U. FLEMMING
R. COYNE
T. GLAVIN
Department of Architecture
Carnegie - Mellon University
Pittsburgh
M. RYCHENER
Design Research Center,
Carnegie - Mellon University
Pittsburgh

ROOSI - Version One of a Generative Expert System for the Design of Building Layouts

Résumé. ROOS1 est la première version d'un système expert générateur de plans de bâtiments, pouvant s'appliquer à différentes sortes de problèmes. Le système ne reproduit pas la démarche des designers, mais plutôt cherche à compléter leur action grace à ses moyens de recherche systématique d'alternatives et à sa capacité à prendre en compte un nombre important de paramètres de design. Le développement du système vise à fournir un aperçu de l'applicabilité des techniques de l'Intelligence Artificielle à l'aménagement de l'espace et à la conception de bâtiments en général. Le système est basé sur une stratégie de génération puis test. Ses composantes principales sont un générateur d'alternatives, un module de test et une structure de contrôle (qui pourra être améliorée afin de devenir un véritable module de planification). Les éléments que le générateur juxtapose sont des rectangles. Les relations spatiales, au-dessus, au-dessous, à-gauche, à-droite, s'appliquent aux paires d'éléments intervenant dans le plan et sont les variables de base permettant de différencier les solutions et de produire les différentes alternatives. Le générateur agit indépendamment de toute contexte, et engendre toutes les façons possibles de disposer un nombre donné d'éléments. Le module de test, au contraire, est lié au type de problème rencontré et contient des informations relatives à la forme de plan demandée. Le système peut s'appliquer à plusieurs domaines en utilisant le module de test adéquat et éventuellement une structure de contrôle spécifique. La structure de contrôle guide la recherche de solutions, en activant alternativement le g'en'erateur et le module de test. Plusieurs versions du système, de complexité croissante, vont être développées. Chacune sera conçue selon une architecture claire et précise, et nous comptons évaluer chaque architecture en termes de possibilités potentielles et limitations relativement à divers domaines. Cet article décrit la première version du système.

Abstract. ROOS1 is the first version of a generative expert system for the design of building layouts that can be adapted to various problem domains. The system does not reproduce the behavior of human designers; rather, it intends to complement their performance through (a) its ability to *systematically* search for alternative solutions with promising trade-offs; and (b) its ability to take a *broad range of design concerns* into account. Work on the system also aims at providing *insights into the applicability of Artificial Intelligence techniques* to space planning and building design in general. The system is based on a general *generate-and-test paradigm.* Its main components are a *generator*, a *tester* and a *control strategy* (which is to be expanded later into a genuine *planner*). The generator is restricted to the allocation of rectangles. The spatial relations *above, below, to the left* and *to the right* are defined for pairs of objects in a layout and serve as *basic design variables* which define differences between solutions and govern the enumeration of alternatives. Within the class of layouts it is able to produce, the generator is completely general and able to generate all *realizable* sets of spatial relations for a given number of objects. In contrast, the tester is domain-specific and incorporates knowledge about the quality of layouts in a specific domain. The system can be applied to various domains by running it with the appropriate tester and, possibly, the appropriate control strategy. The control strategy itself mediates between planner and tester and, when expanded into a planner, is able to streamline the search for alternatives. The system will go through a sequence of *versions* with increasing complexity. Each version will have a conceptually clean and clear architecture, and it is our intention to evaluate each architecture explicitly in terms of its promises and limitations with respect to various domains. The first of these versions is described in the present paper.

1 Goals of Project

The project pursues two major goals:

1. to develop the prototype of a system capable of complementing the performance of human designers through its ability

    - to systematically enumerate alternative solutions which are distinguished by particular trade-offs
    - to take, at the same time, a broad and diverse spectrum of design concerns, performance criteria, guidelines or principles of good design into account

    The underlying assumption is that the human cognitive apparatus is not particularly well-suited to perform either of these tasks.

2. to gain insights into the applicability of Artificial Intelligence techniques and paradigms to problems in architecture and building design.

We selected the area of space planning or layout design as our focus for several reasons: through work that has been going on for almost twenty years, it is better understood than some other aspects of building design; it permeates building design at most of its stages and thus provides a rich context for the definition and investigation of problem domains with various degrees of complexity; and it allows to establish connections with other disciplines that also deal with layout design.

