The Application of Intelligent Knowledge Based Systems in Building Design

Final Report for EPSRC Grant GR/E/18018

Professor J A Clarke, ESRU, University of Strathclyde
D Mac Randal, Informatics, Rutherford Appleton Laboratory
J Rutherford, ABACUS, University of Strathclyde

November 1989

Abstract

Over the past decade a new generation of design tool has emerged which has the potential to simulate any building as an integrated environmental and energy system. While powerful at their core, these computer models suffer from several user interface limitations. This report describes an attempt to solve these problems by developing an Intelligent Front End (IFE) for building performance appraisal in general. The architecture of the developed system (termed IFe to distinguish it from the general case) is described and the use of the system is explained by demonstrating its application in the context of the ESP system for building performance prediction.

1. Introduction

Throughout the 80s several building modelling systems have emerged which can address the range of cost and performance issues of interest to building designers: from realistic visualisations, to a detailed appraisal of the operational performance of the energy, lighting and control systems. In response there have been many attempts to transfer the technology into practice. There are two main incentives for this. Firstly, buildings are complex mechanisms, involving phenomena such as transient conduction and air movement, and there is a growing realisation that traditional design tools cannot cope with this complexity. Secondly, and particularly at the earlier stages of the design process, there is a need for rapid feedback on the cost and performance consequences of alternative design scenarios. The present system of specialist consultants, while adequate for the detailed design and final specification phases, fails to provide this immediate *ad hoc* advice.

Given these incentives, different user types - engineers, architects, educationalist and researchers - are attempting to cost-effectively harness these models. The recent formation of organisations which represent the notion of building performance modelling (the International Building Performance Simulation Association [1] in North America and the Building Environmental
Performance Analysis Club [2] in the UK), the modelling programmes of the UK DEn [3] and US and UK research organisations [4, 5], the model validation and development activities of the
European Community [6, 7] and the workstation/ model acquisitions of the larger design organisations are offered as signal events in this respect.

There remains, however, a major barrier to the effective and routine use of these modelling systems, mainly because of shortcomings in their user interface. These shortcomings derive predominately from the conflict between the necessity for the model to be powerful, comprehensive and rigorous to adequately represent the real world complexity while, at the same time, being simple, straightforward and intuitive to facilitate user interaction. This situation is exacerbated by the divergence of the conceptual framework of the design orientated model user and the technically orientated model developer. To complete this confusion, there is the subtly different terminology of the scientific, engineering and design professions.

The current and laudable trend towards user friendly interfaces carries the risk of negating the power and flexibility of models by restricting the interaction to the lowest common denominator user level. Two of the major fundamental problems, the quantity and nature of the data being manipulated and the expertise and conceptual outlook required of the user, apply to a greater or lesser extent to all models. Although overcoming these problems will ultimately require truly intelligent systems, recent advances in Intelligent Knowledge Based System (IKBS) and
Human-Computer Interface (HCI) techniques offer some scope for medium term alleviation. Using these techniques it is possible to construct a user interface which incorporates a significant level of knowledge in relation to building description - in the face of real world uncertainty and realistic performance assessment methodologies. Such a system would direct a user's line of enquiry, allowing `What do you suggest?' and `Why do you ask?' type responses. It would also be expert enough to devise an appropriate performance assessment methodology and to coordinate model operation against this.

These were the goals of this project, which set out to develop an IFE for computer-aided building performance modelling in general. The objective [8, 9] was to design a machine environment which could act as an expert consultant to assist the user in the problem description phase, recognise her/ his appraisal wishes, commission computer analyses and report back on performance; all in terms which are acceptable to the given user type and design stage.

To achieve these goals, the IFE has to be an intricate synthesis of user modelling, HCI techniques, contextual knowledge manipulation and the interface to the possible performance prediction models at its back-end. In essence, the IFe is a generalised machine environment - a kind of intelligent user interface management system - which can define the mapping from any user's conceptual model of a domain (here building performance assessment) to the data requirements of any performance prediction model, simplified or detailed.

2. Model Users and IFEs

As mentioned above, one of the main objectives of this work was to handle the diverse user types that are found in the field of building performance modelling. To do this, the spectrum of users has initially been divided into three stereotypes as follows.

The Designer

Here the model has, typically, to deal with the earlier stages of the design process, characterised by high level, abstract concepts, incremental and exploratory definition of the issues, lack of focus (from the system viewpoint), tentative data, missing information and so on. What is required is a feel for the building performance, notification about potential trouble spots and information on the consequences of alternative design decisions.

The Engineer

Complementary to the above, once the overall design decisions have been made, there is a wealth of hard concrete information and a set of well defined objectives (albeit within each objective, there is a "designer" type activity to be carried out). The task thus becomes the utilisation of appropriate elements of the appraisal system to provide hard information upon which to make engineering decisions and provide feedback on the performance of the proposed building. Normally, the appraisal system matches this task quite well, but the user does not want to have to deal with the complexity and obtuseness of specific computer programs.

The Modeller

Here the user is probably very familiar with the appraisal system and requires almost direct access to its functionality. The only help required is, perhaps, straightforward assistance with

data preparation, both by providing standard data and by providing sensible default values to minimise data input.

Because the requirements of these three user types were kept firmly in mind during its design stage, the IFe can potentially handle any desired user type. Unfortunately, the resources available only permitted the second category to be implemented in any great detail.

Another, orthogonal categorisation is based on the user's level of expertise. In the field of building performance modelling, experience to date tends to classify users into two categories, the *expert* and the *novice* (note that the novice is usually an expert in his own field; the term is here used only in the context of computer modelling). The requirements of these two classes of user differ substantially. The expert, as usual, is concerned with speed and flexibility of input, direct control of the operation of each module of the system and access to all the resultant output in a structured but flexible manner. The novice, also as usual, wants a clear and coherent interface, where the system provides guidance on options and their implications, error trapping and recovery, and presents results in an easy to understood manner.

As well as distinct classes of user, there are 3 distinct facets of the modelling process, each raising its own set of problems for the user. These are data input, model control and result interpretation.

Data Input
One of the basic difficulties facing designers is the sheer quantity of data required to describe a building and manipulate a model. Not only is gathering this data a time consuming task, but frequently the data has not yet been specified, as is the case at an early design stage. Traditional interfaces tend to offer little help in generating sensible defaults. Also, due to the complex interrelationships, ensuring the integrity of the data can demand very high levels of understanding of the model's theory and mode of operation. Without this understanding, the importance, and hence the required accuracy, of an individual piece of data is very difficult to judge. As well as the question of `what', there is also the problem of `how' to input such a large quantity of highly inter-related data. The various factors associated with data acquisition, together with the user's often idiosyncratic conceptualisation of the inter-relationships, tend to conflict with the rigid question/ answer style of input common to many contemporary programs.

Model Control
Generally, control of the model does not require sophisticated user interaction. The major difficulty is the selection of the computational parameters to produce a sufficient quality and quantity of output to allow a meaningful appraisal of performance - in other words what is the most appropriate performance assessment methodology. For the novice, a lack of understanding of the implications of the selections being made can lead to confusion or even erroneous deductions if the output data is inadequate.

Result Interpretation
It is here that the requirements of the novice and expert differ most. The expert will be trying to detect patterns in, and relationships between, the different building parameters, in an attempt to isolate the dominant causal factors. To do this, all the data generated by the simulation has to be available and capable of being displayed in juxtaposition with any other data. The novice, on the other hand, merely requires a concise summary of performance, preferably in terms of those parameters which are most meaningful to the design team and client. Unfortunately, due to the nature of the program's output, the novice may experience difficulty in relating poor performance to the design parameters under her/ his control.

The limited experience of IFEs to date has generated a number of tentative guidelines, the

main ones being as follows.

- In order to be of much assistance, a front-end must be able to interact at the user's level. This implies that it should have a model of the user's knowledge and problem solving strategies in order to extract all the implicit information and unstated assumptions from the user's statements. Considering the misunderstandings that can arise in person-to-person communication, the difficulty of this task will be appreciated. However, another, more tractable function of a user model is to track the user's mental model of the system and react accordingly. This, of course, must adapt as the user progresses from novice to expert [10]. However, most of the user's progress comes through building this mental model of the complete system: back-end, front-end and even the hardware. Therefore a good front-end must be careful not to hinder or sidetrack this process by changing the characteristics of the interface as the session proceeds. Instead, it should be capable of hastening the process by, firstly, hiding itself, the hardware and any other situational aspects not directly relevant to the user's task, and, secondly, by presenting the underlying concepts of the back-end in an easily assimilable form. Where the user's model of the back-end system is deficient or wrong it should take appropriate remedial action. At this level of operation, an IFE is encroaching on the field of intelligent tutoring.
- The context in which the system is being used and the underlying motivation of the user must be taken into account [11, 12]. Evidently, the type of help proffered by an IFE will be qualitatively different if the user is a student in a School of Engineering, a technician in an architectural practice or a researcher into building physics. In the student's case, the emphasis will probably be on teaching building physics as an intelligent tutor; in the technician's case, the emphasis will be on presenting the system as a friendly consultant; while in the researcher's case, the requirement is for a colleague to help spot errors. These scenarios are sufficiently distinct and it is preferable to deal with them by using different front ends, specialised for the context in which they are to be used. However, even in the more limited scenarios, the IFE should be able to establish the user's objectives and generate a suitable problem-solving strategy without the user having to spell out every step of the process.
- One of the things which distinguish an IFE from other pieces of software is the need to interface to a back-end. This requires it to perform the mapping from the user's model of the back-end to the back-end's model of the user [13]. The interface to the user is a fairly conventional, if difficult, application of established HCI techniques. However, to interface to a back-end requires considerable knowledge about its function and operation. At the extreme, the IFE has to know as much about the methods of solving the user's problem as the back-end. Abstracting this knowledge can be as difficult as deriving an expert system to replace the back-end.
- Not only does the IFE need to have a good interface to the user, it is itself part of the user interface to the back-end. Hence, it must take account of the usual HCI criteria, and support a sufficiently rich dialogue to avoid constraining the user's problem-solving activity.

Several of the components of an IFE have already been investigated, either in their own right or as part of other research - for example, plan formation [14] and dialogue handling [15]. Although a substantial amount of fundamental research has still to be carried out, the results already achieved provide the basic building blocks for the construction of an IFE.

For the uninitiated, Appendix A contains a glossary of terms as used throughout this report.

3. The IFe in Outline

The IFe system is built from several cooperating modules organised around a central communications module, the *Blackboard*, to facilitate multiple use of information. These modules run asynchronously and can examine the Blackboard for information, posting results back to it. Figure 1 shows the IFe architecture.

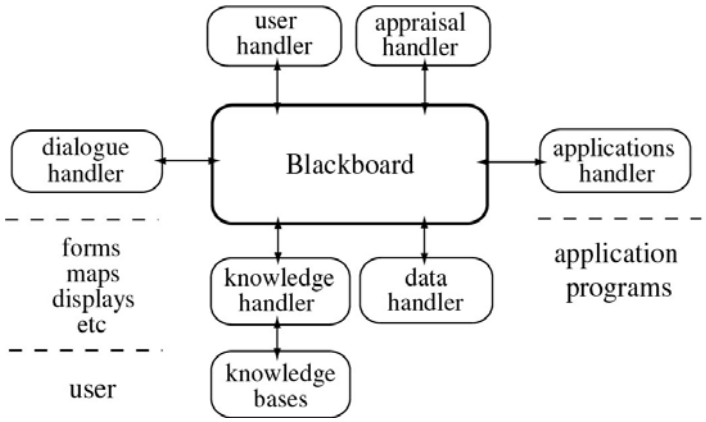


Figure 1: Architecture of the IFe.

The IFe modules include:

- A *Dialogue Handler* to converse with the user in a manner which is tailored to her/ his conceptual class, level of experience and stage reached in the design process.
- A *Knowledge Handler* to verify user entries and, by inference, to complete the building description to the level required by the target application program (for example the ESP system [16] was the target program in the IFe prototype).
- A *User Handler* to track the user's progress and ensure the system responds in an appropriate manner.
- An *Appraisal Handler* to coordinate the performance assessment methodologies.
- A *Data Handler* to create, from the information supplied by the user and the knowledge handler, the building description as required by the application program(s) to which the IFe is interfaced.
- An *Application Handler* to orchestrate an application program against the selected performance assessment methodology and to feed it its required building description data.

The functions to be handled by the IFe therefore include conversing with the user in the appropriate terminology (the Dialogue Handler); generating the description of the building (the Knowledge Handler); collecting, organising and storing this data (the Blackboard); generating the program-specific input data-set (the Data Handler); generating the program-specific control inputs (the Appraisal Handler); and invoking the targeted application program (the Application Handler).

4. The Blackboard

This lies at the heart of the system and has two major functions. Firstly, it stores the data representing the current state of the problem definition as input by a user or inferred by the knowledge handler. Secondly, it acts as a communication centre for its various clients.

4.1 Data Storage

Given that clients are autonomous entities and therefore must not be expected to know the data structures required by the other clients, the Blackboard must be able to receive arbitrarily structured information from one client and pass this on to another. The data structuring is therefore explicit in that it is generated by the client and stored with the data. (There has already been substantial work, under the Esprit CAD*I project [17] and the ISO standards work on STEP [18], examining the nature of CAD data. Unfortunately, this does not cover all the types of data that the IFe must handle, in particular data about the user and about the required appraisal. Furthermore, this is designed much like a database schema, a predefined data structure to be filled in by an application. The Blackboard, on the other hand, uses a rudimentary schema definition language so that clients can create their own schema as

required.) The IFe unit of information is termed a *Tuple* and has the form:

concept <tab> value

for example: "latitude<tab>56.5", hereinafter denoted <concept value>.

A Tuple can be meaningful to one or more client and usually, though not necessarily, will correspond to a concept understood by the user and/ or appraisal program when addressing a particular class of problem (for example, latitude, number of rooms or window width in a building modelling context). There is no restriction imposed by the Blackboard on the contents of the value field of the Tuple. The Tuple, as recorded on the Blackboard contains:

as its 1st element, the concept name;
as its 2nd element, the string "user-set" or "kb-set";
as its 3rd element, the string "nokey" or a list of keys;
as its 4th element, the value(s) associated with the concept.

Tuples are stored in the order of receipt and are `fetched' by the Blackboard, using pattern matching applied to the 3rd element if a key exists, otherwise to the 1st element. This approach to information handling means that the Blackboard can accommodate the data handling requirements of any client (even those added in future) and is able to retrieve information without knowing its structure or meaning.

4.2 Client Communication
Blackboard clients are divided into two classes: those at the user-end (the Dialogue, Knowledge and User Handlers) concerned with extracting from the user the building description and the appraisal definition, and those at the back-end (the Appraisal, Data and Application Handlers) concerned with creating the input data and control instructions to `drive' the application program(s). The Tuples corresponding to these two classes are held within separate communication areas on the Blackboard termed the *user dialogue* and *appraisal* areas.

The Blackboard is not just a passive data structure. Clients can ask it to create new, named areas. The various clients can then post Tuples to these areas, the name of the poster and the time of posting being recorded with each Tuple. Clients can either explicitly ask the Blackboard for information by identifying the area and the Tuple concept, or can ask the Blackboard to keep them informed of any new information posted to any particular area. First, and most important, this avoids clients having to poll the Blackboard. Second, it provides a mechanism whereby two or more clients can create a Blackboard area to serve as a communication channel between them. For example, the user dialogue area is used to pass information between the Dialogue and Knowledge Handlers in order to validate user inputs and provide essential feedback.

All Blackboard clients are autonomous processes running asynchronously and the Blackboard does not impose any selection criteria on incoming messages. Should it become desirable or necessary to provide some form of scheduling and resource allocation for clients, it is envisaged that an area on the Blackboard would be established to act as a control centre.

5. The Dialogue Handler
This client is responsible for managing and monitoring all user interactions and implementing the level of communication as necessary for a given user and task combination. Unlike conventional systems where the interface remains static throughout a session, by keeping track of a user's progress, the Dialogue Handler can, in conjunction with the User and Knowledge Handlers, tailor the dialogue to the user's level of expertise and performance history.

