Parametric Design Problem Solving

Enrico Motta, Zdenek Zdrahal

Knowledge Media Institute
The Open University
Walton Hall,
Milton Keynes,
MK7 6AA
United Kingdom
{e.motta, z.zdrahal}@open.ac.uk

Abstract. The aim of this paper is to understand what is involved in *parametric design problem solving*. In order to achieve this goal, in this paper i) we identify and detail the conceptual elements defining a parametric design task specification; ii) we illustrate how these elements are interpreted and operationalised during the design process; and iii) we formulate a generic model of parametric design problem solving. We then re-describe a number of problem solving methods in terms of the proposed generic model and we show that such a re-description enables us to provide a more precise account of the different competence behaviours expressed by the methods in question.

1. INTRODUCTION

Design is about constructing artifacts. This means that, broadly speaking, any design process is 'creative', in the sense that a design process produces a 'new solution', as opposed to selecting a solution from a predefined set. While recognizing the essential creative elements present in any design process, researchers, e.g. Gero (1990), restrict the use of the term 'creative design' to those design applications where the *design elements* of the target artifact cannot be selected from a predefined set. For instance, when designing a new car model it is normally the case that some design innovations are included, which were not present in previous car designs. In other words it is not (always) possible to characterise the process of designing a new car as one in which components are assembled and configured from a predefined set. Nevertheless, in a large number of real-world applications it is possible to assume that the target artifact is going to be designed in terms of predefined design elements. In such a scenario the design process consists of assembling and configuring these pre-existing design elements in a way which satisfies the *design requirements* and *constraints*, and approximates some, typically cost-related, *optimization criterion*. This class of design tasks takes the name of *configuration design* (Stefik, 1995). In many cases, typically when the problem in hand does not exhibit complex spatial requirements and all possible solutions adhere to a common solution template, it is possible to simplify the configuration design problem even further, by modelling the target artifact as a set of *parameters* and characterizing design problem solving as the process of assigning values to parameters in accordance with the given design requirements, constraints, and optimization criterion. When this assumption is true for a particular task, we say that this is a *parametric design* task. The VT application (Marcus et al., 1988; Yost and Rothenfluh, 1996) provides a well-known example of a parametric design task.

The aim of this paper is to understand what is involved in parametric design problem solving. In order to achieve this goal, in this paper i) we identify and detail the conceptual elements defining a parametric design task specification; ii) we illustrate how these elements are interpreted and operationalised during the design process; and iii) we produce a generic model of parametric design problem solving, characterised at the knowledge level, which generalizes from existing methods for parametric design. We then re-describe a number of problem solving methods in terms of the proposed generic model and we show that such a re-description enables us to provide a more precise account of the different competence behaviours expressed by the methods in question.

Many authors - e.g. (Wielinga et al., 1995) - distinguish between an *analysis* and a *synthesis* phase in the design process. The analysis phase consists of transforming a set of "needs and desires" into a formal set of requirements and constraints, while the synthesis phase is concerned with the construction of a solution design which satisfies the given formal specification. In this paper we will focus almost exclusively on the synthesis phase, although our discussion (i.e. analysis!) of the parametric design task specification is obviously also relevant to the analysis phase, as it precisely defines its output.

The paper is organised as follows: in the next section we formally describe the structure of parametric design applications; in section 3 we present a *problem space* view (Newell, 1980) of parametric design problem solving, and we re-interpret task knowledge for parametric design in terms of the problem space framework. In section 4 we describe a number of problem solving methods for parametric design. In section 5 we illustrate a generic model of parametric design problem solving. In section 6 we apply the generic model to the methods presented in section 4 and we use the resulting re-descriptions to compare and contrast their behaviour. Finally, in section 7 and 8 we compare our work to other relevant contributions and we re-state the main contributions of this paper.

2. PARAMETRIC DESIGN PROBLEM SPECIFICATION

A parametric design application can be characterised as a mapping from a six-dimensional space <P, Vr, C, R, Pr, cf> to a set of solution designs, $\{D_{sol1}, ..., D_{soln}\}$, where

P = Parameters = $\{p_1, ..., p_n\}$;

Vr = Value Ranges = $\{V_1, ..., V_n\}$, where $V_i = \{v_{i1}, ..., v_{il}\}$;

C = Constraints = $\{c_1, ..., c_m\}$;

R = Requirements = $\{r_1, ..., r_k\}$;

Pr = Preferences = $\{pr_1, ..., pr_j\}$;

cf = Global Cost Function.