2 General Approach

The general approach we pursue is based on the generate-and-test paradigm ([6] pages 71-72 and 99-102), which separates the *generation* of a solution from its *evaluation*. In particular, we take advantage of a formal theory that establishes a generator for which we are able to prove some fundamental properties:

- closure (that is, every object produced by the generator belongs to the class of objects we are interested in)
- completeness (that is, every member of that class is generated); and
- non-redundancy (that is, the generation of every object is unique).

To make such a theory possible, we accepted the restriction that we can deal only with layouts composed of rectangles that are pairwise non-overlapping and are placed in parallel to the axes of an orthogonal system of Cartesian coordinates; we call such layouts *loosely-packed arrangements of rectangles*. But within this restriction, the generator is completely general and enables us to investigate the design of layouts in various domains, such as buildings on a site, rooms on a floor, or furniture in a room.

Any rectangle, z, in a layout is completely described by the coordinates of its lower left corner, (x_z, y_z), and by the coordinates of its upper right corner, (X_z, Y_z). The spatial relations *above, below, to the left* and *to the right* can then be defined on sets of rectangles as follows. If c and z are two rectangles, then

$c \uparrow z$ (read *c is above z*) $\Leftrightarrow y_c \geq Y_z$ (1)

$z \downarrow c$ (read *z is below c*) $\Leftrightarrow c \uparrow z$ (2)

$c \leftarrow z$ (read *c is to the left of z*) $\Leftrightarrow X_c \leq x_z$ (3)

$z \rightarrow c$ (read *z is to the right of c*) $\Leftrightarrow c \leftarrow z$. (4)

c and z *do not overlap* if at least one of the relations (1) to (4) holds between them.

Crucial properties of a loosely-packed arrangement of rectangles can be expressed in terms of these relations; examples are adjacencies, alignments or zoned groupings that play an important role in the design of layouts. As a consequence, constraints, criteria and design principles can be expressed as functions of these relations, and layouts that are described in terms of these relations can be evaluated over a broad spectrum of concerns. We therefore use relations (1) to (4) as basic design variables to define differences between alternative layouts.

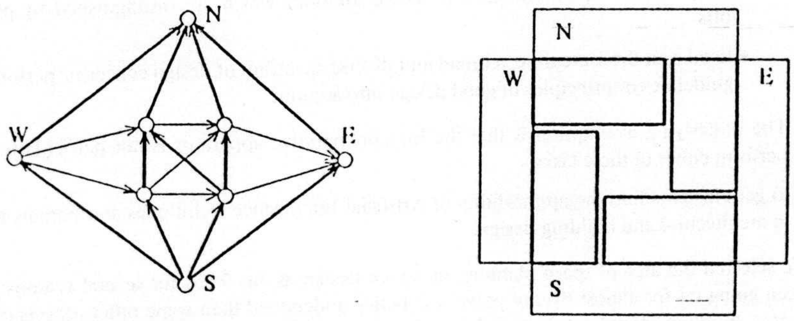

Figure 1: An orthogonal structure representing a loosely-packed arrangement of rectangles

We formally represent the relations that hold between the rectangles in a layout by an *orthogonal structure*, a directed graph whose vertices represent the rectangles in a layout and whose (colored) arcs represent spatial relations between pairs of rectangles. An example is shown Figure 1, which shows the structure representing a configuration of four rectangles forming a pinwheel (we add 'external' vertices N, E, S and W to orient a layout; details can be found in [3]). The conditions of well-formedness or syntactical correctness are known for orthogonal structures; (these are the conditions that assure that the relations depicted by such a graph can be *simultaneously realized* in a layout; see [2] for details).

