Gerhard SCHMITT
Chen-Cheng CHEN
Jean-Christophe ROBERT
James Karl WEEKS
Department of Architecture
Carnegie - Mellon University
Pittsburgh

OPS5 in architecture: four Test Cases

ABSTRACT

The advantage of production systems such as OPS5 over structured programming becomes most evident when they are applied to large, ill-structured problems. These applications are abundant in architectural design. Although efficient algorithms exist for some domains in architecture, empirical knowledge is essential in others. This situation calls for hybrid implementations with traditional efficient procedural packages providing support for production system front ends. OPS5 is a general-purpose production system language. In each one of the four test cases, one particular aspect of OPS5 was explored in-depth.

The first test case is an attempt to capture the rules of thumb, algorithms and decision tables contained in the "Small Office Design Handbook" and to turn it into an interactive computerized design consultant.

The second test case is an attempt to build an intelligent knowledge acquisition tool to extract design knowledge from designers with no or very little programming knowledge, and to transform that knowledge directly into OPS5. The expert system has a problem decomposition module and an expert system building component.

The third test program is a knowledge based ROOF DESIGNER. It addresses a subset of the factors and heuristics that designers use to decide on the shape of roofs. These factors and heuristics were determined with a protocol analysis and then transformed into OPS5 rules.

The last test case uses TOPSI, the IBM AT implementation of OPS5, and DRAW, a GKS based graphics program with three dimensional extensions. DRAW acts as a high quality graphics output device of programs written in TOPSI. A user interface allows the construction of new TOPSI rules from graphic input in DRAW.

RESUME

L'utilisation de systèmes de production tels que OPS5 apparaît nettement préférable à celle de la programmation structurée dans le cas de problèmes complexes et mal structurés. De tels problèmes sont fréquents en design architectural. Bien que des algorithmes efficaces existent pour certains domaines en architecture, des connaissances empiriques sont essentielles dans d'autres cas. Cette situation nécessite le développement de systèmes hybrides comprenant des parties procédurales associées à des modules utilisant un langage de système de production tel que OPS5. Nous allons décrire, pour chacune des quatre applications présentées, un aspect particulier d'OPS5.

Le premier programme tente de reproduire les algorithmes, règles et tables de décision décrits dans le livre 'Small Office Design Handbook', afin d'en faire un outil de design interactif.

Dans le deuxième cas, nous avons tenté de développer un outil permettant d'extraire les connaissances en design de concepteurs sans formation informatique particulière, et de transcrire ces connaissances dans OPS5. Le système comprend deux parties: un module de décomposition de probleme, et un generateur de système expert.

Le troisième programme d'essai est un générateur de toitures, qui utilise des paramètres et des techniques employés par les concepteurs pour définir la forme des toitures. Ces facteurs et raisonnements ont été déterminés par des analyses de démarche, et transcrits en règles de production OPS5.

Le dernier programme a été développé en TOPSI, la version d'OPS5 pour IBM AT, et utilise DRAW, un programme graphique 3D, comme moyen de visualisation graphique. Un interface permet également de créer graphiquement des règles de production OPS5.

1. INTRODUCTION

Traditional algorithmic computer programs have proven their usefulness in certain architectural domains, such as drafting, analysis, and database management. They do not, however, directly address any of the ill-structured and qualitative aspects of a typical architectural design problem. Knowledge based computer applications are evolving as a possibility to supplement the capabilities of the present procedural architectural programs. Out of the growing number of languages available, such as PROLOG, OPS83, and LISP, we selected OPS5 to build four prototypes that seemed promising in terms of modeling quantitative and qualitative design decisions. The four prototypes should be seen not as stand alone programs (although they can function in this manner), but rather as intelligent front ends to existing algorithmic programs. This way it is possible to take advantage of highly sophisticated vertically integrated CAD packages and to combine them horizontally, i.e., simulate the necessary transdisciplinary character of architectural design. Idiosyncratic reactions to design problems, the impact of social and historical factors on architecture, and the important personal stylistic preferences, can be modeled with this approach. The purpose of this research is twofold: to integrate computer use throughout the design process without excluding the qualitative aspects of design, and to expand our knowledge of the architectural design process by externalizing and formalizing the underlying principles. The end result will not be another "Architecture Machine" but a vehicle to explain and demonstrate architectural decision-making.