The basic function of the Dialogue Handler is to pass the user's inputs to the user dialogue area on the Blackboard and to pass messages and requests, as posted on this area by other Blackboard clients, back to the user. To perform this function, the Dialogue Handler has several mechanisms at its disposal to achieve interaction with the user. The primary mechanism is a generic Forms Program which can manipulate a set of forms which correspond to a given user class and capability level - termed a *user conceptualisation*. Each form entity (a labelled field, a button, a multi-option pop-up, etc.) corresponds to a particular `concept' within the user conceptualisation in question (for example window width, number of rooms, project name, etc.). Groupings of related concepts are located on the same form to comprise a *meta-concept* (such as building geometry, construction, control system, appraisal definition and so on). Meta-concepts can contain other meta-concepts, allowing a complete hierarchy to be specified. A set of meta-concepts (forms) then defines a particular user conceptualisation in terms of only those related concepts that are deemed acceptable to the user type the conceptualisation represents. Via these forms, the user can ask about concepts as well as associate values with them.

Thus the Dialogue Handler is essentially a communication switch and protocol converter, sitting between the Forms Program and the Blackboard. The Blackboard protocol has been designed to be as generic (and extensible) as possible, and is not tied to the use of the Forms Program. It is the responsibility of the Dialogue Handler to map the data generated by the Forms Program into the format required by the Blackboard and map the commands emanating from the Blackboard to semantically equivalent commands to the Forms Program. The Dialogue Handler can easily be extended to cope with many types of user interaction, from a simple question/ answer dialogue to full natural language input.

It is important to note that one user conceptualisation may involve many concepts (many user inputs) while another may involve only a few concepts (few user inputs). By relying on a greater degree of inference in the latter case (the function of the Knowledge Handler), the power of any application program (ESP for example) can be offered to both users.

At the time of writing, the Dialogue Handler is able to manipulate a map display program [19] for positional inputs and two geometric modelling programs for geometry definition [20] and perspective view generation [21]. The capabilities of the Dialogue Handler are elaborated in Section 13.1 where the role of the Ife knowledge bases is explained.

5.1 The Forms Program

The objective of the Forms Program is to manipulate a set of forms, passed to it by the Dialogue Handler at the dictates of the User Handler, which correspond to a given user conceptualisation. Any single form is a collection of logically related concepts, each one represented graphically by an identifier and an associated answer field. It is important to appreciate that the Forms Program does not impart any meaning to a user entry; it merely passes on all such entries to the Dialogue Handler in order to be available to the other IFe modules via the Blackboard. It does, however, type-check the user entry. There are two distinct aspects to the Forms Program: form creation and form manipulation.

5.1.1 Creation

Each form (meta-concept) is defined by a template file which establishes the form concepts and their relative positioning by the use of a "form definition syntax". This technique allows any user conceptualisation to be established as a given form-set for use when a user of that type is detected by the User Handler. (Note that to complete a user conceptualisation requires the creation of a matched knowledge base for use by the Knowledge Handler to validate concept values and to make the necessary inferences to complete the building/ appraisal definition to the required level of detail. This process is explained later.)

Forms can be nested to categorise concepts and to provide scrolling regions to accommodate cases where a particular meta-concept contains many individual concepts. Section 13.2, for example, explains a template file and shows the resulting form. It also lists the complete form creation syntax.

All user interactions with a form are controlled by mouse and keyboard inputs. The former is used to select a concept, the latter to make an entry. Two pop-up menus, activated by the mouse, are associated with each concept label. The first (left button down) is common to all concepts and offers several general options which allow a user to request help, ask for an example or instruct the IFe to choose an intelligent default. The second (right button down) is a concept-specific menu which can be assigned run-time options by the Knowledge Handler.

5.1.2 Manipulation

The Forms Program is designed to operate on the basis of instructions received from some external process; the Dialogue Handler in the case of the IFe. This means that the status of any concept or meta-concept can be changed at any time. For example, the options offered on a concept menu or the value associated with a concept can be changed by the Knowledge Handler on the basis of previous user inputs or information received from, say, the User Handler. Alternatively, concepts and meta-concepts can be focussed/ de-focussed (shown/ hidden) in order to direct the user's attention toward or away from a particular concept. A whole form-set can even be dynamically replaced, allowing a user conceptualisation to be changed (shifted, up or down) at the dictates of the User Handler who may, for example, have detected that the user was having difficulties with the currently focussed concepts as a result of an inappropriate, initial user classification. The Forms Program also supports animated graphics (useful for user feedback when addressing geometrical concepts) and dynamic window creation (useful for error reporting).

6. The Knowledge Handler

The function of the knowledge handler is to manipulate several independent knowledge bases which exist to control the user dialogue, collect and validate user entries, make whatever inferences are appropriate and to store on the Blackboard a representation (or representations) of the building and required appraisals. This requires a mixture of conventional procedural programming, event-driven programming and rule-based inferencing. Of the knowledge representation languages widely available at the present time, Prolog is by far the most suitable for these tasks. And so the Knowledge Handler is an autonomous inference engine based on a Prolog interpreter. It monitors the user dialogue area on the Blackboard in order to obtain the

Tuples corresponding to user inputs, to convert the Tuple syntax to a Prolog Predicate (goal), to use this Predicate to validate the user input and, by inference, to further complete the building/ appraisal definition. The Knowledge Handler also performs the reverse operation, namely the filtering out of the Prolog syntax before passing back a new Tuple (the result of the invoked predicate) to the Blackboard.

In practice the Knowledge Handler will have access to several Knowledge Bases, most of which correspond to a particular meta-concept and are loaded when that meta-concept (form) is activated. Each of these Knowledge Bases exists to build on the Blackboard that part of the problem definition to which the meta-concept relates. Based on what is already known about the problem (from previous inputs) a Knowledge Base can deduce what concept values are sensible (that is intelligent input validation), and how to derive appropriate default values for concepts required by the target application but not addressed directly by the active user conceptualisation (that is intelligent defaulting).

These Knowledge Bases are also responsible for coordinating the user dialogue (often a user conceptualisations will require its own unique dialogue). This is achieved by including Prolog code to provide feedback, help and guidance to the user. To this end a Knowledge

Base contains knowledge about the meta-concept to which it relates and about the capabilities of the Dialogue Handler in terms of its control syntax.

The construction of these Knowledge Bases, and their insertion into the IFe is explained in Section 13.1 when the Knowledge Bases corresponding to a particular user conceptualisation are considered in detail.

7. The User Handler
Like the Knowledge Handler, this module is an autonomous inference engine based on a Prolog interpreter. Its function is to set the appropriate user conceptualisation on the basis of a user's class and level within a class. This ensures that the subsequent IFe session is tailored to the user's skill level and that the appropriate level of guidance, feedback or help can be given by the Knowledge Handler during the session. At the present time real, dynamic user modelling has not been attempted. Instead, the user is initially classified, from a database or explicit user input, and the corresponding user conceptualisation knowledge bases established for use by the Knowledge Handler.

During a session, the user's progress is assessed by monitoring the user dialogue area on the Blackboard in order to determine the number of errors, changes of mind, backtracks and Knowledge Handler overrides. On the basis of this information, or upon explicit user request, the User Handler can change the user conceptualisation to a more suitable type/ level. From then on, the Knowledge Handler will automatically pick up those meta-concepts associated with the new user conceptualisation.

8. The Appraisal Handler
In the current implementation of the IFe, the user is explicitly asked to specify her/ his appraisal objectives in terms that the Appraisal Handler can understand. The appraisal possibilities are held within the IFe as discrete, parameterised Unix Shell scripts representing a particular performance assessment methodology. It is the Appraisal Handler's task to select the appropriate
Appraisal Script and, based on the information available on the Blackboard, to select the values for the parameters. These will include the particular application program to be invoked at each stage in the methodology and the appropriate values for the various design or performance parameters on the basis of which decisions will be made. After this task has been completed the data is posted back to the Blackboard where it is accessed by the Data Handler whose function is to prepare the input data required by the identified programs. The form of an Appraisal Script is detailed later.

9. The Data Handler
This module creates an application specific data-set from the building definition as held on the Blackboard. It does this on the basis of a Data Definition Script which defines, in parameterised manner, the data preparation procedure of the targeted program. The parameters of this Data Definition Script define the data which must be obtained from the Blackboard. By executing this script, the Data Handler builds the required data-set. The creation of these scripts is described, by example, in Section 15.

10. The Applications Handler
The Applications Handler passes the Appraisal Script and its matched data-set (as posted on the Blackboard by the Appraisal and Data Handlers) to a Unix Shell where the Appraisal Script is executed. The responses from the script can then either be displayed directly or passed back to the Blackboard for communication to the user via the Dialogue Handler.

11. Status of the IFe
At the present time, a prototype IFe is operational on a Sun3/60 under SunOS 3.5 and SunOS 4.0.3. Specifically, the following has been achieved.

- The blackboard is fully developed and is able to interface with any number of autonomous clients.
- The user dialogue module is based on a generalised forms manipulation program, designed to operate on bit-mapped screen technology under X-Windows or Sun's Suntools. Its function is to manipulate a set of forms which correspond to a particular user conceptualisation.
- The knowledge handler is implemented as a Prolog inference engine. Its mission is to control the forms interface, directing the dialogue session and responding to the user's inputs and requests for help.
- A form-set corresponding to one particular user conceptualisation (a moderately proficient Engineer) has been developed and the knowledge bases, matched to this conceptualisation, created.
- The Appraisal, Application and Data Handlers have been configured in a form suitable for (but not restricted to) use with the ESP system and the requisite scripts installed.

12. The IFe Software

The IFe system exists as a research prototype which will be refined in the coming months and years. As an aid to this refinement process, it is the authors' hope that others will attempt to apply the system, to their particular end user types and application programs. This Section describes the nature of the IFe software as distributed. Then Section 13, by the use of illustrative examples, demonstrates the installation of a new user conceptualisation and the process used to `drive' application programs.

An IFe distribution tape will contain the following sub-directories and files arranged as shown within an IFe home directory (~ife).

Directory	File	Content
~ife/bin	ife_bb	Blackboard
	ife_dh	Dialogue Handler
	ife_kh	Knowledge Handler
	ife_uh	User Handler
	ife_ah	Appraisal Handler
	ife_bh	Data Handler
	ife_ph	Application Program Handler
	forms	Forms Program
	map	Map Program
	change_cpt_set	Script to set user conceptualisation.
	browse	Raster `fetch' program
	perspective	Link to perspective program.
	conlst	Construction database utility.
	vim	Link to geometry modeller.
	conv	Raster conversion program (type "conv" for supported formats).
~ife/sys		Links to third party binaries (src for licence holders only).
~ife/sys/bin	CC	C++ compiler script
	cfront	C++ to C compiler, main program
	munch	C++ to C compiler, aux program
	nip	Prolog interpreter, startup program
~ife/sys/lib	libC.a	C++ function library
	nip.bin	Prolog interpreter, run-time program
~ife/sys/include	plload.h	Prolog header file for use with C
CC/	*	include files for CC
~ife/docs	*	Some IFe reports (including this one) set in

		troff format
~ife/src	*	IFe module source
~ife/src/tmp	*	Some IFe program driver commands (for testing purposes)
~ife/src/ph	*	Application Handler source
~ife/src/bb	*	Blackboard source
~ife/src/dh	*	Dialogue Handler source
~ife/src/forms	*	Forms Program source
~ife/src/bm	*	Data Handler source
~ife/src/browse	*	Browse source
~ife/lib	*	IFe working files, etc.
~ife/lib/maps	*	Maps
~ife/lib/startup	*	Definition of Blackboard, Dialogue Handler and Application Handler clients
~ife/lib/bm_filters	*	Data Handler filters for targeted application program.
~ife/lib/icons	*	Form Icons
~ife/lib/um	*	User model
~ife/lib/um/kbs	*	User Handler knowledge bases and C utilities
~ife/lib/uc	*	User conceptualisations
~ife/lib/uc/initial		Knowledge Handler infrastructure (used as IFe start-up).
~ife/lib/uc/initial/forms	master	IFe top level form (used as IFe start-up)
~ife/lib/uc/initial/kbs	*	IFe infrastructure knowledge bases and installation specific defaults.
~ife/lib/uc/engineer	*	Engineer conceptualisation knowledge bases and conceptualisation specific defaults.
~ife/lib/uc/engineer/forms	*	Engineer conceptualisation forms.
~ife/lib/uc/engineer/kbs	*	Engineer conceptualisation knowledge bases.
~ife/lib/buildings		Directories containing raster images of different buildings for use by the browse program.
~ife/lib/appraisals	*	Appraisal Shell scripts.

In total about 6 MBytes of disk space will be required to accommodate the system which has been designed to operate on a machine with 8 MBytes RAM (4MBytes RAM minimum). Given the prototypical nature of the product it is probable that source code interventions will be required to overcome the inevitable bugs and deficiencies. To facilitate this, source code is available. But note that two aspects of the IFe require proprietary software which it may not be possible to distribute. First, the Blackboard is written in C++ and so, if changed, will require a C++ compiler. Second, all knowledge manipulation is done using a Prolog interpreter. At present this is the NIP system from AIAI at the University of Edinburgh [22]. If another interpreter is used this will probably mean that the distributed user conceptualisation will not work (although any newly developed conceptualisation will). However, any modification required should be straightforward as only standard Edinburgh syntax has been used.

13. Installing User Conceptualisations

Currently there is only one user conceptualisation available, corresponding to an individual who is relatively experienced in energy modelling, though several more have been proposed [23]. This conceptualisation corresponds to a moderately computer literate engineer and is used here as an illustrative example to demonstrate the technique of conceptualisation installation. The targeted application, which defines the scope of the building description as

held on the Blackboard, is the ESP system.

The starting point is to understand the viewpoint of the new user type and to relate this to the data requirements of the application program to be used. For example, the user may well be able (or expect) to input directly much of the data required by a model or, alternatively, a considerable degree of conceptual mapping may be required. In any event there are three stages involved in developing a new user conceptualisation:

1. the creation of a set of knowledge bases to handle those concepts and meta-concepts that is acceptable to this class of user;
2. the addition to, or enhancement of, these knowledge bases to provide the conceptual mapping and inferencing necessary to complete the building/ appraisal definition; and
3. the creation of a matched set of forms presenting the individual concepts in a style which is acceptable to this class of user.

These three stages are now described in turn.

13.1 Creating Knowledge Bases

The prerequisite of knowledge base creation is an understanding of Prolog. But before explaining the process in any detail, it is important to describe the purpose and overall philosophy of these knowledge bases. As described earlier, the Knowledge Handler has two main functions to carry out: building a complete problem definition on the Blackboard and coordinating the dialogue with the user.

13.1.1 Building the Problem Description

This involves collecting the data input by the user and using it to build in the Blackboard area "u_cpt" a complete problem description. As described in Section 4, this is held as <concept value> Tuples. For ease of handling, both by the IFe and by the user, concepts are collected into related sets called meta-concepts. Meta-concepts can, of course refer to subsidiary meta-concepts as well as base concepts, so that a complete tree can be built up. Usually, but not necessarily, each meta-concept has its own knowledge base to handle it's collection of concepts. (If the metaconcept is sufficiently small and simple, it could be handled by its parent meta-concept.) At its simplest, inside each knowledge base, each concept is handled by a number of predicates named after the concept. When the user associates a value with a concept, for example by typing into the appropriate field on a form, the Dialogue Handler posts a <concept, value(s)> Tuple to the "user_dialog" area on the Blackboard. The Knowledge Handler monitors this "user_dialog" area and invokes the predicate named after the concept with the value(s) as its argument(s). For example, within the currently installed user conceptualisation (~ife/lib/uc/engineer) there is a meta-concept of a building (form "forms/building"), handled by a knowledge base called "kbs/building". Within this meta-concept there is the concept of building `function' which is handled by a predicate called `function'. (There may be several similarly named predicates to handle different cases and, of course, there can be many auxiliary supporting predicates.) When the user types (for example "office") into the `function' field on the `building' form, predicates matching "function(office)" are invoked.

```
/* see section 13.1.3 for a definition of undefined predicates */

function( _Function) :-                 /* tried 1st */
        known(bld_function, _Function).         % already set to this?
                                        % ie have we already recorded concept
                                        %      'bld_function' with this value

function( _Function) :-                 /* 'known' above failed */
        check_function(_Function),      % valid function?
        uset(bld_function, _Function).  % yes, so accept it
```

```
function( _Function) :-                 /* invalid function ('check_function' above
failed */
        feedback(bad_function).         % let the user know about error

check_function(_Function) :-            % fail if invalid
        function(_Function, _).         %is there any pred 'function' among
                                        % the defaults with this value?

function(residential).                  /*list of acceptable building functions */
function(office).                       % could be held in separate defaults file
function(factory).
```

The predicate `uset' posts data to the Blackboard area "u_cpt". This is the mechanism by which the Knowledge Handler constructs a Tuple to be added to the evolving problem description.
A related predicate `known' allows the current (that is the latest) value of a concept to be recovered from the Blackboard. Note the symmetry between the arguments for these two predicates.
Thus the code for handling one meta-concept can be completely independent of the code for handling others. Since these predicates operate on any Prolog atom (strings, numbers, even variables), there are no in-built constraints on the type of information that can be stored on the Blackboard. The only constraint is the need for the other Blackboard clients to retrieve the data. To this end, it is recommended that the convention given with the description of uset in Section 13.1.3 is followed.

