), and the suggested climate',
                'file (closest suitable collection).',
                "]).

feedback(location_unknown, expert) :-
        chat_usr([
                'Where is that?',
                "]).


latitude(_Latitude) :-                  /* current latitude */
        known(latitude, _Latitude).

latitude(_Latitude) :-
        uset(latitude, _Latitude),
        tell_usr(latitude, _Latitude),
        ( knownw(longitude, _Longitude, user_set) ->
                position(_Latitude, _Longitude)
        ;
                true
        ).

longitude(_Longitude) :-                /* current longitude */
        known(longitude, _Longitude).

longitude(_Longitude) :-
        uset(longitude, _Longitude),
        tell_usr(longitude, _Longitude),
        ( knownw(latitude, _Latitude, user_set) ->
                position(_Latitude, _Longitude)
        ;
                true
        ).

position(_Latitude, _Longitude) :-  /* known position */
        location_near(_Latitude, _Longitude, _Location),
        (\+check_location( _Location, _Latitude, _Longitude) ->
                tell_usr(location, _Location),
                kset(location, _Location)
        ;
                true
        ).
```

```
check_location( _Location, _Latitude, _Longitude) :-
        knownw(location, _Loc, user_set),
        ( position_of(_Loc, _Loc_Lat, _Loc_Long) ->
                ( near(_Latitude, _Longitude, _Loc_Lat, _Loc_Long, 1.5) ->
                   true
                ;
                   feedback(loc_pos_clash),
                   suggest_usr(location, _Location),
                   suggest_usr(latitude, _Loc_Lat),
                   suggest_usr(longitude, _Loc_Long)
                )
        ;
                true
        ).

position(_Latitude, _Longitude) :- /* unknown position */
        known(location, _Location).                    /* but loc already set */

position(_Latitude, _Longitude) :- /* unknown position */
        tell_usr(location, '???'),
        feedback(position_unknown).                    /* no loc set yet */

feedback(position_known, novice) :-
        known(location, _Location),
        chat_usr([
                'From the position indicated, I am assuming you mean',
                 _Location, ' but I will use your latitude & longitude.',
                'If the assumed location is not acceptable, (and you want',
                'to refer to this exact position again), give a name for',
                'this location by selecting USER in the Location menu.',
                'In that case, you might also want to set the environment',
                'for the new location.',
                "]),
        chat_usr([
                'NOTE. The locations actual coordinates can be obtained',
                'as the default in the Latitude/Longitude boxes',
                "]).

feedback(position_known, expert) :-
        knownw(location, _Location, kb_set),
        chat_usr([
                ['I am assuming you mean location',_Location],
                "]).

feedback(position_error, novice) :-
        knownw(location, _Location, kb_set),
        position_of(_Location, _Loc_Lat, _Loc_Long),
        known(latitude, _Lat),
        known(longitude, _Long),
chat_usr([
        'ERROR!      ',
        [_Location, ' is at ', _Loc_Lat, 'N ', _Loc_Long, 'W.'],
        ['whereas the position you gave was ',_Lat,'N ',_Long,'W.'],
        'You should change the location field to something else,',
        '(invent a name if you want) or modify the position.  I',
        'have set the real position and location as defaults in',
        'the latitude, longitude and location fields',
        "]).
```

```
feedback(position_error, expert) :-
        chat_usr([
                'ERROR!      ',
                'thats not there!',
                "]).

feedback(position_unknown, novice) :-
        chat_usr([
                'If you want to refer to this exact position again, give',
                'a name for it by selecting USER in the Location menu.',
                'You will also need to set an environment for the location.',
                "]).

feedback(position_unknown, expert) :-
        chat_usr([
                'I could do with a name for this location',
                "]).


environment(_Site_type) :-          /* current site type */
        known(environment, _Site_type).

environment(_Site_type) :-          /* known site type */
        uset(environment, _Site_type),
        exposure(_Site_type, _Exposure),
        grnd_rflct(_Site_type, _Grnd_rflct),
        kset(exposure, _Exposure),
        kset(grnd_rflct, _Grnd_rflct),
        feedback(environment_known).

environment(_Site_type) :-          /* unknown site type */
        feedback(environment_unknown),
        ask_usr(exposure, 1),
        ask_usr(grnd_rflct, 2.5),
        uset(environment, _Site_type).

feedback(environment_unknown, novice) :-
        chat_usr([
                'Sorry, I don"t know what that means!  Unless you know',
                'what you are doing, I suggest youselect a standard site',
                'type from the menu.  Otherwise, you MUST give me an',
                'indication of site exposure (int:1-7) and a figure for',
                'the ground reflectivity (real:0-10).  Put these in the',
                'appropriate boxes on the form, (use the defaults given,',
                'if necessary',
                "]).

feedback(environment_unknown, expert) :-
        chat_usr([
                'What does that mean? ',
                "]).
```

```
exposure(_Exposure) :-                  /* current exposure */
        known(exposure, _Exposure).

exposure(_Exposure) :-                  /* new exposure */
        check_exposure(_Exposure),
        uset(exposure, _Exposure).

exposure(_Exposure) :-                  /* new exposure was bad */
        feedback(bad_exposure).

check_exposure(_Exposure) :-
        _Exposure > -1,
        _Exposure < 8.

feedback(bad_exposure, novice) :-
        chat_usr([
                'Sorry, that number does not mean anything to me',
                'Please fix or default it.  (I warned you)',
                "]).

feedback(bad_exposure, expert) :-
        chat_usr([
                'Error 0xf12D9 : Illegal op 841 in stream 0, /dev/tty',
            'or in other words, I think that value will cause problems.',
                'You should reconsider it.',
                "]).

grnd_rflct(_Grnd_rflct) :-   /* current ground reflect */
        known(grnd_rflct, _Grnd_rflct).

grnd_rflct(_Grnd_rflct) :-   /* new ground reflectivity */
        check_grnd_rflct(_Grnd_rflct),
        uset(grnd_rflct, _Grnd_rflct).

grnd_rflct(_Grnd_rflct) :-   /* new ground reflectivity bad*/
        feedback(bad_grnd_rflct).

check_grnd_rflct(_Grnd_rflct) :-
        _Grnd_rflct > grnd_rflct_max,
        _Grnd_rflct < rnd_rflct_min.

feedback(bad_grnd_rflct, novice) :-
        chat_usr([
                'Sorry, that number does not mean anything to me',
                'Please fix or default it.  (I warned you)',
                "]).

feedback(bad_grnd_rflct, expert) :-
        chat_usr([
                'I think that value will cause problems.',
                'You should reconsider it.',
                "]).

function(_Function) :-                  /* current function */
        known(function, _Function).
```

```
function(_Function) :-                  /* new function */
        check_function(_Function),
        uset(function, _Function).

function(_Function) :-                  /* new function was bad */
        feedback(bad_function).

check_function(_Function) :-
        function(_Function, _).

feedback(bad_function, novice) :-
        chat_usr([
                'Sorry, that category does not mean anything to me',
                'I would suggest selecting a category that is on the menu',
                'so that, rathter than asking you, I can select sensible',
                'values for data required in specifying the analysis',
                'methodology.',
                "]).

feedback(bad_function, expert) :-
        chat_usr([
                'Dont know that category',
                "]).

b_geometry(on) :-
        focus_concept(bld_spec, geometry),
        geometry(initialize).

b_construction(on) :-
        focus_concept(bld_spec, construction),
        construction(initialize).

b_useage(on) :-
        focus_concept(bld_spec, useage),
        useage(initialize).

b_connectivity(on) :-
        focus_concept(bld_spec, connectivity),
        connectivity(initialize).

b_site(on) :-
        focus_concept(bld_spec, site),
        site(initialize).

b_shading(on) :-
        focus_concept(bld_spec, shading),
        shading(initialize).

b_airflow(on) :-
        focus_concept(bld_spec, airflow),
        airflow(initializebld_spec).

b_plant(on) :-
        focus_concept(bld_spec, plant),
        plant(initialize).
```

```
b_control(on) :-
        focus_concept(bld_spec, control),
        control(initialize).

b_bld_browse(on) :-                    /* ok to browse */
        focus_concept(bld_spec, bld_browse),
        bld_browse(initialize).
```

```
geometry(initialize) :- %% now addressing geometry specification cpts
        feedback(geometry_sel).

feedback(geometry_sel, novice) :-
        chat_usr([
                'This button switches the focus of discussion to',
                'the geometric and material properties of this',
                'building.  Firstly, the geometry must be given.',
                'Several alternative input mechanisms are provided',
                '']).


geometry(refresh) :-                    %% tell user all known about this meta-cpt
                                        %% get set of existing zone no for menu
        setof(_Zone_no, zone_made(_Zone_no), _Zone_nos),
        offer_usr('zone_form', _Zone_nos).          %%%%%

zone_setup(_New_zn_no):-                            /*common setup for new zone*/
        ( nxt_zone(_Zn_no) ->
                retract(nxt_zone(_))                /* first zone? */
        ;
                _Zn_no is 0
        ),
        _New_zn_no is _Zn_no + 1,
        assert(nxt_zone(_New_zn_no)),
        assert(zone_made(_New_zn_no)),
        ( no_zones(_Num_zn) ->                      /* increment numb of zones */
                retract(no_zones(_))
        ;
                _Num_zn is 0
        ),
        _New_num_zn is _Num_zn + 1,
        assert(no_zones(_New_num_zn)),
        tell_usr('no_zones', _New_num_zn),
        ( g_focus(__) ->
                retract(g_focus(_New_zone_no))
        ;
                true
        ),
        assert(g_focus(_New_zone_no)).      /* currently displayed zone */

b_g_form_fill(on):-
        focus_concept(geometry, g_form_fill),
        g_form_fill(initialize).

b_g_draw(on) :-   /* user wants drafting package */
        focus_concept(geometry, g_draw),
        g_draw(initialize).

b_g_cad_file(on) :-                         /*user has geometry in file*/
        focus_concept(geometry, g_cad_file),
        g_cad_file(initialize).
```

```
g_draw(initialize).
        feedback(draw_geom).

feedback(draw_geom):-
        chat_user([
                'Select the required modelling package',
                'from the panel.'
                ]).

drawing_package(_Package_name):-        /* user has selected a package */
        new_dialogue(draw, _Package_name),
        defocus_concept(geometry).
```

```
g_cad_file(initialize).
        tell_usr(geom_format, viewer),
        feedback(file_geom).

feedback(file_geom, novice):-
        chat_user([
                'This form allows you to import the geometrical',
                'description of a building from an external source',
                '(viewer, autocad, acropolis).'
                ]).

geom_file(_Zn_file):-
        to_bb(application, start,perspective,_Zn_file).

perspective_complete(_Pic_name):-
        tell_usr(geom_file_pic, _Pic_name).
```

```
g_form_fill(initialize):-
        zone_setup(_Zone_no),
        feedback(form_fill_geom).

zone_geom(refresh) :-                   %% tell user all known about current zones
        zone_made(_Zone_no),
        zone_geom(refresh, _Zone_no),
        fail.                           %% iterate over all made zones
zone_geom(refresh).                     %% never fail

zone_geom(refresh, _Zone_no) :-         %% tell user all known about this zone
        'zone_form$'(_Zone_no, on),         %% re-load form
        refresh(zone_name, [_Zone_no]),
        refresh(zone_desc, [_Zone_no]),
        refresh(zone_orientation, [_Zone_no]),
        ( known(zone_origin, [_Zone_no], [_X, _Y, _Z]) ->
                ask_usr(zone_origin_x, _X),
                ask_usr(zone_origin_y, _Y),
                ask_usr(zone_origin_z, _Z)
        ;
                true
        ),
        zone_type(_Zone_no, _Zone_type, _No_Points),
        ( _Zone_type == rec ->
                ask_usr(zone_type, rec),
                refresh(zone_length, [_Zone_no]),
                refresh(zone_width, [_Zone_no]),
                refresh(zone_height, [_Zone_no])
        ; _Zone_type == reg ->
                ask_usr(zone_type, reg),
                refresh(zone_height, [_Zone_no]),
                refresh(vertex_list_2d, [_Zone_no])
                ; _Zone_type == gen ->
                        ask_usr(zone_type, gen),
                        refresh(vertex_list_3d, [_Zone_no]),
                        refresh(surface_list, [_Zone_no])
                ).

refresh(vertex_list_2d, _Zone_no).
refresh(vertex_list_3d, _Zone_no).
refresh(surface_list, _Zone_no).

zone_name(_Zone_name) :-
        g_focus(_Cur_zone_no),
        uset(zone_name, [_Cur_zone_no], _Zone_name).

zone_desc(_Zone_desc) :-
        g_focus(_Cur_zone_no),
        uset(zone_desc, [_Cur_zone_no], _Zone_desc).


zone_orientation(_Orientation):-
        g_focus(_Cur_zone_no),
        uset(zone_orientation, [_Cur_zone_no], _Orientation).
```

```
zone_origin_x(_Xnew):-
        g_focus(_Cur_zone_no),
        ( known(zone_origin,_Cur_zone_no, _X, _Y, _Z)
        ; _X = 0, _Y = 0, _Z = 0
        ),
        uset(zone_origin, [_Cur_zone_no], [_Xnew, _Y, _Z]).

zone_origin_y(_Ynew):-
        g_focus(_Cur_zone_no),
        ( known(zone_origin,_Cur_zone_no, _X, _Y, _Z)
        ; _X = 0, _Y = 0, _Z = 0
        ),
        uset(zone_origin, [_Cur_zone_no], [_X, _Ynew, _Z]).

zone_origin_z(_Znew):-
        g_focus(_Cur_zone_no),
        ( known(zone_origin,_Cur_zone_no, _X, _Y, _Z)
        ; _X = 0, _Y = 0, _Z = 0
        ),
        uset(zone_origin, [_Cur_zone_no], [_X, _Y, _Znew]).


zone_type_Set(rec):-                    /* zone is a rectangle */
        g_focus(_Cur_zone_no),
%%%%%   unask_usr(zone_type),   %%%%% cant change type !!!!!
        ( zone_type(_Cur_zone_no, _Type, _No_points) ->
                unfocus_usr(_Type),
                retract(zone_type(_Cur_zone_no, _Type, _No_points))
        ;
                true
        ),
        assert(zone_type(_Cur_zone_no, rec, 7)),
%%%%% clear form or destroy & recreate
        focus_usr(rec),                 %% display rec subform
        uset(zone_type, [_Cur_zone_no], rec),
        update_vertex(_Cur_zone_no, 0, 0, 0, 0),
        update_vertex(_Cur_zone_no, 1, _X, 0, 0),
        update_vertex(_Cur_zone_no, 2, _X, _Y, 0),
        update_vertex(_Cur_zone_no, 3, 0, _Y, 0),
        update_vertex(_Cur_zone_no, 4, 0, 0, _Z),
        update_vertex(_Cur_zone_no, 5, _X, 0, _Z),
        update_vertex(_Cur_zone_no, 6, _X, _Y, _Z),
        update_vertex(_Cur_zone_no, 7, 0, _Y, _Z),
        update_surface(_Cur_zone_no, 0, [3, 2, 1, 0]),
        update_surface(_Cur_zone_no, 1, [4, 5, 6, 7 ]),
        update_surface(_Cur_zone_no, 2, [0, 1, 5, 4]),
        update_surface(_Cur_zone_no, 3, [1, 2, 6, 5]),
        update_surface(_Cur_zone_no, 4, [2, 3, 7, 6]),
        update_surface(_Cur_zone_no, 5, [3, 0, 4, 7]),
        ask_usr(zone_size_label),
        ask_usr(zone_length),
        ask_usr(zone_width),
        ask_usr(zone_height).
```

```
zone_type_Set(reg):-                          /* constant height zone */
        g_focus(_Cur_zone_no),
%%%%%           unask_usr(zone_type),  %%%%% cant change type !!!!!
        ( zone_type(_Cur_zone_no, _Type, _No_points) ->
                unfocus_usr(_Type),
                abolish(zone_type,3)
        ;
                true
        ),
        assert(zone_type(_Cur_zone_no, reg, 5)),
        focus_usr(reg),                       %% display reg subform
        uset(zone_type, [_Cur_zone_no], reg),
        update_vertex(_Cur_zone_no, 0, 0, 0, 0),
        update_vertex(_Cur_zone_no, 2, _X, _Y, 0),
        update_vertex(_Cur_zone_no, 4, _X, _Y, 0),
        update_vertex(_Cur_zone_no, 1, 0, 0, _Z),
        update_vertex(_Cur_zone_no, 3, _X, _Y, _Z),
        update_vertex(_Cur_zone_no, 5, _X, _Y, _Z),
        update_surface(_Cur_zone_no, 0, [4, 2, 0]),          % floor
        update_surface(_Cur_zone_no, 1, [1, 3, 5]),          % ceiling
        ask_usr(zone_size_label),
        ask_usr(zone_height),
        focus_usr('vertex_list_2d').                  %% display vertex subform

zone_type_Set(gen):-                          /* arbitarily shaped zone */
        g_focus(_Cur_zone_no),
%%%%%           unask_usr(zone_type),  %%%%% cant change type !!!!!
        ( zone_type(_Cur_zone_no, _Type, _No_points) ->
                unfocus_usr(_Type),
                abolish(zone_type,3)
        ;
                true
        ),
        assert(zone_type(_Cur_zone_no, gen, 3)),
        focus_usr(gen),                       %% display gen subform
        uset(zone_type, [_Cur_zone_no], gen),
        update_vertex(_Cur_zone_no, 0, 0, 0, 0),
        update_vertex(_Cur_zone_no, 1, _X, _Y, _Z),
        update_vertex(_Cur_zone_no, 2, _X, _Y, _Z),
        update_vertex(_Cur_zone_no, 3, _X, _Y, _Z),
        focus_usr('vertex_list_3d'),                  %% display vertex subform
        defocus_usr('vertex_list_3d'),
        focus_usr('surface_list').            %% display surface subform

zone_type_Set(_):-                            /* illegal zone type */
        feedback(bad_zone_type).

zone_length(_Length):-
        g_focus(_Cur_zone_no), zone_type(_Cur_zone_no, rec, 7),
        ( vertex(_Cur_zone_no, 0, _X1, _Y1, _Z1), nonvar(_X1) ; _X1 is 0 ),
        _Ln is _X1 + _Length,
        update_vertex(_Cur_zone_no, 1, _Ln, _Y, _Z),
        update_vertex(_Cur_zone_no, 2, _Ln, _Y, _Z),
        update_vertex(_Cur_zone_no, 5, _Ln, _Y, _Z),
        update_vertex(_Cur_zone_no, 6, _Ln, _Y, _Z).
zone_length(_Length):-                % oops, not a rec zone
        feedback(g_focus_error).
```

```
zone_width(_Width):-
        g_focus(_Cur_zone_no), zone_type(_Cur_zone_no, rec, 7),
        ( vertex(_Cur_zone_no, 0, _X1, _Y1, _Z1), nonvar(_Y1) ; _Y1 is 0 ),
        _Wd is _Y1 + _Width,
        update_vertex(_Cur_zone_no, 2, _X, _Wd, _Z),
        update_vertex(_Cur_zone_no, 3, _X, _Wd, _Z),
        update_vertex(_Cur_zone_no, 6, _X, _Wd, _Z),
        update_vertex(_Cur_zone_no, 7, _X, _Wd, _Z).
zone_width(_Width):-
        feedback(g_focus_error).

zone_height(_Height):-
        g_focus(_Cur_zone_no), zone_type(_Cur_zone_no, rec, 7),
        ( vertex(_Cur_zone_no, 0, _X1, _Y1, _Z1), nonvar(_Z1) ; _Z1 is 0 ),
        _Ht is _Z1 + _Height,
        repeat, gen_integer(_N, 4), (                    % dont backtrack into update
                vertex(_Cur_zone_no, _N, _X, _Y, _Z),
                update_vertex(_Cur_zone_no, _N, _X, _Y, _Ht),
        !), _N = 7, !.                  % dont backtrack into repeat loop


zone_height(_Height):-                  % changing height of reg body
        g_focus(_Cur_zone_no), zone_type(_Cur_zone_no, reg, _No_points),
        ( vertex(_Cur_zone_no, 0, _X1, _Y1, _Z1), nonvar(_Z1) ; _Z1 is 0 ),
        _Ht is _Z1 + _Height,
        repeat, gen_integer(_N, 0), (                    % dont backtrack into update
                _N_bot is _N * 2,
                _N_top is _N_bot + 1,
                vertex(_Cur_zone_no, _N_bot, _X, _Y, _Z),
                update_vertex(_Cur_zone_no, _N_top, _X, _Y, _Ht),
        !), _N_top = _No_points, !.             % dont backtrack into repeat loop
zone_height(_Height):-                          % its a gen!
        feedback(g_focus_error).

feedback(g_focus_error,expert):-
        chat_usr([
                'Sorry, are we talking about the same zone?. Could you',
                'please deselect and reselect the zone in question',
                "]).
feedback(g_focus_error,novice):-
        chat_usr([
                'Sorry, I am getting confused. I suspect the zone you are',
                'looking at is not the one I think it is. Could you confirm',
                ' which zone you are addressing (by reselecting it)',
                "]).
```

```
/*      NOT USED, vertices now held relative
 * zone_origin_x(_X):-                          % setting origin of the body
 *      g_focus(_Cur_zone_no),
 *      zone_type(_Cur_zone_no, _Type, _No_points),
 *      repeat, gen_integer(_N, 1), (
 *              vertex(_Cur_zone_no, _N, _Xold, _Yold, _Zold),
 *              (nonvar(_Xold) ->
 *                      _Xnew is _X + _Xold
 *              ;
 *                      _Xnew is _X
 *              ),
 *              update_vertex(_Cur_zone_no, _N, _Xnew, _Yold, _Zold),
 *      !), _N = _No_points, !.
 *
*zone_origin_y(_Y):-
 *      g_focus(_Cur_zone_no),
 *      zone_type(_Cur_zone_no, _Type, _No_points),
 *      repeat, gen_integer(_N, 1), (
 *              vertex(_Cur_zone_no, _N, _Xold, _Yold, _Zold),
 *              (nonvar(_Yold) ->
 *                      _Ynew is _Y + _Yold
 *              ;
 *                      _Ynew is _Y
 *              ),
 *              update_vertex(_Cur_zone_no, _N, _Xold, _Ynew, _Zold),
 *      !), _N = _No_points, !.
 *
 *zone_origin_z(_Z):-
 *      g_focus(_Cur_zone_no),
 *      zone_type(_Cur_zone_no, _Type, _No_points),
 *      repeat, gen_integer(_N, 1), (
 *              vertex(_Cur_zone_no, _N, _Xold, _Yold, _Zold),
 *              (nonvar(_Zold) ->
 *                      _Znew is _Z + _Zold
 *              ;
 *                      _Znew is _Z
 *              ),
 *              update_vertex(_Cur_zone_no, _N, _Xold, _Yold, _Znew),
 *      !), _N = _No_points, !.
 */

'x_coord$'(_N, _X):-
        new_coord(_N, _X, _, _).