2. OPS5 AS A TOOL IN ARCHITECTURE

OPS5 stands for Official Production System version 5. It is a general-purpose production system language and provides a formalism to represent architectural problem-solving knowledge. It can apply rules that correspond to the explicit chunks of knowledge that are believed to be used by human experts to solve design problems [Newell 72]. This feature is a clear advantage over traditional algorithmic programming techniques which solve well defined problems in a narrow domain. The scope of typical architectural problems which are ill-defined and interdisciplinary goes beyond the capabilities of any algorithmic or rule based system at the present. For restricted domains in the architectural context, however, OPS5 can be applied successfully.

Traditionally, OPS5 and the earlier languages of the OPS family have been used for applications in cognitive psychology and Artificial Intelligence. We chose OPS5 to develop four architectural test cases for the following reasons:

- OPS5 is not biased toward particular problem solving strategies or representational schemes. The control mechanism of the OPS5 interpreter, the recognize-act cycle, can easily be adjusted to specific user needs.

- OPS5 allows the flexible placement of knowledge in the production memory, in the working memory, in user defined databases, and in external procedures and functions.

- The meaning of symbols and the relations between symbols can be entirely defined by the programmer and can be easily used to represent experimental knowledge.

While these characteristics of OPS5 are attractive for architectural applications, a number of shortcomings had to be considered from the beginning. They typically occurred when the program reached a considerable size. These shortcomings are slowness, difficulties in debugging, non-transparent behavior, and undesirable interactions among rules [Brownston 85]. Some of these problems could be overcome by building hybrid implementations. In these cases, OPS5 was used only for the ill-defined part of the problem, whereas the procedural model was applied to the well-defined part in which calculations and rigid data formats prevailed. This increased efficiency and execution speed dramatically.

3. CASE ONE: SMALL OFFICE DESIGN CONSULTANT

Summary

The Small Office Design Consultant advises the user in integrating numerous building design factors (building area, height, area of opening, climatic region) that contribute to construction and energy costs of small office buildings. It focuses on energy saving strategies suitable for buildings up to 50000 sqft. The system is organized into five levels. The Small Office Design Consultant will be especially useful at the beginning of the design process when the architectural program is not finalized, but when many decisions affecting the building's energy performance are made. Conventional energy simulation programs like DOE-2.1C take a considerable amount of time to collect the input data and to prepare the output data. Therefore, this task is normally left to experts. The purpose of this program is to simplify the input and to obtain fairly accurate results without extensive hour by hour simulations. This can be achieved with a knowledge based front end.

Knowledge Representation And Control Strategy

Analysis principles for the system match the ASHRAE Standard 90-75 Energy Conservation in New Building Design. The planning and conservation strategies are adopted from the "Small Office Design Handbook" [Burt 85]. The personal experience of Fred Dubin from Dubin, Bloome, and Associates, New York was used to supplement and check the rules found in the "Small Office Design Handbook". His reactions to the system provided the necessary feedback to evaluate the knowledge base.

The reasoning strategy employed by the inference engine is forward-chaining or bottom-up processing [Hayes-Roth 83]. Starting from what is initially known, e.g. building size, climatic region, amount of glazing, the current state of knowledge is used to make a chain of inferences until a goal is reached.

```
; Comments begin with a semicolon and continue to the end of line.

; IF    the building size is small
; AND   % of glazing less than 17
; THEN  modifies the % of glazing to be 10%,
;       goes to level 1 design

(p glazing-1-a
{(goal ↑design-level pre-design ↑status inquire-glazing) <goal>}
{(office ↑size small ↑story 1 ↑glazing { <= 17 }) <office>}
-->
(modify <office> ↑glazing 10)
(remove <goal>)
(make goal ↑design-level 1 ↑status design-assumptions)
)
```

Figure 1: Sample OPS5 rule from the small office design consultant

In this forward-chaining architecture, the contents of the working memory represent what is currently known [Harmon 85]. The inference engine matches the left-hand sides (LHS) of the rules against the working memory, and executes the right-hand sides (RHS) of the rules to update the knowledge base by making changes to the working memory (see figure 1).

