An Intelligent Approach to Building Energy Simulation

J A Clarke
ABACUS CAD Unit
University of Strathclyde

INTRODUCTION

In the field of building design, the development of computer aided design (CAD) systems has been underway for twenty years or so. During this time the economic factor has ensured that the draughting function has received much attention by system vendors and is now well established in the marketplace. Design software, on the other hand, is less well developed. The complexity of design, stemming from its multi-variate nature, makes it difficult to devise a computer-based approach that would perform well and be generally accepted in practice. One aspect of the design activity has received much attention however: the evaluation of building performance throughout the design process. Here, powerful appraisal models have been built that can evaluate the range of cost and performance issues of importance in design: from life-cycle cost estimates at the outline proposal stage, to realistic visualisations prior to construction. At the present time there is a growing demand for systems possessing both the appraisal and draughting functions and, in response, vendors are busy appending appraisal programs to their proprietary draughting packages. The resulting CAD system is then offered on an inexpensive microcomputer or, increasingly, on a single user graphics workstation utilising 32 bit chip technology and state-of-the-art bit-mapped displays.

The point has been reached where different vendors can now offer a seemingly integrated, computer-based solution to designers. But this is not the end of the CAD evolutionary process, or even a point at which the development pace can slow. Two current developments are alone sufficient to fuel further accelerating change.

First, current CAD systems are *function orientated* (Bijl 1986) in that they perform prescribed tasks in a number of areas which comprise the whole design. This type of system will have deficiences:

- The software structure is often inflexible and unyielding because the program was conceived in a now outdated machine environment. This means that the structure is monolithic, imposing extreme management and updating difficulties.
- To date vendors have not found it possible to formulate a common approach to the modelling problem. The lure of the marketplace has served only to stifle inter-vendor collaboration. The result is that each system has customized I/O procedures and a unique internal structure and, because appraisal models are different in each code, theoretical issues are deeply buried with little guidance given on the range of applicability or algorithm validity.
- The application knowledge is often inextricably bound to the source code. Contemporary software engineering favours the separation of the domain specific knowledge from the software. This is a fundamental prerequisite of intelligent systems which, if disregarded, can become a serious barrier to software evolution.

- Many of the currently available systems are of unacceptably low integrity. The real world processes - of cost, energy, comfort *etc*. - have been crudely approximated and causality has not been addressed. An unsuspecting user is then left to struggle with the inadequacies of a model that has unacceptably degraded the real world in an attempt to achieve an elegant computational scheme.
- Finally there exists no consensus statement on the goals of CAD. As a consequence many existing systems are not well tailored since each author organisation has been forced to address every element of the problem: I/O protocols, database design/management, user interface graphics, software structure, validation, documentation *etc*. It is clear that no single organisation will possess the necessary expertise in all areas. Each system is then promoted in a manner which implicitly undermines the development effort expended on its contemporaries.

In the short term, vendors will attempt to hide these problems beneath a veneer of interfaces; with an increasing number of retrofits being applied to each CAD system in support of the vendor's marketing claims. The only real solution to these problems lies in the complete re-structuring of CAD systems in a manner that allows state-of-the-art pursuit within a task sharing environment. The current move to an *object orientated* approach to software development and maintenance is a mechanism intended to enable this new architecture. *Objects* define the primitive operations of the application and are combined, automatically, according to a template defining a particular appraisal procedure. Templates defining one system can be readily modified to create a new procedure offering an improvement on the old. Community-wide collaboration is enabled since *objects* emerging from the research community can be made available to all. And state-of-the-art developments can be fostered through ease of integration of new objects as they emerge. In other words, such developments will allow current CAD systems to be recast in a form amenable to an evolutionary, task-sharing approach.

Second, and potentially far reaching, is the current move to computer systems which incorporate a degree of intelligence. Undoubtedly the 80's will see the demise of the solely functional software approach. Future systems will possess a very different architecture (probably constructed in object-oriented fashion) in which knowledge bases, inference engines, user models and the like will be combined with simulation primitives to endow computer systems with human-like characteristics.

This paper is concerned with the second issue. Developments in object-oriented software construction are reported elsewhere (Goldberg *et al* 1983, Clarke 1986).