'y_coord$'(_N, _Y):-
        new_coord(_N, _, _Y, _).

'z_coord$'(_N, _Z):-
        new_coord(_N, _, _, _Z).
```

```
new_coord(_N, _X, _Y, _Z):-
        g_focus(_Cur_zone_no),
        zone_type(_Cur_zone_no, reg, _No_points),
        var(_Z),                                % no Z input for reg
        _Pt_no is _N + _N,
        update_vertex(_Cur_zone_no, _Pt_no, _X, _Y, _),
        _Ceiling_pt_no is _Pt_no + 1,
        update_vertex(_Cur_zone_no, _Ceiling_pt_no, _X, _Y, _Ht),
        ( _Ceiling_pt_no = _No_points ->  %%% open new coord 'field'
                _N2 is _N + 1,
                _Nxt_point is _No_points + 1,
                update_vertex(_Cur_zone_no, _Nxt_point, _, _, 0),
                _Nxt_ceiling_point is _No_points + 2,
                update_vertex(_Cur_zone_no, _Nxt_ceiling_point, _, _, _Ht),
                ask_usr(['vert_no$', _N2], _N2),
                ask_usr(['x_coord$', _N2]),
                ask_usr(['y_coord$', _N2]),
                retract(zone_type(_Cur_zone_no, reg, _)),
                assert(zone_type(_Cur_zone_no, reg, _Nxt_ceiling_point))
        ;
                true
        ),
        ( vertex(_Cur_zone_no, _Pt_no, _X1, _Y1, _), nonvar(_X1), nonvar(_Y1) ->
                _Face is _N + 1,                        % got point, update surfaces
                _Last_pt_no is _Pt_no - 2,
                _Last_ceiling_pt_no is _Pt_no - 1,
                update_surface(_Cur_zone_no, _Face, [_Last_pt_no, _Pt_no,
                                        _Last_ceiling_pt_no,_Ceiling_pt_no]),
                update_surface(_Cur_zone_no, 0, _Pt_no, _),             % floor
                update_surface(_Cur_zone_no, 1, _, _Ceiling_pt_no)      % ceiling
        ;
                true
        ).

new_coord(_N, _X, _Y, _Z):-
        g_focus(_Cur_zone_no),
        zone_type(_Cur_zone_no, gen, _No_points),
        update_vertex(_Cur_zone_no, _N, _X, _Y, _Z),
        ( _N = _No_points ->
                _Nxt_point is _No_points + 1,
                update_vertex(_Cur_zone_no, _Nxt_point, _, _, _),
                ask_usr(['x_coord$', _Nxt_point]),
                ask_usr(['y_coord$', _Nxt_point]),
                ask_usr(['z_coord$', _Nxt_point]),
                retract(zone_type(_Cur_zone_no, gen, _)),
                assert(zone_type(_Cur_zone_no, gen, _Nxt_point))
        ;
                true
        ).

new_coord(_N, _X, _Y, _Z):-                             /* should not occur for rec*/
        g_focus(_Cur_zone_no),
        zone_type(_Cur_zone_no, rec, _No_points).
```

```
'surface$'(_N, _Points):-
        g_focus(_Cur_zone_no),
        zone_type(_Cur_zone_no, gen, _No_points),
        name(_Points,_Pt_lst),
        chars_to_words(_Pt_lst, _List),
        check_surface(_List, _No_points),
        update_surface(_Cur_zone_no, _N,  _List).

check_surface([_Pt|_List], _No_points):-    %% should do coplanar check!!!!!
        nonvar(_Pt),
        _Pt =< _No_points,!,
        check_surface(_List, _No_points).
check_surface([_Pt], _No_points):-
        _Pt =< _No_points.
check_surface([], _No_points).
check_surface(_Pt, _No_points):-
        nonvar(_Pt),
        _Pt =< _No_points.
check_surface(_L, _No_points):-
        write("check_surface failed given: ["),                    %%%%% to stderr!!!!!
        write(_L),
        write("], "),
        write(_No_points),
        nl.

update_vertex(_Cur_zone_no, _N, _X, _Y, _Z):-
        nonvar(_Cur_zone_no), nonvar(_N),
        ( retract(vertex(_Cur_zone_no, _N, _Xold, _Yold, _Zold)) ; true ),
        update_coord(_X,_Xold,_Xnew),
        update_coord(_Y,_Yold,_Ynew),
        update_coord(_Z,_Zold,_Znew),
        assert(vertex(_Cur_zone_no, _N, _Xnew, _Ynew, _Znew)),
        ( nonvar(_Xnew), nonvar(_Ynew), nonvar(_Znew),  /* got full vertex */
                uset(vertex, [_Cur_zone_no, _N], [_Xnew, _Ynew, _Znew]),
                feedback_geom(_Cur_zone_no)
        ;
                true
        ).

update_surface(_Cur_zone_no, _N, _List):-
        nonvar(_List), nonvar(_Cur_zone_no), nonvar(_N),
        ( retract(surface(_Cur_zone_no, _N, _L)) ; true ),
        assert(surface(_Cur_zone_no, _N, _List)),
        uset(surface, [_Cur_zone_no, _N], _List),
        feedback_geom(_Cur_zone_no).

update_surface(_Cur_zone_no, _N, _Prelist, _List):-
        nonvar(_Prelist), nonvar(_Cur_zone_no), nonvar(_N),
        ( surface(_Cur_zone_no, _N, _List) ; true ),
        ( nonvar(_List), member(_Prelist, _List)
;       ( nonvar(_List), append([_Prelist], _List, _L)
        ; _L is [_Prelist]
        ),
        update_surface(_Cur_zone_no, _N, _L)
).
```

```
update_surface(_Cur_zone_no, _N, _List, _Postlist):-
        nonvar(_Postlist), nonvar(_Cur_zone_no), nonvar(_N),
        ( surface(_Cur_zone_no, _N, _List) ; true ),
        ( nonvar(_List), member(_Postlist, _List)
        ;       ( nonvar(_List), append(_List, [_Postlist], _L)
                ; _L is [_Postlist]
                ),
                update_surface(_Cur_zone_no, _N, _L)
        ).

update_coord(_A,_Aold,_Anew) :-
        ( nonvar(_A) ->
                _Anew is _A
        ;
                ( nonvar(_Aold) ->
                        _Anew is _Aold
                ;
                        true
                )
        ).

feedback_geom(_Cur_zone_no).      %% perspective feedback?
```

```
construction(initialize) :- %% now address construction specification cpt
        conlst(1, _List),  %% get list of available materials from database
        offer_usr(material_type, _List),
        feedback(construction_sel).

feedback(construction_sel, novice) :-
        chat_usr([
                'This button switches the focus of discussion to',
                'defining the materials and construction primitives',
                'eg. a particular external wall construction to be',
                'used. The lower half of the form provides access',
                'to some standard databases. If a materials is not',
                'shown here, its thermophysical properties will be',
                'required.',
                "]).

b_construction(refresh) :-          %% tell user all known about this meta-cpt
        true.


material_type(_Material) :-                 /* browser */
        conlst(2, _Material, _List),
        offer_usr(material, _List),
        ask_usr(material).
```

```
%%%%%b_bld_browse(on) :-                    /* browser */
%%%%%          no_zones(_Zn_no),                    %% should fail or be 0
%%%%%          _Zn_no > 0,
%%%%%          feedback(browse_error).

bld_browse(initialize) :-                  /* ok to browse */
        (browsing ->
                true                       /* only start one browser */
        ;
                assert(browsing),
                to_bb(application,start,browse)
        ),
        feedback(browse_geom).

feedback(browse_error, novice):-
        chat_usr([
                'Sorry, the browse facility is designed to create a',
                'complete building geometry and you already have started',
                'inputting some zone descriptions.  In order to use the',
                'browse facility, you will have to destroy the geometry',
                ' already input - use the "zap" button on this form iff',
                'you really want to do this',
                "]).

feedback(browse_error, expert):-
        chat_usr([
                'You already have a geometry!  Zap it',
                "]).


feedback(browse_geom, novice):-
        chat_usr([
                'The browse facility first asks you to select the type',
                'of building, then offers a number of topologies before',
                'finally permitting a limited dimensioning facility',
                'Browse cannot be used with any other geometry input',
                'method, as it is designed to create a complete building',
                'geometry.  However, the resulting geometry, once passed',
                'back to the ife, can be modified using any other method',
                "]).

feedback(browse_geom, expert):-
        chat_usr([
                'The browse facility first asks you to select the type',
                'of building, then offers a number of topologies before',
                'finally permitting a limited dimensioning facility',
                "]).

building_class(_Class_list):-
        offer_usr(browse_type, '??'),
        offer_usr(browse_image, '??'),
        offer_usr(browse_class, _Class_list).

browse_class(_Item):-
        offer_usr(browse_type, '??'),
        offer_usr(browse_image, '??'),
        to_bb(application, inform, browse, [['class:',_Item]]).

building_types(_Types_list):-
        offer_usr(browse_type, _Types_list).
```

```
browse_type(_Item):-
        to_bb(application, inform, browse, [['type:',_Item]]).

building_images(_Image_list):-
        offer_usr(browse_image, _Image_list).

browse_image(_Building_name):-
        current_building(_Building_name) ->
                retract(current_building,1)
        ;
                true
        ),
        assert(current_building(_Building_name)).

browse_select(on):-
%%      current_building(_Building_name) ->
%%              uset(building_name,_Building_name),
%%              to_bb(application, start, init_rove),
                shell("bin/init_rove &"),
                to_bb(application, start, etherlink).
%%      ;
%%              true
%%      ).
```

```
analysis(initialize) :-        %% now addressing analysis specification cpts
        offer_usr(m_analysis1, 'comfort ann_cons peak_load' ),
        feedback(analysis_sel).

feedback(analysis_sel, novice) :-
        chat_usr([
                'This button switches the focus of discussion to',
                'the sort of analysis required. Information about',
                'the design stage is asked for so that only',
                'appropriate analysis (and data input) will be',
                'carried out.',
                 "]).
analysis(refresh) :-                    %% tell user all known about this meta-cpt
        true.

b_mono_functional_methodology(on):-
        focus_concept(analysis, mono_functional_methodology).

m_analysis(comfort):-
        shell("lib/uc/uc/appraisals/comfort &").

comfort_analysis(_results,complete):-
        offer_usr(analysis_results,_results).

m_analysis(energy):-
        shell("lib/uc/uc/appraisals/energy &").

energy_analysis(_results,complete):-
        offer_usr(analysis_results,_results).

m_analysis(condensation):-
        shell("lib/uc/uc/appraisals/condensation &").

condensation_analysis(_results,complete):-
        offer_usr(analysis_results,_results).


b_multi_functional_methodology(on):-
        focus_concept(analysis, multi_functional_methodology).

mm_analysis(control_systems):-
        shell("lib/uc/uc/appraisals/control_systems &").

control_systems(_results,complete):-
        offer_usr(analysis_results,_results).
```

```
feedback(_Concept) :-
        chat_usr(['', '']),
        ( user_level(_User_level) ->
                feedback(_Concept, _User_level)
        ;
                feedback(_Concept, novice)
        )
        ; true.

focus_concept(_Focus_type, _Meta_concept):-
        defocus_concept(_Focus_type),                   %% defocus relevent meta_cpt
        focus_usr(_Meta_concept),                       %% enable input, flag as currently
        assert(focus(_Focus_type, _Meta_concept)),      %%  available for input
        assert(called(_Meta_concept)),                  %% has been addressed (data input?)
        name(_Meta_concept, _Mcpt_str),                 %% load handler for this meta-concept
        append("lib/uc/uc/kbs/", _Mcpt_str, _Mcpt_str2),
        name(_Meta_concept2, _Mcpt_str2),
        [-_Meta_concept2].

defocus_concept(_Focus_type):-
        ( focus(_Focus_type, _Meta_concept) ->
                name(_Meta_concept, _Mcpt_str), %% turn off button; b_ prefix
                append("b_", _Mcpt_str, _Mcpt_str2),
                name(_Meta_concept2, _Mcpt_str2),
                tell_usr(_Meta_concept2, 'off'),
                unfocus_usr(_Meta_concept),
                retract(focus(_Focus_type, _Meta_concept))
        ;
                true
        ).

tell_usr(_Concept, _Value) :-
        to_bb(user_dialog, tell_user, _Concept, _Value).
ask_usr(_Concept) :-
        to_bb(user_dialog, ask_user, _Concept).
ask_usr(_Concept, _Default) :-
        to_bb(user_dialog, ask_user, _Concept, _Default).
unask_usr(_Concept) :-
        to_bb(user_dialog, unask_user, _Concept).
focus_usr(_Concept) :-
        to_bb(user_dialog, focus_user, _Concept).
unfocus_usr(_Concept) :-
        to_bb(user_dialog, unfocus_user, _Concept).
defocus_usr(_Concept) :-
        to_bb(user_dialog, defocus_user, _Concept).
offer_usr(_Concept, _Values) :-
        to_bb(user_dialog, offer_user, _Concept, _Values).
suggest_usr(_Concept, _Value) :-
        to_bb(user_dialog, suggest_user, _Concept, _Value).
new_dialogue(_Type, _Cmd) :-
to_bb(user_dialog, new_dialog, _Type, _Cmd).
```

```
to_bb(_A, _B, _C, _D, _E) :-        % eg u_cpt concept keys args who_set
        to_bb(_A), csep,
        to_bb(_B, _C, _D, _E).
to_bb(_A, _B, _C, _D) :- % eg user_dialog tell_usr concept value
        to_bb(_A), csep,
        to_bb(_B, _C, _D).
to_bb(_A, _B, _C) :-                % eg user_dialog ask_usr cpt
        to_bb(_A), csep,  % or notify_me area id_string
        to_bb(_B, _C).
to_bb(_A, _B) :-            % eg mk_area area
        to_bb(_A), csep,
        to_bb(_B), nl, ttyflush.
to_bb([_Head]) :-
        writeline(_Head).
to_bb([_Head|_Tail]) :-
        to_bb(_Head), ksep,
        to_bb(_Tail).
to_bb([]).
to_bb(_Arg) :-
        write(_Arg).

chat_usr(['']):-                        % text separator on its own
        chat_usr([]).
chat_usr(_Text):-
        write('user_dialog'), csep,
        write('chat_user'), csep,
        writetext(_Text).
writetext([_Head]) :-
        writeline(_Head), nl.
writetext([_Head|_Tail]) :-
        writeline(_Head), tsep,
        writetext(_Tail).
writetext([]) :-
        tsep, nl.
writetext(_Arg) :-
        write(_Arg), nl.
writeline([_Head|_Tail]) :-
        writeline(_Head),
        writeline(_Tail).
writeline([_Head]) :-
        writeline(_Head).
writeline([]).
writeline(_Arg) :-
        write(_Arg).

csep :-                                 % bb's concept separator
        write('    ').                  %%      = \t
ksep :-                                 % bb's concept key separator
        write('    ').                  %%      = \t
tsep :-                                 % bb's text separator
        write('\n').                    %%      = '\n'
```

```
uset(_Concept, _Value) :-              /* user set concept value */
        nonvar(_Concept),
        set(u_cpt, _Concept, nokey, _Value, user_set).
uset(_Concept, _Keys, _Value) :-
        nonvar(_Concept), nonvar(_Keys),
        set(u_cpt, _Concept, _Keys, _Value, user_set).
kset(_Concept, _Value) :-      /*knowledge handler set concept value*/
        nonvar(_Concept),
        ( known(_Concept, nokey, _Value, user_set)
        ;       set(u_cpt, _Concept, nokey, _Value, kb_set)
        ).
kset(_Concept, _Keys, _Value) :-
        nonvar(_Concept), nonvar(_Keys),
        ( known(_Concept, _Keys, _Value, user_set)
        ;       set(u_cpt, _Concept, _Keys, _Value, kb_set)
        ).

set(_Area, _Concept, _Keys, _Value, _Who_set) :-
        _Old_Term =.. [_Area, _Concept, _Keys, _, _],  %%%% hold locally ??
        ( retract(_Old_Term) ; true ),
        _Term =.. [_Area, _Concept, _Keys, _Value, _Who_set],
        assert(_Term),
        to_bb(_Area, _Concept, _Keys, _Value, _Who_set).

known(_Concept, _Value) :-
        nonvar(_Concept), var(_Value),
        query(u_cpt, _Concept, nokey, _Value, _).
known(_Concept, _Key, _Value) :-
        nonvar(_Concept), nonvar(_Key), var(_Value),
        query(u_cpt, _Concept, _Key, _Value, _).
knownw(_Concept, _Value, _Who_set) :-
        nonvar(_Concept), var(_Value),
        query(u_cpt, _Concept, nokey, _Value, _Who_set).
knownw(_Concept, _Key, _Value, _Who_set) :-
        nonvar(_Concept), nonvar(_Key), var(_Value),
        query(u_cpt, _Concept, _Key, _Value, _Who_set).
query(_Area, _Concept, _Key, _Value, _Who_set) :-
        _Term =.. [_Area, _Concept, _Key, _Value, _Who_set],  % check locally
        _Term.
/*known(_Area, _Concept, _Value, _Who_set) :-      %%%% held locally for now
 *      write(query), write('          '),
 *      write(_Area), write('          '),
 *      to_bb(_Concept, '?', '?'),
 *      read_ans(_Concept, _Value, _Who_set),
 *      _Value \== '?'.
 */
```