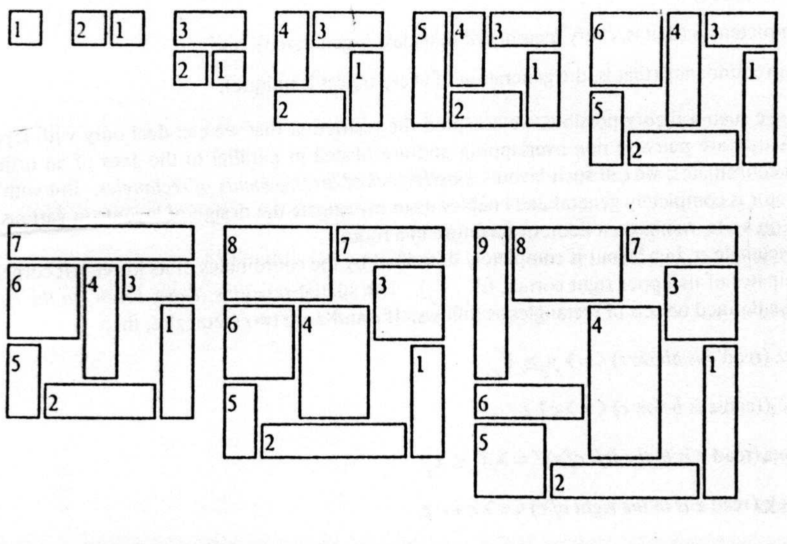

Figure 2: Layouts generated by successive applications of generation rules

Based on these conditions, we formulated rules that allow us to construct well-formed structures from well-formed structures. Examples of the effect of these rules are shown in Figure 2 which depicts a layout and its stepwise expansion by application of various rules, adding one rectangle at a time. The orthogonal structures that represent each layout were generated by the program LOOS, a fore-runner of the generator used in the present system [2]. The figure does not show the structures themselves; for ease of comprehension, it presents layouts represented by these structures. The dimensions of the rectangles in each layout are arbitrary and the overall layout can have an irregular outline; LOOS selects dimensions so that the relations that make up the underlying structure can easily be inferred from an inspection of a layout.

The generator can be complemented by various testers that use domain-specific knowledge about the quality of layouts in a particular domain to evaluate layouts in that domain. Unlike the generator, a tester is not established a priori for any domain. Various testers are to be built during the project through the process of 'knowledge acquisition' that seems to be typical for the development of expert systems: it consists of a series of iterations in which experts observe the performance of the program; criticize it; inspect the knowledge base that has been used and suggest additions to or modifications of it.

This approach is attractive to us because many of the principles and criteria used by human designers are not systematically documented and accessible; rather, they are accumulated over years of practice and remain largely unarticulated until a particular stimulus brings this expertise to the surface. A particularly effective mechanism to trigger this response is to confront experts with flawed solutions and ask them to criticize them. In our experience, this mechanism can be very successful in building design, and we are very much interested in exploring its applicability in the present context.

Finally, we plan to implement the system in a *series of versions* of increasing complexity. Each version will have a conceptually clean and clear architecture, and we plan to evaluate each version explicitly from both a methodological and practical point of view. The first version, ROOS1, will be described in the following section (a more detailed description is given in [4]).

3 ROOS1 - Components

ROOS1 consists of the five basic components shown in Figure 3. Each of these can be modified (in fact, replaced) independent of the other components. Furthermore, each component is programmed as a collection of production rules, using the OPS83 language [5].

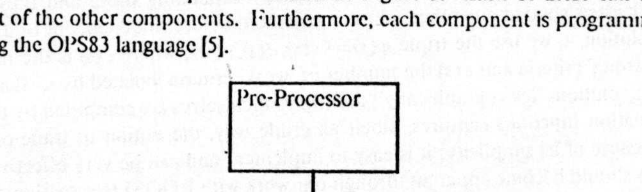

Figure 3: ROOS1 - components

Pre-Processor

The pre-processor accepts from the user a problem statement consisting of a *context description* and a *list of objects* to be allocated. The simple domains we have used to set up the system deal with furniture and fixture layouts in small spaces such as bathrooms or residential kitchens. The context typically consists of a specification of walls, windows and doors forming the boundary of a space; to these can be added existing fixtures such as radiators or plumbing stubs. All of these objects are considered rectangles, and their shape and position are specified by the user through their corner coordinates. From these specifications, the pre-processor generates an orthogonal structure that represents the configuration of context elements; it functions as the starting configuration for the search (an example is shown in Figure 4).