Sample Session

The user starts the program by selecting the location of his project. For this purpose, the United States are divided into five major climatic regions. The program then leads through five levels of proposed energy conscious design considerations:

1. Level 1: energy savings can be achieved by making sure that the proposed small office building meets the minimum requirements of ASHRAE Standard 90-75.

2. Level 2: energy saving can be achieved by wisely selecting basic building characteristics such as amount of glazing, HVAC system type, and Heating fuel.

3. Level 3: energy savings are achieved by applying a set of lighting, mechanical, and daylighting strategies to improve upon the design already achieved in Level 1 and Level 2.

4. Level 4: strategies are presented which are either unique to a given climate or which require a strong commitment to energy conservation, such as increased manual switching of lights or wide temperature deadbands (expansion of the thermal comfort zone).

5. Level 5: the final level presents a checklist of responsibilities for all members of the design and construction team to insure that building turnover and operation will be successful.

The system presently runs on a VAX 11/780. It takes about two minutes to load the literalization module, some 500 production rules and the weather data. The user input is kept in working memory. According to the forward-chaining control mechanism, the energy design decisions will be made level by level.

4. CASE TWO: KBS BUILDER

Summary

KBSBuilder is a tool for developing knowledge based systems (KBS) for design. KBSBuilder is used to address two major knowledge areas in a design oriented KBS:

- Descriptive Knowledge. This knowledge describes the design elements, constraints, and the design context in a frame-like representation.

- Procedural Knowledge. This knowledge can be described as the heuristic knowledge or rules of thumb that an architect uses to manipulate the descriptive knowledge into a coherent design.

KBSBuilder is a helpful tool to students of architecture who have little or no expertise in constructing a KBS to model their design process. A welcome addition to KBSBuilder would be a graphical representation of its inference paths and design hierarchies. This addition would give KBSBuilder the ability to construct more complex and powerful KBS's than is currently possible.

Descriptive Knowledge Representation -

The descriptive knowledge within KBSBuilder is stored in a semantic network. The semantic network is built in a hierarchical fashion according to levels of design abstraction. Once the student has finished building the descriptive knowledge base, KBSBuilder converts that knowledge into OPS5 working memory. For example, the representation of the design element wall could look like this:

- meta-frame: house
  design-element: wall
  is-part-of: exterior
  constraints: length width height

Constraint knowledge is attached to each design element. This knowledge is converted to OPS5 working memory when the student has finished building the descriptive knowledge base for a KBS. An example for a wall constraint could be represented as follows:

- constraint-name: wall-length
  design-element: exterior-wall
  attribute: length
  measure: >=
  value: 2(feet)

The construction process for the descriptive knowledge base adheres to the top down model, where the student starts with the highest level of design abstraction and describes all of the design elements and their respective constraints at each successive level until she or he arrives at the lowest level of design detail (see figure 2).

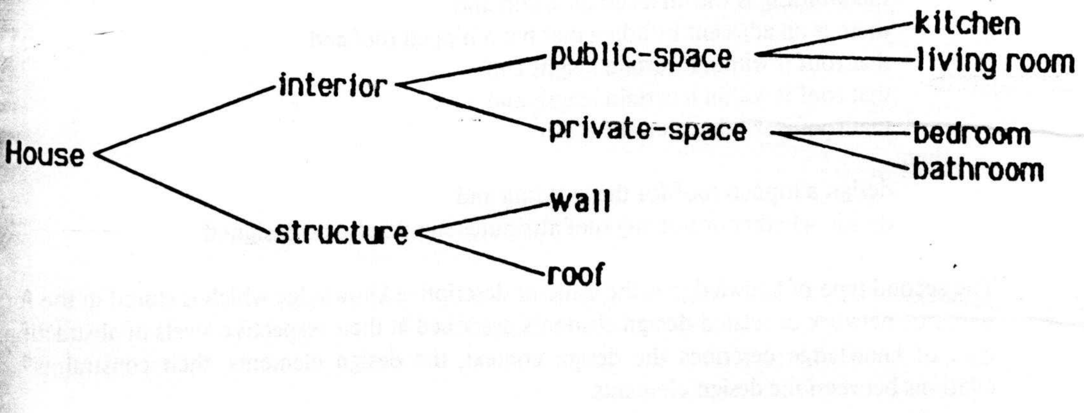


Figure 2: Descriptive hierarchy for a simple house.

