Is design different?

William Fawcett

Unpublished manuscript of 1984.

william.fawcett@carltd.com
IS DESIGN DIFFERENT?

William Fawcett*

February 1984
(Revised April 1984)

This paper considers whether established problem-solving techniques can be applied to architectural design. Assumptions underlying a successful non-architectural design system are analysed. Taking floor planning as an architectural example, the use of optimising, enumerative and knowledge-based methods are compared. The importance of higher-level primitives is emphasised.

Submitted for the IFIP Working Conference on Knowledge Engineering in Computer-aided Design, Budapest, September 1984

* Lecturer, University of Hong Kong, School of Architecture, Pokfulam Road, Hong Kong
1. INTRODUCTION

Writers on design, and many design practitioners, seem to enjoy making out that design is somehow different from any other human activity. One implication is that problem-solving techniques which have been successful in any other domain can be rejected in the domain of design. Advocates of this position are not uncommon in the sub-field of architectural design. And it must be admitted that scepticism has been reinforced by the weakness of some early attempts at systematic design methods. So the question is still open: is design different?

2. CHARACTERISTICS OF A SOLVABLE DESIGN PROBLEM

A recent survey of knowledge engineering defines design as "configuring objects under constraints" (1). This corresponds closely to the knowledge-based expert system which stands out as the most convincing example so far of knowledge engineering techniques applied to a design or design-like problem - McDermott's R1 (2,3). I am sure that many would aspire to emulate R1's reported successes - but how typical is it of problems that we familiarly regard as design problems? We can rephrase the question: is design different from R1?

The starting point for R1 is a schedule of distinct physical components, each with distinct attributes. They are actually computer hardware components, but that does not seem to be important. The task is to put all these components together in a satisfactory configuration. Let us consider some important features of this particular task:

1. the schedule of components is certain to be complete (there is a pre-configuration step to check that this is so)

2. there is an unambiguous criterion for success or failure - all legal configurations work
3. the step-by-step development of a configuration never needs to backtrack

4. there is no concern for better or worse configurations, where alternative configurations exist for a given schedule of components.

These are powerful simplifications, which presumably contribute to RI's success, and they do not apply to architectural design problems as conventionally perceived. Indeed some people seem to argue that it is precisely an ill-defined quality that makes a problem a design problem; but this is circular - it means that systematic design is excluded by definition. It also means that the domain of design is necessarily shrinking with every successful systematic problem-solving strategy - and that the computer configuration problem ceased to be a design problem as soon as RI solved it. But this is absurd, and we shall go on to consider whether RI's simplifying assumptions could be applicable to other design problems, and in particular to architectural design problems.

3. SIMPLIFYING ASSUMPTIONS

RI's task, as the definition states, is to configure objects under constraints. A natural reaction is to assert that design is much more than configuring objects under constraints; or at least, that design professionals spend a lot of time doing other things. What things? Discussions with clients, scanning catalogues, checking codes of practice, checking cost or functional or other criteria, etc, and also actually producing documents and drawings. Many of these activities are far from unique to design professionals. It might be tempting to say that design is not another distinct activity, but rather the name given to the collection of activities of designers - but this simply perpetuates wooliness. Perhaps if a design office is distinguished from other professional offices by what it produces, instead of what goes on
in the office, it is clearer: that designers produce designs, and in
general terms designs are configurations of objects, with associated data.
If the output of design is fairly clear, what is the input?

3.1 Complete schedule of components This is the central point in
matching RI's first simplifying assumption. Suppose that the complete
process of moving from client requirements to a design solution can be
divided into two stages - programming and design. The task of programming
is to translate a problem which is perceived in behavioural and functional
terms, into a problem stated in terms of spaces, dimensions, adjacencies,
and other physical attributes (I use the terminology of architecture, but
I assume the principle can be generalised). Then the design stage starts,
with a problem stated in terms of objects and attributes that has to be
solved by generating a configuration of those same objects. The program-
ning/design model is a hierarchical model, with a client level and a
design level. Programming translates between levels, design operates
within one level.

To return to RI's first simplifying assumption, we can match RI's problem
statement of a complete schedule of components if - a big if - programming
skills are adequate. This may be difficult. Programming may have its own
share of uncertainty, judgement and expertise - but is it a design skill?
Architects may like to fuse the two, programming and design, into an
undifferentiated bundle, arguing that they are highly interrelated.
Certainly they are interrelated, to the extent that ill-defined
programming gives an ill-defined design problem. But I would be
reluctant to see difficulties in programming stand as a barrier to
advances in design. It is worth proceeding on the hypothesis that the
programming problem can be solved, and I argue that RI's first
simplifying assumption can in principle be generalised to other design
problems.