INTELLIGENT KNOWLEDGE BASED SYSTEMS

The spectrum of possibilities for the design of *intelligent knowledge based systems* (ikbs) is vast. At the present time many first generation *expert systems* are emerging. To a large extent these systems are restricted to problems which are themselves rules based; such as regulations advising, process control and data interpretation. In the approach, a knowledge base is established to contain domain specific facts, rules and relations. A control system then acts on this stored knowledge in order to make logical deductions in response to a posed problem. The expert system will then reproduce the expert view which may or may not be correct. For example, in the case of building

regulations or HVAC component selection the system's advice will be ‘correct’ in that the problem domain is totally rules based: obey the rules and the answer emerges. But given another application – say building comfort and energy performance assessment at the design stage – problem solution requires more than just a set of prescriptions expressed as facts, rules and relations. Now we need an important new element: the prediction of future reality at the design stage in order to quantify the performance criteria identified in the knowledge base. In other words even an expert will have need to refer to data – monitored or computed - which characterizes building performance. Of course it is conceptually possible to establish the laws of thermodynamics as a set of facts and relations so that the expert system can gain access to numerical fact through a rules-based protocol. This is a very long term prospect however.

At the ABACUS CAD Unit two approaches to the design of a building energy ikbs are being pursued: one, a short term and pragmatic approach, is currently operational, the other, a more fundamental approach, is in the early development stage. The paper now continues with a description of both approaches.

A PRAGMATIC APPROACH TO AN ENERGY IKBS

In essence this approach involves the use of the Unix Bourne shell (Bourne 1982) as a pseudo expert system shell. Shell scripts are designed to coordinate the operation of a number of objects (programs) against the rules and relations of a particular performance assessment methodology. Shell script rules, although hardwired, can be replaced by the designer at script invocation. Within the shell script ESP-r and Unix objects act to perform set operations as a function of the in-built rules which relate to each performance appraisal. The computational path to be followed at any stage in the script will depend on the performance data to emerge at run time and on the embodied rules. Each shell scripts can be viewed as a design assistant: the performance assessment and program operation knowledge is known to the assistant; the designer is free to focus attention on design decision making. The scripts are developed to operate on a Unix workstation offering a bit-mapped display controlled by a window manager (for example, a SUN2 or Whitechapel MG-1).

A thorough appreciation of shell script programming can only be obtained from study of the syntax involved. However, to demonstrate the possibilities and the technique, one ESP-r script is reproduced here. Its purpose is to undertake a comfort analysis with the following mission:

- To determine an appropriate simulation boundary condition by selecting a climatic context collection with a severity rating matched to the building's geographical location and function.
- To initiate and control the simulation processing over a period of time determined as a function of severity criteria.
- 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.
- To initiate a sensitivity study, focusing on the causal energy flow paths, and to so rank order the options for design intervention.
- And to provide a comprehensive report on comfort performance, including problem causes

and potential cures.

With reference to Figure 1, which shows the program modules of the ESP system, this script is constructed as six interrelating sub-scripts. The first runs the climate module **clm** to determine the climate context then runs **sim** 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 function-related comfort criteria. Summary statistics on the worst offenders are then output. The third script investigates the cause of any discomfort, while the fourth commissions a sensitivity analysis. 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-r modules. Because of the complexity of the syntax, these two scripts are not reproduced here. Also, since script 4 is effectively the iterative operation of the first three, it is only cursorily explained. The actual scripts are as follows.