```
u_cpt_got(_Concept, _Value) :-          /* someone else setting values */
        %%%%% take appropriate action!!!!!
        true.

quitrqst :-                                     /* user said quit */
        to_bb(quitrqst, kb-uc),
        halt.
quitrqst(_) :-
        to_bb(quitrqst, kb-uc),
        halt.
quitrqst(_,_) :-
        to_bb(quitrqst, kb-uc),
        halt.
```

```
position_of(glasgow, 55.9, -4.5).
position_of(london, 51.5, 0.1).
position_of(edinburgh, 55.9, -3.1).
position_of(manchester, 53.4, -2.5).
position_of(birmingham, 52.5, -1.9).
position_of(exeter, 50.7, -3.5).
position_of(belfast, 54.6, -11.6).
position_of(dublin, 53.4, -6.2).

environment(glasgow, city_centre).
environment(london, city_centre).
environment(edinburgh, city_centre).
environment(manchester, city_centre).
environment(birmingham, city_centre).
environment(exeter, city_centre).
environment(belfast, city_centre).
environment(dublin, city_centre).
environment(lenzie, urban).
environment(wantage, urban).
environment(abingdon, urban).
environment(oxon, rural).
environment(wilts, rural).
environment(berks, rural).
environment(chilton, rural).

exposure(city_centre, 1).
exposure(urban, 3).
exposure(rural, 4).
exposure(reference, 6).

grnd_rflct(city_centre, 2.5).
grnd_rflct(urban, 1.75).
grnd_rflct(rural, 1.25).
grnd_rflct(reference, 1).

climate_set(glas, 4.5, 55.9).
climate_set(kew, 0.2, 51.5).

climate_type(glas82, very_bad_week, '7/1', '15/1').
climate_type(glas82, very_bad_day, '7/1', '7/1').
climate_type(glas82, bad_day, '9/1', '9/1').
climate_type(glas82, average_week, '7/1', '15/1').
climate_type(glas82, good_week, '15/7', '21/7').
climate_type(glas82, very_good_day, '17/7', '17/7').

function(residential, 1).
function(industrial, 2).
function(office, 3).
function(hospital, 4).
function(school, 5).
```

```
%%% The following predicates should be C functions
getenv('IFE_LOC',uk).
conlst(1, _List):-
        _List =
'asbestos asphalt_and_bitumen brick carpet concrete earth glass insulation metal plaster screeds_
and_renders stone tiles wood'.
conlst(2, wood, _List):-
        _List =
'block hardboard(medium) hardboard(standard) fir_(20%_mc) flooring cork_board chipboard weather
board oak_(radial) plywood softwood'.
```

```
refresh(_Concept) :-
        known(_Concept, _Value),
        ask_usr(_Concept, _Value).
refresh(_).                                          %% always succeed
refresh(_Concept, _Keys) :-
        known(_Concept, _Keys, _Value),
        ask_usr(_Concept, _Value).
refresh(_, _).                                       %% always succeed

near(_X1, _Y1, _X2, _Y2, _Criteria) :-
        number(_X1), number(_Y1), number(_X2), number(_Y2),
        _Criteria > ((_X1 - _X2) ^ 2 + (_Y1 - _Y2) ^ 2).

near(_X1, _Y1, _X2, _Y2, _Criteria) :-
        ( number(_X1) -> _X3 = _X1 ; flt(_X1, _X3) ),
        ( number(_Y1) -> _Y3 = _Y1 ; flt(_Y1, _Y3) ),
        ( number(_X2) -> _X4 = _X2 ; flt(_X2, _X4) ),
        ( number(_Y2) -> _Y4 = _Y2 ; flt(_Y2, _Y4) ),
        _Criteria > ((_X3 - _X4) ^ 2 + (_Y3 - _Y4) ^ 2).

location_near(_Latitude, _Longitude, _Location) :-
        position_of(_Location, _Lat, _Lng),
        near(_Latitude, _Longitude, _Lat, _Lng, 0.3).


%% gen_integer(X,Seed) generates integers, incrementing from Seed.
%% NG if arg1 is nonvar or arg2 is var.

gen_integer(X, Seed) :-
  var(X),
  integer(Seed),
  gen_integer_vc(X, Seed).

gen_integer_vc(X, Seed) :-
  X is Seed;
  (New_seed is Seed+1,
   gen_integer_vc(X, New_seed)).


%% append(First_part, Second_part, List) iff List is the
%% concatenation of the first two arguments.

append([], List, List).
append([Elem | First_part], Second_part, [Elem | List]) :-
  append(First_part, Second_part, List).

member(X,[X|_]).
member(X,[_|T]) :-
        member(X,T).

%% chars_to_words(Chars, Words)
%% parses a list of characters (read by read_until) into a list of words,
%%                striped down to only handle ints

chars_to_words(Chars,Words) :-
     chars_to_words(Words,Chars,[]).
```

```
chars_to_words([Word|Words],A,B) :-
        chars_to_word(Word,A,C), !,
        chars_to_words(Words,C,B).

chars_to_words([],A,A).


chars_to_word(Word,A,B) :-                    %% this bit repeated for valid tokens
         'C'(A,Char,C),
         is_digit(Char), !,
         Init is Char-48,
        chars_to_integer(Init,Word,C,B).

chars_to_integer(Init,Final,A,B) :-
        'C'(A,Char,C),
        is_digit(Char), !,
        Next is Init*10-48+Char,
        chars_to_integer(Next,Final,C,B).

chars_to_integer(Final,Final,A,A).

is_digit(Char) :-
          Char >= "0", Char =< "9".           % decimal digit

chars_to_word(Word,A,B) :-
        'C'(A,Char,C),
        is_space(Char), !,
       chars_to_word(Word,C,B).
is_space(32).                                 % ``
is_space(95).                                 % `_`          %% for bb input strings
is_space( 9).                                 % `\t`
is_space(10).                                 % `\n`
is_space(11).                                 % `\v`
is_space(12).                                 % `\f`
is_space(13).                                 % `\r`
```

```
# include       <stdio.h>
# include       "plload.h"

# include       <sys/time.h>
get_date()
{       struct tm       *now;
        long    nowtime;
        char    date[12];
        static char *month[]={"Jan","Feb","Mar","Apr","May","Jun",
                              "Jul","Aug","Sep","Oct","Nov","Dec",0};
        nowtime = time(0);
        now = localtime(&nowtime);
        sprintf(date,"%2d:%s:%d",now->tm_mday,month[now->tm_mon],now->tm_year);
        if (!putatom(1, date))  return(FAIL);
        return(SUCCEED);
}

get_env() /** get environment varaible **/
{
        extern char* getenv();
        char* env_name;
        char* env_var;

fprintf(stderr,"get_env\n");
        if(!getatom(1, &env_name))  return(FAIL);
fprintf(stderr,"env_name = `%s'\n",env_name);
        if((env_var = getenv(env_name))==NULL) return(FAIL);
        if(!putatom(2,env_var)) return(FAIL);
fprintf(stderr,"env_var = `%s'\n",env_var);
        return(SUCCEED);
}

proj_exists()
{       char    *name;

        if (!getatom(1, &name))  return(FAIL);
        if (strcmp(name, "old_proj"))  return(FAIL);
        if (!putint(2, 5))  return(FAIL);               /* session */
        if (!putatom(3, "01/11/87"))  return(FAIL); /* date started */
        if (!putatom(4, "old_proj_data"))  return(FAIL);/* data file */
        if (!putatom(5, "old_proj_log"))  return(FAIL);/* log file */
        return(SUCCEED);
}

#include        <math.h>
#include        <ctype.h>
flt()
{       char    *str, *ch_ptr;
        double  f;

        if (!getatom(1,&str))  return(FAIL);
        for (ch_ptr = str; *ch_ptr != '\0'; ch_ptr++)
                if (!isdigit(*ch_ptr)  &&  *ch_ptr != '.')
                        return(FAIL);
        f = atof(str);
        if (!putfloat(2,f))  return(FAIL);
        return(SUCCEED);
};
```

APPENDIX E

3D object viewing and manipulation (visulising time variant data).

E. 3D OBJECT VIEWING AND MANIPULATION (VISULISING TIME VARIANT DATA).

An integral part of any CAD package is a geometrical description of the product. Following on from the generic architecture identified in chapter 3, used in the forms package, a 3D object viewing and manipulation environment has been developed to aid the visualisation of time variant data. The program has been developed on a Silicon Graphics Iris workstation taking full advantage of the geometry pipeline. ROVE (Real-time Object Visualisation Environment) developed by Malcolm Shamwana, formally of ABACUS, has been used as a reference model. The extensions provide the user with basic mouse and keyboard control over viewing parameters (see camera E.4) and will eventually be able to interact directly with individual objects which will format utterances containing manipulation events which will then be passed to an appropriate knowledge base and interpreted.

The same principles developed for the forms package apply to the 3D manipulation package in that coneptual templates are defined hetra-archically with each node pointing to an instance of a 3D body. By simply grouping bodies in this way concepts may be represented and mainpulated graphically.

Control mechanism may exist externally and pass messages via IP and TCP to the package. Alterntively control sequences may be embeded within the object itself and fired at will. This is achieved using an introverted software IC (E.5.1) and object scripts.

E.1. OBJECT DEFINITION

An object is composed of two sets of data:

i) generic

ii) instance

In the case of 3D modelling the genric data consists of the following generic attribtutes:

- origin
- orientation
- scale
- colour
- Material properties may also be included.

while instance data is specific to the form of 3D representation.

This very simple distinction between generic and instance data enables any 3D body to be manipulated in a generic manner. In the same way in which the forms package utilises interchangeable concept interpreters so gemetrical interpreterations of a concept may be interchanged dynamically without affecting control knowledge.

This results in a multi-representational system. For example a specific object may be represented either symbolically or litterally by a complete topological and topographical description. Symbolic and literal representations of objects within a model may exist in parralel. In the case of building design it may be appropriate to display a building in wireline while services are represented using some form of symbolic notation.

Another advantage is one of extendibility. Since the only reference to a body is by means of its generic attributes new object primitives may be integrated quickly and easily without disturbing the rest of the system. The following is a complete list of the current primitive set:

ABACUS Viewer:
EXT
GEN
LIN
PLA
RAG
REC
REG
TIL
RGB
FIL

Other (parametric) primitives include:

bottle volume of revolution- this expects a profile which is swept at run time.

patch bicubic patch

tree fractal trees

Light sources are also to be added together with 3D raster imaging capabilities.

E.2. TREES

A fractal tree generator [PETRIC 88] has also been integrated within the system. The original program was written in FORTRAN by J. Petric (ABACUS) and in order to integrate models of trees with other 3D primitives it was necessary to generate, from essentially a parrametric description, a 3D vertex description of a tree. The resulting

files typically contained upto and over three thousand line bodies which placed enormous demands upon the storage device.

The approach taken, during integration (owing to my complete hatred of the Fortran programming language), was to re-use the original fortran code and write a C interface arround it. This interface simply reads a parametric desciption from file and makes a subroutine call to trees. Therefore, rather than create a complete model file, a FORTRAN to C interface was written enabling 3D vetcors to be pushed onto the geometry pipeline (after some post processing) resulting is a much more compressed tree description file; typically a tree is generated from a file containing 10-20 parrameters.

Fortran is a static language in that memory must be assigned at initialisation. C on the other hand is dynamic enabling memory to be allocated and deallocated during execution.

The advantages of the C language are exploited fully throughout. In this instance array sizes (determined by certain parameters in the tree description file) are dynamically allocated in the C portion of the code and their respective memory addresses passed as arguments to the Fortran subroutine. From a practical viewpoint the program is much more efficient and only a single (compressed) representation of the model is held on disc.

Parametric descriptions are seen as being the key to multi-representaional systems enabling different interpretations of data to be generated. This was the basis for the re-phrasing mechanisms for the forms package.

By creating a series of instance interpreters many different representaions of the same instance data are possible.

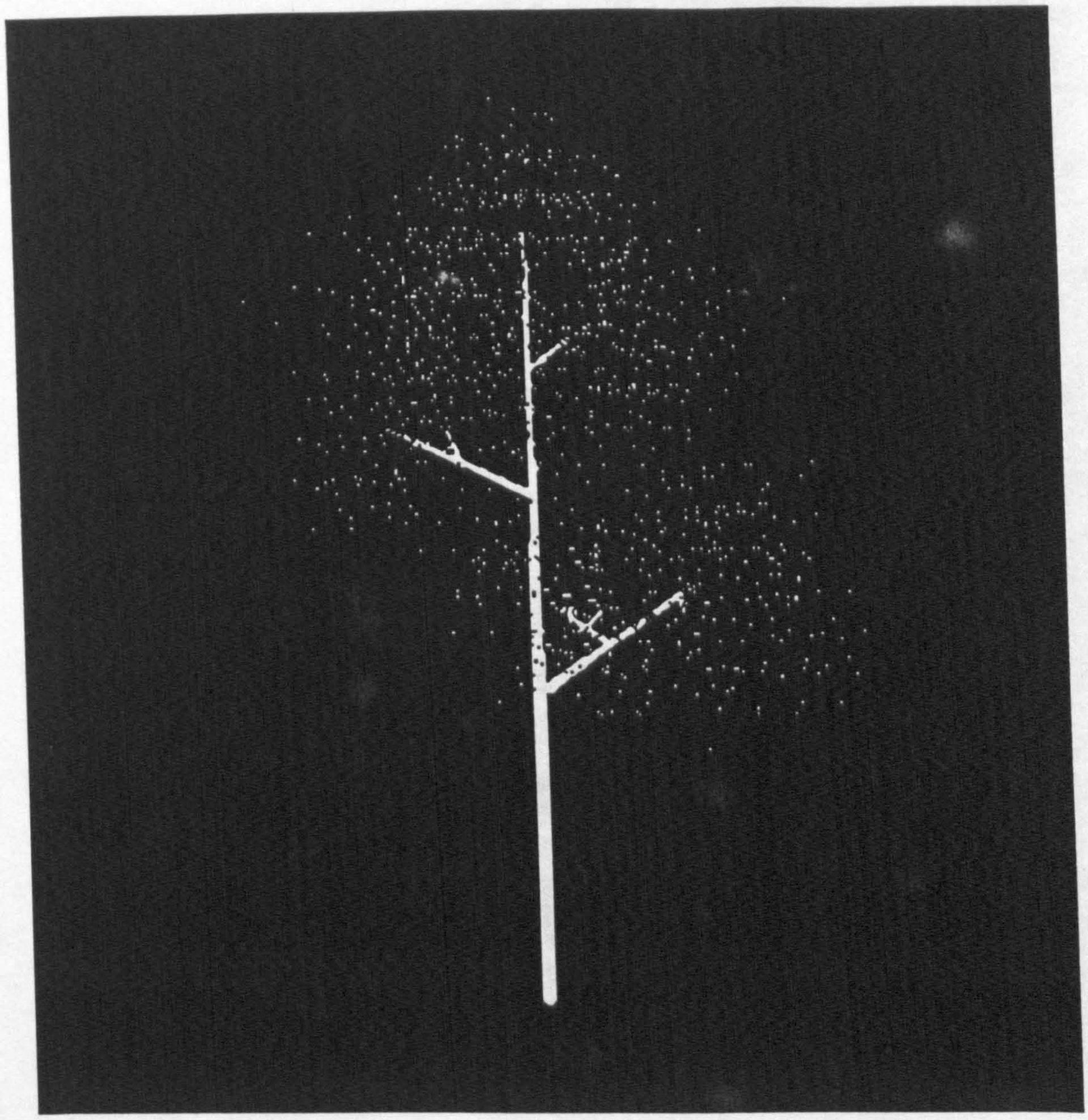


Figure E.2.1. Fractal tree primitive with impresionistic foliage.