These concepts are described in the next sections.

2.1. Valid designs

2.1.1. Parameters and design models. In accordance with the problem space model - see figure 1 - the design process can be characterised as a search through a space of possible *design states*, where each design state is uniquely defined by an associated *design model*, consisting of an assignment of values to a set of parameters. In the rest of the paper we will refer to a design model, say D_k, by using the notation $D_k = \{<p_i, v_{ij}>\}$, where $p_i \varepsilon P$, and $v_{ij} \varepsilon V_i$. The following definitions characterise different types of design models:

- A design model D_k is *complete* if each parameter, say p_i, in P has a value in D_k.
- A design model D_k is *admissible* if no constraint in C is violated in D_k.
- A design model D_k is *valid* if it is *admissible* and satisfies all applicable requirements.
- A design model D_k is a *solution* to a parametric design problem if it is complete and valid.

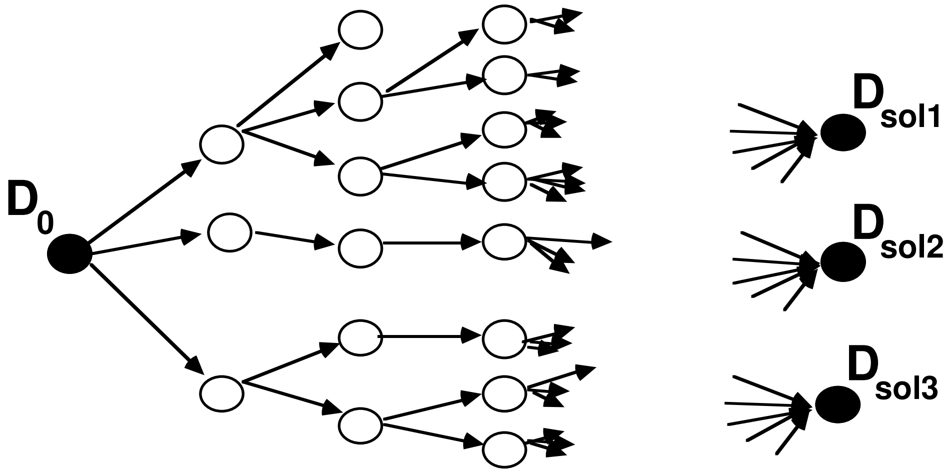

Figure 1. Problem space view of the design process

2.1.2. Constraints and requirements. A constraint specifies a condition which has to be satisfied by an admissible design. For instance, the VT elevator design application includes constraints such as "The cab height must be between 84 and 240 inches, inclusive". Requirements are also constraints and, as discussed in (Wielinga et al., 1995), the difference between requirements and constraints is conceptual rather than formal. Requirements typically have a 'positive' connotation, in the sense that they describe the desired properties which must be satisfied by the solution, while constraints have a 'negative' connotation, in the sense that they limit the space of admissible designs, by expressing the applicable technological or physical constraints. Moreover, constraints normally specify case-independent restrictions, while requirements are typically case-specific. For instance, a requirement in the VT application specifies the desired capacity of the target elevator, which is of course case-specific.

2.1.3. Legal values. For each parameter, say p_i, we assume that a value range, V_i, is given, which specifies the set of legal values which can be assigned to p_i. In principle, this value range can be either finite or infinite. For instance, the value of parameter motor-model in the VT domain can be one of six possible types of motor models, while the value of parameter hoist-cable-traction-ratio is constrained to a real number in a certain interval and therefore is potentially infinite. However, a value range of infinite cardinality does not imply that a parameter can really be assigned anyone of an infinite number of values. For instance the value of hoist-cable-traction-ratio in the VT domain is functionally determined by the values of other parameters, whose value range is finite. Therefore its value range is in practice finite too. Our experience with a number of parametric design applications suggests that it is very unlikely that the value of a parameter can be really drawn from an infinite range. Typically, when the range of a parameter is potentially infinite and it is not the case that its value is functionally determined by the values of other parameters, additional heuristic knowledge is brought in, either as part of the task specification itself, or as part of domain-specific, heuristic problem solving knowledge, to limit the number of available options - see discussion in section 3.1.2.