Procedural Knowledge Representation

Procedural knowledge (design rules) are built within the KBSBuilder sequentially. The user is prompted by KBSBuilder to supply information for the LHS and the RHS of the design rules. Upon completion of this process, KBSBuilder converts the rules from their internal representation into OPS5 production rules.

5. CASE THREE: ROOF DESIGNER

Summary

The roof is an integral part of any building. Its shape is determined by a number of factors and design rules of thumb that vary from architect to architect and from culture to culture. The Roof Designer(RD) is a knowledge-based system written in OPS5 that designs an "appropriate" roof for a given building. RD developed out of a research project that investigated the representation of roof shapes in a 3D geometric solid modeler. The Roof Designer is a very simple expert system. It has been a vehicle for understanding two aspects: a selected part of the design process and the construction of an expert system. The Roof Designer has shown that it is possible to recreate the design process of an architect in a very restricted domain and to come up with some tangible results that will enable us to build larger and more complex design expert systems.

Knowledge Representation and Control Strategy

The Roof Designer contains procedural and descriptive knowledge in its knowledge base.

Procedural knowledge is used for the inference chaining and heuristics. It is stored in form of production rules. An example of the procedural knowledge in English is:

```
        rule roof7
    if
        there is a building that needs a roof and
        the building has a northern latitude location and
        the building has a rectangular shape and
        the building is within a certain length and
        the building is within a certain width and
        there is an adjacent building that has a hipped roof and
        that roof is within a certain height and
        that roof is within a certain length and
        that roof is within a certain width
    then
        design a hipped roof for the building and
        decide whether or not any roof attributes should also be designed
```

The second type of knowledge is the static or descriptive knowledge which is stored in the form of a semantic network of related design elements described at their respective levels of abstraction. This type of knowledge describes the design context, the design elements, their constraints and the relations between the design elements.

An example of the descriptive knowledge, addressing the design of a dormer:

- meta-element: house1
  design-element: simple-dormer
  is-part-of: roof
  shape: local coordinates
  typical-roofs: hipped-roof, saltbox

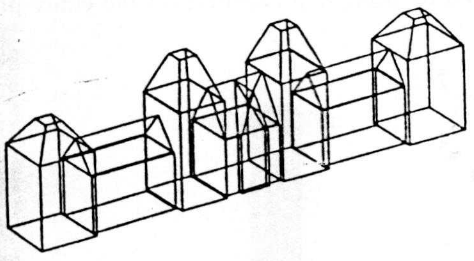
<u>Constraint knowledge</u>
constraint name: simple-dormer-max-width
design-element: simple-dormer
design-element-attribute: width
attribute-measure: <=
attribute-value: 5(feet)

<u>Relationship knowledge</u>
building: house
adjacency: apartment
length-of-adjacency: 40(feet)
distance-between-bldgs: 10(feet)
height-difference: 15(feet)
direction-of-adj: east
adj-aesthetics: type1

The user interface of RD is embedded in a geometric solid modeler, VEGA, developed at Carnegie-Mellon by Professor Robert Woodbury and Greg Glass. VEGA is a menu driven graphics package wherein RD is an operation that can be "picked" from the menu at any time.

A Sample Session

The architect initializes VEGA and builds the design context. Next, the designer picks the "Design a Roof" operation on the VEGA menu. The Roof Designer prompts the designer for some critical information such as site latitude, orientation, and building occupancy. The Roof Designer then transforms the graphical information supplied by VEGA into relational information that will be used in designing the roof. Once this has been completed, the Roof Designer designs the roof (see figure 3). The Roof Designer looks at the space below the roof, building occupancy, direction of the roof slope, along with other information to decide whether or not any other roof attributes need to be designed. The Roof Designer then designs dormers, and places them along the roof (see figure 4).