```
tree
0
    2   1   5   9    6    0.00 140.   0
1    0.001   0.001   1.0  1.0   0
    2   2
   1.0 3.0
  70.0 65.0
1
    0   0   0
    0.0
    0.0
    0.0
    0.0
0   0   0   0   0   5
    0   0   0   0   0   0   0
```

Figure B.2.2. Fractal tree parameters [PETRIC 88]

In the case of the trees primitive, although not yet coded, a series of intepreters are under development to enable trees and foliage to be displayed at different levels of

detail; from literal to impresionistic representations of individual and grouped objects. Figure E.2.1 illustrates an impressionistic representation of tree foliage (simple points randomy placed within a sphere at each twig) with the corresponding growth paramenters in figure E.2.2. Figure E.2.3 illustrates several instances of the tree primitive clumped together.

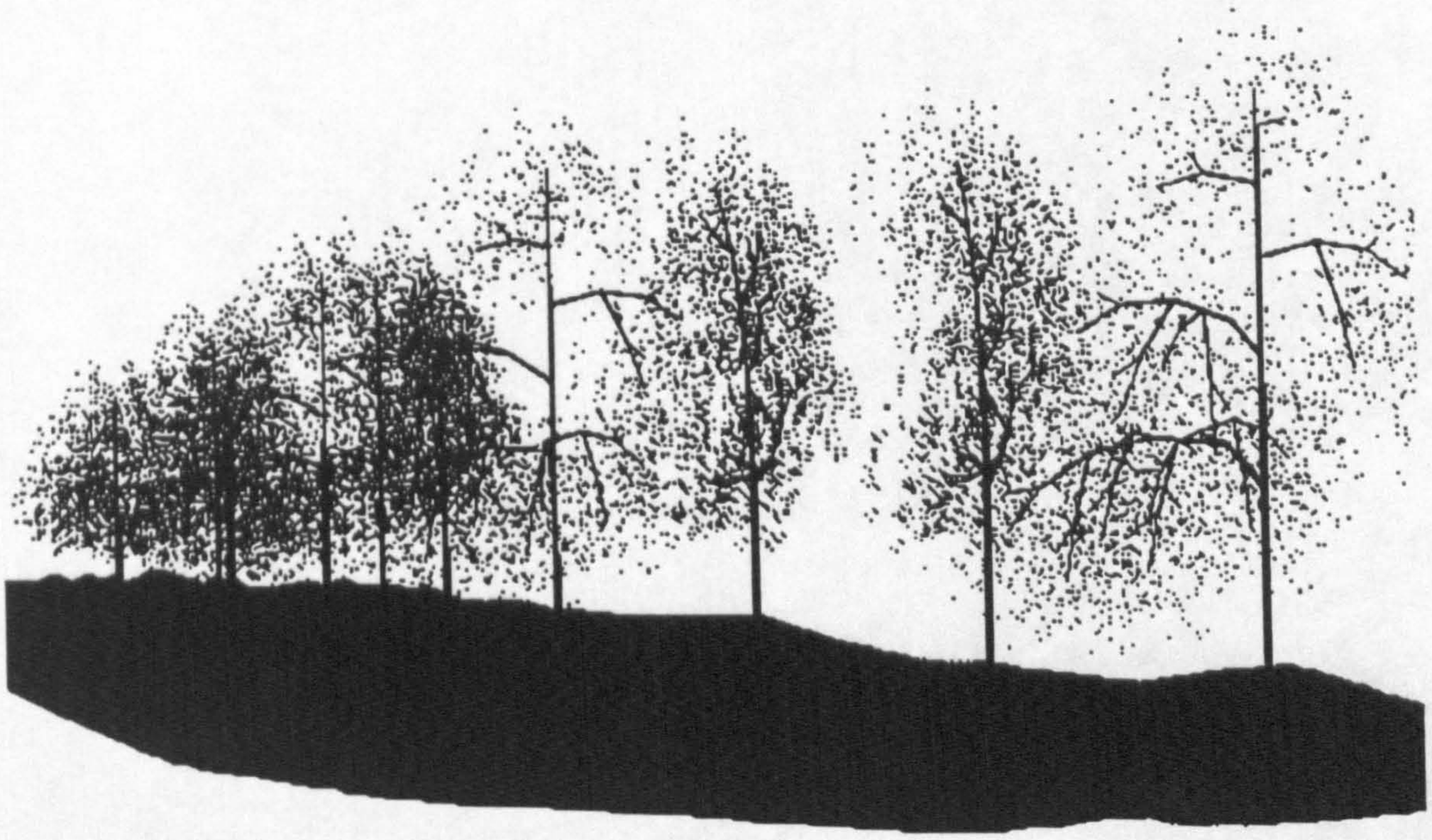


Figure E.2.3. Fractal trees with impressionistic foliage clumped together.

More general 3D interpreter, again although not fully implemented, are:

wireline

flat shaded

gouraud shaded

E.3. MULTIPLE INSTANCES

Since generic and instance data have been separated the program enables several (N) objects to be defined, each with their own instance attributes, while pointing to the same instance data. This is achieved usging the following syntax:

refer instance_base_name * number required	
t(x y z) s(x y z) o(x y z)	(base translation, scale and orientation)
dt(x y z) ds(x y z) do(x y z)	(translation, scale and orientation increments to be applied to succesive references.

The order in which the trasformations are specified is important. The modelling routines within the Iris Graphics Library are not commutative, figure E.3.1:

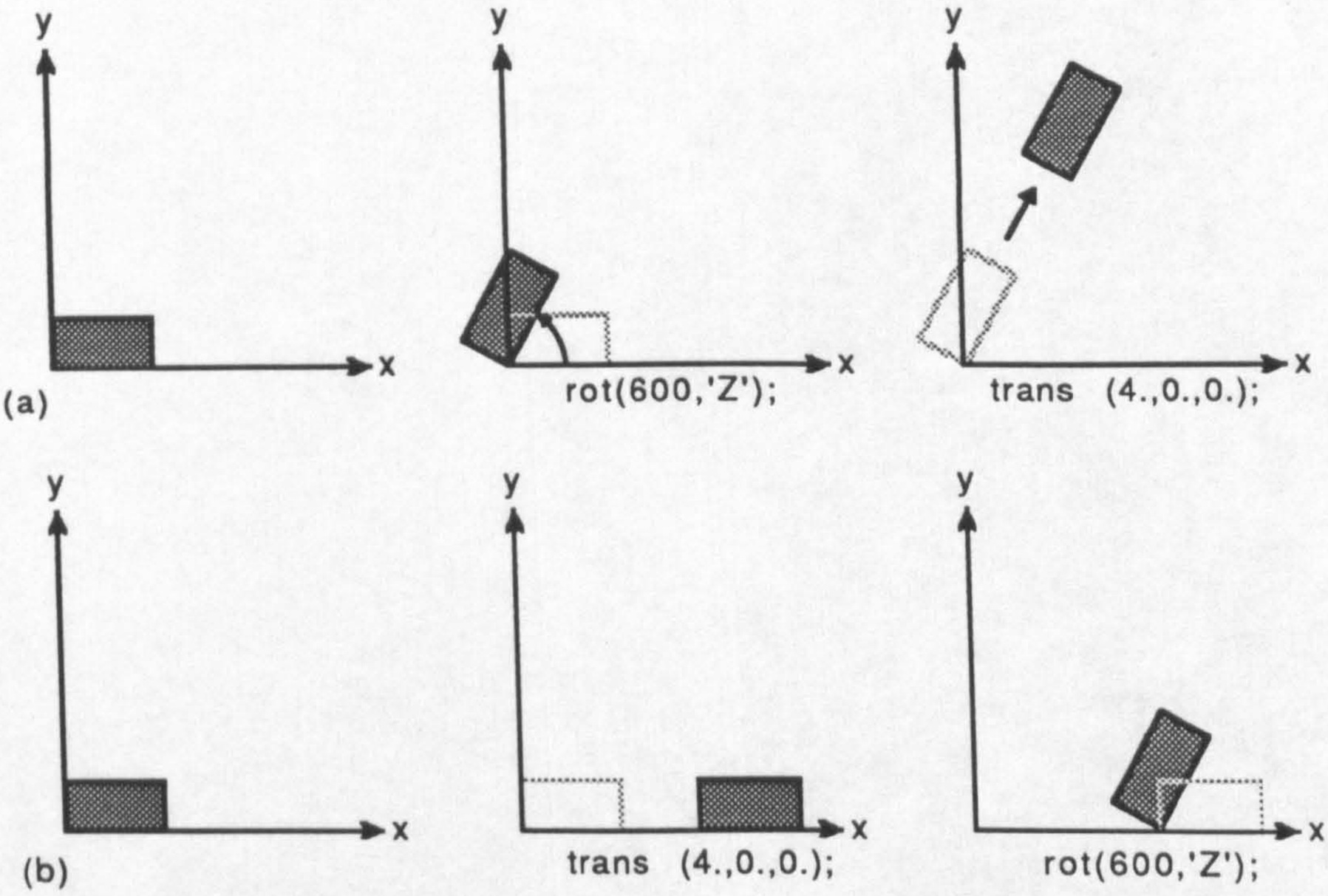


Figure E.3.1. Translate and rotate (from Graphics library programming guide, 7-20 , IRIS 4D Series, Silcon Graphics)

Figure E.3.1a illustrates a rotation of 60 degrees followed by a translation of 4.0 units along the x-axis. Figure B.3.1b shows the same operations but performed in the reverse order.

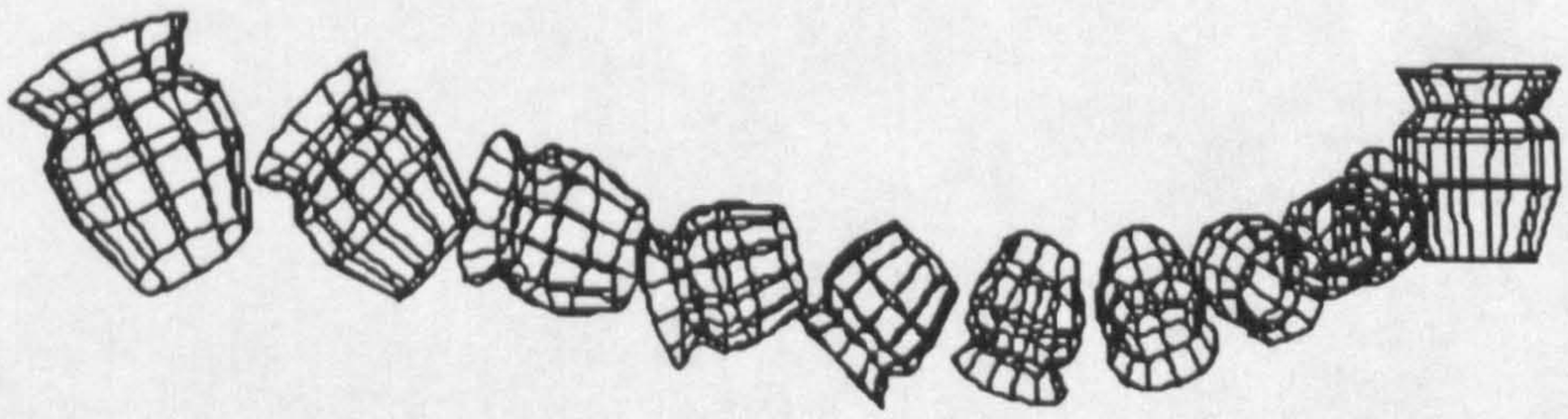


Figure E.3.2. Object referencing.

Figure E.3.2 illustrates each of these transformations radiating from a base model. The corresponding declaration is listed bellow in figure E.3.3.

```
bottle.prim:
        bottle bottle
        5
                5 0
                7 6
                7 10
                5 12
                7 15

bottles.fil:
        FIL
        models/bottle.prim
        0 0 0 1 1 1 0
        refer bottle * 10
        s(.5 .5 .5) o(0 0 30) t(-12 0 0)
        ds(1 1 1) do(36 0 0) dt(-12 0 0)
```

Figure E.3.3. Example of Object referencing.

This facility is useful when defining sequences of repretitive elements such as columns or trees and significantly reduces physicaly demands upon memory (only one description is held). Another future development will allow the same instance data to be referenced N times with each occurance having a different interpreter. This will be useful when rows of objects pass through several peception boundaries (near, middle and far) each requiring a different form of representation (detailed to impresionistic).

E.4. THE CAMERA

The camera primitive is the most important in the system. It provides a viewing port into this virtual reality. Essentailly a camera consists of a lens and a frame. The lens which may be switched interactively between camera->projection (s):

a) perspective

b) orthographic and

c) window

points to a private object list or scene. Objects in the camera's scene are then projected by the current lens routine on to the display film which is essentially a graphics window. A number of viewing parameters may also be set interactively facilitating: camera->viewing of:

a) polar

b) lookat

Other camera attributributes include:

camera->name

camera->eye {ex,ey,ez}

camera->focus {fx, fy, fz}

camera->fov

camera->aspect

camera->twist

camera->clipping {near, far}

camera->window {w_left,w_right, w_bottom, w_top}

camera->azim

camera->dist

Since each instance of the camera contains its own personal object list, this list need not be the same for every camera. Therefore it is possible to view and manipulate objects in isolation or in context simultaneously, figure E.4.1.

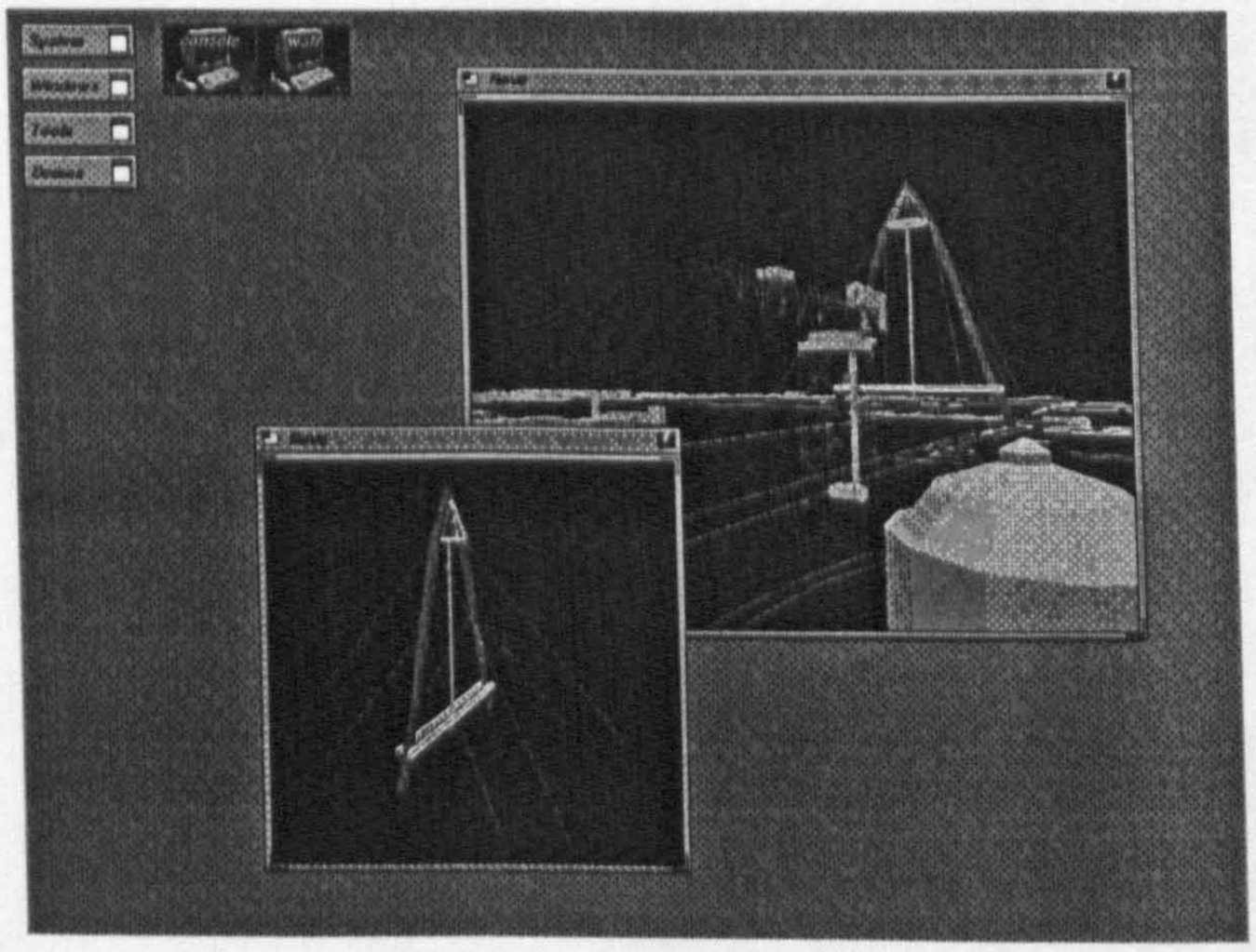


Figure E.4.1. The Glasgow Flourish: in isolation and in context.

E.5. SOFTWARE IC'S AND OBJECTS SCRIPTS

The concept of software IC's was introduced in chapter 4. Here the idea is extended further forming the concept of an introverted software IC (IIC). An IIC in addition to

allowing external access to it's methods, enables control sequences (scripts) to be defined and embeded within it, figure E.5.1.

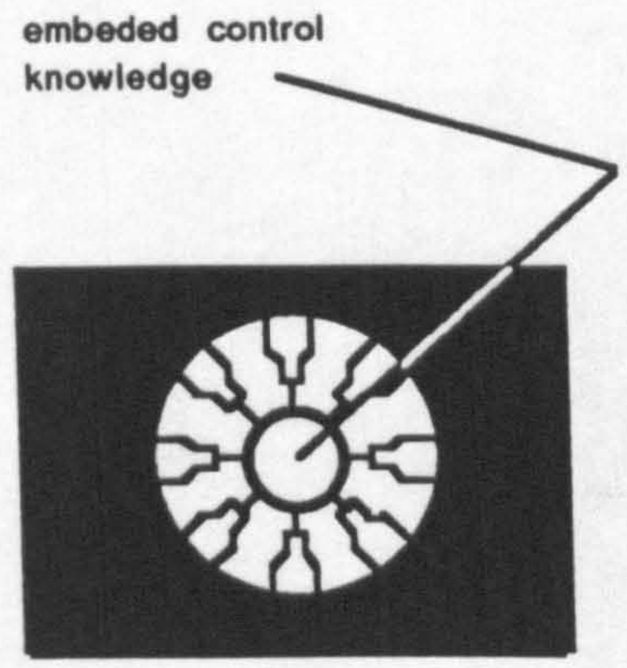


Figure E.5.1. Introverted Software IC

Scripts are defined in a script file using a simple notation, E.5.1, and are compiled at initialisation into a linear list of pointers to object methods. This run time linear list is then played back (simply by running through the list of function pointers) at the dictates of some external source (a knowledge base) modifying data within the object itself, which if monitored by objet interpreters results in realtime motion, articulation, or even growth simulation. The latter form of script may be thought of as a form of software DNA. The following sections describe motion and articulatory scripts. Growth scripts are the subject of further investigation.

E.5.1. OBJECT SCRIPTS

An object script file consists of the (ASCII) definition of one or more objects together with a description of their relative or absolute motion. An object is a collection of bodies. Therefore by defining an object template geometrical bodies are effectively grouped together.

Figure E.5.1.1, below illustrates a single object script. Each rectangular box represents a time step. For each time step a number of methods may be called; these are represented by circles attached to the objects. Many strings of instructions may exist in parallel.

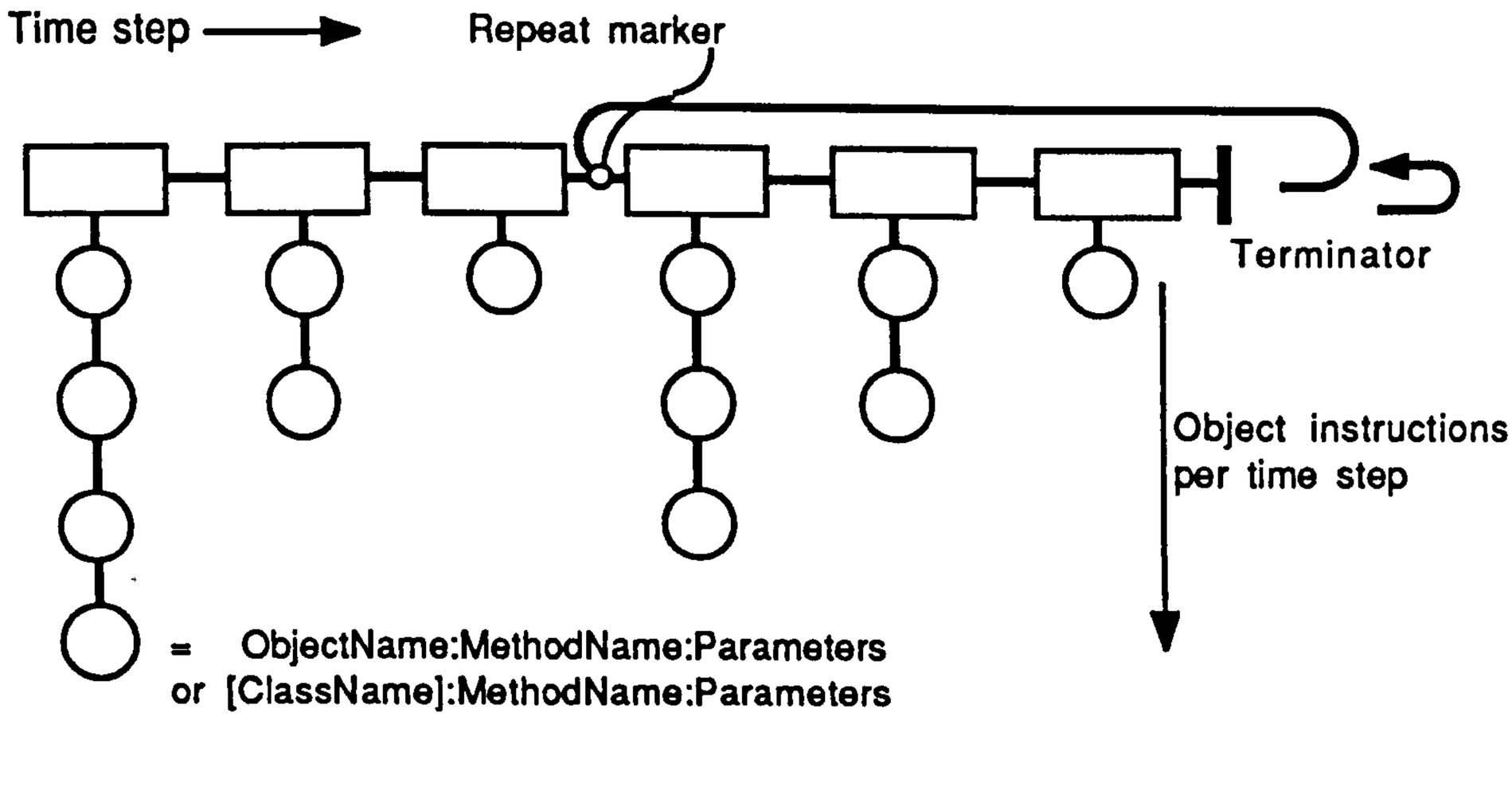


Figure E.5.1.1. Object script

The following example illustrates the basic definition of an object held in a file called a motion script file. All attributes must be typed on a new line and the syntax observed closely. The file must also begin with the keyword "*template*".

```
template
        new object
                name:object_name
                geom:body_1\
                     body_2\
                     body_3\
                     body_4
                script:description
                     {
                          .:method:n n n
                          .:method:n n n
                     }
                     {
                          .:method:n n n
                     }
                          :
                          :
                     {
                          .method:n n n
                     }
                end script
end object
```

E.5.2. REFERENCES TO GEOMETRICAL BODIES.

The model file will contain the physical definition of a number of bodies. Each body is defined by a key word followed by numerical data:

```
GEN
    nvertex nsurfaces
        x y z

        n v1 v2 v3 ...
```

In order to reference a body it must be given a unique tag. This is a single character string placed after the key word describing the type of body.

eg:

```
GEN body_1
```

A pre-processor program called *number* takes as it's arguments a geomtry input and output file, numbering each body and writing a list to stdout which may be re-dirrected into a template file:

```
number model.vewmodel_n.vew >> template
```

The following convention must also be observed during the specification of methods:

```
ObjectName:Method:Parameters
[ClassName]:Method:Parameters
```

For scripts contained within an object, the object name should be omitted or specified as a single full stop '.' (current object). Care must be taken when using this method to mesh the sequence of instructions with other objects. This may be achieved using the "do nothing" fill-in method The object or class name is only used for a externally controled models.

E.5.3. SCRIPT SYNTAX

The following syntax is used to define an object script. As illustrated in figure E.5.1.1 (above) a series of instructions may be specified for each time step. A single time step is defined by enclosing instructions within brackets { }.