Script 1

```
                                        # Variable ESPdir enables script to be
if test "X$ESPdir" = "X"                # run from any directory.
then
        echo "Please set up 'ESPdir' shell variable (and 'export' it)"
        exit
fi

/usr/ucb/clear                          # Clear the window and
echo " ESP rules script 20 - comfort expert." # display header.

echo "SCRIPT COMMENCES ....." >$ESPdir/tmp/s20_trace

comfort_index="res"                     # Resultant temperature set as
criteria=28                             # default comfort index. Other
                                        # options include "air" and
                                        # "set".
building=$ESPdir/tmp/system_cfg
climate=$ESPdir/tmp/climate
s20_lock=$ESPdir/tmp/s20_lock
s20_results=$ESPdir/tmp/s20_results
demo_mode="no"
start="17 7"                            # In demo mode we only need
finish="17 7"                           # a one day simulation and
timesteps=1                             # a one hour time step.
information="no"
                                        # Process command line options.
if test $# -ne 0
then
        for i do
```

```
                case "$i" in
                -h)     information="yes"; shift;;
                -help)  information="yes"; shift;;
                -i)     comfort_index=$2;shift;shift;;
                -o)     criteria=$2;shift;shift;;
                -f)     s20_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;;
                -d)     demo_mode=$2; shift; shift;;
                --)     shift;;
                -*)     echo "Unknown option: $i. Type comfort -h"
                        exit 2;;
                esac
        done
fi

if test $information = yes                      # Respond to help request.
then
echo
echo " Command line options: information (help)          -h or -help"
echo "                        comfort index              -i index"
echo "                       (res, air or set)
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 "                       demo mode                   -d yes/no"
echo
echo " Default: comfort -d no -i res -o 28"

exit
fi
                # echo files used

echo "                  Files used    : ${building}"
echo "                                    ${climate}"
echo "                                          from ${start} to ${finish}"
echo "                  Files created: ${s20_results}"
echo
echo "                  Comfort index: ${comfort_index} at ${criteria}"
echo
```

```
echo " wait ....."
echo

rm -f $ESPdir/tmp/s20_*
rm -f ${s20_results}                    # Might be user-defined file.

                                                # Determine climatic context
                                                # as a function of geographical
                                                # location and building type.
if ${demo_mode} = no
then
        clm   >>$ESPdir/tmp/s20_trace   2>$ESPdir/tmp/climate   <<~
        -6
        selection_request                       # Commands to 'drive' the
        ${building}                             # climate module of ESP.
        f
        ~
${start}='cat $ESPdir/tmp/s20_ps'       # The analysis period is
${finish}='cat $ESPdir/tmp/s20_pf'      # set from the results of
${timesteps}='cat $ESPdir/tmp/s20_pts'          # the clm run.

fi
                                                # Simulate using parameters,
                                                # saving output for analysis.
echo "SIMULATION COMMENCES ....." >>$ESPdir/tmp/s20_trace

sim   >>$ESPdir/tmp/s20_trace     <<~           # The simulation module of ESP.
-6
${climate}                              #
1                                       #
${building}                             #
y                                       #
1
3                                       # Commands to 'drive' sim.
${s20_results}
${start}                                #
${finish}                               #
${timesteps}                            #
s                                       #
n                                       #
y                                       #
>                                       #
-                                       #
f                                       #
~
```

```
echo " Simulation complete; analysis commencing,"
echo "                          output 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

        if test $comfort_index = set
        then
        comfort_index="e"
        graphpic="g"
        fi
                                                # Export required data
                                                # for use by other scripts.
export s20_results s20_lock criteria comfort_index

                                                # Run script 2 in a new window.
        newwin -t "worst zone" -f title -x 0 -y 8 -w 653 -h 600 -ix 944 -iy 656 -iw 56 -ih 56 -u --
$ESPdir/scripts/script2

while test ! -f ${s20_lock}                     # Script 1 now sleeps until the
do                                              # script 2 lock file appears.
sleep 5
done
                                                # It appears.
                                                # Check for discomfort;
if test -s ${s20_lock}                  # lock file identifies zone.
then
                                                # Lock file has worst zone(s)
                                                # in it. Determine cause.
        newwin -t "cause" -f title -x 3 -y 352 -w 550 -h 500 -ix 944 -iy 592 -iw 56 -ih 56 -u --
$ESPdir/scripts/script3

                                                # Now get frequency distribution.
WZ=`cat ${s20_lock}`                            # from the output module of ESP.
echo "COMFORT ANALYSIS; GET FREQUENCY DISTRIBUTION FROM out ....."
>>$ESPdir/tmp/s20_trace
```

```
out >>$ESPdir/tmp/s20_trace 2>$ESPdir/tmp/s20_graph <<.
-6
${s20_results}
y
4
y
${WZ}
c
&
${graphpic}
10
y
1
4
-
-
f
.
                                        # Draw frequency distribution plot.
tekem -t "frequency distribution" <$ESPdir/tmp/s20_graph


fi
```