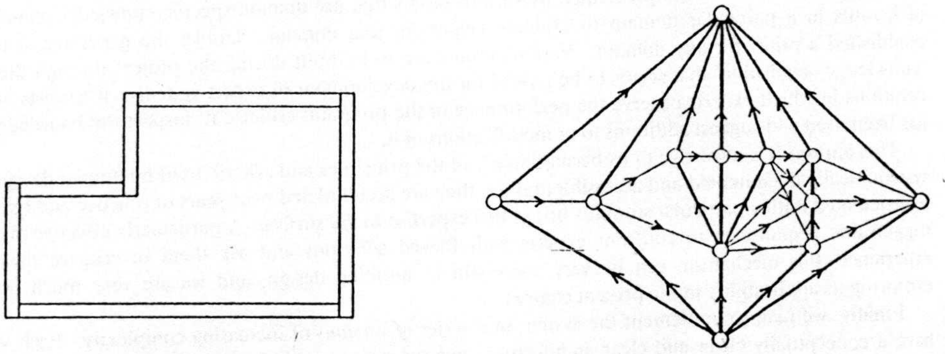

Figure 4: Example of a context and its internal representation

Control Strategy

The control strategy directs the search for alternatives into promising directions and prunes the search tree. For ROOS1, we selected a *branch-and-bound* strategy expanding those and only those intermediate solutions which are at least as good as any other solution generated before (independent of its depth in the search tree). Tc evaluate a solution, s, we use the triple $\varphi(s)=\langle c(s), d(s), e(s)\rangle$, where $c(s)$ is the number of constraints, $d(s)$ the number of 'strong' criteria and $e(s)$ the number of 'weak' criteria violated by s. Based on these triples, the search strategy ranks solutions 'lexicographically'; the triples themselves are computed by the tester.

The evaluation functions captures, albeit an crude way, the notion of trade-offs between alternatives. We selected it because of its simplicity; it is easy to implement and can be very effective for simple layout problems. Its limitations should become apparent through our work with ROOS1 (see section 4).

The constraints and criteria evaluated by the function depend on the spatial relations that characterize a layout. Our control strategy works if it can be guaranteed that these relations do not change between rectangles once they have been allocated; that is, spatial relations are *invariant on each branch of the search tree*, a property that must be assured by the generator.

Generator

The generator used in ROOS1 consists of the single rule specified in Figure 5. The rule is a recursive re-write rule that can be applied to an orthogonal structure containing the left-hand side of the rule as a sub-structure; the application consists in substituting the right-hand side for the left-hand side in the structure. Intuitively, one can view the rule as 'pushing' rectangles $v_1,\ldots,v_m$ 'to the side', thus creating space for the insertion of a new rectangle, n. In Figure 5, the rule is specified in a particular orientation. But it should be noted that it can be applied also in rotated versions.

The rule is sufficient to generate alternative spatial relations for a small number of rectangles. But it does not generate all possible sets of relations. In order to become complete, the generator will have to expand through the addition of rules, a task we plan to accomplish with later versions.