```
{
        .:Method 1:parameters
        .:Method 2:parameters
}
{
        .:Method 1:parameters
        .:Method 3:parameters
}
```

The script is played through sequentially from beginning to end. By placing a repeat marker between any of the time steps the script will be repeated in the direction specified from the marker to the end of the script.

```
{
        .:Method 1:parameters
        .:Method 2:parameters
}
<repeat marker>
{
        .:Method 1:parameters
        .:Method 2:parameters
}
{
        .:Method 1:parameters
        .:Method 3:parameters
}
```

The following are recognized repeat instructions:

repeat - play-back from marker

oscillate - oscillate between end and repeat marker

The above will loop for ever. A number of iterations may be specified in square brackets [] after the repeat command:

repeat [5]

oscillate[4]

After the specified number of iterations the script will terminate. Future modifications to the script handler will enable the continuation of the script.

Although motion scripts are held as ascii instructions it is not really the intention to type complex paths. Where the motion of an object may be described by a proven mathematical law a program should be written to generate the script or communicate directly with the object using interprocess control.

E.5.4. METHODS

The following list indicates the current range of methods available together with the expected parameters. These are generic 3D transformations enabling any object to be manipulated regardless of it instance attributes. Three parameters are expected for each method.

translate: tx ty tz	Absolute translation from 0,0,0.
rotate: rx ry rz	Absolute rotation from 0,0,0.
scale:sx sy sz	Absolute scaling from 1 1 1
Colour: r g b	Set the colour of the objects current colour index.
Translate:tx ty tz	Relative translation
Rotate: rx ry rz	Relative rotation
Scale:sx sy sz	relative scaling by increments
Colour: r g b	Relative colour increment/decrement
do nothing: - - -	Does nothing - useful for padding scripts when meshing is an issue

Note that the object origin for translation is its initial position while its rotational origin is at 0 0 0 unless specified otherwise.

In order to load a script the template definition may be included within the model file or more efficiently in a separate file which then must be included in FIL file referencing the model:

```
FIL
model.vew
x y y sx sy sz 0
INCLUDE
model.script
```

E.6. EXAMPLES

In addition to the example given in Chapter 6, the following are illustrations of how the application may be used to represent time variant data.

E.6.1. GLASGOW FLOURISH

The example below illustrates a proposed structure over the River Clyde at Finnieston. The Glasgow Flourish (Scottish Sculpture Trust), standing 205 m above the river, is intended to carry passengers across the river in a ball suspended at the base of a pendulum. In order to represent this feature a mathematical model of a pendulum, simply generating a simple harmonic wave (no damping) was encoded as a C program operating on a Sun workstation. Using the etherlink application transformations are broadcast across the network to the 3D viewing and manipulation program operating on a Silcon Graphics Iris. The composite application (shown as a montage) is illustrated in figure E.6.1.1.

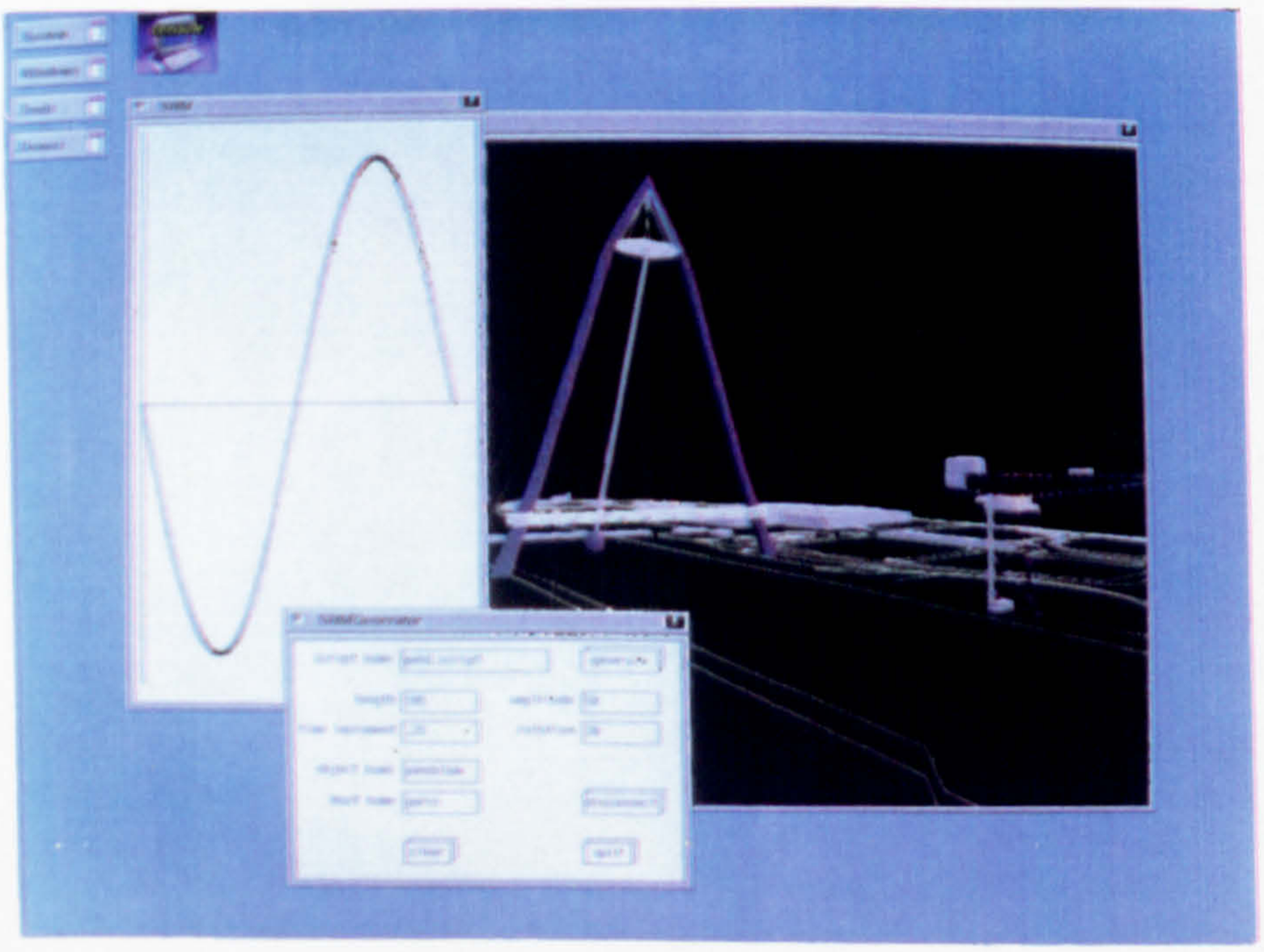


Figure E.6.1.1. Glasgow Flourish and interface.

The geometrical representation of that model is then translated accordingly. In order to fine tune the SHM script generated, a forms interfaces was bolted onto the script generator enabling the length, amplitude and time increment to be adjusted. Since the model lies in a plane at 30 degrees of north a third, orientation, parameter is used to rotate the script so that the pendulum swings normal to the banks of the river. Other attributes controlling the inter-client communication are also included within the proforma template. In addition to broadcasting messages to the application a script file is also created enabling the articulation commands to be encapsulated within the model itself once refined.

E.6.2. NEWTON'S CRADLE

Using the same principles as above a sinusoidal motion script (split at π) is assigned to the outer most spheres of a Newtons cradle, figure E.6.2.1. Although not a true representation of the principles of the conservation of momentum, an accurate model could be implemented to achieve a real time simulation of the colliding spheres.

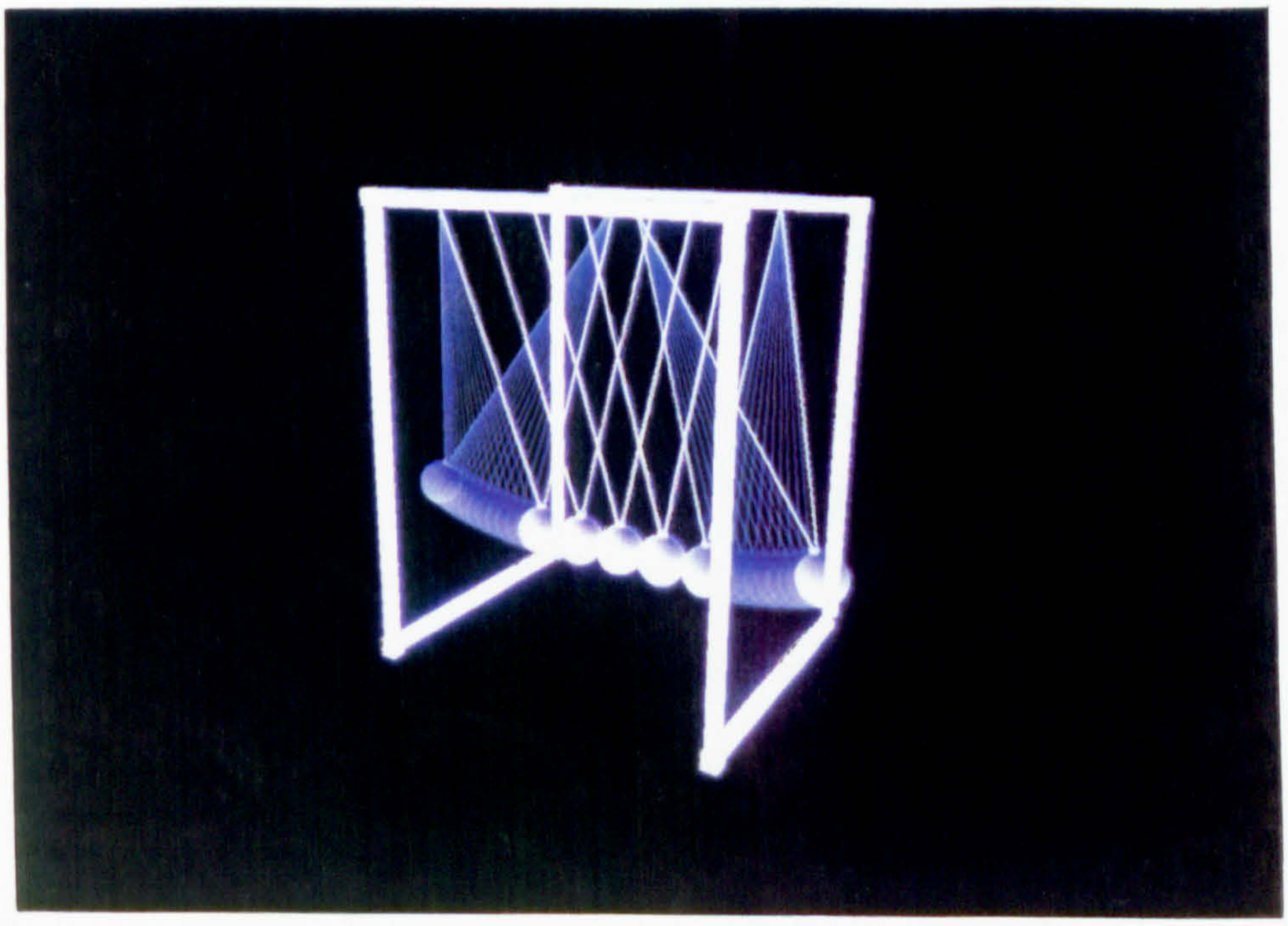


Figure E.6.2.1. Newton's Cradle.

The user would also be able to pick and pull back one or more spheres setting the model in motion.

E.6.3. INTERACTIVE 3D MODELS

The example of a chess set, figure E.6.3.1, illustrates the ability to perform multiple operations within a single time step. The complete model is constructed by referencing only three objects (a tile, a pawn and the queen). Each reference is given a unique identifier and translation and may therefore be manipulated independently.

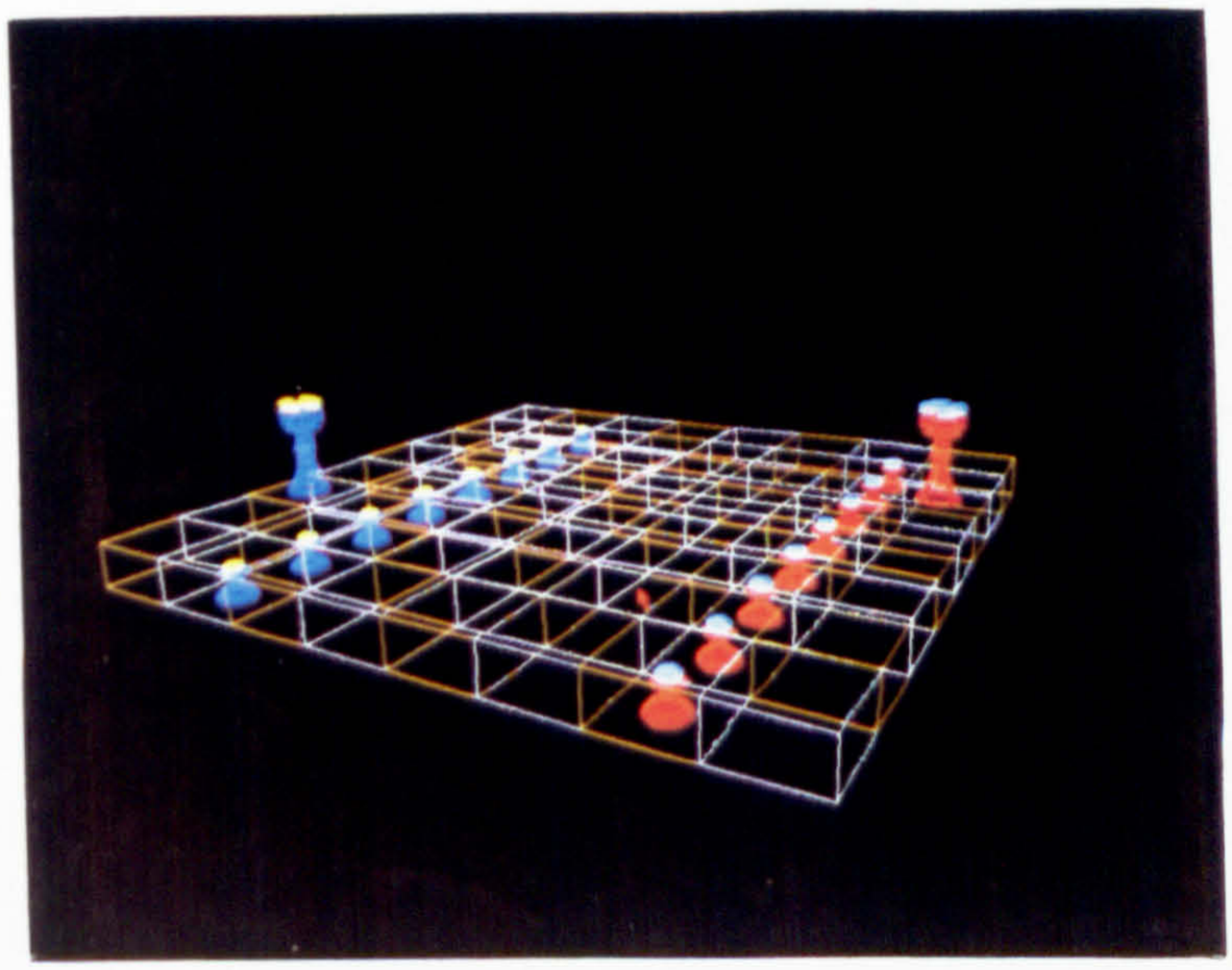


Figure E.6.3.1. Wireline model of chess set - note the pawns are squashed to fit within the board.

Here the pawns are scaled so that they fit within the depth of the chess board, figure E.6.3.1 and 2. At the beginning of a new game all the pieces are scaled and translated to their original size and position,figure E.6.3.3 rising out of the board to defend their queen. Although not complete the model would enable the user to interact directly with individual pieces which would format natural language utterances and would then be interpreted by a knowledge base containing valid and counter moves.

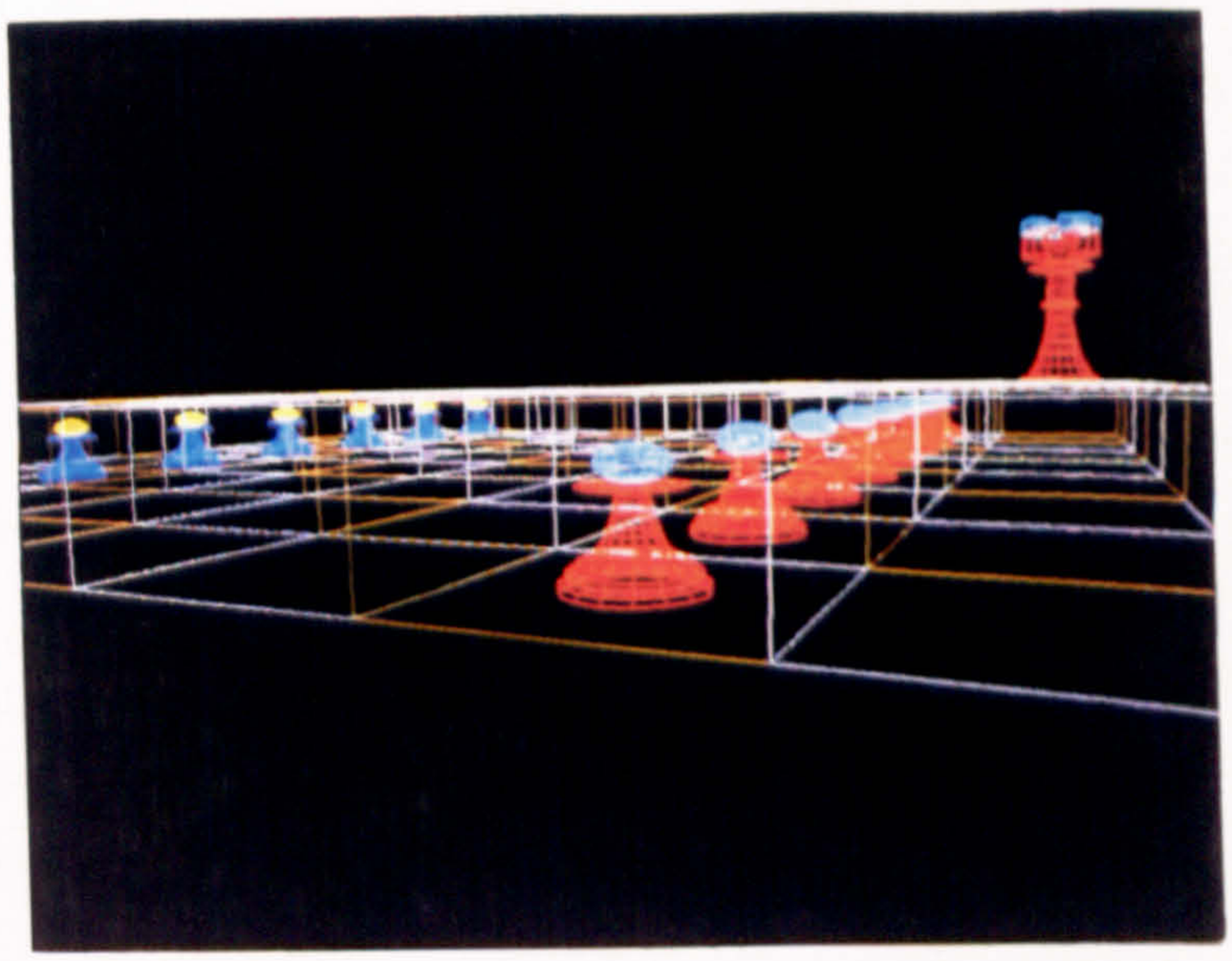


Figure E.6.3.2. Initial model state.

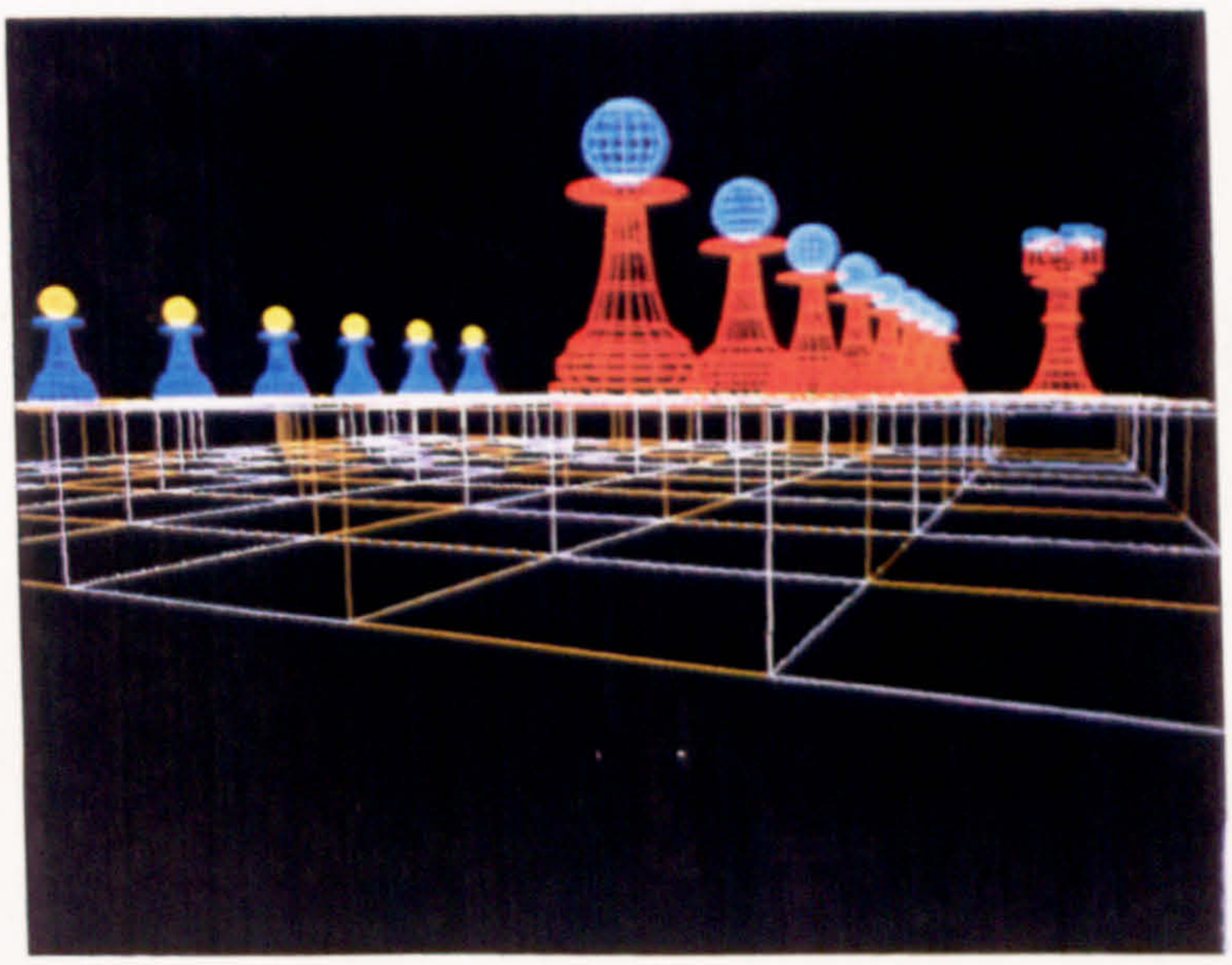
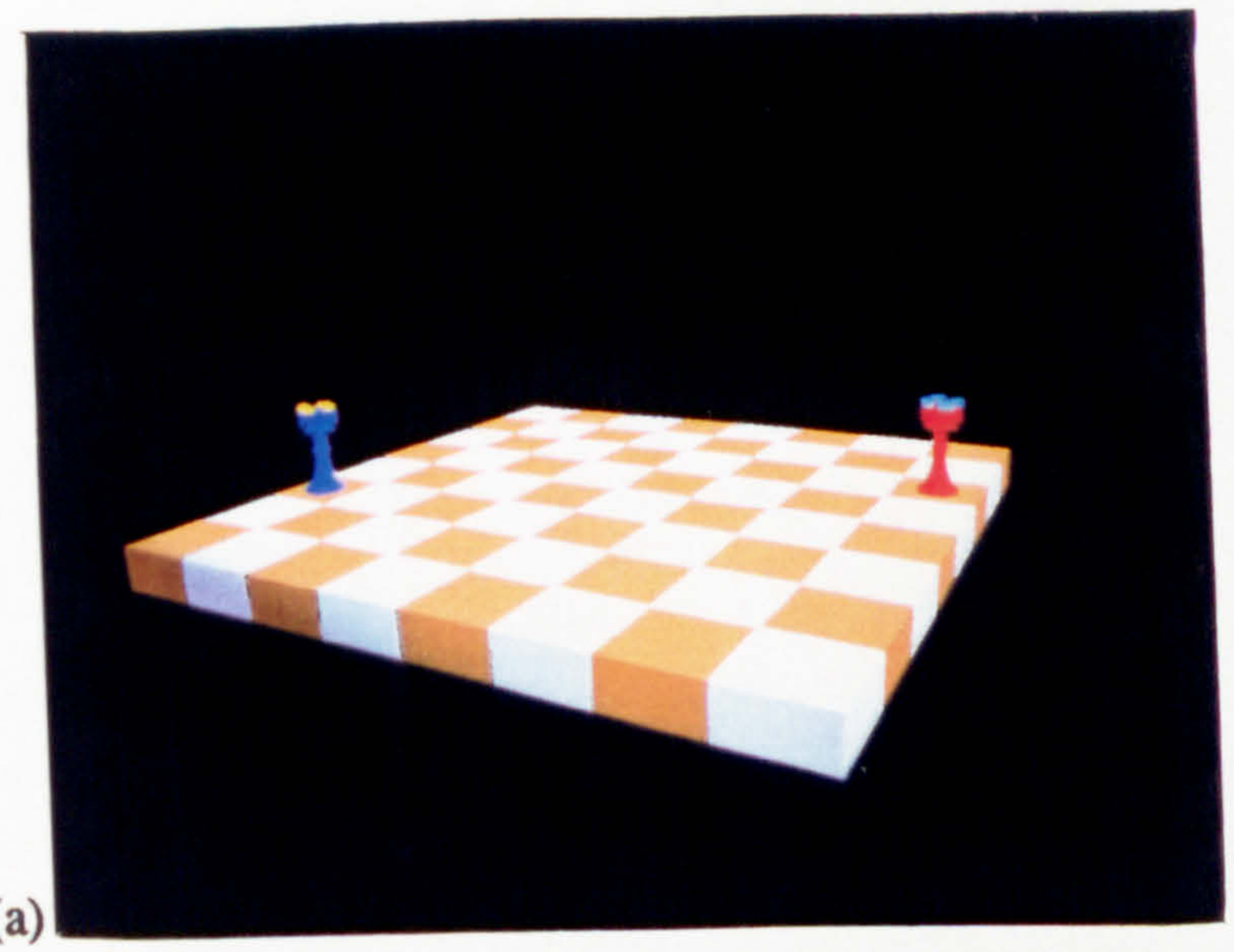
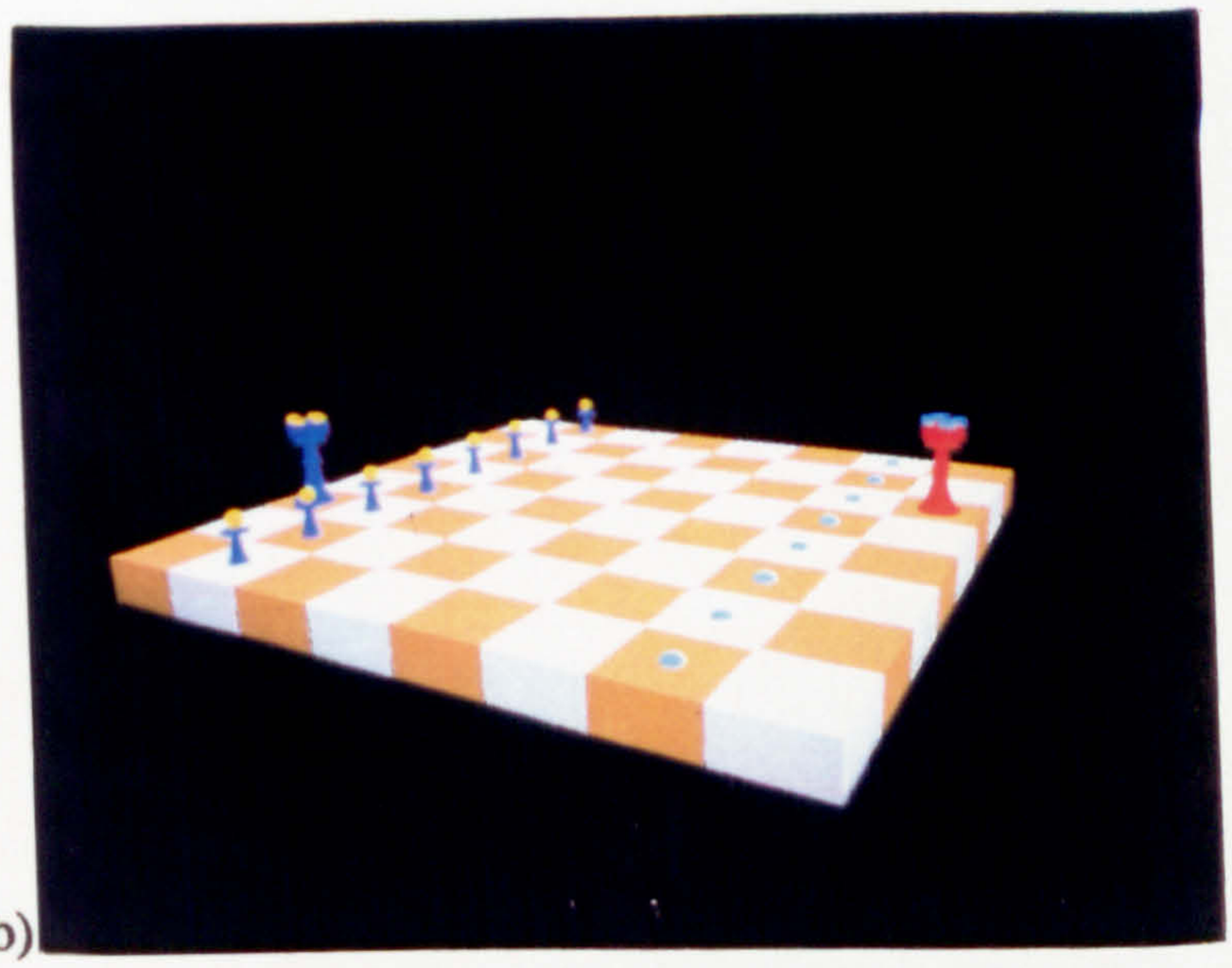


Figure E.6.3.2. Final model state.

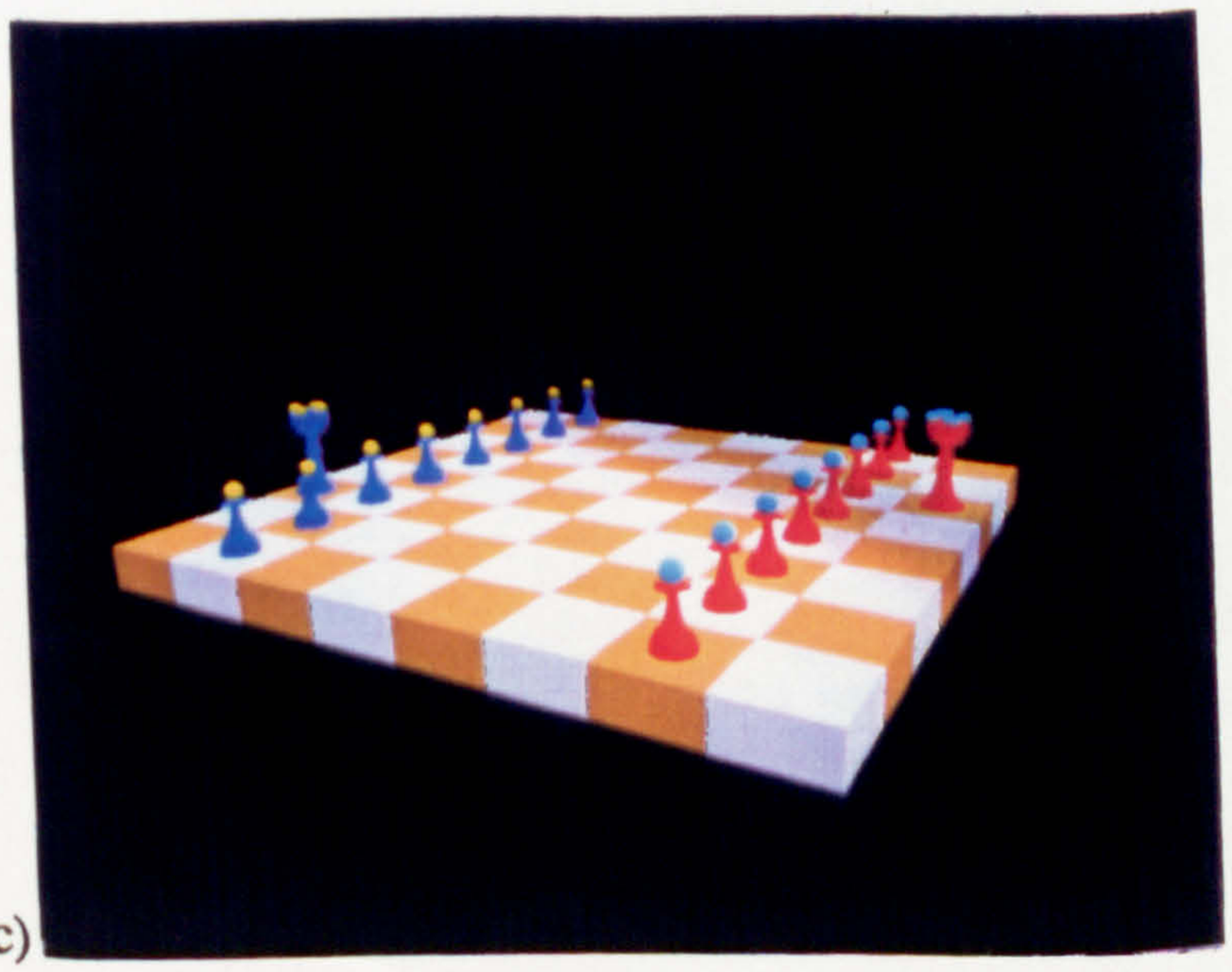
Figures E.6.3.4a - c illustrate the intermediate scaling and translation. The user would not be confined to any one viewpoint but could either select any viewpoint or be shown the board from the current piece, figure E.6.3.5, adding a new dimension to the game.



(a)



(b)



(c)

Figure E.6.3.4. a-c. Intermediate scaling.

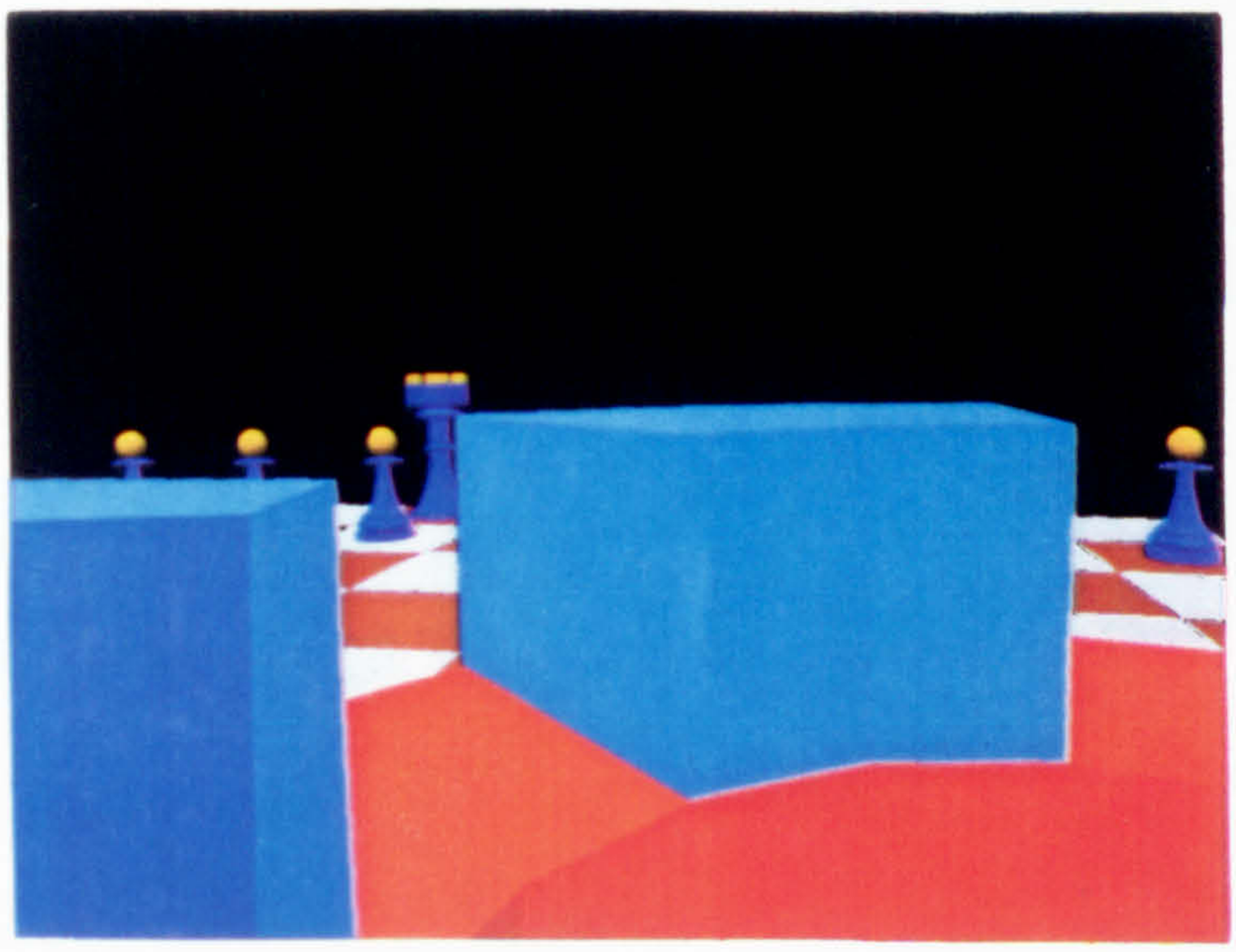
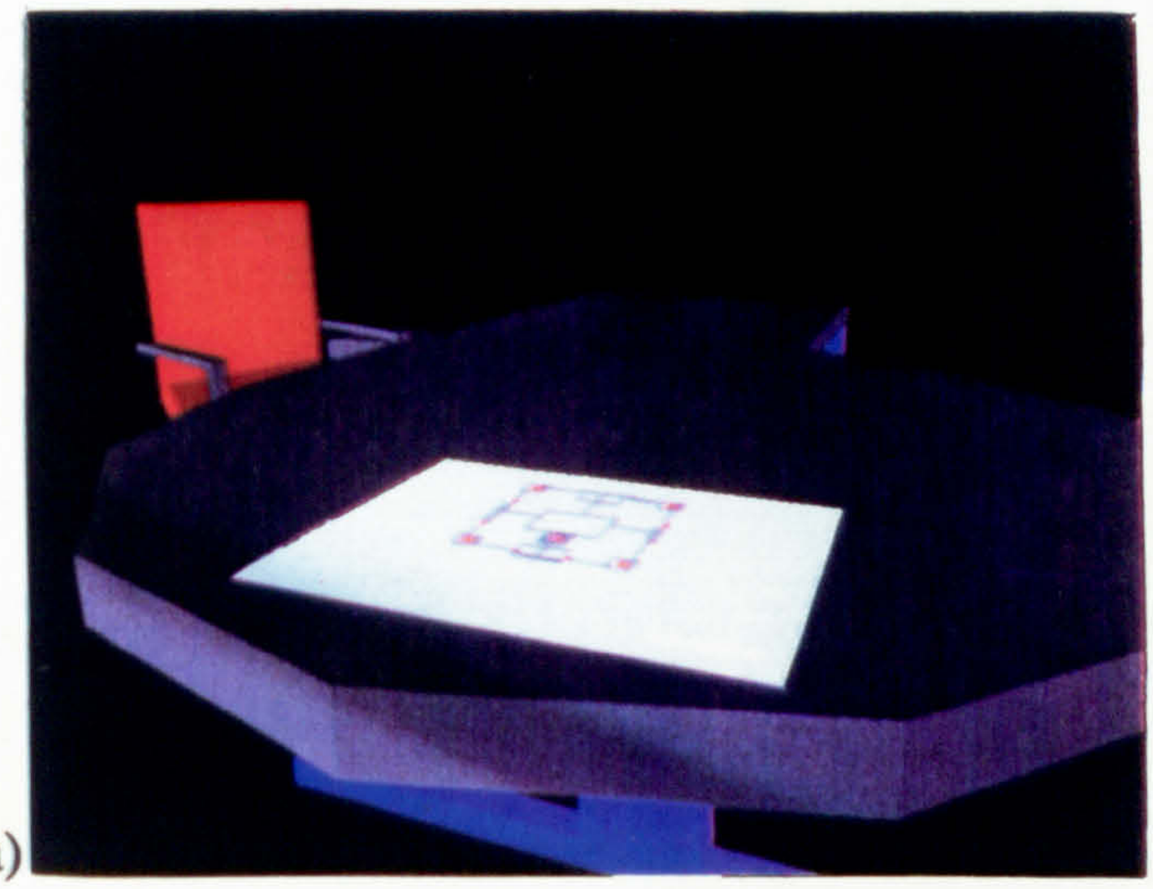


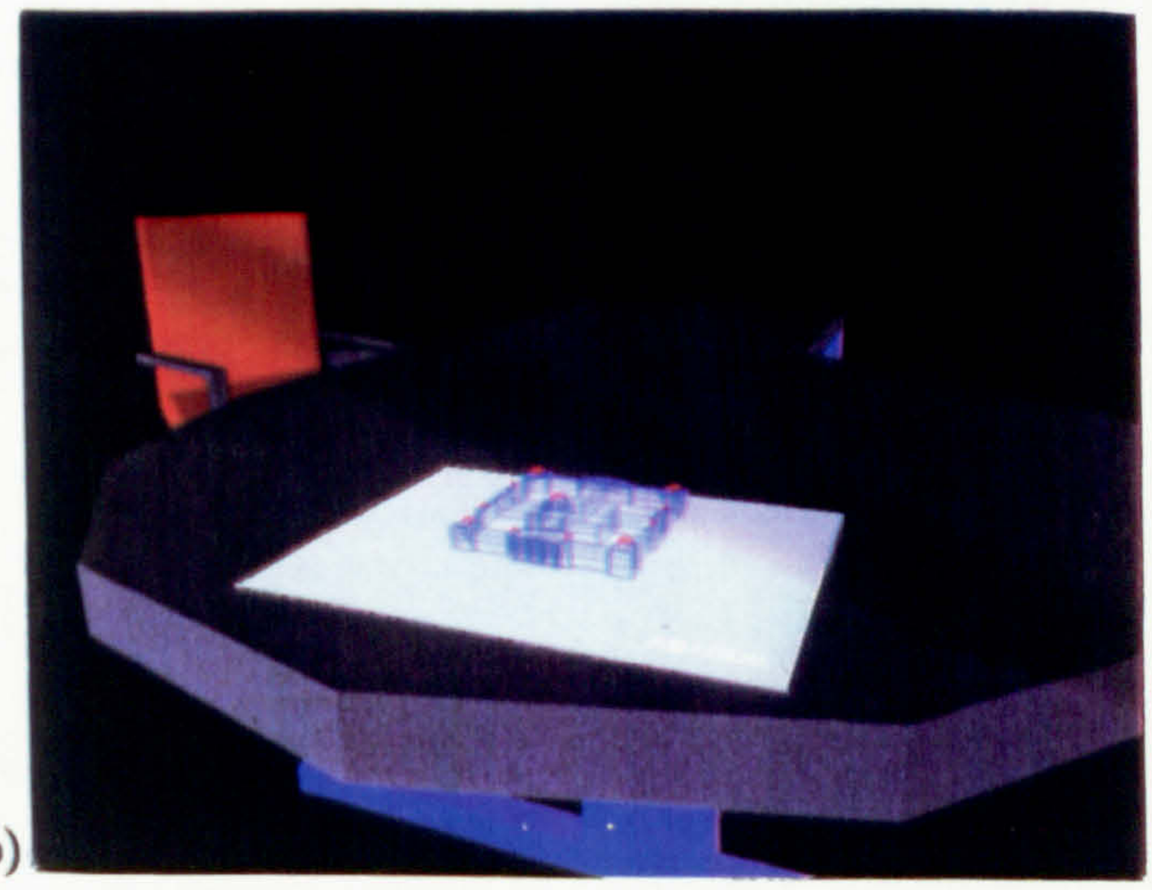
Figure E.6.3.5. Alternative views of the same problem space.

The image frames, figures E.6.3.6a-c illustrate a wireline model of Glasgow City Chambers being extruded out of a drawing placed on a desktop.

(a)



(b)



(c)

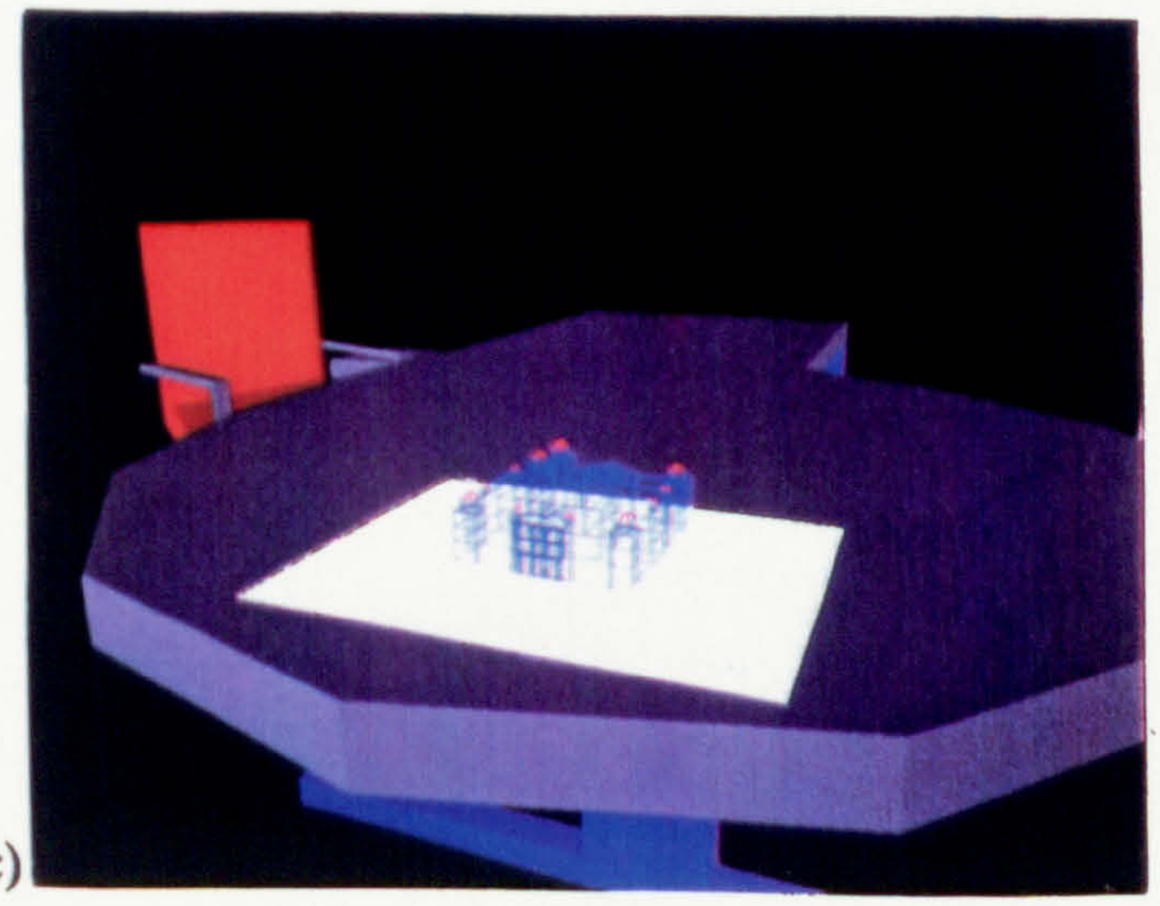


Figure E.6.3.6.a-c. Simple Scaling. (note also the change in the colour of the drawing title).

These principles would also be useful in orchestrating or choreographing video sequences but, rather than having to develop a specific application to articulate bodies, various 3D representations may be manipulated by a general mathematical (deep) model of movement and high-level (abstract) messages broadcast.

The 3D viewing and manipulation program is (in its present state) sufficiently general and extendible to enable other applications to be constructed.

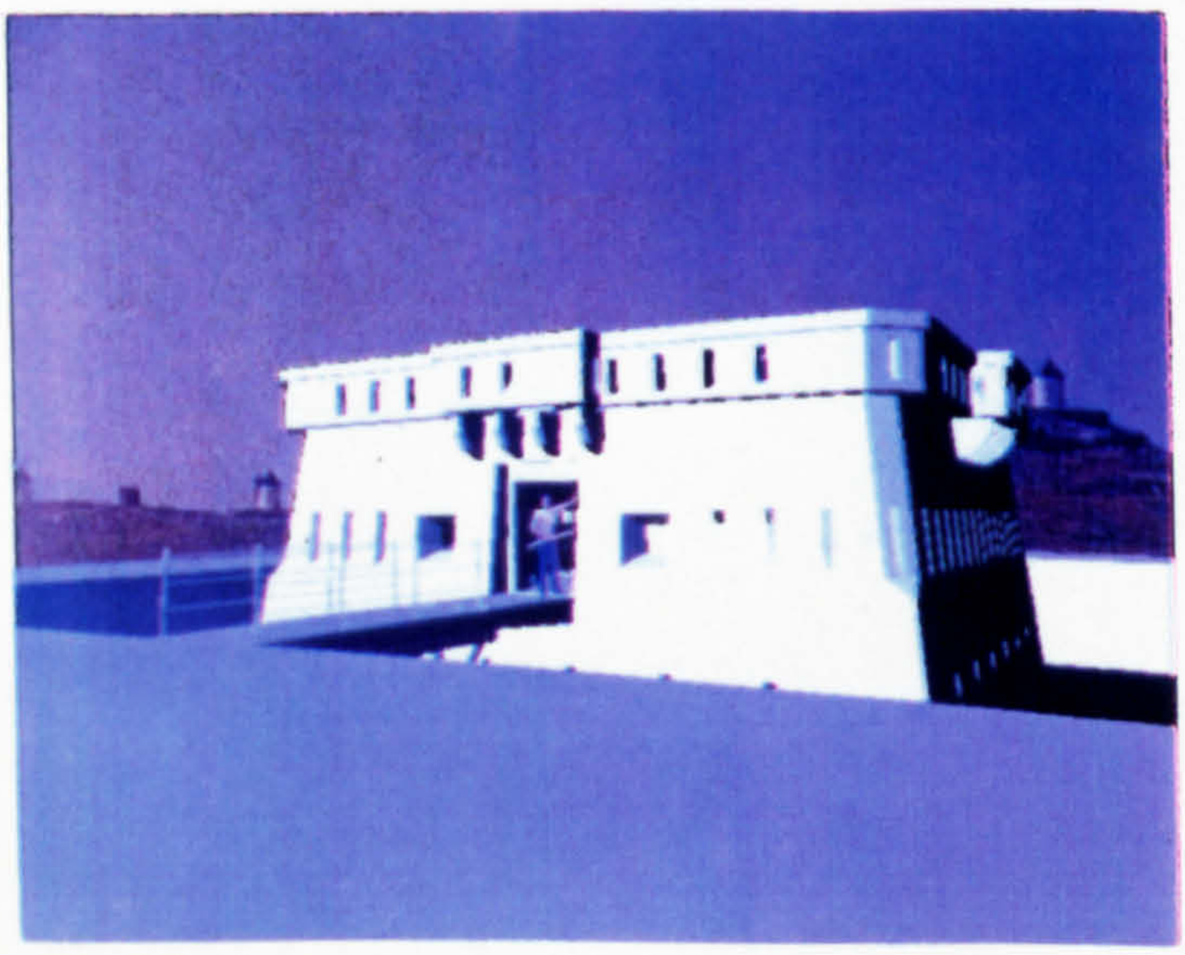


Figure E.6.3.6. Virtual reality - a user's perception of him/herself as an interacting part of a larger system. The image shows a 3D model of a 19th century Napoleonic fort in front of a frame-grabbed image with the author positioned in the doorway.

By extending the range of primitives to include raster images, for instance, realistic contextual backdrops may be incorporated creating an experiential virtual reality within which the user perceives himself as an interacting part of a larger system, figure E.6.3.6, improving the communication of concepts and ideas.

The models featured in the above illustrations have been assembled from existing models created by students and staff at ABACUS. Motion or articulation scripts have been developed by the author to illustrate specific applications. Acknowledgements are given for the following models:

Newton's cradle	Morag Boyd, advanced computer graphics course, department of architecture.
Chess set	Unknown.
Table and chair	Andrew Anderson and Mike Grant, ABACUS.
Glasgow model	Various.

All other components and models are by the author.

REFERENCES/BIBLIOGRAPHY

[ALTY 83] Alty J.L. (1983).
Path Algebras - a useful CAI/CAL Analysis technique.
Man-Machine Interface Group, Department of Computer Science, University of Bristol, April 1983.

[ALTY 83] Alty, J.A., Coombs, M.J. (1983).
Computational Approaches to Representation and Control:The representation of domain knowledge.
In:J.L. Alty and M.J. Coombs, Expert Systems, Concepts and Examples, NCC Publications, Chpt3.3 pp. 60-76

[ALTY 84] Alty J.L. (1984).
Use of Path Algebras in an Interactive Adaptive Dialogue System
Proc. Alvey IKBS Research Theme workshop, University of Sussex, UK, 10-11 July 1984

[APPELT 82] Appelt, D.E. (1982).
Planning natural-language utterances
SRI International, Menlo Park, California,
In. AAAI 82.

[ARISTOTLE] Aristotle
From the W.D. Ross edition, 12 Volumes. Oxford, 1908-52.
In Therories of the symbol, Tzvetan Todorov, Cpht 1.: The birth of Western Semiotics.

[ASIF 89] Asif M., and Horner R.M.W. (19989).
Economical Construction Design Using Simple Cost Models.
Proc. Int. Conf. on Structural Faults and Reparir Vol.2, London, June 1989

[AVRAHAMI 89] Avarahami Gideon, Brooks P. K. & Brown M.H. (1989).
A two view approach to constructing user-interfaces.
DEC Systems Research Centre, Computer Graphics, Vol. 23, No. 3, July 1989.

[BAECKER 87] Baecker R.M., Buxton W.A.S. (c1987).
Readings in human-computer interaction : a multidisciplinary approach
Los Altos, CA : Morgan Kaufmann.

[BARKER 89] Barker, Philip G. (1989).
Basic principles of human-computer interface design
London: Hutchinson.

[BENNING 73] Benning W. (1973).
The logic of Scientific Discovery
In: W.J. Horst, Rittel and M.M. Webber (eds) Dilemas in general theory of planning,
Policy Sciences.

[BIG BOOK] Big Book on User Interfaces
Programming techniques and tools
In: The BIG BOOK on User interfaces, Chapter 12, pp. 555-558.

[BIRMINGHAM 88] Birmingham W., Brennan A., Gupta A.P., and Siewiorek D.P. (1988),
MICON: A single board computer synthesis tool
IEEE Circuits and Devices Magazine (January 1988).

[BIRMINGHAM 89] Birmingham W.P., Kapoor A., Siewiorek D.P., and Vidovic N. (1989).
The design of an integrated environment for the automated synthesis of small computer systems
EECS University of Michegan and ECE, Carnegie Mellon University.

[BJORK 89] Bjork B.C. (1989).
Basic Structure of a proposed building product model
CAD Vol. 21, No. 2, March 1989.

[BOIES 74] Boies S.J. (1974).
User Behavior in an interactive computer system.
IBM Systems Journal, 13, pp. 1-18.

[BOULAY 82] Boulay, Ben du. (1982).
Modelling.
Alvey IKBS Theme:Intelligent Front Ends, p. 39.

[BRIDGES 89] Bridges A. and Rutherford J.H. (1989).
Informal discussion
ABACUS computer room (in front of persi).

[BUCHANAN 89] Buchanan J.T. (et al) (1989).
Distributed Asynchronous Hierarchical Problem Architecture Applied to Plant Scheduling.
AI in Engineering, Springer Verlag, July 1989.

[BUNDY 82] Bundy, A. (1982).
Frames and Logic.
Alvey IKBS Theme: Inference, pp. 4-6.

[BUNDY 84] Bundy A. (1984).
An Architecture for Intelligent Front Ends.
Proc. Alvey IKBS Resarch Theme Workshop
University of Sussex, UK, 10-11 July, 1984.

[BUXTON 89] Buxton W., Lamb M.R., Sherman D., and Smith K.C. (1989).
Towards a comprehensive user interface management system
Computer Systems Research Group, University of Toronto, Canada M56 1A4,
In: [BARKER 89], pp. 576-583.

[CARLSON 89] Carlson P.A. (Rose Hulman Institute of Technology, Indiana) (1989).
Hypertext and Intelligent Interfaces for text retrieval
In: The Society of text: hypertext, hypermedia and the social construction of information.