3.2 Unambiguous criterion of success Having a yes/no test is handy,
but cannot be considered an essential feature for systematic design. It
makes for elegance and efficiency, but many knowledge engineering systems
deal with uncertain knowledge and achieve significant results.
3.3 No backtracking  This is another bonus for Rl. It is both unlikely and unnecessary to find it repeated in all design problems. It presumably results from particular features in Rl's problem domain. Many inference mechanisms used in knowledge engineering involve orderly backtracking procedures, and their use in design need not cause a problem.

3.4 No concern for better configurations  To differentiate better from worse configurations some criteria of quality are required. In Rl's case the existence of the unambiguous criterion of success makes it feasible to ignore finer questions - an acceptable standard is assured. But where in Rl is the standard assured? It is not articulated as an explicit objective function or goal statement, against which design decisions are tested: it is not part of the problem statement. Rather it is implicit in the operation of Rl's rules, which contain knowledge about acceptable standards. In Simon's terms (4) we have constrained generation of designs, not a generate-and-test sequence.

Rl's rules of course contain multiple criteria for a successful configuration. Whilst "success" could be considered as a single performance objective, in fact it is highly fragmented. Nor could the mould of of a multi-criterion objective function fit these diverse criteria comfortably. And this, surely, is the innovation of the knowledge engineering paradigm, which distinguishes it from the operational research/optimisation tradition: it can accept large quantities of knowledge, which is often fragmentary, and achieve good performance by holding knowledge in the operation of problem-solving techniques. It is knowledge-directed decision making, rather than goal-directed decision making.

When this principle is applied to other design problems without Rl's unambiguous criterion, questions of quality, of better and worse designs, must be contained in operational knowledge. This is not the same as moving to full optimisation - but the imposition of of pragmatically acceptable performance, which might be pushed incrementally forward.

Thus, although other design problems may not be able to share Rl's lack of concern for better or worse configurations, the way that Rl
achieves a required performance standard by operational knowledge, without an articulated objective function, is something one would expect to be generally applicable to design problems.

To sum up, it seems that RL may be unusually fortunate in its problem structure, making the system very elegant and efficient, but RL does not seem to make any fundamental assumptions that are incompatible with more general application of systematic methods to design problems.

4. EXAMPLE: FLOOR PLANNING

Let us take as an example a very well-established model of an architectural design problem, the layout problem. This has a long and possibly honourable history, though heroic failure may describe the net result achieved.

Intuitively, the problem is to design a building plan. This type of problem was formalised by Koopmans and Beckmann in 1957 as the quadratic assignment problem. In essence, the quadratic assignment problem requires as input a list of components and a matrix of cost or flow data relating each pair. There is a simple objective function: to achieve a configuration that minimises the cost penalty of interaction between components. In quadratic assignment algorithms candidate configurations are tested against this single criterion and the best is selected. Normally the number of possible configurations is so great that an exact solution cannot be guaranteed. An extraordinary amount of work has gone into this problem, and recent results, eg. by Liggett (5), suggest that performance thresholds have been reached.

There have been many criticisms of quadratic assignment algorithms. One interesting observation is that the output of algorithms is highly sensitive to marginal variations in input data (6). This is disturbing, as nobody claims that quadratic assignment give more than a general indication of satisfactory layout, leaving realistic detail to be added. It follows that stable solution-types without detail would be more use than unstable solutions encumbered with unhelpful detail.
This suggests an alternative approach to the question of architectural layout, studying the range of possible solutions rather than searching for unique solutions. The emphasis is on enumeration, not optimising. The analysis of possible architectural layouts has received attention under the name of configurational studies (7), but exhaustive enumeration hits severe size limits. Comparison of spatial types is less limited by size, but less rigorous (8). The problem of matching a problem statement to all possible spatial solutions has been attempted on a small scale, with 7 rooms (9). In this example the quadratic assignment problem's objective function was not used, but was replaced with an alternative single-criterion objective function measuring the floor area cost of candidate solutions. An optimum solution to this problem has been claimed by Gero (10).