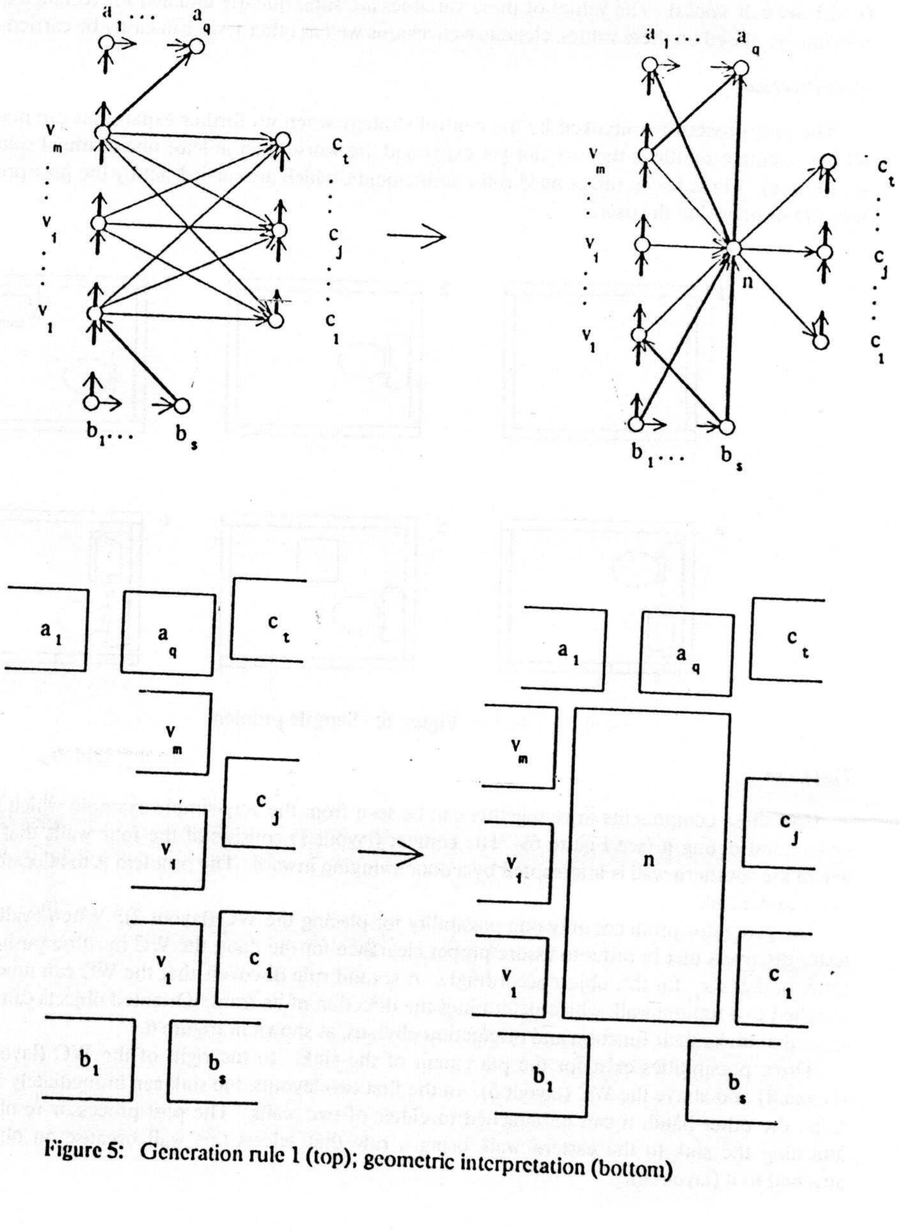

Figure 5: Generation rule 1 (top); geometric interpretation (bottom)

Tester

At the time of this writing, the tester contains very limited knowledge about layouts. It checks (a) if any newly allocated object can fit physically into a layout, and (b) if front and side clearances required for certain objects are in fact available. While carrying out test (a), the tester determines for each newly allocated rectangle coordinates x_{lo}, x_{hi}, y_{lo} and y_{hi}, which give lower or upper limits for the dimensions of its corner points; the tester also calculates the maximum distances by which a rectangle can move between these limits in the x and y direction (which we call *slacks*). The values of these variables are subsequently updated for rectangles that were allocated previously. Based on these values, clearance checks, as well as other tests, can easily be carried out.

Post-Processor

The post-processor is invoked by the control strategy when no further expansions are possible; that is, when all intermediate solutions that are not yet expanded are worse than at least one terminal solution (or leaf of the search tree). These leaves might need some refinements, which are carried out by the post-processor. In the end, they are displayed to the user.

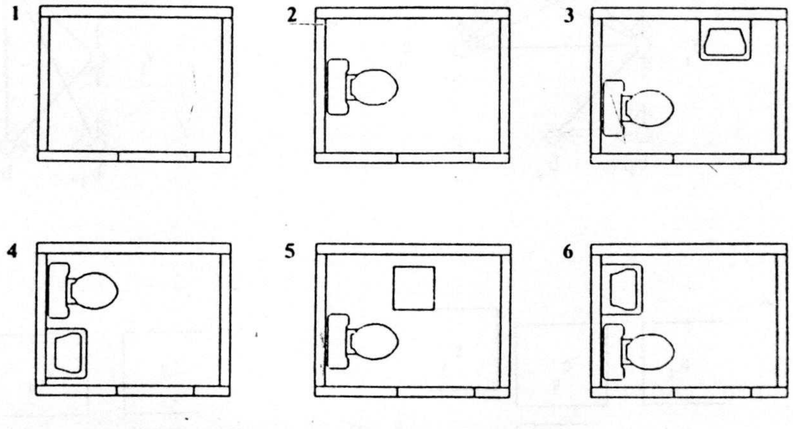

Figure 6: Sample problem

Test Case

How these components work together can be seen from the very simple example which we used to set up the system and debug it (see Figure 6). The context (layout 1) consists of the four walls that enclose a bathroom, where the southern wall is interrupted by a door swinging inward. The problem is to allocate in that room a water closet and a sink.

The generator produces only one possibility for placing the WC (layout 2). When evaluating this layout, the tester discovers that in order to assure proper clearance for the door, the WC must be pushed to the left, and the tester updates x_{hi} for this object accordingly. A second rule discovers that the WC can now be 'oriented', that is, attached to a unique wall, which determines the direction of its front. Oriented objects can be displayed through icons that make their function and orientation obvious, as shown in Figure 6.