Having received and validated the user input and recorded it on the Blackboard, there is one further task which is performed by the Knowledge Handler. From the data just received, the previously input data and the in-built knowledge, it may be possible to make inferences which will remove the need to explicitly ask the user for further data. As an example, the following code shows how, if the user describes the building's environment as one of the known types, the relevant exposure index and ground reflectivity can be inferred.

```
/* see section 13.1.3 for a definition of undefined predicates */

environment( _Site_type) :-             % already set?
        known(environment, _Site_type).         % yes

environment( _Site_type) :-             % no, new site type
        uset(environment, _Site_type),  % accept any site type string
        exposure(_Site_type, _Exposure),        % have we an exposure
        grnd_rflct(_Site_type, _Grnd_rflct),    %  or ground ref. for this type
        kset(exposure, _Exposure),      % yes, kset them
        kset(grnd_rflct, _Grnd_rflct),
        feedback(environment_known).    % feedback if novice

user_said(environment, _Site_type) :-   % unknown site type
        feedback(environment_unknown),          % feedback
        ask_user( exposure, 1),         % ask user for
        ask_user( grnd_rflct, .2).      %     exp and gr

feedback(environment_unknown, novice) :-
    chat_usr([
        'Sorry, I don't know what that means!  Unless you know',
        'what you are doing, I suggest you select 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.-1.).  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(urban,4).                              /* in defaults file? */
exposure(rural,5).
exposure(city-centre,2).
grnd_rflct(city-centre, .2)
grnd_rflct(urban, .2)
grnd_rflct(rural, .15)
```

In the above, a predicate `kset' was used in place of `uset'. The only difference between these two is that uset flags the concept as having been explicitly set by the user while kset flags it as having been inferred by the Knowledge Handler. User set values cannot be overridden by the Knowledge Handler, but, of course, the user can set any concept to whatever value is required, whether or not the Knowledge Handler has asked for the information.

Clearly, the above is targeted at a user who can, if necessary, supply the exposure index and ground reflectivity. For a different user type, it would be possible to generate sensible defaults using previous information such as site location. Very much more complex inferences may be made:

```
infer_from(position, _Latitude, _Longitude) :-
          ( \+known(location, _),                       % \+known = not known
            position(_Location, _Lat, _Long),           % guess lat, long
            near(_Latitude, _Longitude, _Lat, _Long, 0.5) ->  /* 30 miles */
                  kset(location, _Location),            % close enough
                  infer_from(location, _Location) % make inferences from loc
          ; true
          ),
          ( \+known(climate_set, _),                    % guess climate file
                        /* need idea of proposed building appraisal */
            ( known(analysis_climate_requirement, _Climate_type), % suitable?
                  /* suits analysis requirements */
                  climate_type(_Climate_set, _Climate_type, _, _)
            ;     true                % no particular analysis requirements (yet)
            ),
            climate_set(_Climate_set, _Lat, _Long),   % have we a suitable one
            near(_Latitude, _Longitude, _Lat, _Long, 1.5) ->  /* 100 miles */
                  kset(climate_set, _Climate_set),
                  infer_from(climate_set, _Climate_set)
          ; true
          ).

          /* If the user overrides the above inference, or resets the location
           * we need to retract all the inferences made above*/
retract_infer(position) :-
          retract(u_cpt, location), retract_infer(location),
```

retract(u_cpt, climate_set), retract_infer(climate_set).

13.1.2 Coordinating the User Dialogue

The function of the Dialogue Handler and the overall structuring of the dialogue into concepts and meta-concepts has already been described (Section 4). Basically, the Dialogue Handler runs the Forms Program (and any other interactions with the user - for example the Map Program).

It does this by monitoring a Blackboard area called "user_dialog", implementing the commands it finds there and posting back the user actions. To maintain independence from any specific interaction program, the Dialogue Handler deals with generic requests, such as "tell_user" (associate a default with a concept), and "new_dialog" (invoke a new form or interaction program), and converts these into the syntax required by the interaction program, usually the Forms Program. The Knowledge Handler, under the direction of the various knowledge bases available to it, uses these requests to interact with the user.

A major feature of the way the Knowledge Bases are designed - independent concepts each handled by a suite of predicates - is that there is no predefined order in which they are accessed.

The user can, at any stage and in any order, volunteer the data associated with these concepts. Since this is potentially confusing for the user, difficult to manage in the Knowledge Base and very error prone, one aspect of the interaction handling that the Knowledge Handler has to deal with is the overall guidance of the user through all the concepts that are necessary to complete the problem definition. The hierarchical organisation of the concepts into meta-concepts offers a way of directing the dialogue at a gross level. The user can be requested to indicate which meta-concept she/ he wishes to address, for example zone geometry, and then their attention can be focussed specifically on the concepts appropriate to that meta-concept, for example zone name, orientation, origin, height, walls (a subsidiary meta-concept), etc. This focussing of attention can be complete, that is no other concepts/ meta-concepts are available until the user completes/ quits this meta-concept, or partial, that is a selected subset of other concepts/ meta-concepts are available. For example, when specifying the construction elements of a zone, it would be useful to have access to the geometry specification so that minor change to the geometry can be made, without changing the current focus of attention.

To give a finer level of control, each concept can independently be brought to the user's attention, or excluded from the collection available for input. Thus, a collection of concepts can be gone through one at a time, giving a question/ answer style of interaction, or they can all be raised at once and the user left to respond in whatever order they wish, a command-driven interaction style. In practice, a mix of these would be required because often a concept's value can have implications, either on the values acceptable for other concepts, or on whether those concepts have any meaning. In this case, it would be better to wait for the user to set the value of this concept before raising those other concepts affected by it. However, for other cases, letting the user choose the order in which concepts are addressed leads to a much more friendly user interface (albeit slightly harder to learn).

At this low level, some other benefits of the knowledge based approach can be applied. In a number of cases, the Knowledge Base can infer an "intelligent default", that is one that takes into account the context in which it is being offered and the data that has already been input by the user. This could then be suggested to the user. Alternatively (or as well), the Knowledge Base may have a list of preferred options (for example standard constructional elements) about which it has a lot of built-in knowledge, and this list could be offered to the user for perusal. Both of these possibilities have been built into the user conceptualisation as currently installed within the IFe.

The requests currently available for controlling the interaction are as follows.

new_dialog
This is a request to initiate a new interaction with the user, usually by starting a new interaction program (for example the map utility for inputting positions). It takes 3 arguments, the name of the utility, the filename containing the executable (binary or shell script), and the argument string to be passed to the program. For example, the Tuple posted on the user_dialog area of the Blackboard in order to start the map utility for inputing positions is:

< new_dialog ife_map_prog ~ife/bin/map "-s -o -e" >.

Currently, only one Forms Program can be run, as this can already deal with multiple dialogues in multiple windows.

focus_user
This request directs the user's attention to a new meta-concept. (The user can, of course, still address the previous meta-concepts.) The intention is to notify the user that the concepts dealt with by this meta-concept are now available for input. This is usually a response to a user input, for example a request to specify a zone's geometry, and has a Tuple like:

< focus_user geometry >.

In the Forms Program, this causes the form with name `geometry' to be (re)displayed. The current focus of attention is highlighted for easy identification If the form is not already in memory, it is loaded from a file of the same name in the form's directory for the current user conceptualisation - that is ~/lib/uc/?/geometry (but see Section 13.1.4).

unfocus_user
This request removes the given meta-concept from the domain of discourse. It indicates to the user that the included concepts are no longer available for input. Its main function is to keep the number of concepts available to the user down to a manageable number (for the user). A subsidiary benefit is to permit reuse of concept names in different meta-concepts. For example, when the user has finished specifying the zone geometry and moved on to plant specification, the zone geometry concepts can be disabled by the Tuple:

< unfocus_user geometry >.

In the Forms Program, this causes the `geometry' form to be removed from the screen (hidden).

ask_user
When focusing on a meta_concept, some concepts are implicit and will be addressed by the user without further prompting. For example they may already be displayed on the form when it was first activated. Others are only relevant later in the dialogue, once some basic information has been entered and considered by the Knowledge Handler. In this case, the user is explicitly asked for the subsidiary information. For example, if the user gives an unknown (to the Knowledge Handler) location for the building, the user must be asked for its latitude and longitude. This can be done by means of the Tuples:

< ask_user latitude>
< ask_user longitude>.

Sometimes it is possible to suggest a default value, and this can be added as a further element in the Tuple:

< ask_user latitude 55.7 >.

In the Forms Program, this request causes a field to be (re)displayed. The field must have been previously defined, usually as a hidden field in the form definition file.

unask_user
In some cases, a concept that the user has been asked to address becomes irrelevant because of subsequent information. For example, if the user specifies that she/ he wants an annual simulation, it is unnecessary to ask for the simulation's start and finish dates. In this case, the concept can be removed from the discourse using the Tuples:

< unask_user start_date >
< unask_user finish_date >.

In the Forms Program, this causes these fields to be hidden.

new_query
This is the mechanism by which a new, previously undefined concept can be brought to the user's attention. In this case, not only must the Dialogue Handler be given the concept name, it must also be told how to ask the question, what syntax checks to apply to the input and so on. This is clearly interaction program specific. In the Forms Program, for example, fields are normally defined in the form specification file. To create a field not so specified, the Tuple supplies the concept name and a string to be parsed by the Forms Program itself:

< new_query user_age "field \\nname user_age\\ntype real\\n..." >.

tell_user
This is a request to inform the user of a concept's value. It takes 2 arguments, the concept name and the value associated with it. For example, to set the value in the "no_of_zones" field to 6:

< tell_user no_of_zones 6 >.

In a number of cases, it is necessary to distinguish between different instances of the same concept, for example, the different zones to which the concept "zone_name" applies. In this case the concept name has a `$' appended to it, and the 1st argument is the index of the required instance. For example, to set the name of zone 4 to `kitchen':

< tell_user zone_name$ 4 kitchen >.

suggest_user
In many cases, the Knowledge Handler can suggest a sensible default value for a concept based on data input so far and built-in knowledge. Rather than just adopting this default and telling (or not telling) the user, it is possible to suggest to the user that this is an appropriate value. The value of the concept isn't set until the user confirms the default or explicitly sets the value. (Usually, it is much simpler for the user to confirm a default than to input the same value.) For example, to handle the concept of time, the referenced time zone must be known. Assuming GMT is the most likely, the Knowledge Handler can send the Tuple:

< suggest_user time_zone GMT >.

The Forms Program will display this value in a different font to indicate that it has not been input by the user.

offer_user
In some cases, the Knowledge Handler may be able to make extensive inferences about a

subset of the possible values for a concept, but can still handle the others. For example, if the user selects a material that is described in one of the databases available to the Knowledge Handler, it need not ask for the material properties. If any other material is selected, the user will have to supply this information. One option open to the Knowledge Handler is to offer the user a list of the known materials, using the Tuple:

< offer_user material_type paper,wood,brick,concrete,stone >.

In the Forms Program, this menu appears when the right mouse button is down over the label of the concept.

In summary, the requests currently acceptable to the Dialogue Handler are:

Request	Meaning	Example
new_dialog	Switch to an new interaction program.	Run 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 re-set a field's default value.
offer_user	Present sensible options to the user.	Set menu options.
focus_user	Direct the user 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 data.	Display a concept field.
unask_user	Withdraw a request for data.	Hide a (displayed) concept field.
new_query	Ask an unexpected question.	Create a new concept field.

13.1.3 Built-in Utilities

This section lists the utility predicates built-in to the Knowledge Handler. Most of these are given in the Prolog files in ~ife/lib/uc/initial/kbs. The files contain:

File	Contents
initialise	The startup predicate and the command loop that monitors the Blackboard.
interface	Predicates to interact with the Blackboard.
utilities	Various utility predicates.
master	Predicates to handle the "master" form (see later).
defaults	Default facts (installation specific).

Any changes to these files, other than adding to the defaults file, will necessitate a rebuilding of the Knowledge Handler. This is done as follows.

```
$ cd ~ife/lib/uc/initial/kbs
$ nip -U128
   .
   .
| ?- [load_kb].
   .
   .
kb_initial created
Prolog terminated
```

$ mv kb_initial ..

The predicates of interest for the creation of a new user conceptualisation are given in the following table. In this 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).

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 (see environment example, Section 13.1.1).
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_usr(C)	
unfocus_usr(C)	
ask_user(C)	
ask_user(C, V)	
ask_user(C, K)	
ask_user(C, K, V)	
unask_usr(C)	
unask_usr(C, K)	
tell_usr(C, V)	
tell_usr(C, K, V)	
suggest_usr(C, V)	
suggest_usr(C, K, V)	
offer_usr(C, V)	
offer_usr(C, K, V)	
char_usr(T)	Appends the list of strings T to concept user_chat.
The following post a Tuple on the "u_cpt"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 the value came from.
known(C, W, K, V)	
u_cpt_got(C, V)	user_supplied, invoked when someone else sets a concept value.
The above are specialisations of:	
to_bb(A1)	These create and send a Tuple to the Blackboard.
to_bb(A1, A2)	A1 is the Blackboard area to post to.

to_bb(A1, A2, A3)	A3 is a string (usually indicates 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.
The following are general utilities in 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 to give list L.
member(E,L)	Checks if element E is on the list L.

The file "master" contains the predicates to handle the master form, defined by ~ife/lib/uc/initial/forms/master. This form is the top level IFe form, which sets up the user type and level, gets the problem identification, and opens communication with the user via a "chat area". Once this is established, the top level meta-concepts "bld_spec" and "analysis" are enabled.
Then control passed to the predicates defined in the user_conceptualisation files.

The file ~ife/lib/uc/initial/kbs/defaults contains those defaults common to all users of this installation of the IFe, such as common locations and their attributes, default climate sets, analyses and so on. This file is consulted at run-time when the Knowledge Handler is started, so can be added to at any time.

Finally, there are a number of utilities which access the Unix environment external to the IFe. Most of these are simple system calls, to get a date or a login name for example, but some invoke external database systems to extract required information. An example of this is "conlst", which accesses a database of materials and their thermo-physical properties. Since they need operating system facilities, these utilities have to be written in `C'. However, to the Knowledge Handler, they are invoked in exactly the same manner as any other predicate, for example "getdate(_Date)" will instantiate the variable _Date to today's date. The requests currently acceptable to the Dialogue Handler are:

Predicate	Function
getdate(_D)	Gets today's date and returns it in D.
proj_exists(N, S, D, I, L)	Checks for the existence of a project N, and if it exists returns the latest session number S, the start date, D, files containing the current data, I, and IFe system state, L, for reloading.
flt(N, F)	Converts an integer to a float.
get_env_var(E,V)	Puts the value of an environment variable into V.

13.2 Creating Form-Sets
It is envisaged that all IFe users will be presented with the same initial form, with subsequent forms tailored to their individual needs. The template or definition file corresponding to this initial form has been installed as

~ife/lib/uc/initial/forms/master

and is reproduced here, in annotated format, to demonstrate the use of the syntax.