The objection to all this by a competent architect is that it is, architecturally, so hopelessly unsophisticated. Can the knowledge engineering paradigm raise the level of architectural attack?

5. AN ARCHITECTURAL APPROACH TO FLOOR PLANNING

Let us look at a small and simple example - 9 component spaces for which a configuration is required. The list of spaces is supplemented by a matrix giving the weight of interaction (fig.1a). It is assumed that the configuration is to be set out on a square planning grid. In this problem statement we have already met Rl's first simplifying assumption. This input is quite sufficient for an automated layout algorithm. It is not necessary to say what activities take place in the component spaces, and the algorithms could not use the information anyway. A typical output is shown (fig.1b).

I gave this example to some architectural students, and, assuming they needed a bit more information, I also told them the activities intended for each space; that is, the spaces were given names (fig.2a). Their configurations are shown (fig.2b). Although the students were not asked to optimise with respect to the cost-minimising objective function,
their results can be compared with the automated design on the basis of cost-minimising (fig.3).

The configurations can also be evaluated differently - in what could be called, for want of a better word, an architectural way. This is rather hard to define, and rather subjective. The ranking of solutions is then different - my own architectural criteria are revealed by a few designs that I selected as being particularly good or bad (fig.3b). We have two standards - minimum cost penalty and "architectural"; how can they be compared? I would argue that the architectural standard itself contains cost-penalty considerations as one factor.

Before sketching a strategy for formalising the architectural approach, one comment. Architecturally the problem is too simple - there is not enough information to work with. What is the organisation? what are the interactions? what about circulation? what is the site? what is the orientation? is is in a multi-storey building or on the ground? All these and other fragments of information may affect the configuration, and without them the architect is almost paralysed. The algorithms with explicit objective functions, on the other hand, have no way of responding to additional information.

The architect, however, is not completely reliant on massive data inputs. He can use experience, make sensible assumptions, and so on; in fact, use his architectural knowledge.

6. A KNOWLEDGE-BASED APPROACH TO FLOOR PLANNING

I attempted this floor planning exercise myself, and arrived at two configurations that I was reasonably satisfied with - that is, I didn't think it was worth continuing to search for better ones (fig.4). By introspection, it seemed to me there were three main criteria I was applying, in order of priority:
1. coherent overall configuration

2. coherent component shapes

3. correct adjacencies.

Now, R1 can accommodate any amount of knowledge about individual objects to be configured, and hence can also know how objects can or cannot relate to each other in a configuration. This kind of knowledge could readily be incorporated in a floor planning system, though in the example there is actually a dearth of knowledge. But what about "coherence"?

In this context it simply means "fairly neat and tidy" - and is a relatively high-level concept, deriving from the pattern formed by the aggregate of objects, or cells in an object. If there are performance criteria concerning the aggregate which cannot be expressed as constraints on individuals, then it is hard to see how backtracking can be avoided. It seems that R1 avoided the use of high-level concepts of this type, and gained thereby in efficiency.

A systematic method for floor planning, if it is to satisfy my criteria, must deal with higher-order configurational concepts, like coherence. And I don't think it can do so in a generate-and-test way, because the random generation of configurations produces so few that are coherent: generate-and-test would probably never succeed. Constrained generation is required.

Perhaps the most direct expression of pattern or order, or coherence, in configurations is symmetry. It is also a well-defined concept (11). I propose that a first step in giving a floor planning system the higher-level knowledge it needs, is to formalise symmetry as a higher-level primitive. For 2-dimensional configurations on a square grid there are five possible symmetries: $D_1$, $D_2$, $D_4$, $C_2$, $C_4$ - dihedral and cyclic symmetry of restricted types (fig.5). Clauses can be written in, for example, PROLOG to check whether an array of cells has symmetry, and of what type. It is also possible to check whether symmetry is exhibited
in the overall configuration, or in the internal arrangement of objects as well (fig.6) - perhaps using the idea of "walls" and "partitions" (12).

This is a mechanism for testing, but I said that constrained generation would be needed. It is of course a characteristic of PROLOG clauses that they can be used to seek instantiation of any variables. Thus if a given symmetry is specified, we should be able to generate an array of cells to achieve it. I envisage that the role of this knowledge in a design system could be to complete partial configurations, by adding the residue of cells in locations which create a symmetrical goal configuration (fig.7). Objects are placed in accordance with the goal configuration, but if they cannot be placed, the system backtracks and identifies an alternative goal configuration with different symmetry.