2.2. 'Better' and 'worse' solutions

The concepts discussed so far (i.e. parameters, constraints, requirements, and design models) provide us with the required conceptual vocabulary to discuss parametric design applications in which the goal is to produce valid, complete designs. However, practical parametric design tasks are not just concerned with finding valid and complete designs, but they often introduce optimization aspects, usually associated with cost criteria. In a simple scenario the cost of a design can be characterised as the sum of the monetary costs of its design elements, while in other cases more complicated metrics might be needed. For example, the cost criterion used by VT design experts characterises the cost of a solution design in terms of the 'distance' between this design and an ideal one. This distance is computed by considering all the 'corrective steps' (*fixes* in the VT domain terminology) required to produce a design solution. Intuitively, this informal criterion tries to assess the cost of a design not just in terms of the cost of the individual components, but in terms of the broader costs, which include the cost of the design process itself, and the expected maintenance costs. In a nutshell, the cost factor in a parametric design application is of paramount importance and cannot be necessarily reduced simply to the sum of the financial costs of the individual design elements.

In order to account for the cost-related aspects of parametric design problems our framework includes two additional concepts: *preferences* and *cost function*, which are discussed in the next two sections.

2.2.1. Preferences. Preferences describe task knowledge which, given two design models, D_1 and D_2, allows us to make a judgement about which of the two - if any - is the 'better' one, in accordance with some criterion. For instance, the Sisyphus-I room allocation problem (Linster, 1994) includes informal statements such as "Secretaries should be as close as possible to the head of group" and "Project synergy should be maximised". Given that these statements specify desired solution features, we could consider modelling them as requirements. If we take this approach, then we define a solution allocation model as one which (among other things) maximises project synergy and in which the distance between secretaries and head of group is minimized. Although this approach is possible, it is not entirely satisfactory. A statement such as "secretaries should be as close as possible to the head of group" appears to indicate a criterion which might or might not be entirely satisfied - depending on the distance between secretaries and head of group - not one which is either satisfied or violated. In other words, our feeling is that to interpret these statements as requirements would unduly restrict the space of solutions. A better approach is therefore to consider the above statements as expressing preference knowledge and defining two criteria by which solutions to the Sisyphus-I problem can be assessed. In particular, if take this approach, the first statement could be interpreted as stating that, given two admissible design models for the Sisyphus-I problem, say D1 and D_2, D_1 is 'better' than D_2 if the distance between the secretaries' room and the head of the group's in D_1 is less than in D_2. Of course, given an informal design specification, there are in general alternative admissible formal task specifications which can be produced during the analysis stage of the design process. Deciding on the 'best interpretation' in a real scenario is very much the result of an iterative analysis and negotiating process between designers and clients and it is outside the scope of the present discussion. The important aspect here is that the triadic partition of the problem specification into preferences, constraints, and (proper) requirements provides us with an adequate conceptual framework to analyse and represent the desired features and constraints on the solution design.

From a formal point of view a preference can be represented as a binary relation predicated over design models. Thus, we will use the notation `prefer` (D_1, D_2) to indicate that design model D_1 is to be preferred over design model D_2. For instance, we can represent our preference for allocation models which minimize the distance between secretaries and the head of group in the Sisyphus-I example by means of the following rule, expressed in the OCML language (Motta, 1995).

```
((prefer ?d1 ?d2)
      if
      (secretary ?sec)
      (head-of-group ?head)
      (member (?sec . ?sec-room1) ?d1)
      (member (?head . ?head-room1) ?d1)
      (member (?sec . ?sec-room2) ?d2)
      (member (?head . ?head-room2) ?d2)
      (< (compute-distance ?sec-room1 ?head-room1)
         (compute-distance ?sec-room1 ?head-room1)))
```

2.2.2. Global cost function. As shown by the aforementioned statements concerning the Sisyphus-I problem, a task specification typically includes a number of preferences which express different criteria for distinguishing good solutions from 'less good' solutions. In order to harmonise the possible multiple preference criteria uncovered during the analysis phase, it is useful to introduce the notion of *global cost function*, to provide a global cost criterion for ordering solution designs. Such a criterion should be constructed by combining preference-specific cost criteria. In order to do this, it is necessary to specify preference-specific cost functions first, expressing the cost criteria defined by each preference. Having done this, it is then possible to define the global cost function for a parametric design application as the combination of the preference-specific cost criteria. More precisely, given a task specification <P, Vr, C, R, Pr, cf>, where Pr = $\{pr_1, ..., pr_j\}$, we define the global cost function cf = $cf(pr_1)$ o......o $cf(pr_j)$, where $cf(pr_i)$ is the cost function associated with preference pr_i.