Syntax	Associated Data	Meaning
new desk \ \ name	IFe #2.2a of September 1989	Beginning of the master or desk form definition: all other forms

		are nested relative to this form. The name of the desk is "/". The associated data is the title of the parent window which contains the desk (for information only).
\\ shade	white	Form background colour.
\\ data font	…fonts//fixedwidthfonts/serif.b.10	Font to be used for data.
\\ label font	…fonts//fixedwidthfonts/cour.b.10	Font to be used for labels.
\\ application font	…fonts/fixedwidthfonts/cour.r.10	Font to be used for default data.
\\ origin	3 2	Origin in characters (of the specified font size).
\\ size	132 46	Form size: dx and dy in characters.
\\ help	This is the IFe master form.	A terse help message to be displayed if the help option is selected.
\\ border	1	The line thickness for the form border.
\\ selection border	2	The line thickness for the form border, to be used when the form is selected.
end desk	End of desk form definition.	
new form		Start of a new form defined relative to the desk form.
\\ name	chat_area	Name of this form: subsequently referred to by the Dialogue Handler as "/chat_area".
\\ shade		
\\ origin	63 1.5	Origin of form in characters relative to the desk form.
\\ size	74 17	
\\ data font	…fonts/fixedwidthfonts/serif.r.10	
\\ label font	…fonts/fixedwidthfonts/serif.r.10	
\\ application font	…fonts/fixedwidthfonts/serif.r.10	
\\ border	2	
\\ selection border	2	
end form		End of form definition.
new field		Start of a concept definition on previously defined form; the parent form.
\\ name	chat_user	Name of concept; subsequently referred to by the Dialogue Handler as "/chat_area/chat_user".
\\ type	character	Field type; here a character field allowing only character inputs.
\\ origin	0.5 1.5	Origin relative to parent form.
\\ size	73 14	
\\ label string	Feedback	Concept label.
\\ label position	fit above	Put the label above concept field.
\\ help	This is a text area for general chit-chat from the IFe.	A brief message to be output if the user requests help.
\\ description	This field is used by the IFe to display non-urgent messages,	

	suggestions and session status info. It scrolls, so previous messages can be recovered.	
\\ border	0	
\\ selection border	0	
end field		End of field definition.
parent form	/	Change the parent form to be, in this case, the desk form.
new field		
\\ name	user_name	Name of concept; subsequently referred to by the Dialogue Handler as "/user_name".
\\ type	alpha + .	Field type; here an alphanumeric field plus a ".".
\\ origin	13 1.5	
\\ size	25 1	
\\ label	string	Your Name
\\ label position	left	Put the label to the left of the concept field.
\\ help	You should enter your name here.	
\\ description	This field is used to maintain a chronological record of the individuals working on the project.	
end field		
new field		
\\ name	user_type	
\\ type	popup	Field type; here a `pop-up' selections of icons defined by files of the name given under the `menu' entry below and located in directory ~ife/lib/icons. If these icon files do not exist then the file names are shown.
\\ style	key	The style of the pop-up icon.
\\ origin	40 1.5	
\\ size	64 64 pixels	Field size in pixels.
\\ label string	User type	
\\ label position	fit above	
\\ selection border	0	
\\ border	8	
\\ menu	engineer architect modeller	Icon file names or pop-up panel labels.
\\ help	User type (engineer, architect, or modeller).	
\\ description	This button identifies the currently selected user type. This dictates the style and content of the forms to be displayed during this session. An attempt will be made to ask only for information liable to be available to you, using the internal knowledge	

	to provide values for the more technical aspects. All feedback given will also reflect this bias.	
end field		
new field		
\\ name	user_level	
\\ type	button	Field type; here a two state button offering the options defined under menu below.
\\ style	key	
\\ origin	50 1.5	
\\ size	64 64 pixels	
\\ label string	User Level	
\\ label position	fit above	
\\ menu	expert novice	Button states.
\\ help	User level (expert or novice).	
\\ description	This button identifies the currently selected user level. This dictates the level and style of help/guidance given during the interaction and also influences the content of some of the forms shown. An attempt will be made to ask only for information liable to be available to this user, using the internal knowledge to provide values for the more esoteric aspects.	
end field		
new field		
\\ name	date	
\\ type	date	Field type: here a date field that offers an in-built graphical calendar facility.
\\ origin	117 0.2	
\\ size	10 1	
\\ label string	Date	
\\ label position	left	
\\ help	Today's date.	
\\ description	This field is used to time-stamp any modifications made during this session. It is used to maintain a chronological record of project modifications.	
end field		
new field		
\\ name	started	
\\ type	date	
\\ origin	25 5.5	
\\ size	10 1	
\\ label string	Date Started	
\\ label position	left	
\\ start	hidden	On form start-up do not show this concept.

\\ help	Date of first session for this project.	
\\ description	This field gives the start date for the work on this project.	
end field		
new field		
\\ name	session	
\\ type	integer	Field type.
\\ origin	57 5.5	
\\ size	2 1	
\\ label	string	Session Number
\\ label position	left	
\\ start	hidden	
\\ help	Session number of this consultation.	
\\ description	This field is used to identify subsequent sessions targeted on an active project.	
end field		
new field		
\\ name	project	
\\ type	character	
\\ origin	13 6.5	
\\ size	47 1	
\\ label string	Project Name	
\\ label position	left	
\\ help	Project name or identifier.	
\\ description	This field is used to identify the project so that the data can be subsequently retrieved/ used.	
end field		
new field		
\\ name	m_focus	
\\ type	label	Field type: a label only, that is no input required.
\\ origin	19.5 9.5	
\\ size	0 0	
\\ label string	Focus for discussion.	
\\ label position	fit above	
\\ help	Push a button to change topic of discussion.	
\\ description	These buttons switch the focus of discussion to the requested topic. The relevant forms will be displayed below (existing ones may disappear). It is suggested that buttons are worked through from left to right to minimise specification of redundant information.	
\\ selection border	0	
\\ border	0	
end field		
new field		
\\ name	b_analysis	

\\ type	button	
\\ origin	8 10	
\\ size	9 1	
\\ label position	fit above	
\\ label colour	black	Colour of label only.
\\ menu	analysis	Only a single state is defined.
\\ start	hidden	
\\ help	Select to enter analysis definition mode.	
\\ description		This button switches the focus of discussion to the appraisal specification.
\\ selection border	0	
\\ border	1	

end field

new field	
\\ name	b_bld_spec
\\ type	button
\\ origin	22 10
\\ size	9 1
\\ label position	fit above
\\ label colour	black
\\ menu	building
\\ start	hidden
\\ help	Select to enter building description mode.
\\ description	This button switches the focus of discussion to building description.
\\ selection border	0
\\ border	1

end field

The result of this template file is shown in figure 2 which can be reproduced by issuing the command "forms master".

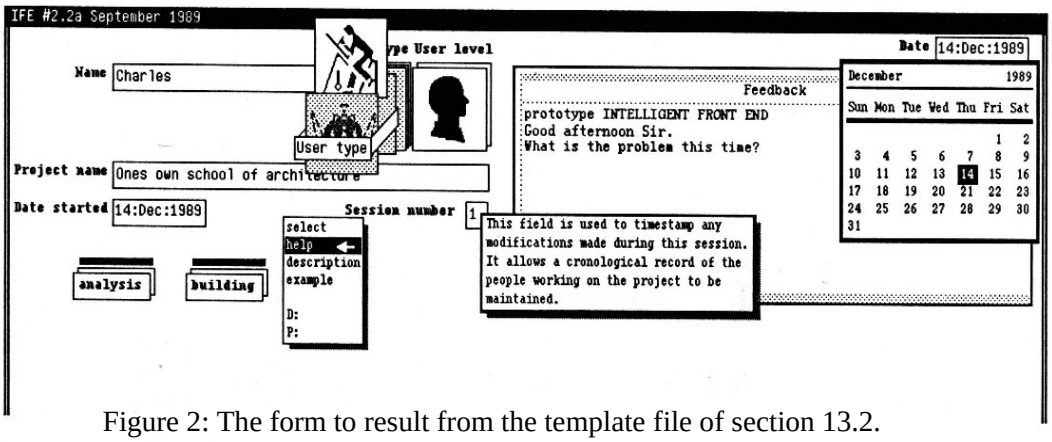


Figure 2: The form to result from the template file of section 13.2.

The complete form creation syntax is listed in the following table.

Field Attribute	Option	Description
new/ end	desk	start/ end of desk (root form) definition
	form	start/ end of form definition

	field	start/ end of field definition
parent form	form name (or \ for desk)	to return form or field definition to being relative to named form
name	any unique character string	assigns a name to a field or form
type		assigns a data template to a field:
	alpha num	alphanumeric field
	character	character field
	integer	integer only
	real	real only
	date	system date set + pop-up edit facility
	popup	displays a stack of images or text
	button	displays one image (or text) from a list
	file	allows a file identifier to be loaded from a directory browser
	graphics	raster graphics field
	label	non-editable text field
	menu	displays dynamic menu
	viewer	displays single viewer .pic file
start		defines the field state on start-up:
	visible	displays the field
	hidden	hides the field
origin	x y	positions field in parent form character units
	x y pixels	positions field in pixels
size	x y	field size in parent form character units
	x y pixels	field size in pixels
shade		determines the background shade for forms:
	white	the default
	light grey	
	mid grey	
	dark grey	
	black	
	left hatching	
	right hatching	
label string	any character string	field label
label position		position of the label string:
	left	
	right	
	above left	
	below left	
	above right	
	above	
	fit above	
	fit below	
	fit right	
	fit left	
label font	font name	define the label font
data font	font name	define the field font
application font	font name	define the application font
menu	menu entries separated by a newline	defines the menu options
border	x	width of border around fields and forms
selection border	x	width of border around fields on

		selection
description	text	attaches to the description option of the menu associated with a field's label
help	text	attaches to the help option of the menu associated with a field's label
default value	data	defines the default field data
current value	data	defines the start-up field data
previous value	data	defines the previous data on the stack
icon image	exrep bitmap file name	defines icon for form

Using this syntax it is possible to create a complete hierarchy of nested forms of arbitrary complexity. If any given form contains fields which are defined outside the physical boundary of the form, then that form will be scrollable.

The engineering conceptualisation as installed within the IFe has the following forms defined (located in ~ife/lib/uc/engineer/forms).

Form	Purpose	Invocation
analysis	Definition of required appraisals.	Button on IFe master form (/).
bld_spec	High level building description information (for example location and function).	Button on IFe master form (/).
bld_spec_focii	A scrollable form of buttons allowing a user to focus on specific topics (such as building geometry, construction or operation).	Defined on bld_spec.
bld_browse	A building design browse facility	Button on bld_spec.
geometry	Building geometry definition on a zone-by-zone basis.	Button on bld_spec_focii.
g_form_fill	Zone geometry definition by form filling.	Button on geometry.
vertex_list_2d	Vertex list form.	Defined on g_form_fill.
g_cad_file	Building geometry definition by importing a foreign CAD file	Button on geometry.
g_draw	Invoke geometry modeller to define building/ zone geometry.	Button on geometry.
construction	Constructional attribution of defined geometry.	Button on bld_spec_focii.
usage	Zone operation definition.	Button on bld_spec_focii.
connectivity	Definition of building boundary conditions.	Button on bld_spec_focii.
plant	Definition of plant components and networks.	Button on bld_spec_focii.
control	Definition of building/ plant control loops.	Button on bld_spec_focii.
airflow	Definition of building leakage and pressure distribution.	Button on bld_spec_focii.
site	Definition of site obstructions.	Button on bld_spec_focii.
shading	Definition of shading time-series.	Button on bld_spec_focii.

Figure 3 shows a typical screen image resulting from an IFe session with the engineering conceptualisation active. It is suggested that the interested reader should study the relationship between these forms, the corresponding knowledge bases and the IFe session dialogue.

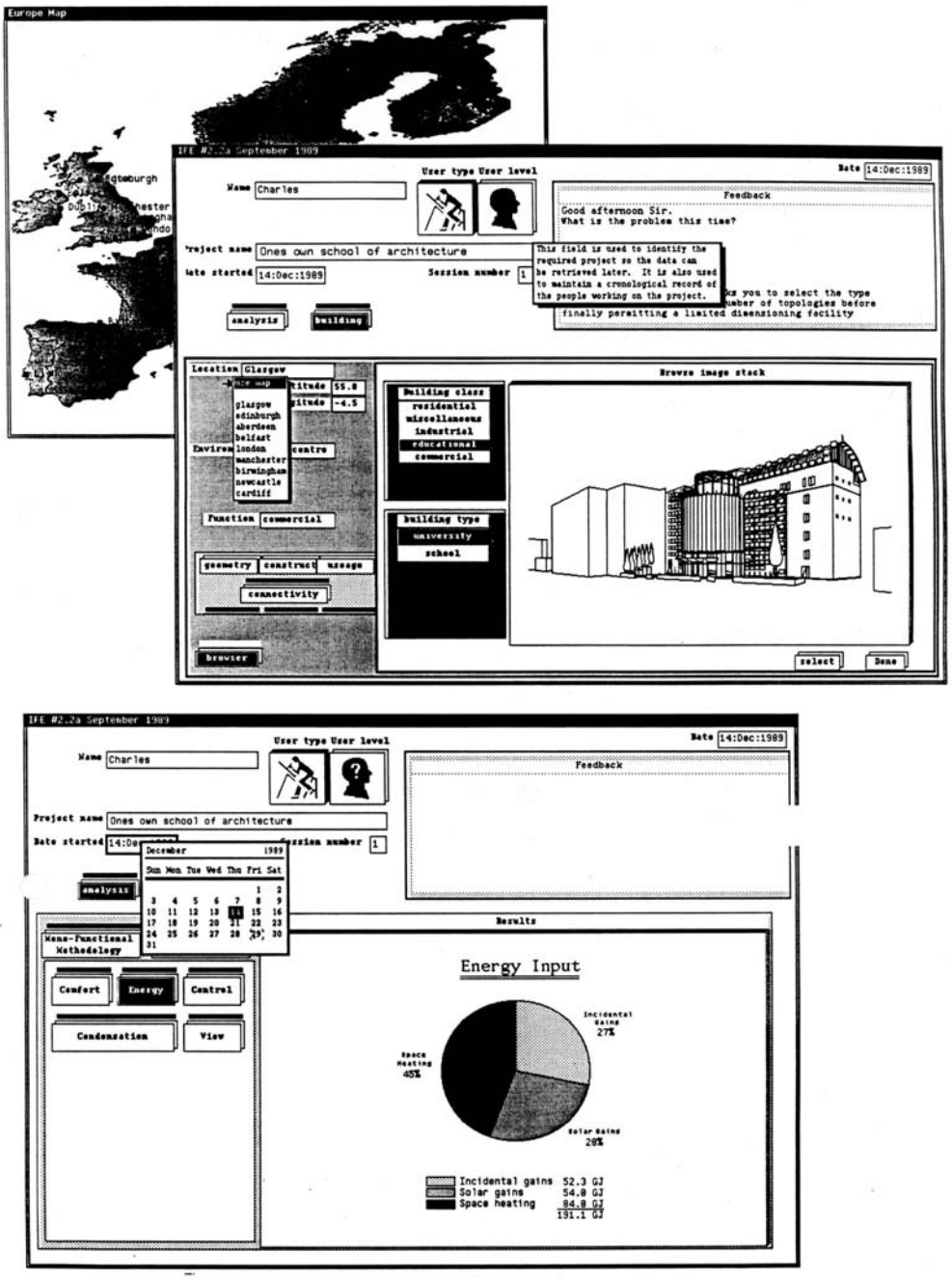


Figure 3: The engineer user conceptualisation.

One interesting feature of this user conceptualisation is a geometry browse feature. This is established as form "bld_browse" which is invoked from a button on the "bld_spec" form. Associated with this button is a Prolog predicate which starts the browse form and posts a Tuple to the Blackboard requesting that the browse program be run. The browse program scans a standard directory (~ife/lib/uc/uc/buildings), which contain a set of subdirectories corresponding to the different building types deemed appropriate to the current user conceptualisation. Within each of these directories may be placed any number of sub-directories which, in turn, contain any number of images held in exrep format. Exrep files are generated from a source bitmap - typically, but not necessarily, a Sun raster – using the conv raster conversion program (file ~ife/bin/conv). The browse facility allows, for example, design exemplars to be offered to an IFe user so that entire models can be defined without the need to explicitly enter topographical and topological information. It would also permit

related attribute data (constructional information and the like) to be stored and retrieved. This would require the incorporation of an additional predicate within the "bld_spec" knowledge base.