Script 2

```
                                        # Use out to get comfort statistics.
echo "COMFORT ANALYSIS; GET ZONE COMFORT STATISTICS FROM out ....."
>>$ESPdir/tmp/s20_trace

out >>$ESPdir/tmp/s20_trace   2>$ESPdir/tmp/s20_pt   <<.
-6
${s20_results}
y
b
${comfort_index}
n
-
f
.
                                        # Use awk to get overheating
                                        # zones, ranked in PZ. File s20_awk_pt
                                        # has awk driver commands.
PZ=`(echo OHT $criteria ; cat $ESPdir/tmp/s20_pt) | awk -f s20_awk_pt `

if test "X$PZ" = "X"
```

```
then
      echo "Congratulations!"                  # Whole building is
      echo "No overheating occurs."            # comfortable.
      echo
      echo "Zone summary table follows ...."
      echo
      cat $ESPdir/tmp/s20_pt
      > ${s20_lock}
      echo
      read xa                    # Hang around until told to go away.
      exit 0                     # Normal exit (0) to indicate no discomfort.
fi

                                 # Tell designer about problem zones.
echo "The following zones (rank ordered) overheat, $PZ."
echo
                                 # 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   <<.                   # The ESP module impb is used to test
         $i                            # a zone for occupants.
    .
    if test $? -eq 1             # Impb error exits if zone occupied.
    then                         # $? is last exit status.
        echo "Occupied."
        echo "Worst discomfort occurs in occupied zone $i."

        OZF=1                    # Worst zone is occupied
        WZ=$i
        break
    else
        echo "Not occupied."
    fi
done
if test "X$OZF" = "X"
then
                                 # No impb error exit.
    echo "Worst discomfort occurs in unoccupied zone $WZ."
fi
echo
```

```
echo "Zone summary table follows ...."
cat $ESPdir/tmp/s20_pt                  # Cat is the Unix 'type file' command.
echo $WZ >$s20_lock                     # Put WZ in lock file for access by script 3.
read xa                         # Then hang around until told to go away.
```

Script 3

```
                                        # Get worst zone from lock file.
        WZ=`cat $s20_lock`
                                        # Create graph for worst zone.
echo "COMFORT ANALYSIS; GET GRAPH OF WORST ZONE FROM out ....."
>>$ESPdir/tmp/trace

out   >>$ESPdir/tmp/s20_trace   2>$ESPdir/tmp/s20_wzt <<~
-6
${s20_results}
y
c
4
y
${WZ}                                   # Worst zone variable is passed
a                                       # to out.
f
g
b
!
-
f
~
                                        # Draw graph in new window.
tekem -t "worst zone profiles" <$ESPdir/tmp/s20_wzt

                                        # Get energy balance for worst zone.
echo "COMFORT ANALYSIS; GET ENERGY BALANCE FOR WORST ZONE FROM out
....." >>$ESPdir/tmp/s20_trace

out   >>$ESPdir/tmp/s20_trace   2>$ESPdir/tmp/s20_eb   <<~
-6
${s20_results}
y
b
p
${WZ}
2
-
```

```
f
~
cat $ESPdir/tmp/s20_eb                  # Display energy balance.

                                        # Get offending energy flowpaths
                                        # using awk.
OHC=`awk -f   $ESPdir/scripts/s20_awk_eb   $ESPdir/tmp/s20_eb`
echo
echo "The cause of the zone ${WZ} overheating is"
echo "$OHC."

for i in $OHC                           # Insert plot commands into variable GP
do                                      # for later use to select graph.
    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 "COMFORT ANALYSIS; GET CAUSAL FLOWPATHS FOR WORST ZONE FROM out
....." >>$ESPdir/tmp/s20_trace

out   >>$ESPdir/tmp/s20_trace   2>$ESPdir/tmp/s20_cause   <<.
-6
${s20_results}
y
```

```
c
$GP
-
f
.
                                        # ..... and display it.
tekem -t "causal flowpaths" <$ESPdir/tmp/s20_cause

export $OHC                             # Export cause and initiate
                                        # sensitivity analysis in new window.
        newwin -t "sensitivity" -f title -x 3 -y 300 -w 550 -h 500 -ix 944 -iy 500 -iw 56 -ih 56 -u --
$ESPdir/scripts/script4
```