For example, Balkany et al (1994) characterise the preferences which apply to the Sisyphus-I domain as statements with the form "x should be close to y". In accordance with this approach they define a cost function which values a solution in terms of the distance between the relevant <x,y> pairs. In order to extend this criterion with one which also takes into account the preference about maximizing project synergy, we need to define a preference-specific cost function for this second criterion and then integrate it with the one defined by Balkany et al. A possible cost function measuring the degree of project synergy denoted by a model could be defined as one which scores positively (say +1) each shared allocation which involves researchers belonging to different projects and which scores negatively (say -1) each shared allocation which violates this criterion. The two preference-specific cost functions could be combined in different ways, for instance by trying to come up with commensurable criteria or by giving priority to one of the two and assessing each allocation by means of a two-dimensional vector $\langle cf_1, cf_2 \rangle$, where cf_1 is the cost function associated with the 'closeness' preference, and cf_2 is the cost function associated with the 'synergy' preference. The OCML lambda expression shown below defines a possible formalization of a cost function for the 'synergy' criterion.

```
(lambda (?design-model)
          (in-environment
              ((?triplets . (setofall (?room ?x ?y)
                                  (and (room ?room size 2)
                                       (in-room ?x ?room)
                                       (in-room ?y ?room)
                                       (<> ?x ?y)))))
               (vector 0
                        (apply +
                               (map
                                 (lambda (?triple)
                                   (if
                                     (and (has-project (second ?triple) ?n)
                                          (has-project (third ?triple) ?m)
                                          (<> ?m ?n))
                                     0.5
                                     -0.5))
                                 ?triplets)))))
```

As shown by the above definition, a cost function, whether global or preference-specific, can be characterised as a function which takes as input a design model and whose output is a n-dimensional vector. We assume that a lower score denotes a better solution. Thus, we say that a design model $D_{sol\text{-}k}$ is an *optimal solution* to a parametric design problem if it is a solution, and there is no other solution, say $D_{sol\text{-}j}$, such that $cf(D_{sol\text{-}j}) < cf(D_{sol\text{-}k})$.

Other examples of cost functions can be found in an earlier paper of ours, (Zdrahal & Motta, 1995), in which we discuss a number of cost functions for the VT application, which attempt to model the revision-oriented cost model used by the domain experts. As already mentioned, the domain experts in the VT problem assess the quality of a design by means of an informal metric which attempts to characterise the 'distance' between the design produced and an ideal, low cost design. This metric was 'compiled' in the Propose & Revise design method used in the original

VT application (Marcus et al., 1988). Thus, the cost of an elevator configuration was computed in terms of the individual costs associated with each revision step required to fix the constraints violated during the design process. In our 1995 paper we formalised the informal criterion given in the VT problem specification by means of a number of different metrics, which used both archimedian and non-archimedian criteria for combining individual revision costs.

3. PARAMETRIC DESIGN PROBLEM SOLVING

In the previous section we have analysed the structure of parametric design applications - i.e. we have characterised the *generic task* which has to be solved. The goal of this section is to understand parametric design problem solving. In particular, our analysis aims to achieve the following objectives:

- To produce a generic framework for parametric design problem solving, characterising the 'essential' knowledge-intensive activities and knowledge structures used when solving parametric design problems.
- To show how the different elements of a parametric design task specification are re-interpreted and 'operationalised' in a process-oriented way, so that they can be effectively used during the design process.
- To examine a number of existing problem solving methods for parametric design and assess, compare, and contrast them in terms of the generic framework devised here.

In the statement of the first goal above, we use the expression 'essential knowledge-intensive activities' to indicate that i) we are interested in design as a knowledge-based process and that ii) we want to devise a 'parsimonious' conceptual framework for characterizing it. The first point spells out our view of (parametric) design as a complex decision-making process where task and domain specific knowledge is brought in to reduce the complexity of a problem. In accordance with such a position, in this paper we identify the decision-making activities (i.e. subtasks) required for parametric design in general, and we characterise them at the *knowledge-level* (Newell, 1982; Van De Velde, 1994), i.e. in terms of their knowledge requirements and abstract inference structures. Having identified the relevant subtasks, we will then use them to compare and contrast different problem solving methods.

Finally, we say that our framework should be 'parsimonious' to emphasize the fact that we want our framework to be both complete, i.e. useful for examining any specific problem solving method, and minimal, i.e. not introducing unnecessary distinctions.

3.1. Design as search

As shown in figure 1, parametric design problem solving can be characterised as the problem of navigating a design space efficiently. Each node in the design space, i.e. a design state, say S_i, is uniquely defined by the associated design model, D_i. Given a state S_i, with design model D_i, we use the notation CV_i to indicate the set of constraints violated by D_i and the notation cf_i to indicate the cost of D_i.