13.3 Creating/ Installing User Conceptualisations

The mechanism by which the IFe selects between different user conceptualisations is described in section 13.4. In essence, the directory ~ife/lib/uc contains one subdirectory for each user conceptualisation. This directory is given the same name as the conceptualisation, for example engineer. When the User Model wishes to utilise a particular user conceptualisation, it creates a symbolic link ~ife/lib/uc/uc which points to the required directory. If the user conceptualisation changes during a session, the User Handler will change this link accordingly. When the user first focuses on a meta-concept, the focus_concept predicate accesses the required predicates by (re)consulting the Prolog file of the same name in ~ife/lib/uc/uc/kbs. In this way the predicates appropriate for the current user conceptualisation are automatically accessed. It should be clear, therefore, that the naming and positioning of new user conceptualisations in the IFe directory structure is critical.

In summary, there are ten stages in creating a new user conceptualisation as follows.

1. Identify the concepts that the user can use to specify the building and the required analysis.
2. To help organise the collection and processing of the user inputs, collect these concepts into related groups to form a hierarchy of meta-concepts. (It is probable that the identification of concepts in Stage 1 will already have been based on such a hierarchy.)
3. For each concept, write the predicates to validate and store the input. These are placed in a file with the same name as the enclosing meta-concept.
4. Design the form that corresponds to these concepts. This requires selecting the best field type (text, menu, button, etc) to help the user input the data.
5. Add the predicates to handle the switch of focus. Usually, when a meta-concept is selected, some initialisation is necessary - for example it may be appropriate to inform/ remind the user of the information already held/ inferred (by re-displaying the form for example). Similarly, when a meta-concept is de-selected this might trigger the calculation of default values for un-set concepts.
6. Add to the form the necessary fields (usually buttons) to allow the user to switch the focus of discussion to another meta-concept (form) and back again.
7. Install the form-set in ~ife/lib/uc/?/forms and the knowledge bases in ~ife/lib/uc/?/kbs, where ? is the typename for this user conceptualisation.
8. Update the user_type field on the master form to present the new option - in file ~/lib/uc/initial/forms - and add the names of users of this type to the User Handler defaults file in file ~ife/lib/um/kbs (there is no need to recompile this knowledge base).
9. Test the knowledge base by, firstly, running nip and invoking the predicates directly and, secondly, running ~ife/bin/ife_kh and typing Tuples directly into it. See the file ~ife/src/tmp/bb_to_kh.script for a sample session.
10. Finally, test the form-set by, firstly running ~ife/bin/forms with the forms, and then by running ~/bin/ife_dh and typing Tuples into it. See the file ~ife/src/tmp/bb_to_dh.script for a sample session.

13.4 User Handler

The User Handler is, like the Knowledge Handler, an inference engine based on Prolog. Together with its associated knowledge bases it provides the IFe's `user modelling' function. Since the provision of a full user model would be a complete research topic in itself, only a fairly rudimentary facility has been developed to date. Basically, the user is stereotyped according to the categorisation given in Section 2 and, based on this, a pre-defined interaction style/ content is used. While this has proved adequate for the current system, all the necessary hooks have been developed to allow the later development of a much more sophisticated user model as and when this proves necessary.

The User Handler has two main tasks. The first is the collection of the information upon which a user model would operate. The second is the modification of the interaction with the user in accordance with the recommendations of the user model. These will be addressed in turn.

13.4.1 Monitoring the User Dialogue

The underlying data for this monitoring is clearly the raw user inputs, as picked up from the user_dialog area on the Blackboard. The Dialogue Handler posts here everything the user inputs as a user_said Tuple. It also posts requests for help, as a `user_help' Tuple and any error it detects as a `user_error' Tuple. These are picked up by the User Handler and used to trigger the appropriate predicates in the user model. The mechanism used is almost identical to the Knowledge Handler as already described in Section 13.3. A typical predicate for analysing the input would be:

```
user_said(_Concept, __, __):-          % don't care about the concept
        user_said(_Concept).           % value so strip it off
user_said(_Concept, __):-
        user_said(_Concept).
user_said(_Concept):-                  % handle new user input
        time(_Now),                    % get current time
        retract(lasttime(_Lasttime)),  % get time of last user input
        _Delay is _Now - _Lasttime,
                                       % collect info on how long user took to
                                       % assimilate the new meta_concept
                                       % (i.e. to read & understand the form)
                                       % by looking at the delay until he/she
                                       % inputs the 1st piece of data
        ( focussed(_Meta_concept) ->   % we have 1st input for this mc
                assert(mc_assim(_Meta_concept, _Delay)), % record delay
                ( _Delay < 120, _Delay > 20,
                        feedback_user(assim)    % having difficulties?
                ;
                        true      % delay > 2 mins => gone to coffee
                ),
                retract(focussed(_Meta_concept))
        ;
                true                   % wasn't new mc
        ),
                                       % collect info on how often the user
                                       % vacillates about this concepts value
                                       % (ie cycles through menu values)
                                       % as this indicates uncertainty
        (retract(previous_concept(_P_c)) ; true), % get last inputs
        (retract(penultimate_concept(_Pp_c)) ; true),
        ( ( _Concept = _P_c ; _Concept = _Pp_c ) ->
                ( retract(vacillate(_Concept, _No)) ; _No is 0),
                _New_no is _No + 1,
                assert(vacillate(_Concept, _New_no)),   % record
                ( _New_no > 5 ->        % >5 changes => confusion?
                        feedback(vacil),
                        assert(vacillate(_Concept, 0))    % reset
                ;
                        assert(vacillate(_Concept, _New_no))    %record
                )
```

```
        ;
                true                    % wasn't same cpt
        ),
        assert(previous_concept(_Concept)),             % update
        assert(penultimate_concept(_P_c)),
                                        % check if the user is having to override
                                        % the Knowledge Handler often
        ( retract(no_inputs(_No_i)) ; _No_i is 0 ),
        ( knownw(_Concept, __, __, kb_set) ->   % is overriding kb?
                ( retract(kb_override(_Num)) ; _Num is 0),      % yes
                _New_num is _Num + 1,
                ( _New_num > _no_i / 5 ->       %limit of 20% overrides
                        feedback(override),
                        assert(kb_override(0))  % reset counter
                ;
                        assert(kb_override(_New_num))           % record
                )
        ;
                true                    % not set by kb
        ).
user_said(_Concept).                    % user_said predicates should never fail
```

The first few lines just act to strip off the concept's value, since only the fact that the concept has been addressed by the user is of significance. The predicate then goes on to extract some statistical information about the user interaction and makes some recommendations. This particular predicate doesn't take any action, leaving that to the user, but clearly it could be enhanced to make decisions and carry them out. Three main pieces of information are extracted from the input: the rate of interaction, the amount of dithering on the user's part and the error rate of the Knowledge Handler's suggestions:

- A lengthy delay before user responses generally implies either that a reasonable amount of thought is going into the input or that the user has been interrupted. If there is a consistently slow response, this could indicate that the user is having trouble with what is being asked. Consistently slow responses to just one class of concepts could indicate that the requested data is not easily available in the required form.
- If a user is constantly backtracking and changing the last few inputs, this might suggests that she/ he is uncertain, either about what is being asked or about the data that is being supplied.
- Or if the user is constantly overriding the default values suggested by the Knowledge Handler, this might suggest that the current user conceptualisation is inappropriate.

The above should give a feel for the potential of the User Handler to monitor the user interaction. Of course, the difficult question is deciding what to do about the conclusions. Above, we just make recommendations to the user and leave it to her/ him to make the decisions. Extra information - such as the user's goals and the rate of progress towards achieving them, or the proportion of irrelevant information volunteered – could be made available to help the user model to make the decisions itself. This is a major research area.

13.4.2 Modifying the User Interaction

On start up, after an explicit user request, or as the result of a user model recommendation, it may become necessary to change the way the user interaction is being handled. Corresponding to the different ways of categorising the user given in Section 2, it may be wished to modify that aspect of the user interaction to a simpler or to a more sophisticated manner. The easiest course of action, from the User Handler's point of view, is to change the

level of expertise the user is assumed to possess. This is handled within the Knowledge Handler. By posting a `user_level' Tuple to the Knowledge Handler, the quantity and content of the feedback to the user will be changed (see the `feedback' predicate described in Section 13.3). Furthermore, the Knowledge Handler may modify the questions being asked of the user, or even omit some questions altogether, depending on the assumed expertise of the user. Similarly, there is the potential to ask the Dialogue Handler to modify the style of the interactions by switching to a question/ answer format rather than form-filling.

Of more interest here is the modification of the user conceptualisation underlying the Knowledge and Dialogue Handlers. As mentioned earlier, each user conceptualisation consists of a set of knowledge bases together with a matched set of forms. These are held in a directory called after the user type to which they apply, e.g. engineer, architect and so on. However, the Knowledge and Dialogue Handlers both access these files via fixed pathnames, ~ife/lib/uc/uc/kbs in the case of the Knowledge Handler. The second uc directory is actually a symbolic link pointing to a user conceptualisation directory (engineer for example). By changing this link, the User Handler can switch the current user conceptualisation to an alternative. From that point on the new conceptualisation is used whenever the Knowledge or the Dialogue Handlers want to interact with the user. All existing input, stored on the Blackboard, is preserved. The only thing left to do is to ask the Knowledge Handler to redisplay the existing data in accordance with the new user conceptualisation. The mechanism to achieve this is shown in the following predicate, which performs the initial categorisation of the user.

```
classif_user :-
        get_login(_User_name),                          % find out who user is
        user_type(_User_name, _User_type),              % get their default type
        ( u_type(_User_type, __)                        % is it already set?
        ;
                ( u_type(__, kset)                      % if user set it, leave it
                ;
                        kset(user_type, _User_type),    % record new type
                        ( abolish(u_type,2) ; true ),
                        assert(u_type(_User_type, kset)),
                        tell_usr(user_type, _User_type),
                        name(_User_type,_User_type_str), % change the uc link
                        append("${IFE_HOME}bin/change_cpt_set
",_User_type_str,_Cmd),
                        shell(_Cmd)
                )
        ),
        user_level(_User_name, _User_level),            % as for user type
        ( u_level(_User_level, __)
        ;
                ( u_level(__, kset)
                ;
                        kset(user_level, _User_level),
                        ( abolish(u_level,1) ; true ),
                        assert(u_level(_User_level)),
                        tell_usr(user_level, _User_level) % tell the KH
                )
        ).
```

The current user type is held locally as a `u_type' Prolog fact. If it hasn't changed, or was set by the user, this startup classification predicate shouldn't modify it. The modification of the uc link mentioned earlier is actually carried out by a Shell script, `change_cpt_set', invoked by

the Prolog `shell(_Cmd)' predicate.

The default user type is held in a defaults file, ~ife/lib/um/kbs/defaults. This contains all the user/ installation data that the user model requires. Because it is loaded at run-time, it can be updated without needing to recompile the user model knowledge base (kb_um) or to modify the Knowledge Handler's knowledge bases.

14. Linking to an Application Program

The IFe can be made to `drive' any application program which runs under the Unix operating system. To do this a `performance assessment script' is developed and, after testing, installed within the ~ife/lib/uc/?/appraisals directory, where ? is the name of the user conceptualisation to which the script relates (it is envisaged that different users will require different performance assessment methodologies). Script invocation is via a Prolog predicate usually, but not exclusively, matched to a form button (see the "~ife/lib/uc/engineer/forms/analysis" form template file and its “~ife/lib/uc/engineer/kbs/analysis" matched knowledge base).

The function of the Appraisal Handler is to manipulate discrete performance assessment methodologies established in parameterised form. When an IFe user requests a particular appraisal, it is the job of the Appraisal Handler to locate the corresponding methodology script, determine the appropriate substitutions for its parameters and post the script to the Blackboard (where it will be matched to a corresponding data-set posted by the Data Handler before being passed to the Application Handler for execution).

In the approach the Unix Bourne Shell [24] is used as a pseudo expert system shell. Shell scripts are designed to coordinate the operation of any number of application programs against the rules and relations of a particular performance assessment methodology. These rules and relations are established in parameterised form so that they can be determined at script invocation. Within the script application programs (such as ESP), Unix programs such as editors and graphics entities such as window managers can be used to perform the operations required by the methodology. The computational path to be followed at any stage in the script will then depend on the performance data to emerge at run-time and on the embodied rules. In this sense a script can be viewed as a modestly intelligent design assistant with knowledge of how to operate and sequence the various application programs while adhering to an imposed methodology.

A thorough appreciation of Shell programming can only be obtained from study of the syntax involved and practice. As an aid to such a study, two ESP scripts are presented here - one simple, the other comprehensive. These have been selected from the following list as currently developed for use within the IFe environment.

Heating plant sizing.
Cooling plant sizing.
Climatic severity assessment.
Plant control strategy appraisal.
Condensation report.
Summer overheating analysis.
Building zone dimensions take-off.
Comfort analysis
Annual energy demand and causal breakdown.
Solar utilisation report.
Air flow analysis.
Perspective view generation.
Animated view.
Cost-in-use report.
Regulations compliance report.

The simple script corresponds to heating plant sizing and has the following form.

```
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 "Useage <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
~
```

Firstly, the variable IFe_HOME is located. 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 assigned a default value before being reassigned as appropriate on the basis of the command line options 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 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.

ESP Script 1: Heating Plant Sizing.

Climate file: /usr2/ife/lib/tmp/climate
Configuration file: /usr2/ife/lib/tmp/building
Configuration description: ESP standard test

Control file name: /usr2/ife/lib/tmp/control

Building save option: No results saved
No. of warning messages: 0

Simulation period: 1 day – 9th January
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: 9th January

Zone	Max Air temp (°C)	Min Air temp (°C)	Max heating (kW)	Max cooling (kW)	Heating energy (kWh)	Cooling energy (kWh
1	16.00 @ 08:30	6.58 @ 06:30	0.91 @ 08:30	0. @ 00:30	9.9	0.
2	20.00 @ 08:30	7.58 @ 06:30	1.81 @ 08:30	0. @ 00:30	19.6	0.
3	20.09 @ 17:30	8.08 @ 06:30	0.89 @ 10:30	0. @ 00:30	10.4	0.
4	16.71 @ 23:30	6.65 @ 09:30	1.78 @ 22:30	0. @ 00:30	3.6	0.
5	-0.14 @ 13:30	-4.22 @ 08:30	0. @ 00:30	0. @ 00:30	0.0	0.

All zones :

Max. Temp. = 20.1 in Zone 3 @ 17:30 hrs
Min. Temp. = -4.2 in Zone 5 @ 8:30 hrs
Max. Heating = 1.8 in Zone 2 @ 8:30 hrs
Max. Cooling = 0. in Zone 1 @ 0:30 hrs

Total heating energy = 43.45 kWh
Total cooling energy = 0.0 kWh

A more comprehensive performance assessment script now follows. In this case the script is endowed with more intelligence so that it can make decision on the basis of the predicted performance. Its purpose is to undertake a comfort analysis with the following mission:

- to initiate and control the simulation processing;
- to seek out building zones which are uncomfortable according to user specified (or default) comfort criteria;
- to recover and present statistics on comfort prevailing in uncomfortable areas;
- to determine the cause of the problem; and
- to provide a comprehensive report on comfort performance, including problem causes.

With reference to figure 4, which shows the program modules of the ESP system, this script is constructed as five interrelating sub-scripts.