I have only mentioned one higher-level configurational primitive, symmetry, but there can be others. It is interesting to consider a broad definition of configurational order corresponding to algorithmic information theory (13).

This is still only a sketch. I anticipate some difficulties:

1. on what basis does the system start - I have only indicated a proposal for completing partial configurations

2. the number of goal configurations may be unmanageably large (even though they are a very small subset of possible configurations)

3. how bad does the placing of objects within a goal configuration have to get before the attempt fails and triggers a backtrack

4. if a goal configuration is satisfied, do you go on looking for superior goals.

Other high-level concepts may be relevant to these problems, for example my own approach to the example problem involved identifying coherent sub-configurations of related objects, before attempting a complete configuration.
7. CONCLUSION

The main extension to the problem-solving features of RI that I propose for a knowledge-based floor planning design system, is the use of higher-level concepts about configurations.

The use of higher-level primitives is common to advances in computer applications in many fields - indeed it can be seen as the general characteristic of "very high level languages" (14). In architectural CAD the same principle is seen in attempts to build intelligent drawing systems (15). The description of high-level primitives in a language with a procedural interpretation could perhaps synthesise the tasks of representation and problem-solving.

From this perspective design takes a natural place in the spectrum of problem-solving activities, and leads to a clear negative answer to the original question, "is design different?".
REFERENCES


2. J McDermott (1982a) "Domain knowledge and the design process" *Design Studies* vol.3, pp.31-36


8. T M Willoughby (1975b) "Building form and circulation patterns" *Environment & Planning B* vol.2, pp.59-88


12. L Combes (1976) "Packing rectangles into rectangular arrangements" *Environment & Planning B* vol.3, pp.3-32


15. A Bijl (1982) "Dumb drawing systems and knowledge engineering" *CAD82* Brighton
FIGURES

<table>
<thead>
<tr>
<th>SPACE</th>
<th>AREA</th>
<th>INTERACTION WITH</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1   2   3   4  5  6  7  8  9</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>2   2</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>0   2  0</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>4   5  2  1</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>2   2  2  0  4</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>2   2  2  0  3  3</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>1   1  2  2  3  4  2</td>
</tr>
<tr>
<td>9</td>
<td>11</td>
<td>1   1  2  1  1  2  3  2</td>
</tr>
</tbody>
</table>

(a)

\[ C = 76 \]

(b)

Figure 1. (a) Data input for quadratic assignment algorithm.
(b) Typical output by sequential allocation algorithm (CORELAP).
A value for the cost penalty of interaction (C) is given.

<table>
<thead>
<tr>
<th>SPACE</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>reception &amp; waiting</td>
</tr>
<tr>
<td>2</td>
<td>conference</td>
</tr>
<tr>
<td>3</td>
<td>toilets</td>
</tr>
<tr>
<td>4</td>
<td>kitchen</td>
</tr>
<tr>
<td>5</td>
<td>private office 1</td>
</tr>
<tr>
<td>6</td>
<td>private office 2</td>
</tr>
<tr>
<td>7</td>
<td>private office 3</td>
</tr>
<tr>
<td>8</td>
<td>general office</td>
</tr>
<tr>
<td>9</td>
<td>drawing office</td>
</tr>
</tbody>
</table>

(a)

Figure 2. (a) Additional data input for architectural students.
Figure 2 (continued). (b) Configurations designed by architectural students.
Figure 3. Selected configurations designed by architectural students, based on "architectural" evaluation: (a) particularly good, (b) particularly bad. Values for the cost penalty are also given.

Figure 4. The author's configurational designs. Values for the cost penalty are given.

Figure 5. The five types of symmetry on a 2-dimensional square grid: dihedral symmetries (a) $D_1$, (b) $D_2$, (c) $D_4$; cyclic symmetries (c) $C_2$, (d) $C_4$. 
Figure 6. Two configurations for the same schedule of cells, with the bounding "walls" drawn solid and the internal "partitions" with broken lines. (a) A configuration with overall symmetry \(D_1\) but no internal symmetry; (b) a configuration with overall and internal symmetry \(D_1\).

Figure 7. (a) A partial configuration of 14 cells, to be completed by the addition of 10 cells. Goal configurations for different symmetries:
(b) \(D_1\), (c) \(D_2\), (d) \(D_4\), (e) \(C_2\), (f) \(C_4\).

The goal configurations relate only to overall symmetry.