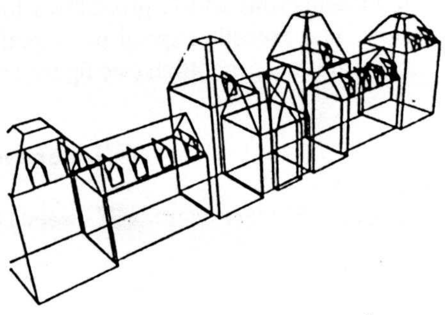


Figure 3: (Left) Design context, built in VEGA, and roofs proposed by RD.
Figure 4: (Right) The building with roofs and dormers proposed by RD.

6. CASE FOUR: MEDIEVAL ARCHITECTURE CONSULTANT

Summary

The Medieval Architecture Consultant combines an expert system and a graphics program. It provides the expert system with a graphical output and with a graphical knowledge acquisition option. The prototype in its present form deals with the relationship between the construction of religious buildings in medieval France and historical conditions. It features the generation and graphic display of various possible religious building types, their graphic insertion into a map of France, and the inference of a historical statement from the distribution, type, and number of the buildings in a certain area of France. This prototype can easily be enhanced and adapted to other uses in the architectural design or evaluation process. For instance, for building diagnostic applications, the program would present a set of possible building failures, ask for their location in the building plan, and suggest strategies for correction. Another possible application is interactive participatory design. The user designs within given constraints and receives feedback whenever constraints are violated. A more elaborate graphic knowledge acquisition module is under development. It features general facilities for site and symbols definition, insertion of symbols into the site, and incorporation of the resulting knowledge into the expert system environment.

Knowledge Representation and Control Strategy

The expert system is implemented in TOPSI, the IBM AT version of OPS5. All historical and factual knowledge is represented in form of rules. The graphics program, DRAW, and the knowledge acquisition module are written in C and are based on the IBM Graphical Kernel System. The program is thus a hybrid implementation. The control strategy is forward chaining. The user interface is menu driven in the TOPSI environment, and icon driven in the graphic parts.

The graphic output for the expert system is performed by calling DRAW, after creating a data file containing the drawing in DRAW format. An earlier version of the program used TOPSI rules for the calculations and C procedures for the writing operations of the data file. It proved to be too slow. The execution speed was greatly improved by using C procedures for the entire process of graphics data generation (see figure 5).

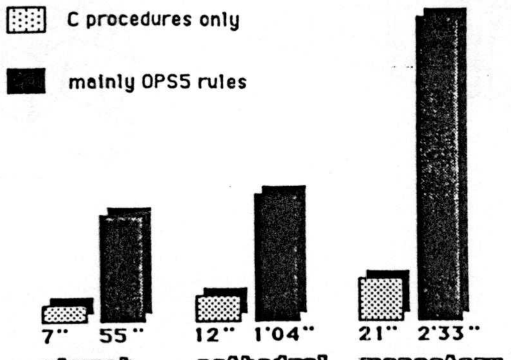


Figure 5: Comparison of DRAW data file creation time.

The graphic knowledge acquisition program writes a file containing OPS5 rules that represent the distribution, type, and number of symbols graphically entered into the site.

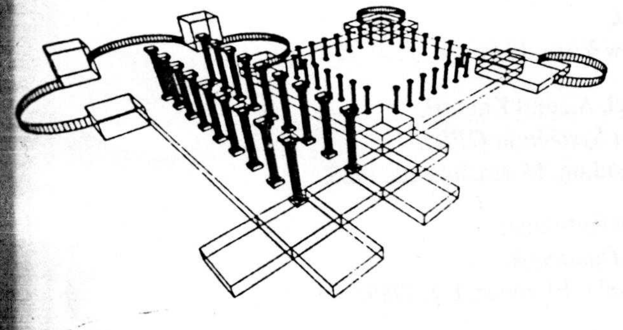
TOPSI is very suitable for this application for the following reasons:

- Its capability to call external C procedures, which enables the communication with the graphics program. In addition, the use of C procedures for some well-defined functions of the programs saves computation time.