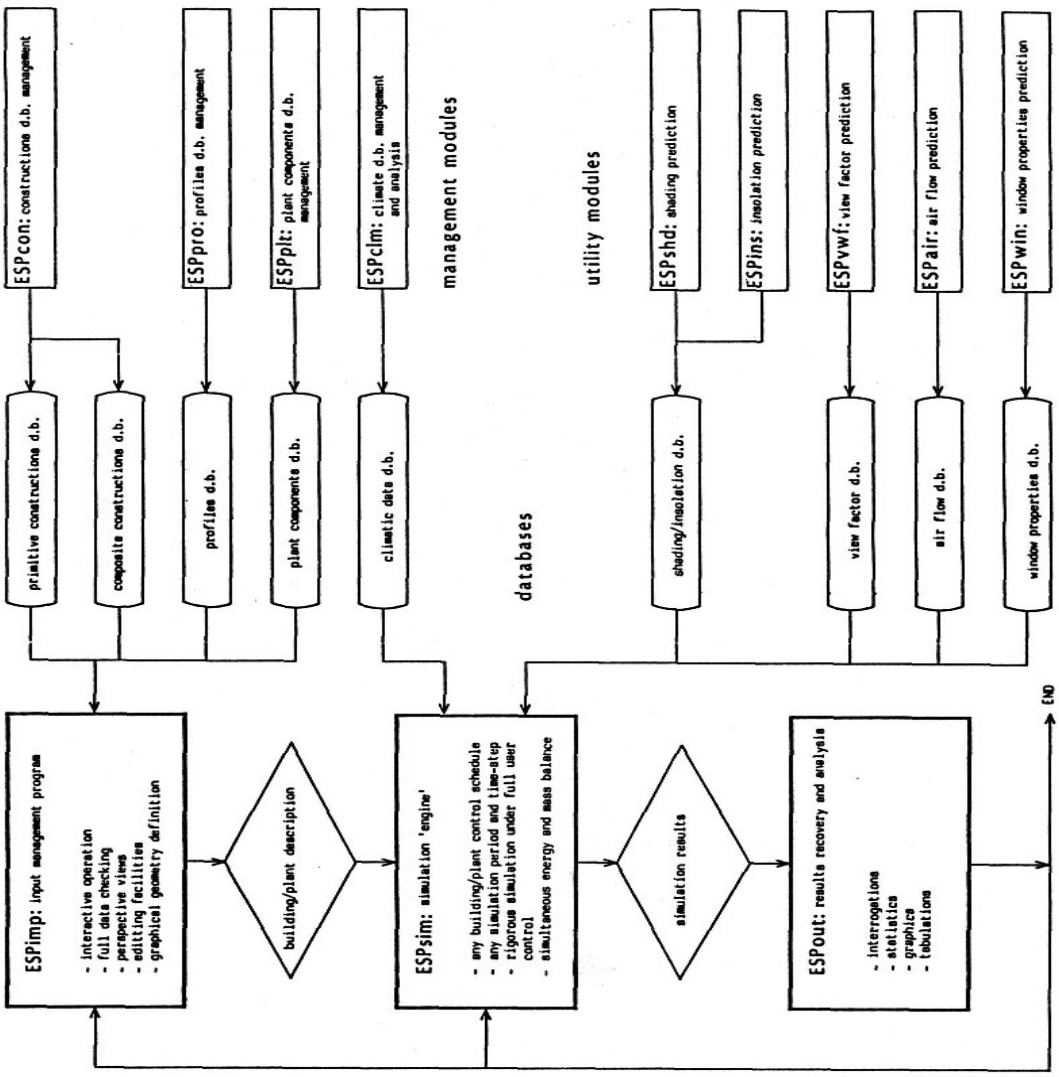
The first sub-script performs a simulation before spawning a number of new windows required later for results reportage. The second recovers the state variables which quantify comfort and groups zones according to whether or not they violate the comfort criteria. Summary statistics on the worst zone offenders are then output. The third script investigates the cause of any discomfort. The last two scripts are special in that they exist to compliment the Unix process awk: in essence they control the extraction of information – such as the cause of discomfort or the location of the worst zone - from the data sets transferred to awk from the ESP modules.

The actual scripts are as follows.

***** Script 1 *****

```
        # Comfort Assessment: Version 1.4 of January 1988
clear
# Test to determine if IFe directory path has been
        # set up in .cshrc file.

if test "X$dir" = "X"
then
        echo "Please source .iferc_b (Bourne Shell) or .iferc_c (C Shell)."
```



Shown here are the connections between the various program and database modules which comprise ESP. These include:

ESPimp A program which allows the interactive definition of the combined building/plant network to be subjected to some weather influence and simulated over time. All items, as input, are processed through legality and 'within acceptable range' checks before being located in an output file for transfer to the simulation engine ESPSIM. Other facilities include a range of interactive editting commands and automatic access to construction, casual gain profile and plant component databases.

ESPcon A program to manage (create, modify, delete, list) a primitive constructions database containing the thermophysical properties (conductivity, density, specific heat, solar absorptivity and emissivity) of a number of miscellaneous homogeneous elements. This program will also allow the creation and editting of a second project-related database containing whole composite multi-layered constructions formed from elements extracted from the primitives database.

ESPpro A program to manage a casual gain profiles database containing any number of standard or project-specific profiles defining the time varying heat gain from occupants, lights etc.

ESPplt A program to manage the plant components database containing component descriptions, component models (in matrix form) and essential manufacturers data.

ESPshd A program to predict the time-series shading of external opaque and transparent surfaces as caused by surrounding site and facade obstructions.

ESPins A program to predict the time-series insolation of internal opaque and transparent surfaces as caused by solar penetration through windows.

ESPvwf A program to compute view factors between the surfaces bounding any building zone.

ESPair A program which simulates the flow of air throughout any multizone system (where each zone is connected by flow resistances representing windows, doors, cracks, etc) subjected to boundary wind loading.

ESPwin A program to perform a spectral analysis of a multilayered window system to determine overall transmission, reflection and absorption.

ESPclm A program to manage the climatological database allowing creation (by prediction), modification and analysis of the hourly time-series values of the various climatic parameters retained.

ESPsim Is the central simulation 'engine' which simulates energy flow by a heat balance matrix inversion technique of implicit enumeration. The technique is unconditionally stable for all computational time-increments and operates in conjunction with hourly values of various climatological parameters recovered from the climatological database. As the model steps though the user-defined time-increments it continuously predicts and environmental and energy status of the simulated building. This is done by placing 'nodes' at many points of interest, for each node performing a heat balance between all interacting regions (over time and space dimensions), and then simultaneously solving the resulting set of equations at each time-step.

ESPout is the output program which operates on the simulation results located in the results database by ESPSIM. A variety of output facilities are available: visualisation, result interrogation; graphical display; statistical analysis; and tabulations.

Figure 4: Program modules of the ESP system.

```
                exit
fi

echo " Start of comfort script."
echo
echo "***** COMFORT SCRIPT COMMENCES" >$dir/tmp/comfort_trace
echo >>$dir/tmp/comfort_trace
                                        # Set up defaults and
                                        # process command line options.
comfort_index="res"                     # Resultant temperature or "air"
oh_tmp=28                               # for air temperature.

comfort_lock=$dir/tmp/comfort_lock
comfort_results=$dir/tmp/comfort_results
building=$dir/defaults/defcfg
climate=$dir/defaults/defclm
start="17 7"
finish="17 7"
timesteps=1
information=0

if test $# -ne 0
then
        for i do
                case "$i" in
                -h)     information=1; shift;;
                -help)  information=1; shift;;
                -i)     comfort_index=$2;shift;shift;;
                -o)     oh_tmp=$2;shift;shift;;
                -f)     comfort_results=$2; shift; shift;;
                -c)     climate=$2; shift; shift;;
                -b)     building=$2; shift; shift;;
                -p)     start="$2 $3"; shift; shift;
                        finish="$2 $3"; shift; shift; shift;;
                -t)     timesteps=$2; shift; shift;;
                --)     shift;;
                -*)     echo "Unknown option: $i. Type comfort -h"
                        exit 2;;
                esac
        done
fi

if test $information -eq 1
then

echo
echo
echo " Command line options: information            -h"
echo "                       comfort index (res or air) -i index"
echo "                       comfort criteria          -o criteria"
echo "                       results file             -f filename"
echo "                       climate file              -c filename"
echo "                       building                 -b filename"
echo "                       period                  -p sd sm fd fm"
```

```
echo "                    timesteps               -t timestep"
echo " Default: comfort -i d -o 28 -p 17 7 17 7 -c clm67"

exit

fi
                    # echo files used
echo
echo "                Problem: ${building}"
echo "                Climate: ${climate}"
echo "                        Analysis period from ${start} to ${finish}"
echo "                Comfort index is ${comfort_index} at ${oh_tmp}"
echo
echo " Simulation commencing, please wait ....."
echo

rm -f ${comfort_results}                        # Might be user-defined file.
rm -f ${comfort_lock}

                                                # Simulate using parameters,
                                                # saving output for analysis.

echo >>$dir/tmp/comfort_trace
echo "***** comfort: run sim" >>$dir/tmp/comfort_trace
echo >>$dir/tmp/comfort_trace

sim  >>$dir/tmp/comfort_trace    <<~
-6
${climate}
1
${building}
y
1
3
${comfort_results}
${start}
${finish}
${timesteps}
s
n
y
>
-
f
~

echo
echo " Simulation complete; analysis commencing, output"
echo "                    directed to separate windows."


          if test $comfort_index = res
          then
            comfort_index="d"
            graphpic="f"
```

```
        fi

        if test $comfort_index = air
        then
          comfort_index="c"
          graphpic="a"
        fi
                                                # Export required data.
export comfort_results comfort_lock oh_tmp comfort_index graphpic
                                                # Get perspective from
                                                # out & viewer.

rm -f $dir/tmp/comfort_persp_in

echo >>$dir/tmp/comfort_trace
echo "***** comfort: running out to get viewer input file" >>$dir/tmp/comfort_trace
echo >>$dir/tmp/comfort_trace

out >>$dir/tmp/comfort_trace 2>/dev/null <<.
-6
${comfort_results}
y
n
a
$dir/tmp/comfort_persp_in
f
.

echo >>$dir/tmp/comfort_trace
echo "***** comfort: running viewer to get perspective" >>$dir/tmp/comfort_trace
echo >>$dir/tmp/comfort_trace

viewer >>$dir/tmp/comfort_trace 2>$dir/tmp/comfort_persp_out <<.
-6
$dir/tmp/comfort_persp_in
v
s
b
.
                                # display graph
mv $dir/tmp/comfort_persp_out $dir/tmp/comfort_graph
tektool -Ws 850 706 -Wp 200 100 -WP 1078 750 -Wl "perspective view" -r
comfort_graph_sun &
                                                # Get zone temperatures
                                                # from script s20_worstz.
echo >>$dir/tmp/comfort_trace
echo "***** comfort: spawning new window with comfort_worstz running"
>>$dir/tmp/comfort_trace
echo >>$dir/tmp/comfort_trace

shelltool $dir/binsh/comfort_worstz -W1 "worst zone" -Wp 10 12 -Wh 22 -WP 1080 512 -
WL "worst" &

while test ! -f ${comfort_lock}                 # Hang around until lock file appears.
do
```

```
sleep 5
done                                    # It's here: check for overheating,
                                        # lock file identifies zone.
if test -s ${comfort_lock}
then
                                        # Lock file has worst zone in it.
echo >>$dir/tmp/comfort_trace
echo "***** comfort: spawning new window with comfort_cause running"
>>$dir/tmp/comfort_trace
echo >>$dir/tmp/comfort_trace

shelltool $dir/binsh/comfort_cause -W1 "cause" -Wp 10 412 -Wh 27 -WP 1080 592 -WL
"cause" &

fi


***** Script 2 *****

echo "I will now locate the worst zone....."

                                # Use out to get max & min zone temperatures.
echo >>$dir/tmp/comfort_trace
echo "***** comfort_worstz: get zone comfort temperatures from out"
>>$dir/tmp/comfort_trace
echo "***** comfort index is ${comfort_index}" >>$dir/tmp/comfort_trace
echo >>$dir/tmp/comfort_trace

out >>$dir/tmp/comfort_trace  2>$dir/tmp/comfort_pt  <<.
-6
${comfort_results}
y
n
b
${comfort_index}
n
-
f
.

                        # Use awk to get overheating zones, ranked in PZ.
PZ=`(echo OHT $dir/tmp/oh_tmp ; cat $dir/tmp/comfort_pt) | awk -f
$dir/binsh/comfort_awk_pt `

if test "X$PZ" = "X"
then
    echo "Congratulations!"
    echo "All zones are comfortable according to the selected criteria."
    echo
    echo
    echo "Zone summary table follows ...."
    echo
    cat $dir/tmp/comfort_pt
    > $dir/tmp/comfort_lock
    echo
```

```
    echo
    echo
    echo
    echo "Type ctl-c to terminate."
    read xa                         # Hang around until told to go away.
    exit 0                          # Normal exit (0) to indicate no overheating.
fi

echo "The following zones (rank ordered) suffer discomfort: $PZ."

                                    # Locate worst occupied (or worst) zone.
WZ=
for i in $PZ
do
   echo "Now checking zone $i for occupants."
   if test "X$WZ" = "X"
   then
         WZ=$i                              # Worst Zone
   fi
   impb  <<.
      $i
.
   if test $? -eq 1                 # impb error exits if zone occupied.
   then
                                    # impb error exit.
         echo "Occupied."
         echo "Worst discomfort therefore occurs in occupied zone $i."

         OZF=1                      # Occupied Zone Found
         WZ=$i
         break
   else
         echo "Not occupied."
   fi
done

if test "X$OZF" = "X"
then
                                    # No impb error exit.
   echo "Worst discomfort therefore occurs in unoccupied zone $WZ."
fi
echo
echo "Zone summary table follows ...."
echo
cat $dir/tmp/comfort_pt
echo $WZ >$dir/tmp/comfort_lock     # Put WZ in lock file for comfort_cause.
echo
echo "Graphs of zone performance will appear in separate windows."
echo "Type ctl-c to terminate."
read xa                             # Hang around until told to go away.


***** Script 3 *****

echo " I will now determine the cause of the"
```

```
echo " discomfort in the worst zone"
echo
echo " A zone energy balance follows....."

                                        # Get worst zone from lock file
                                        # or user if no lock file.
if test "X$comfort_lock" = "X"
then
        comfort_results=$dir/tmp/comfort_results
            echo -n "Zone to be examined? "
            read WZ
else
            WZ=`cat $comfort_lock`
fi
echo >>$dir/tmp/comfort_trace
echo "***** comfort_cause started: worst zone is ${WZ}" >>$dir/tmp/comfort_trace
echo >>$dir/tmp/comfort_trace
                                        # Now get frequency distribution.
echo >>$dir/tmp/comfort_trace
echo "***** comfort_cause: get freq. dist. from out" >>$dir/tmp/comfort_trace
echo "***** graphpic is ${graphpic}" >>$dir/tmp/comfort_trace
echo >>$dir/tmp/comfort_trace

out >>$dir/tmp/comfort_trace 2>$dir/tmp/comfort_graph <<.
-6
${comfort_results}
y
n
4
1
${WZ}
c
d
${graphpic}
10
y
1
4
-
-
f
.

tektool -Ws 850 706 -Wp 200 100 -WP 1078 750 -Wl "frequency distribution" -r
comfort_graph_sun &
                                        # Create graph of temp of worst zone.
echo >>$dir/tmp/comfort_trace
echo "***** comfort_cause; get graph of worst zone from out" >>$dir/tmp/comfort_trace
echo >>$dir/tmp/comfort_trace

out  >>$dir/tmp/comfort_trace  2>$dir/tmp/comfort_wzt <<~
-6
${comfort_results}
y
n
```

```
c
a
4
1
${WZ}
a
f
g
b
!
-
-
f
~
                                        # Draw graph in new window.
cp $dir/tmp/comfort_wzt $dir/tmp/comfort_graph
tektool -Ws 850 706 -Wp 200 100 -WP 1078 750 -W1 "worst zone profiles" -r
$dir/binsh/comfort_graph_sun &

                                        # Get energy balance for worst zone.
echo >>$dir/tmp/comfort_trace
echo "***** comfort_cause: get energy balance pie from out" >>$dir/tmp/comfort_trace
echo >>$dir/tmp/comfort_trace

out >>$dir/tmp/comfort_trace 2>$dir/tmp/comfort_pie <<~
-6
${comfort_results}
y
n
4
1
${WZ}
c
e
c
-
-
f
~

cp $dir/tmp/comfort_pie $dir/tmp/comfort_graph
tektool -Ws 850 706 -Wp 200 100 -WP 1078 750 -Wl "causal energy" -r
$dir/binsh/comfort_graph_sun &

echo >>$dir/tmp/comfort_trace
echo "***** comfort_cause: get energy balance for worst zone from out"
>>$dir/tmp/comfort_trace
echo >>$dir/tmp/comfort_trace

out  >>$dir/tmp/comfort_trace  2>$dir/tmp/comfort_eb  <<~
-6
${comfort_results}
y
n
b
```

```
p
2
${WZ}
2
-
f
~
cat $dir/tmp/comfort_eb                 # Display energy balance.

                                        # Get offending energy flowpaths.
OHC=`awk -f  $dir/binsh/comfort_awk_eb  $dir/tmp/comfort_eb`
echo
echo " From this data I have determined that the (rank ordered)"
echo " cause of the zone ${WZ} discomfort is"
echo " $OHC."

for i in $OHC                           # Insert plot commands into GP.
do
   case $i in
   "Infilt")        GP=$GP'l
';;                                     # 's needed to put \\n into GP.
   "Vent")          GP=$GP'm
';;
   "WcondE")        GP=$GP'n
';;
   "WcondI")        GP=$GP'n
';;
   "DcondE")        GP=$GP'o
';;
   "DcondI")        GP=$GP'o
';;
   "Solair")        GP=$GP'p
';;
   "CasConv")       GP=$GP'q
';;
   "Surfconv")      GP=$GP'r
';;
   "Plant")         GP=$GP'k
';;
   esac
done
GP=$GP'!'                               # Add draw command.

                                        # Get graph of causal flowpaths.
echo >>$dir/tmp/comfort_trace
echo "***** comfort_cause: get causal flowpaths for worst zone from out"
>>$dir/tmp/comfort_trace
echo "***** graphpic is ${GP}" >>$dir/tmp/comfort_trace
echo >>$dir/tmp/comfort_trace

out  >>$dir/tmp/comfort_trace  2>$dir/tmp/comfort_cause  <<.
-6
${comfort_results}
y
n
```

```
1
d
d
d
1
${WZ}
c
a
${GP}
-
-
f
.

echo
echo " A graph of flowpath interactions will appear"
echo " in the next window. Type ctl-c to terminate."

cp $dir/tmp/comfort_cause $dir/tmp/comfort_graph
tektool -Ws 850 706 -Wp 200 100 -WP 1078 750 -Wl "causal flowpaths" -r
$dir/binsh/comfort_graph_sun &

read xb                                          # hang around.
```