Three possibilities exist for the placement of the sink: to the right of the WC (layout 3); below the WC (layout 4) and above the WC (layout 5). In the first two layouts, the sink can immediately be oriented. In layout 5, on the other hand, it can be attached to either of two walls. The post-processor resolves this ambiguity by attaching the sink to the eastern wall, using a rule that selects this wall because an object has already been attached to it (layout 6).

4 Outlook

At the time of this writing, the limits of ROOS1 have not yet been reached. Based on our experience so far, however, we expect to implement the following changes when we start work on version 2 of the system.

It is pleasing from an aesthetic point of view to program every component consistently as a collection of rules. This gives formal unity to the system and introduces a discipline of its own. We have discovered, however, (as have other workers in the field) that the computational efficiency of the resulting system is low and starts to become an obstacle as soon we go beyond the most trivial problems. Version 2 will therefore be a *hybrid system* in which those components that are well-understood (the pre-processor and generator) are programmed in the traditional, procedural way (OPS83 as well as other shells make the switch from one style to the other within the same system easy). Only those components which will take shape through work *with* the system (notably the tester and post-processor) will remain rule-based. Our experience with ROOS1 shows that these parts tend to change and expand in unpredictable ways, and that a rule-based approach is ideal for supporting the process of knowledge acquisition, at least in the present context.

The generator itself will have to contain more rules if the number of objects increases and if the context becomes more complex. Under these circumstances, we also expect our simple control strategy to break down. In subsequent versions, we plan to expand it into a genuine *planner* (see [1]), and we intend to explore two types of planners:

1. *Strategic planner.* It determines the most promising directions in which the search should proceed. In the design of service cores for high-rise office buildings, for example, it might be advantageous to determine, at the outset, the few ways in which the given elevators can be banked and to arrange the remaining areas around these clusters; this contrasts with a blind search that would generate all possible elevator configurations only to discover later that most of them make little sense.

2. *Hierarchical planner.* This planner divides the generation process into *phases* through which the design evolves at decreasing levels of abstraction. In space planning, this happens typically when rooms are combined into larger components, which are allocated first and in which sub-components or individual rooms are placed at later phases. It should be noted that within the limitations accepted by the present system, the same generator can be used to find alternatives at different levels of abstraction.

Implementation of a planning capability will commence, if not with version 2, then with version 3, work on which is planned to start in the fall of 1986. With these versions, we will enter new terrain (in the context of space planning) and the most interesting phase of the current project.

Acknowledgements

Work reported here is supported by a grant from the National Science Foundation, with additional funds from the Center for Art and Technology, College of Fine Arts, Carnegie-Mellon University. Thanks to Jean-Christophe Robert for providing us with a translation of our abstract!

References

[1] Buchanan, B.; Sutherland, Georgia and Feigenbaum, E.A. HEURISTIC DENDRAL: a program for generating explanatory hypotheses in organic chemistry. In Meltzer, B. and Michie, D. (editor), *Machine Intelligence 4*, pages 209-254. Edinburgh University Press, Edinburgh, 1969.

[2] Flemming, U. On the representation and generation of loosely-packed arrangements of rectangles. *Planning and Design* 12, 1985. [forthcoming].

[3] Flemming, U.; Coyne, R.; Glavin; T. and Rychener, M. A generative expert system for the design of building layouts. In R. Adey, and Sriram, D. (editors), *Applications of Artificial Intelligence in Engineering Problems*, pages 811-821. Springer, New York, 1986a.

[4] Flemming, Ulrich; Coyne, Robert; Glavin, Timothy and Rychener, Michael. *A Generative Expert System for the Design of Building Layouts - Version 1.* Technical Report, Center for Art and Technology, Carnegie-Mellon University, Pittsburgh, PA, 1986b.

[5] Forgy, Charles L. *OPS83 User's Manual and Report.* Production Systems Technologies, Pittsburgh, PA, 1985.

[6] Hayes-Roth, Frederick, Waterman, Donald A. and Lenat, Douglas B. (editors). *Building Expert Systems.* Addison-Wesley, Reading, MA, 1983.

U. Flemming, R. Coyne, T. Glavin
Department of Architecture, Carnegie-Mellon University,
Pittsburgh, PA 15213, U.S.A.
(412) 268-2363

M. Rychener
Design Research Center, Carnegie-Mellon University,
Pittsburgh, PA 15213, U.S.A.
(412) 268-8777