Script 4

```
        # For each of the causal flowpaths identified in script 3, and
        # exported to this script in variable OHC, scripts 1 and 2 are
        # re-run against appropriate changes to the building description
        # selected to establish the sensitivity of the flowpath to design
        # intervention. The possible changes are passed to impc
        # in table form. This table contains the percentage changes that
        # may be applied to the relevant parameters which define the building
        # to ESP.

for i in $OHC
do
    case $i in
    "Infilt")    parameter=1;;
    "Vent")    parameter=2;;
    "WcondE")        parameter=3;;
    "WcondI")parameter=4;;
    "DcondE")        parameter=5;;
    "DcondI") parameter=6;;
    "Solair")   parameter=7;;
    "CasConv")        parameter=8;;
    "Surfconv")        parameter=9;;
    "Plant")    parameter=10;;
    esac

echo "COMFORT ANALYSIS; MODIFY BUILDING DESCRIPTION FOR $parameter ....."
>>$ESPdir/tmp/s20_trace

impc   >>$ESPdir/tmp/s20_trace   <<.                # The ESP process impc is
-6                                          # used to modify the building
$ESPdir/tmp/sensitivity_table                       # description for the
```

```
$ESPdir/tmp/building                    # sensitivity analysis.
.

echo "COMFORT ANALYSIS; RE-SIMULATE FOR $parameter ....." >>$ESPdir/tmp/s20_trace

sim   >>$ESPdir/tmp/s20_trace      <<~              # And re-simulations are
-6                                      # initiated.
${climate}
1
${building}
y
1
3
${s20_results}
${start}
${finish}
${timesteps}
s
n
y
>
-
f
~

echo "COMFORT ANALYSIS; GET ENERGY BALANCE FOR PREVIOUSLY WORST
ZONE FROM out ....." >>$ESPdir/tmp/s20_trace
                                        # Finally new output is taken.
out   >>$ESPdir/tmp/s20_trace   2>>$ESPdir/tmp/s20_sensitivity   <<~
-6
${s20_results}
y
b
p
${WZ}
2
-
f
~
done
                                        # Now display results of sensitivity analysis.
cat $ESPdir/tmp/s20_sensitivity
```

When invoked, this script, after a computational effort which depends on the complexity of the building and the length of the required simulation, will produce the screen image of Figure 2. 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).

The environment variable ESPdir is used everywhere. This is set (in .profile or .login) to define the location of the main shell script directory. This allows ESP-r files to exist in, and the shell scripts to run from, any directory. Any variables used must then be exported so that other scripts can use them. The 'echo' process is used throughout to inform the user of progress or to write to a trace file (s20_trace) to provide a complete record of the performance assessment.

The script is executed by typing the command *comfort*, perhaps followed by one or more of the command line options -h, -i, -o, -f, -c, -b, -p, -t or -d to modify the script's action. At the start of Script 1, variables are assigned their default values. The command line is then scanned to determine if the user wishes to invoke one or more of the options. For example, the command *comfort -i set -o 25* executes the script with standard effective temperature defined as the comfort index and the cut-off temperature for overheating set at 25 C. Obviously there are many permutations. **Clm** is now run to determine a climatic collection of an appropriate severity (in terms of the building type). **Sim** is then run to determine the building's behaviour against this weather boundary condition. Selection of the -c option allows the designer to force the use of a specified weather collection.

At a certain point in its operation, Script 1 hangs until a special file is created by Script 2 to indicate that the latter has finished. Script 2 will have filled this file with data that identifies those zones which are uncomfortable in terms of the selected criteria. If the file is empty, all zones are within the comfort zone. Otherwise Script 3 is run to continue the appraisal. The **newwin** process opens a new window/icon according to the specified arguments. The **tekem** process is a graph display device which is used to display any ESP-r 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**, 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. **Impb** is the ESP-r process that 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.

Script 3 again uses the **awk** process 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.

Script 4 is then run. For each causal flowpath, **impc** will modify the building description as a function of a relational table linking flowpaths to possible design changes. **Sim** and **out** are then rerun to establish the effect of each design change before the final performance data is output.