***** Script 4 *****

```
# This awk script selects the zones with overheating problems and sorts them
# into descending order.  The temperature used to indicate overheating is
# obtained from a recode in the file containing "OHT nn", where nn is the temp.
# If this record doesn't exist, a default is used.  Script comfort_worstz
# provides the OHT record.
#
# This script is totally dependent of the formats used by ESPout.
# Its output is used by the script comfort_awk_worstz. If ESPout changes,
# this script may have to be modified, and if this script is changed, script
# comfort_worstz may require modification.
#
BEGIN           { ORS = "";                      # Stop print writing NL
                  OHT = 25;                      # Assume default temp
                  nz = 0;                        # as showing overheating
                }

$1 ~ /^OHT$/    { OHT = $2 }                     # Pick up overheating
                                                 # temp (from
                                                 # comfort_worstz)

$1 ~ /^[1-9]/   { if ($2 > OHT) {                # Select line starting
                        PT[nz] = $2;             # with no.
                        PZ[nz] = $1;             # 1st no. is zone number
                        nz++;                    # 2nd no. is max temp.
                  }                              # Only save o'heat zones
                }

END             { for (i=0; i<nz; i++)           # Exchange sort temps.
```

```
                for (j=i+1; j<nz; j++)          # and assoc. zones
                        if (PT[i] < PT[j]) {
                                tmp = PT[i]
                                PT[i] = PT[j]
                                PT[j] = tmp
                                tmp = PZ[i]
                                PZ[i] = PZ[j]
                                PZ[j] = tmp
                        }
                for (i=0; i<nz; i++)                    # print zone nos. (1 line)
                        print PZ[i] " "
                printf "\\n"
               }
```

***** Script 5 *****

```
# This awk script selects the worst energy flowpaths contributing to the
# zone overheating and prints them (currently any non zero flowpath is printed).
#
# This script is totally dependent of the formats used by ESPout.
# Its output is used by the script comfort_cause. If ESPout were to change,
# this script would probably need to be modified, and if this script is changed,
# script comfort_cause may require modification.
#
BEGIN           { ORS = ""                      # stop "print" outputing NL
                  C[0] = "Infilt"           # Causal Classes
                  C[1] = "Vent"                 # These are in the order that
                  C[2] = "WcondE"               # imp outputs them
                  C[3] = "WcondI"               # They are printed as shown for
                  C[4] = "Dcond"                # script comfort_cause to use
                  C[5] = "DcondI"
                  C[6] = "Solair"
                  C[7] = "Casconv"
                  C[8] = "Surfconv"
                  C[9] = "Plant"
                }

#
# NOTE: ESPout OUTPUTS GAINS & LOSSES ON LINE AFTER TITLE AND
STARTING WITH ^F
#
$2 ~ /^[+-.0-9]*$/  &&  $3 ~ /^[+-.0-9]*$/ && NF == 3 {
                  G[c] = $2                     # Select lines with only two
                  L[c] = $3                     # numbers.  These are the casual
                  c++                           # gains/losses, in the order
                }                               # output by ESPout

$1 == "No" && $2 == "plant"    { C[9] = "" }    # Correction due to missing
                                                # gain/loss nos. if no plant

END             { for (i=0; i<10; i++)
                        if (G[i] != "0.") # Test for o'heating contributor
                                print C[i] " "  # Print contributors (1 line)
                  print "\\n"
```

}

When invoked, this script, after a computational effort that depends on the complexity of the building and the length of the required simulation, will produce the screen image of figure 5.

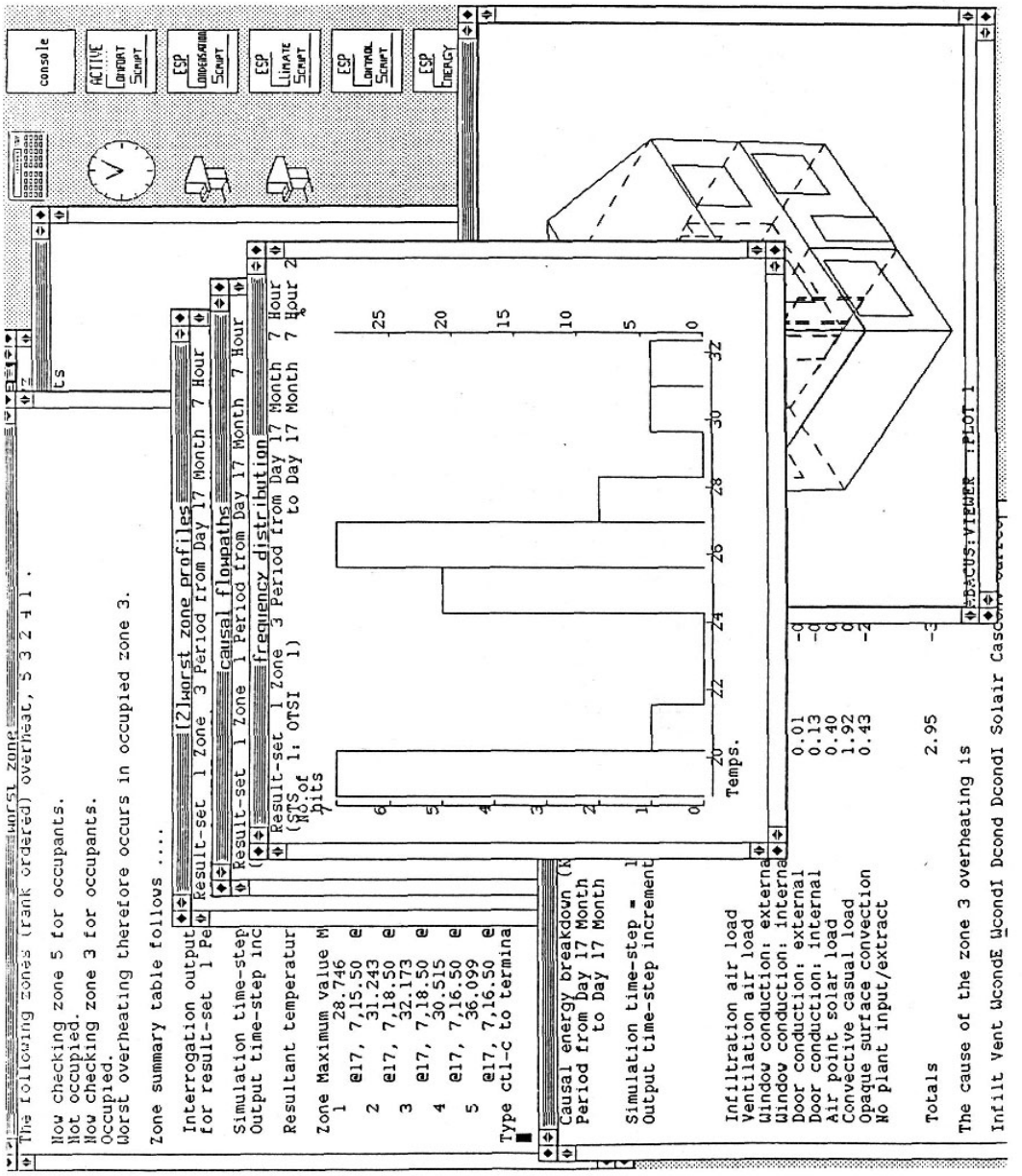


Figure 5: Output from the comfort assessment script.

The methodology is obviously more involved. Only the more salient features of the scripts are discussed here since much effort would be required to fully explain the subtleties of script syntax. The interested reader should consult the appropriate Unix manual entry (man sh) in order to fully grasp the data redirections used throughout (the `>', `>>', `<', `<<' and `|' symbols) and the variable substitutions (${?}). Although the sub-scripts are designed to work together, they can be used independently. For example, to get an energy balance for a particular zone, script 2 could be invoked.

The environment variable IFe_HOME is used throughout. This is set in .iferc_c (C Shell) or

.iferc_b (Bourne Shell) to define the home directory of the IFe. This allows the ESP files to exist in, and the script to run from, any directory. Any variables used must then be exported so that other scripts can use them. For example, here IFe_HOME is exported. The `echo' program is used throughout to write to a trace file (comfort_trace) in order to provide a complete record of the performance assessment.

It is the Appraisal Handler's job to determine which, if any, of the command line options (-h, -i, -o, -f, -c, -b, -p, -t or -d) should be invoked and to supply the corresponding arguments. At the start of script 1, variables are assigned their default values. The command line is then scanned to determine what substitutions are appropriate. For example, "comfort -i set -o 25" would cause the script to execute with standard effective temperature defined as the comfort index, with a temperature level of 25°C defined as the overheating cut-off. Obviously there are many premutations. ESPsim is now run to determine the building's behaviour against the default climatic boundary condition. Selection of the -c option allows the Appraisal Handler to force the use of a user-specified climate collection.

At a certain point in its operation, script 1 hangs until a special file is created by script 2 to indicate that it has finished. Script 2 will have filled this file with data which identifies those zones which are uncomfortable in terms of the selected criteria. If the file is empty, all zones are within the comfort range. Otherwise script 3 is run to continue the appraisal. The "suntools" command opens a new window according to the specified arguments. The tekem program is a graph display utility which is here used to display ESP graphics in a window of any size, positioned anywhere on the bit-mapped display.

In script 2 the back quotes (`) cause whatever is between them to be run, with the output replacing the quoted string. To generate the input for awk, the two commands echo and cat are placed in brackets. This tells the Shell to execute them sequentially, but to treat them as one command as far as the rest of the line is concerned. First the echo output is piped to awk and then the cat output is sent down the same pipe. This means that the variable PZ will contain the results of the awk process: an ordered list of the problem zones in this case. The variable WZ is then set to the worst zone - that is the first one in the PZ list. This zone is then tested for occupants. ESPimpb is the ESP process which does this. It returns 0 if a zone is unoccupied; it does no output. The worst zone is then written to the lock file in order to restart script 1 and feed script 3. Finally a perspective view is produced and displayed.

Script 3 again uses the awk program to determine the cause of any thermal discomfort. OHC contains the rank ordered causal list - for example, *Infilt Solair Surfconv* for infiltration, window solar absorption convected inward to the air, and internal surface convection respectively. Variable `i' is set to each string in OHC. Then the case adds the ESPout plot command to GP, depending on the value of `i'. GP is set to the old GP with the new plot letter and a newline appended. The plot letter and \\n are enclosed in quotes to stop the Shell molesting them.

15. Preparing Application Data

The function of the Data Handler is to prepare the input data-set as required by the target application program to which the IFe is front-ended. What was required was a general mechanism which could be used to do this irrespective of the data and format requirements of the application program. The only external action required on the part of a person attempting to configure the IFe is the establishment of a *Data Definition* script. This is a generalised Shell script which defines a typical data preparation session but with the actual data identified as requests to the IFe Blackboard. Within the Script an editor might be used to create the data-set or, conversely, some input management program may be used. For example, the following Shell script is essentially a session with the input management program of ESP (called ESPimp) mixed with Shell commands and a special syntax which allows information to be extracted from the IFe Blackboard:

```
        # ESP zone geometry file builder: version 1.0a of November 1988

        # Script to build ESP zone geometry input files
        # from data supplied by the IFE building model.

"imp << EOI"
"-6"
"a"

% nz = no_zones          nokey
forall i upto ${nz:=0}
do
%   zsht = zone_type     $i
  if test X${zsht} = Xrec
  then
    "c"
    "rec"
%     zo = zone_origin   $i
    "${zo:='0    0         0'}"
%     zl = zone_length   $i
%     zw = zone_width    $i
%     zh = zone_height   $i
    "${zl}  ${zw}  ${zh}"
%     zor = zone_orientation      $i
    "${zor:='0'}"
    nsur=6
  elif test X${zsht} = Xreg
  then
    "c"
    "reg"
%     nsur = no_surfaces          $i
    nw=` expr $nsur - 2 `
    "${nw:=0}"
%     f_ch = zone_height          $i
    "${f_ch}"
%     angrot = zone_orientation   $i
    "${angrot:=0}"
    forall j upto ${nw}
    do
      jj=' expr $j + $j '
%       xyz = vertex      $i        $jj
      set `echo $xyz`
      "$1 $2"
    done
  elif test X${zsht} = Xgen
  then
    "c"
    "gen"
%     nver = no_vertices$i
    "${nver:=0}"
%     nsur = no_surfaces          $i
    "${nsur:=0}"
%     angrot = zone_orientation   $i
    "${angrot:=0}"
```

```
    forall j upto ${nver}
    do
%       xyz = vertex      $i        $j
      "${xyz}"
    done

    forall j upto ${nsur}
    do
%       ver_order = surface       $i        $j
      set `echo $ver_order`
      "$# ${ver_listr}"
    done
  fi

  total_no_wins=0
  no_windows=
  forall j upto ${nsur}
  do
%    no_win = no_windows      $i        $j
    total_no_wins=` expr $total_no_wins + 0${no_win:=0} `
    no_windows="$no_windows ${no_win:=0}"
  done
  "$no_windows"
  if test 0${total_no_wins} -gt 0
  then
    forall j upto ${nsur}
    do
%      no_win = no_windows    $i        $j
      if test 0${no_win} -gt 0
      then
        forall k upto ${no_win}
        do
%           win_xzdxdz = window          $i        $j        $k
          "${win_xzdxdz}"
        done
      fi
    done
  fi
  total_no_doors=0
  no_doors=
  forall j upto ${nsur}
  do
%    no_dr = no_doors  $i        $j
    total_no_doors=` expr $total_no_doors + ${no_dr:=0} `
    no_doors="$no_doors ${no_dr:=0}"
  done
  "$no_doors"
  if test 0${total_no_doors} -gt 0
  then
    forall j upto ${nsur}
    do
%      no_dr = no_doors          $i        $j
      if test 0${no_dr} -gt 0
      then
```

```
        forall k upto ${no_dr}
        do
%           door_xzdxdz = door $i      $j      $k
          "${door_xzdxdz}"
        done
      fi
    done
  fi
%   insolation_data = zone      $i      insolation_data
  if [ X${insolation_data} = X ]
  then
    "ife_z$i.shd"
  else
    "${insolation_data}"
  fi
  "ife_z$i.geo"
  "-"
done
"-"
"f"
"EOI"
```

To explain the syntax, consider the first 12 lines of this script.