- Its flexibility, guaranteed by the representation of the knowledge in form of rules. This enables the addition of the rules resulting from the graphic knowledge acquisition.

Sample Session

The user starts by defining the kinds of buildings that will later be used for the history session. A number of choices are presented to him in form of short menus. Choices include the time period, the building type, the building location, the kind of religious order, and the financial situation of the builders [Brooke 74]. If these parameters prove to be insufficient to characterize a building adequately, more special rules can be added easily. If enough evidence is assembled for one particular building type, the program will pass control to the graphics program DRAW. This way, the user "learns" from the program by trying out several input combinations and seeing immediately the architectural result, displayed by DRAW (see figure 6).



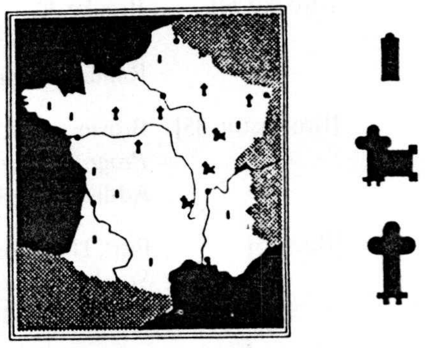


Figure 6: (Left) One of the possible building types displayed with DRAW.
Figure 7: (Right) Screen image for the Graphic Knowledge Acquisition module.

The user then has the choice to insert the different possible configurations into the map of France. France is divided into five regions for this purpose. The user picks an icon and inserts it into the appropriate region (see figure 7). The necessary knowledge can be taken from an appropriate source, such as an architectural history book. By inserting the object into the appropriate region, the user "teaches" the program. The program will accept the input in graphic form and convert this information into TOPSI rules. Once control is passed back to the KBES, the result of the graphic interaction will be analyzed and the most likely political situation, based on the input, will be described. If this political analysis conflicts with the user's knowledge, two reasons can be isolated. First, the graphic input could be wrong. This can be corrected immediately. Second, the rules in production memory that assess the political situation based on the building distribution in the region are incomplete or do not apply to this region. In this case, the production memory knowledge has to be changed or enhanced.

7. CONCLUSION

The general-purpose production system language OPS5 was applied to four architectural test cases. The Small Office Design Consultant provides a tool to evaluate the energy appropriateness of a building in the early design stages. The Roof Designer simulates the reasoning of architects in designing roofs. The KBSBuilder allows a logical breakdown of the design process and provides a simple interface for building OPS5 rules. The Medieval Architecture Consultant provides the possibility to view relations between quantitative and qualitative (historic) conditions. The advantage of these programs lies in their flexibility and their "learning" and "teaching" capabilities. The development of the programs provided an important means to clarify the decision making process in selected architectural domains. After the prototypes had been tested, some of the rules could be translated into algorithms to increase efficiency. In the conviction that future architectural computer applications will be of hybrid nature with knowledge based modules supporting algorithmic utilities, we intend to further develop and integrate the prototypes.

8. REFERENCES

[Brooke 74] Brooke, C.
The Monastic World.
Random House, New York, 1974.

[Brownston 85] Brownston, L., Farrel, A., and Kant, E.
Programming Expert Systems in OPS5.
Addison-Wesley, Reading, Massachussets, 1985.

[Burt 85] Burt, Hill, Kosar and Rittelman.
Small Office Design Handbook.
Van Nostrand Reinhold, Florance, Ky, 1985.

[Harmon 85] Harmon, P. and King, D.
Expert Systems.
Wiley Press, New York, NY, 1985.

[Hayes-Roth 83] Hayes-Roth, F., Waterman, D.A. and Lenat, D.B.
Building expert systems.
Addison Wesley, Reading, MA, USA, 1983.

[Newell 72] Newell, A. and Simon, H.A.
Human Problem Solving.
Prentice-Hall, Englewood Cliffs, NJ, 1972.

Gerhard Schmitt - Chen-Cheng Chen - Jean-Christophe Robert - James Karl Week
Department of Architecture, Carnegie-Mellon University, Pittsburgh, PA 15213