Cambridge M.A. MIT Press. pp. 59-76.

[CARRARA 88] Carrara G., Kalay Y.E., Novembri G. (1988).
A computational framework for supporting creative architectural design
La Sapeinza University, Rome, Italy.

[CHIGNELL 89] Chignell M.H., Hancock P.P. (University of South California) (1989).
An introduction to Intelligent interfaces.
In: Intelligent Interfaces: theory, research and design.

[CHIN 87] Chin, D.N. (1989).
Modelling what the user knows in UC
In:User Models in dialogue systems, Symbolic Computation, Springer Verlag, chpt4, pp. 74-107.

[CLARKE 86] Clarke J., McLean D. (1986).
ESP: A Building and Plant Energy Simulation System
Reference manual 5.3

[CLOWES 87] Clowes I.
Requirements for User Interface Management Systems.
Logica Cambridge Limited. 15/9/87.

[DENNIS 80] Dennis J.B. (1980).
Data-flow supercomputers
Computer 13(11), pp. 48-56.

[DIGITAL 88] Digital Technology LTD (1988).
FormsDesigner Reference Manual
Digital Technology Ltd, 10 Victory Business Centre, Worton Road, Islewort, Middlesex, TW7 6DB.

[DODSON 87] Dodson D.C. (1987).
A short report on the second meeting of the intelligent interface.
SIG. Department of Computer Science, City University, London.

[DREYFUS 85] Dreyfus S.E. (1985).
The nature of expertise
IJCAI 85, Vol 2, 1306.

[EDMONDS 81] Edmonds E. A., Gains R.D. (1981).
Adaptive Man-Computer Interfaces.
In: M.J. Coombs & J.L. Alty (eds), Computing Skills and the User-interface

[EDMONDS 82] Edmonds, E. (1982).
A note on IFE's.
Alvey IKBS Theme:Intelligent Front Ends, p..6.

[ELKERTON 89] Elkerton J., Williges R.C. (University of Michegan) (1989).
Dialogue Design for Intelligent Interfaces.
In: Intelligent Interfaces:theory, research and design.

[ELLIS 80] Ellis C.A. and Nutt G.L. (1980).
Office information systems and computer science
ACM Computing Surveys, 12(1), pp. 27-60.

[ENCYCLOPEA 74] Encyclopedia Britanica Inc. (1974).
Micro Media
Vol 3, page 45 Encyclopedia Britanica 15th Edition.

[ENGLEMORE 88] Englemore R.S. and Morgan A.J. (eds) (1988).
Blackboard Systems
(Insight series in artifical intelligence): Adison Wesley.

[ENSOR 88] Ensor J.R. and Gabbe J.D. (1985).
Transactional blackboards
In: Proceedings of the Ninth International Joint Conference on Artifical Intelligence (IJCAI-79) reproduced in chapter 24, [ENGLEMORE 88].

[ERMAN 88] Erman L.D., Hayes-Roth F., Lesser V.R. and Reddy D.R. (1988).
The Hearsay II speech-understanding system: Integrating knowledge to resolve uncertainty.
In: R. Englemore and T. Morgan (eds) Blackboard Systems; Adison Wesley, pp. 31-86.

[FEIGENBAUM 89] Feigenbaum E.A. (1988).
Knowledge assembly
In: Blackboard Systems, Englemore R.S. and Morgan A.J. (eds), pp 5-8, Adison Wesley.

[FOLEY **] Foley J.D., Van Dam A. (19)
The Design of User-Computer Graphic Conversations.
Fundamentals of Interactive Computer Graphics, Chpt. 6.
Foley and Van Dam

[FROHLICH 86] Frohlich D.M., Crossfield L. P. & Gilbert G.N. (1986).
Requirements for an Intelligent form-filling interface.
Alvey DHSS Demonstrator, Department of Sociology, University of Surrey.

[GEISOW 89] Geisow A.D. (1989).
Recognition and Generation of Symbolic Diagrams.
In: Geometric Reasoning, IBM UK Scientific Centre, chpt 6.1.
Oxford University Press.

[GERO 86] Geor J.S., and Coyne R.D. (1986).
Semantics and the organisation of knowledge in design
Computer Applications Research Unit, Department of Architectural Science, The University of Sydney, NSW 2006 Australia.

[GUEDJ 80] Guedj R.A. (1980).
Remarks on some aspects of man-machine interaction
Methodology of interaction, North-Holland Publishing Company, pp.235-238.

[GUPTA 90] Gupta A.P., Sudhalkar A., Vidovic N., Siewiorek D.P., and Prinz F. (1990).
Concurrent engineering of computer systems
Engineering Design Research Centre, Carnegie Mellon University.

[HAEBERLI 88] Haeberli P.E. (1988)
ConMan: A visual programming language for interactive graphics
Silicon Graphics, Inc. Mountain View, CA 94043,
In: Computer Graphics, Vol. 22 No. 4, August 4, 1988, pp. 103-110.

[HAENEN 87] Haenen P. (1987).
User Interface Management Systems
Laboratoire d'Etudes Methodologiques Architecturales.

[HAMALAINEN 88] Hamalainen M. (1988).
An approach to developing intelligent interfaces to conventional software.
In: STeP 88,Vol.2 p765-775. Finnish AI Symposium.
August.

[HANCOCK 89] Hancock P.A., Chignell M.H. (1988).
Intelligent Interfaces: theory research and design
ISBN:0 44487313 9

[HARRISON 85] Harrison M.D., and Thimbleby H.W. (1985).
Formalising guidlines for the design of interactive systems
In: P. Johnson and S. Cook (eds), People and Computers: Designing the interface (HCI-85), Cambridge University Press, p. 161-171.

[HART 82] Hart, P. (1982).
Directions for AI in the Eighties.
SIGART Newsletter. 79, pp. 11-16.

[HASHIMSHONI 78] Hashimshoni R., ShavivE., and Wachman A. (1978).
A decomposition model as a tool for evaluation of multicell solutions: A case study
In: Proceedings of CAD 78Third International Conference on Computers in Engineering and Building Design, IPC Science and Technology Press, Guildford.

[HAYES-ROTH 85] Hayes-Roth B. (1985).
A blackboard architecture for control
Artifical Intelligence Vol. 26, pp. 251-321.

[HENDERSON 86] Henderson A. (1986).
The Trillium User Interface Design Environment
Proc. CHI 1986, pp. 221-227.

[HILL 86] Hill R. (1986).
Supporting concurency, communication, and sychronisation in Human-computer interaction - The Sassafras UIMS
ACM Transactions on Graphics, 5(3), pp. 179-210.

[HMSO 78] HMSO (1978)
An introduction to dimensional co-ordination
London: Her Majesty's Stationery Office

[HOPKINS 89] Hopkins T. (1989).
Introduction to Object-Oriented Programming
EASE Education and Awareness Seminar; Object Oriented Programming Systems
Department of Computer Science, University of Manchester
18th September 1989.

[HUTCHINGS 86] Hutchings A.M.J. (ed) Bowen D.L., Byrd L., Chung P.W.H., Pereira F.C.N., Pereira L.m>, Rae R. and Warren D.H.D. (1986).
Edinburgh Prolog (The New Implementation)
User's Manual Version 1.5, AI Applications Institute, University of Edinburgh.

[IFE 88a] Clarke, J.A, Rutherford J.H., MacRandal D. (1988).

[IFE 88b] Clarke, J.A, Rutherford J.H., MacRandal D. (1988).

[IFE 89] Clarke J.A., MacRandal D., Rutherford J.H. (1989).
The Application of Intelligent Knowledge Based Systems in Building Design.
SERC Grant GR/E/18018 Final Report, November 1989.

[JENKIN 82] Jenkin J.M.
Some principles of screen design and software for their support
Computers in Education, 6, pp. 25-31.

[JEON 87] Jeon, Y.I. (1987).
Working Memory Aid For Designers (WOMAD) A design Experience Supprot System.
PhD thesis, Department of Architecture,University of Strathclyde, Glasgow.

[KAEMMERER 86] Kaemmerer W., Larson J. (1986).
A graph-oriented knowledge representation and unification technique for automatically selecting and invoking software functions
AAAI-86, Philadelphia, Pennsylvania, Reproduced in [HAMALAINEN 88].

[KASIK 82] Kasik D.J. (1982).
A User Interface Management System.
Computer Graphics 16(3), pp. 99-106.

[KOBSA 89] Kobsa A. and Wahlester W. (eds)(1989).
User Models in Dialogue Systems
Symbolic Computation: Springer Verlag.

[KRIPPENDORFF 89] Krippendorff K. (1989).
On the Essential Contexts of Artifacts or on the Proposition that "Design Is Making Sense (of things)"
Design Issues: Vol. V, Number 2, Spring 1989.

[LANSDOWN 86] Lansdown J. (1986).
Requirements for knowledge based systems in design
In: A. Pipes (ed), CAAD Futures, Proceedings of International Conference on CAAD, Spetember 1985, London: Butterwoths.

[LAWSON 80] Lawson B.R. (1980).
How designers think
London: Architectural Press.

[LEHMAN 88] Lehman J.F. and Carbonell J.G. (1988).
Learning the user's language: A step towards automated creation of user models.
In [KOBSA 89] pp. 163-194.

[LESSER 78] Lesser V.R. and Corkill D.D. (1978).
Cooperative distributed problem solving: A new approach for structuring distributed systems
Technical Report 78-79, Department of Computer and Information Science, University of Massachusetts at Amherst.

[LEWIS 69] Lewis B.N., and Cook J.A. (1969).
Toward a theory of telling.
International Journal of Man-Machine Studies (1), pp. 129-176; cited in [GUEDJ 80].

[LOD 84] Little Oxford Dictionary (1984).
page 606.

[MAANEN 89] Maanen J. Van and Mead M. (1989).
The STEP standard for engineering data: Application protocols
News Letter, Informatics Divsion, Rutherford Appleton Laboratory, Chilton Didcot.

[MacCALLUM 85] MacCallum K.J., Duffy A. & Green S. (1985).
An Intelligent Concept Design Assistant
IFIP Conference on Design Theory for CAD, Tokyo, October 1985.

[MARKUS 72] Markus T.A., Maver T.W., Whyman P., Morgan J., Whitton D., Canter D., and Fleming J. (1972).
Building performance.
New York: Halsted Press. 1972.

[MARTIN 73] Martin J. (1973).
The design of Man-Computer Dialogues.
Englrwood Cliffs, N.J.: Prentice Hall.

[MARTIN 87] Martin. M. M. (1987).
Foundations of a Toolkit.
Eurographics Workshop, August 1987.

[MAVER 90] Maver T.W., Rutherford J.H., MacCallum K.J., MacCleod I.A. (1990).
Intelligent Design Assistant:A Collaborative IT Demonstrator Project in the Domain of Building Design.
SERC Grant Application, February 1990.

[MCCALLUM 85] McCallum K.J., Duffy A. and Green S. (1985).
An Intelligent Concept Design Assistant.
IFIP Conference on Design theory for CAD, Tokyo, October 1985.

[MCKEOWN 85] McKeown K.R. and Paris C.L. (1985).
Text generation: using discourse strategies and focus constraints to generate natural language text