Line	Content	Purpose
1-3	# etc	Comments
4	"imp << EOI"	Command to run ESPimp with input taken from this file until string "EOI" is encountered.
5	"-6"	ESPimp required the terminal type to be defined. Here we choose non-graphics mode ("-6").
6	"a"	At this stage in ESPimp a menu appears. We choose option "a" which allows us to define zone geometry.
7	% nz = no_zones nokey	We now ask the Blackboard for the number of zones in the problem. Note that requests to the Blackboard are preceded with a "%", the concept to be matched follows the "=" and the result is assigned to the variable preceding the "=". The string nokey is part of the Blackboard Tuple.
8-9	forall i upto ${nz:=0} /do	We now establish a Shell "forall" loop to process each zone.
10	% zsht = zone_type $i	The first piece of zone information required by ESPimp is the shape type and so we ask the Blackboard.
11-12	if test X${zsht} = Xrec /then	If the shape type is rectangular, we proceed with that ESPimp option.
13-		

Apart from the use of normal shell syntax, the things to note are the usefulness of the "forall" syntax extension and the retrieval of data from the Blackboard using the "%" syntax. It should be clear that, in order to retrieve data from the Blackboard, the format of that data must be known.
Mismatches between the data held on the Blackboard and that required by the package can cause some complexity in the script. For example, in the script above, the Blackboard does

not hold the total number of windows for each zone but holds the number per surface. Thus, in the script, each surface has to be polled once to calculate the total number of windows and again to retrieve the window data. (This example is rather contrived, as it would be simple to modify the Knowledge Handler knowledge base to record this information. However, it does show how a user conceptualisation created with one modelling package in mind can be used to drive a different one).

Using this mix of Blackboard querying and Shell programming allows any data structure to be defined. It is this script that is then used by the Data Handler to produce the final data-set. This is achieved in two stages as follows. Firstly the Script is filtered by the Unix process awk, against a set of patterns. This action substitutes the Blackboard information and modifies the Shell commands to produces a second script which can actually be run. Each line of the Data Definition script (DDS) is read and converted as follows:

DDS	Shell Equivalent	Meaning
file output	file=output	from now on, output to a file named "output"
"xyz" echo	"xyz" >$file	line must only have whitespace outside the quotes; copy quoted line literally into the output file
forall j upto n	j=-1\\nwhile j=` expr $j + 1 ` && [j -lt n]	equivalent to "for(i=0 ;i<n ;i++)" or "do ??? i=0,n,1"; simplifies scripts
% val = cpt args	echo query; read query answer	sends "query u_cpt ? cpt args" to the Blackboard; retrieves "answer u_cpt u_set cpt args values"; assigns values to val

The actual pattern-set used by awk is as follows:

```
BEGIN                   { print "file=\"bm.output\"\\n>$file"
                          ORS=""
                          OFS=""
                        }
# lines beginning with "file" change the output file - default "bm.output"
/^file/                 { print "file=$2\\n"
                          next
                        }
# matches Blackboard querys, ie lines of the form "% n = no_zones" are converted into
# a query of the u_cpt area of the Blackboard for the data associated with "no_zones"
/^%/                    { print "echo \"query     u_cpt    " $4
                          for (i = 4; i <= NF; i++ )
                                  print "   " $i
                          print "\" >&1; read junk junk junk"
                          for (i = 4; i <= NF; i++ )
                                  printf " junk"
                          printf " " $2 " <&0\\n"
                          next
                        }
# match literal data - ie lines completely enclosed in quotes
/^[       ]*".*"[   ]*$/    { print "echo " $0 " >> $file\\n"
                          next
                        }
```

```
# match "special" for loops, ie lines of the form "   forall i upto $n
/^[        ]*forall[           ][^        ]*[        ]upto[   ]/ {
                                print $2 "=-1\\n"
                                print "while " $2 "="
                                print     "` expr ${" $2 "} + 1 `  || true  &&"
                                print     "  [ $" $2 " -lt " $4 " ] \\n"
                                next
                               }
# matches everything else - eg variable assignment, if-for-switch cmds, etc
/.*/                           { print $0 "\\n"
                                 next
                               }
```

This filtering is equivalent to "awk -f pat_fil DDS", where "pat_fil" is the preceding awk pattern file.

Secondly, and finally, the output script (bm.output) is passed to a Bourne Shell and executed. This results in the final data-set.

The Data Handler is run once the Knowledge Handler and Appraisal Handler have collected the necessary data on the Blackboard. If several data files are required, they can be prepared one at a time as the data becomes available on the Blackboard. Currently, the user has to indicate that she/ he has completed data input, although this could, of course, be inferred by the Knowledge Handler. The Knowledge Handler then requests the Blackboard to run the Data Handler by sending it the command: (see section 14.3)

"new_dialog IFE_data_handler ife_bm data_definition_script".

This causes the Blackboard to start the Data Handler client (held in the file ~ife/bin/ife_bm) and pass it the Data Definition script name. The script, which is held in the directory ~ife/lib/bm_filters, is converted to a pure shell script and executed. Once the data files have been created and the Shell script exits, the Data Handler also dies.

16. Future Work

The IFe has been conceived and progressed to the research prototype stage in a two year period. Given the complexity of the system, it will require a further R&D effort to evolve the IFe into a robust product which can be used routinely by others to create intelligent interfaces for their particular applications and user types. In particular the IFe could be refined by improving the efficiency and flexibility of the knowledge handling, conceptualisation entry and data manipulation functions. In the first two cases this could be achieved by the development of software tools to assist in knowledge base creation and conceptualisation entry. This would reduce the level of computing science expertise required and so allow a greater number of application specialists to work with the IFe directly. In the last case a simple schema definition language could be defined which is capable of handling the requirements of the IFe as well as the STEP data exchange standard [18]. The Blackboard would then be enhanced by the addition of a mechanism to search the schema, so that clients can request specific data without specifying where in the schema it is stored. This would greatly increase the efficiency with which new or existing clients could be serviced, and by reducing the need to know about the data structuring used by other clients, will further improve the IFe's modularity. This, in turn, would ensure that major new components, such as new appraisal methodologies or user conceptualisations, could be added more easily in future.

Other potential refinements include:

- the development of a plan recognition function for the Appraisal Handler to enable it to address the wider, less focussed appraisal objectives characteristic of users such as architects; and
- further work on the User Handler to endow it with genuine use modelling capabilities.

Much further work will also be required to ensure that the Ife meets the diverse needs of the building design community and to ensure that the system offers a sufficiently generalised range of performance appraisals. This can be achieved by

- Adding new user conceptualisations to represent the range of possible user types - from a practising designer to a student; from an educationalist to a legislator; and from a client to an occupier. This would allow the IFe to satisfy the diverse needs of all potential model users.
- And extending the performance assessment methodologies to other application programs and domains so that the user can examine the range of cost and performance as a design evolves from inception, through the schematic and detailed design phase, to completion and post occupancy.

Finally, much of the IFe prototype is in a form suitable for inclusion into the Energy Kernel system [5]. This would allow future model developers to rapidly prototype the technical and user interface dimensions of a model, modifying both in response to user reaction and technical deficiencies exposed through use.

17. Conclusions

A prototype IFE has been developed which enables a human-sensitive approach to building performance appraisal in the context of the multiplicity of models now emerging (from the advanced simulation systems to the regulations orientated structures such as Eurocode). This system is operational within a Unix workstation environment and already several groups throughout the world are attempting to apply it.

The principal advantage of an IFE is that it helps users to deal with the varying scientific, engineering and design vocabularies. This should help in the technology transfer process by easing the learning curve associated with the adoption of the new modelling technologies.

By giving the profession access to the power of contemporary and future software systems from a single building description achieved from only the information the user is able to give, the possibility of truly integrated, multi-criteria design appraisal is enabled. This, in turn, will allow the designers to make the necessary trade-offs in the search for an optimum solution and so arrive at more robust designs.

18. Acknowledgements

The authors are indebted to the Science and Engineering Research Council for funding this work and to our many research colleagues in the UK and abroad who have been so ready to field trial the product.

19. References

1. IBPSA Bylaws and Charter Statement.
2. BEPAC Rules of Association.
3. Energy Technology Support Unit, `Project Brief: Performance Analysis Service', Harwell, Didcot, 1987.
4. Winkelmann F, `Developments in Object-Orientated Programming in the Area of Building Performance Appraisal', Various Publications, Simulation Research Group, Lawrence Berkeley Laboratory.
5. Clarke J A, `Proposal to the UK SERC to Develop an Energy Kernel System for

Building
6. Energy Simulation', Energy Simulation Research Unit, University of Strathclyde, Glasgow, 1988.
7. Gicquel R, `Programme PASSYS: Status Report', Ecole des Mines, Sophia Antipolis, 1987.
8. Hatten D van (ed), `Proceedings of a Workshop on the Future of Building Performance Modelling', EUR Report Reprint, March 1988.
9. Mac Randal D and Clarke J A, `A Proposal to Develop an Intelligent Front End for Building Energy Simulation', Science and Engineering Research Council, Grant Case for Support, 1986.
10. Clarke J A and Mac Randal D, `The Application of Intelligent Front Ends in Building Design', *Artificial Intelligence in Engineering*, pp360-370, Computational Mechanics Press, 1987.
11. Ross P, `The Virtues and Problems of User Modelling' Colloquium on Intelligent Knowledge Based Systems - the Path to User Friendly Computers, *Digest 1984/104*, Institute of Electrical Engineers, Savoy Place, London, 1984.
12. Kidd A L and Cooper M B, `Man Machine Interface for an Expert System' , *Proc. 3rd BCS Conf. on Expert Systems*, Cambridge, UK, Dec 1983.
13. Moralee S, `Intelligent Front Ends', *Proc. Alvey IKBS Research Theme Workshop*, Cosners House, Abingdon, England, 26-27 September 1983.
14. Bundy A, `An Architecture for Intelligent Front Ends', *Proc. Alvey IKBS Research Theme Workshop*, University of Sussex, UK, 10-11 July, 1984.
15. Tate A, `Generating Project Networks', *Proc. International Joint Conference on Artificial Intelligence*, Cambridge, Mass., USA, 1979.
16. Alty J L, `Use of Path Algebras in an Interactive Adaptive Dialogue System', *Proc. Alvey IKBS Research Theme Workshop*, University of Sussex, UK, 10-11 July 1984.
17. Clarke J A, *Energy Simulation in Building Design*, Adam Hilger Ltd, Bristol and Boston, 1985.
18. Thomas D, van Maanen J & Mead M (Eds), *Specification for Exchange of Product Data*, Springer Verlag, 1989.
19. Standard for Exchange of Product Data (STEP), ISO Draft Proposal 10303, Dec 1988.
20. Wilson M and Mac Randal D, `The Map Utility', *Report IFE/SDN5/88*, Rutherford Appleton Laboratory, Chilton, Didcot, Oxon, UK.
21. Stearn D D, `Vim: a geometry input utility', *ABACUS User Manual*, University of Strathclyde, Glasgow, 1988.
22. Stearn D D, `The Viewer System', *ABACUS User Manual*, University of Strathclyde, Glasgow, 1988.
23. Hutchings A M J (ed), `Prolog User Manual', *Report AIAI/PSGm1/86*,AI Applications Institute, University of Edinburgh, Edinburgh, UK.
24. Clarke J A et al, `Project IBIPS: Intelligent Building Input for Performance Simulation', A Research Proposal to the Commission of the European Communities under the Joule Programme of DGXII, 1989.
25. Bourne S R, *The UNIX System*, Addison-Wesley, 1982.

Appendix A: Glossary of Terms

Term	Meaning
Animated Graphics	Moving pictures.
Application Handler	The IFe module which executes the required back-end application programs.
Appraisal Definition	The script describing how to perform the required appraisal.
Appraisal Handler	The IFe module which coordinates Shell scripts which represent performance assessment methodologies.
Asynchronous Process	A stand-alone program, potentially executing in parallel with other programs.
Autonomous Inference Engine	A program which makes inferences based on the supplied knowledge, (in the IFe these are Prolog predicates).
Autonomous Process	A stand-alone program, independent of any other program.
Back-End	The Application Handler, the Appraisal Handler, the Data Handler and parts of the Knowledge Handler.
Blackboard	An active, dynamic information store, available for clients. Sometimes it provides synchronisation between clients.
Blackboard Client	An autonomous program which accesses the information on the Blackboard.
Blackboard Communication Area	A distinct part of the Blackboard containing a particular category of information.
C++	An object-oriented programming language.
Concept	The user's mental unit of information; the smallest data item handled by the IFe.
Data Handler	The IFe module for formatting data-sets as required by any target application program.
Data Structure	A description of the relationships between a collection of data items which collectively comprise an entity.
Dialogue Handler	The IFe module for user communication.
Dynamic Memory Allocation	Obtaining extra memory from the operating system when it is needed, instead of reserving large quantities initially.
Dynamic User Modelling	Adapting the system's handling of the user on the basis of the ongoing interaction.
Energy Kernel System	An object-oriented machine environment for the construction of advanced building energy and environmental simulation models
ESP	A building energy and environmental simulation system
Event-Driven Programming	A style of programming where the actions taken depend on external events, for example user input.
Expert System	A rule based system.
Form Definition Syntax	The syntax used to specify the entities (fields, buttons, pop-ups, etc). which represent the concepts associated with a particular user conceptualisation.
Forms Program	A program which manipulates a set of forms, each of which correspond to a particular meta-concept.
Form Template File	A file containing predefined display instructions for each concept.
Human-Computer Interface	The User Interface, together with the computer

	software/ hardware utilised.
Inference	The deduction of further information from that supplied.
Intelligent Default	A default value for some concept, selected on the basis of the context within which the concept is being used.
Intelligent Knowledge Based System	A system in which knowledge is explicitly represented, for example in Prolog clauses.
Intelligent Tutoring System	An IKBS for teaching; uses a user model to adapt the teaching to suit the particular user.
Knowledge Base	A file containing the Prolog predicates relating to the concepts and meta-concepts.
Knowledge Handler	The IFe module for knowledge manipulation.
Meta-Concept	A collection of concepts which are in themselves a unit; an abstract, higher level concept.
Object-Oriented	A style of programming in which the data, not the operations, are central; usually involves encapsulation of operations with their data, abstraction of behaviour into a Class and inheritance of behaviour (and code) between Classes.
Pattern Matching	Searching the Blackboard for a Tuple containing the given text.
Performance Assessment Methodology	The rules and relations against which a system's performance is evaluated.
Plan Formation	The development of a series of primitive actions that will achieve the desired result.
Plan Recognition	The deduction, based on the user's input and the user model's data, of what the user is trying to achieve.
Pop-Up Menu	A menu associated with a concept's label, activated by the left or right mouse button.
Poster	A Blachboard Client who is sending information to the Blackboard for storage.
Predicate	A series of preconditions (possibly null) that must be met for the fact represented by the predicate to be true.
Procedural Programming	A style of programming in which the machine is told what to do one instruction at a time.
Prolog	A knowledge representation language based on predicate logic. Predicates may contain facts (knowledge) or preconditions.
Protocol Converter	A procedure that maps messages from one syntax to another, without affecting the semantics.
Rule-Based Inferencing	A rule is a series of actions to be taken when the specified conditions are met; inferencing from information generated by other rules may be used to deduce if the conditions have been met.
Schema Definition Language	A language for describing data structures, used especially for databases.
STEP	An ISO draft standard for Product Data Exchange.
Tuple	The IFe Blackboard's unit of information. Consists of a list of text fields.
Unix Shell	The command interpreted supplied with the Unix (and SunOS) operating system.
Unix Shell Script	A file containing commands which can be executed by the Unix Shell.
T}	A set of concepts and meta-concepts deemed
User Conceptualisation	acceptable to a particular user type.

User-End	The Dialog Handler, the User Handler and parts of the Knowledge Handler.
User Handler	The IFe module for handling to the different user types.
User Interface	The part of a computer program that interacts with the user.
User Interface Management System	A computer system that aids the construction of high quality interfaces.
User Model	The data/ knowledge about the user, held in the system and used to provide a personalised interaction style
User Type/ Class	In the IFe, the user is stereotyped into standard types, for example an architect or an engineer.