Finally a script is run in a new window to produce a perspective view of the building under analysis for zone identification purposes.

The following scripts are available to drive ABACUS programs.

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

A number of new scripts are now under development to aid in the building description process. Here the intention is to match scripts to different building types and levels of knowledge. For example, one script might accept a simple description of a passive solar building and generate the full data set as required by ESP-r. Another script might accept a detailed description of an air handling unit, adding its own typical office description to allow an meaningful plant appraisal.

A MORE SOPHISTICATED APPROACH

Although the shell script technique offers an intelligent interface to energy simulation, it suffers from three fundamental limitations. Firstly, it is constrained to the performance appraisal aspect of the problem; it does not, for example, address the problematic issues surrounding data preparation in the face of uncertainty. Secondly, each script is considered as an independent design assistant. This implies that the user must be able to coordinate script selection and to act as the overall integrator. And thirdly scripts do not allow `Why do you ask' type responses. They have no real understanding of the system they address; they are merely clever prescriptions.

It would obviously be attractive to design an ikbs which could act as an expert consultant, recognising the user's plan, commissioning simulations and reporting back on overall performance. This is the goal of the next version of the ESP-r system. What is envisaged is the system architecture shown in Figure 3.

- A central communications center exists to manage information traffic. Each module of the system can examine this center for relevant information, posting results where appropriate.
- The knowledge base, implemented in Prolog, holds both application knowledge (concerning energy in buildings) and modelling knowledge defining simulation strategies).
- The dialogue handler is the user communication mechanism. This controls the consultation session, allowing a user to volunteer information, to redirect the systems line of inquiry, or

to make 'Why do you want to know' type responses to the systems prompts.

- The ikbs will possess more than one user model. For each user type - architect, engineer, energy modeller, student *etc*. - at least two categories, naive and proficient, are envisaged.
- The plan recognition module exists to interpret the user's objectives and to generate a list of performance assessments which will meet these.
- The simulation planner provides the complementary function: namely the generation of a list of simulation requests which match the required performance assessments.
- The model builder assembles the data structure required by the ESP-r program modules. These data can come directly from a user, if available, or from the knowledge base in the form of dynamic defaults which may depend on the user dialogue.
- And, lastly, there is ESP-r itself. This is a multi-process system which, against some design hypothesis, can undertake a dynamic energy simulation. Simulations can be tailored to allow a range of performance assessments: including comfort checks, control system appraisal, condensation prediction, passive solar analysis and whole building energy reports.

The development of such a system is obviously a major undertaking, and one which is made difficult by the fact that many of the foregoing elements are not yet well advanced. A research project has been formulated (Clarke *et al* 1986) which should create a prototype within two years.

CONCLUSIONS

In addition to its native interactive graphics mode, the ESP-r system can now be operated as a pseudo expert system. A number of Unix Shell Scripts have been developed to control ESP-r's program modules against rules which relate to particular performance assessment methodologies. Each Script is then equivalent to a design assistant, endowed with knowledge of ESP-r and the application domain.

The shortcoming of this approach is that the internal representation of knowledge is primitive and inflexible. For example, it is not possible for the designer to interact with a script once invoked. To remedy this, a more ambitious ikbs system is needed. Such a system is now under development.

ACKNOWLEDGEMENTS

I am indebted to Damian MacRandal, a project leader at the Intelligent Knowledge Based Systems Group at Rutherford Appleton Laboratory, for his help in structuring the Unix Shell Script approach detailed in this paper and for helping me to better appreciate future possibilities in the area of artificial intelligence.

REFERENCES

Bijl A 1986, AI in architectural CAD *Proc. Int. Conference on CAD and Robotics in Architecture and Construction*, Marseille.

Bourne S R, 1982, *The UNIX System*, Addison-Wesley.

Clarke J A, 1986, The Energy Kernel System, *Proc. Int. Symp. On Systems Simulation*, University of Liege.

Goldberg A and Robson D 1983, *Smalltalk-80 The Language and its Implementation*, Addison-Wesley, Reading, Massachusetts.

Clarke J A, MacRandal D and Maver T W, 1986, The Application of Intelligent Knowledge Based Systems in Building Design, *Grant Application to the UK Science and Engineering Research Council*.