Cambridge Univ. Press.

[MCLEOD 88] MacLeod I.A. and Rafiq M. (1988).
Integrated Computer Aided Structural Design of Buildings.
Structural Engineer, Vol. 66, No.20, October 1988.

[MEYER 88] Meyer, Bertrand. (1988).
Object-oriented software construction
Englewood Cliffs, N.J. : Prentice-Hall.

[MILLER 56] Miller G. (1956).
The Magical Number Seven, Plus or Minus Two
Pyschological Review, 63(1956):81-97
Cited In: H.A. Simon, The Sciences of the Artificial pp. 81. and
I. Sommerville, Software Engineering, pp. 230-231.

[MILLER 76] Miller L.A. and Thomas J.C. jr. (1976).
Behavioural issues in the use of interactive systems
In:Interactive Systems, Proc, 6th Informatik Symposium IBM Germany, Bad HomBurg v. d. H., September 1976.

[MORALEE 83] Moralee S. (1983).
Intelligent Front Ends.
Proc. Alvey IKBS Research Theme Workshop
Cosnors House, Abingdon, England, 26-27 September 1983.

[MORELAND 83] Morland D.V. (1983).
Guidelines for technical interface design
Communications of the ACM, 26(7), pp. 484-94.

[MORTON 79] Morton J., Barnard P.J., Hammond N. and Long J. (1979).
Interacting with the computer: a framework
In E.J. Boutmy and A. Denthe (Eds.) Teleinformatics 79. Netherlands: Noth Holland.

[MYERS 86] Myers B., Buxton W. (1986).
Creating highly interactive and graphical user interfaces by demonstration.
Computer Graphics 20(3), pp. 249-258.

[NEWALL 62] Newall A. (1962).
Some problems of the basic organisation in problem-solving programs.
In: Proceedings of the Second Conference on Self-Organising Systems, Yovitis M.C., Jacobi G.T., and Goldstein G.D. (eds), pp. 393-423, Spartan Books (cited in Blackboard Systems, Englemore R.S. and Morgan A.J., 1988).

[NEWALL 75] Newall A. (1975).
A tutorial on speech understanding systems
IEEE symposium: Accedemic Press, New York, pp. 3-54
In: [ERMAN 88].

[NEWMAN 68] Newman W.M. (1968).
A graphical technique for numerical input.
Computing Journal 11, pp. 63-64.

[NEWTON 86] Newton P.D., Marsden P.H., Rector A.L. (1986).
What kind of system does an expert need?
Department of behavioural Sciences, Hudersfield Polytechniqh.

[NII 86] Nii H.P. (1988).
Blackboard Systems (part 2). AI Magazine 7(3), 82-106.
In [ENGLEMORE 88].

[NORM 86] Norman D.A. and Draper S.W. (1986).
User Centred System Design: New pespectives on human-computer interaction.
In [KOBSA 89] p. 9.

[NORMAN 85] Norman, D.A. (1985).
Design principles for human-computer interfaces.
Department of Psychology and Institute of Cognitive Science,san Diego La Jolla, California.

[NORMAN] Norman D.A.
Design Principles for human computer interfaces.
Department of Psychology & Institute of Cognitive Science,
University of California.chpt 11.

[ORACLE 87] Oracle Corporation (1987).
*SQL*Forms Designer's Reference Manual, Version 2.0*
Oracle Corporation, Belmont, California, USA.

[PARIS 89] Paris C.L. (1989).
The use of explicit user models in a genration system for tailoring answers to the user's level of experience.
In: [KOBSA 89] pp. 200-232.

[PETRIC 88] Petric J.(1988).
Computer modelling of landscapes
Phd Thesis, Department of Architecture and Building Science, University of Strathclyde, May 1988.

[POLLIT 82] Pollit, S. (1982).
Dialogue Handling.
Alvey IKBS Theme: Intelligent Front Ends.

[POLSON] Polson P.G.
A quantitative theory of human-computer interaction

[POWELL 88] Powell J.A. (1988).
Towards the Integrated Environment for Intelligent Building Design.
pp. 233-248, pub. UNICOM.

[POWELL 88] Powell J.A. (1988).
Towards the Integrated Environment for Intelligent Building Design.
pp 233-248, pub UNICOM.

[PRIME 88] Prime M. (1988).
User Interface Management Systems: A current product review.
Informatics Division, Rutherford Appleton Laboratory.

[PSTI 86] Production Systems Technology Incorporated (1986).
OPS/83 User's manual and report version 2.2.

[RADFORD 90] Radford A. (1990).
Guest Lecture held at the Department of Architecture, University of Strathclyde.

[RAFIQ 88] Rafiq M. and MacLeod I.A. (1988).
Automated Structural Component Definition from a Spatial geometry model.
Engineering Structures, Vol.10, No.1, January. 1988.

[REILLY 87] Reilly R.G. (1987).
Communication failure in dialogue and discourse.
North-Holland Publishing Company.

[RICH 89] Rich E. (1989).
Natural Language Interfaces
The University of Texas Austin
In: [KOBSA 89], Chpt 10. pp. 442-450

[RILEY X]

[ROSC 83] Rosc, E. (1983).
Prototype Classification and logicl Classification: The two systems.
In E. Scholnick, ed.: New Trends in Cognitive Representation: Chalenges to Paigets theory. Hillsdale, NJ.

[RUBENSTEIN 84] Rubenstein R. and Hersh H. (1984).
The human factor
Designing Computer Systems for People, Digital Press, Burlington, MA.

[RUTHERFORD 89] Rutherford J.H. (1989).
Intelligent Design Assitance (pilot Study).
ABACUS, Department of Architecture, Strathclyde University, November 1989.

[RUTHERFORD 90] Rutherford J.H. (1990).
View to DXF: A bridge between ABACUS viewer and DXF.
ABACUS technical bulletin (internal).

[SCHMUCKER 86] Schmucker, K.J. (1986).
MacApp: An Application Framework.
Byte 11(8), August 1986, pp. 189-193.

[SEHEIM]
[SELFRIDGE 59] Selfridge O. (1959).
Pandemonium: a paradigm for learning.
In: Proceedings of Symposium on the Mechanisation of Thought Processes, pp. 511-29, HMSO, London.

[SHACKEL 69] Shackel B. (1969).
Man-Computer interaction -- The contribution of the human sciences.
Egonomics, 12, pp. 485-499.

[SHAVI 79] SHAVI and GALI 79

[SHNEIDERMAN 87] Shneiderman B. (1987).
Designing the user-interface
Strategies for effective human-computer interaction, Addison-Wesley, Reading, MA.

[SIMON 69] Simon H.A. (1969).
The Psychology of Thinking
The Sciences of the Artificial Second Edition pp. 63-98: The MIT Press London.

[SMITH 86] Smith S.L. and Moiser J.N. (1986).
Guidelines for designing user-system interface software
Technical Report ESD-TR-86-278, Hanscom Air Force Base, MA, USAF Electronic Systems Division. In: [BARKER 89].

[SOMMERVILLE 85] Sommerville I. (1985).
The user interface
Software Engineering: Addison-Wesley International Computer Science Serries, Second edition, pp. 228-251.

[STEARN 90] Steam D. (1990).
Viewer unformated binary picture format
ABACUS internal technical bulletin.

[STEP 88] STEP: General AEC Reference Model (GARM) (1988).
Standard for the Exchange of Product data.
ISO TC184/SC4/NGI (Draft) Document 3.2.2.1, 12th October 1988.

[SUFRIN 86] Sufrin B. (1986).
Formal methods and interface design
In: M.D. Harrison and A.F. Monk (eds), People and Computers: Designing for Usability (HCI-86), Cambridge University Press, pp. 44-64.

[SUN 90] Sun (1990).
Reference Manual, Release 3.3
Sun MicroSystems, April 1990.

[SUTCLIFFE 88] Sutcliffe A.G. (1988).
Human-computer interface design
Macmillan Education Ltd.

[SUTTCLIFFE 83] Suttcliffe A. G. (1983).
Use of Conceptual Maps As Human Computer Interfaces.
Computational Department UMIS.

[SZEKELY 89] Szekely P. (1989).
Structuring programs to support intelligent interfaces
Information Science, University of South California.
ISI/RR-89-231

[TANNER 83] Tanner, P., Buxton W. (1983).
Some issues in future User Interface Management Systems (UIMS) Development.
IFIP WG 5.2 Workshop on User Interface Management.

[THOMAS 89] Thomas D. and OWEN J. (1989).
Use of CALS (Computer Aided Acquisition and Logistics Support) for product data exchange

[TOFFLER 74] Tofler A. (1974).
Futureshock
In: Nigel Cross (ed), Man Made Futures, Readings in Society, Technology and Design,
Open University Press.

[TREU 75] Treu S. (1975).
Interactive command language design based on required mental work.
International Journal of Man-Machine Studies (7), pp.135-149.

[UTGOFF 82] Utgoff, P.E., Mitchell, T.M. (1982).
Acquisition of Appropriate Bias for Inductive Concept Learning
Department of Computer Science, Rutgers University, New Brunswick,
In: AAAI 82.

[VEERKAMP 88] Veerkamp P., Akman V., Beruns P. and ten Hagen P. (1988).
IDDL: A language for intelligent interactive integrated CAD systems
Department of interactive systems, Centre for Mathematics and Computer Science (CWI), Amsterdam, The Netherlands.

[VIDOVIC 90] Vidovic N. (1990).
Design Framework Toolkits for Concurrent Engineering Environments,
Engineering Design Research Centre, CMU, Pittsburgh, PA 15213, 1990

[WAHLSTER 84] Wahlster W. (1984).
Cooperative access systems.
Future Generation Computer Systems 1, pp. 103-111.

[WAHLSTER 89] Wahlster W, and Kobsa, A. (1989).
User Models in Dialogue Systems.
User Models in Dialogue Systems, Symbolic Computation, Springer Verlag, chpt 1.

[WARTIK 86] Wartik, S.P. and Penedo, M.H. (1986).
Fillin: A Reusable Tool for Form-Oriented Software
IEEEE Software, 3(2), pp. 61-9.

[WASSERMAN 85] Wasserman, A.I. (1885).
Extending State-Transition Diagrams for the Soecification of Human-Computer Interaction.
IEEE Transactions on Software Engineering 11(8) , pp. 699-713.

[WEISS 82] Weiss, S., Kulikowski, C., Apte, C., Uschold, M. (1982).
Department of Computer Science, Rutgers University.
Patchett, J., Brigham, R., Spitzer, B., Amoco Production Research
Building expert systems for controlling complex programs
In: AAAI 1982, pp. 322-326.

[WILLIAMS 88]

[WILSON 87] Wilson M.D. (1987).
Task analysis for knowledge acquisition
Proc. SERC Workshop: Knowledge Acquisition for Engineering Applications,68-80, Cosener's House Abingdon, 18th - 19th July 1987.

[YANKELOVICH 88] Yankelovich N., Haan B.J., Meyrowitz N.K., and Drucker S.M. (1988).
Intermedia: The concept and the construction of a seamless information environment
IEEE Computer (January 1988).