3.1.1. Key design parameters. Assuming an average of k possible values for a parameter, and n parameters, it follows that the size of the design space associated with a parametric design application is k^n. This of course is a very large number even for relatively small applications. However, it turns out that many parameters do not contribute to the size of the search space, given that there is no degree of freedom associated with their assignment. This is the case when the possible value of a parameter is functionally determined by a constraint or requirement. For example, the value of the parameter door-operator-weight in the VT domain (Yost & Rothenfluh, 1996) is calculated as the sum of the door operator engine weight and the door operator header weight. This means that from a design point of view the value of this parameter is functionally determined by a domain constraint, and therefore a design problem solver does not need (or indeed cannot) make any decision about it. In the rest of the paper we will use the expression *functionally bound* to denote parameters whose value is uniquely determined by a functional constraint or requirement. The parameters which are not functionally bound are called *key design parameters*. Key design parameters and their possible values define the degrees of freedom in the design process and, consequently, the 'real' size of the design space. In other words, the essential decision-making activity during the design process is to decide upon the values to be assigned to key design parameters; the values of the other parameters can then be determined by propagating the relevant functional constraints.

In the rest of the paper we will use the attribute *constructive* to refer to those requirements and constraints which functionally determine parameters. Non-constructive constraints and requirements will be referred to as *restrictive*. The role of restrictive requirements and constraints in the design process is to eliminate certain combinations of parameter values.

It is important to point out that deciding which are the key design parameters is not just a matter of analysing the syntactical format of constraints and requirements. On the contrary, such a decision is part of the analysis process during which task knowledge is interpreted in terms of the operational framework provided by a problem solving method. In other words, domain knowledge is needed to distinguish key parameters from the others. For instance, the expression "door operator weight = door operator engine weight + door operator header weight" can be formulated in three different ways, each of which defines different key parameters. It is domain knowledge[1] which tells us that the dependent parameter in the expression is door operator weight and not one of the other two.

The concept of key design parameter is very useful both because it allows us to focus the design process only on the 'important' design parameters and also because it reduces the size of the search space. For example, only 24 of the 199 parameters given in the VT problem specification are key design parameters, which means a reduction in the complexity of the search space from k^{199} to k^{24}, if we assume again that on average each parameter can have no more than k possible values. Moreover, given a parametric design task specification, it is possible to define an isomorphic problem, which is specified only in terms of key design parameters. Hence, we can take advantage of this property and we can define parametric design as the process of assigning values to key design parameters. A consequence of this approach is that, with no loss of generality, we can limit the discussion to design models comprising only key design parameters.

3.1.2. Heuristic value range narrowing. In section 2.1.3 we indicated that while typically the task specification for a parametric design problem specifies (either directly or indirectly) a finite value range for each parameter, it is possible that some parameters have an infinite number (or at least a very large set) of possible values. In this case, additional domain knowledge can be brought in, either as part of the task specification itself, or as part of domain-specific heuristic problem solving knowledge, to narrow down the number of possible values. An example of this kind of 'heuristic value range narrowing' can be found in the VT domain and is used to restrict the potentially very large space of values for parameter counterweight-to-hoistway-rear down to two choices.

As shown in figure 2, the parameter counterweight-to-hoistway-rear specifies the distance between the counterweight and the rear of the hoistway. From a theoretical point of view the counterweight can be positioned anywhere in the space defined by the distance between the platform and the rear of the hoistway (this space is called "counterweight space"). However, the description of the VT application (Yost & Rothenfluh, 1996) indicates that only two options are ever considered by the VT domain experts, when deciding on the position of the counterweight. The preferred solution locates the counterweight half way between the platform and the U-bracket. If this solution does not work (because the counterweight is too close to the U-bracket), then the alternatives are: i) to reduce the depth of the counterweight; ii) to move the counterweight closer to the platform; and iii) to increase the counterweight space (presumably by decreasing the depth of the platform). In

---

[1] Although in this example, one only needs common-sense knowledge to decide on the 'direction' of the constructive constraint.

practice all this means that a VT domain expert only considers two positions for the counterweight: either half way between the platform and the U-bracket, or, in case the previous approach fails, a position ensuring that its distance from the U-bracket is more than 0.75 inches. Because we have no access to the original VT domain experts, it is not possible for us to ascertain whether this refinement of the value range is really part of the task specification (i.e. whether or not these two positions are the only ones which make sense for the counterweight) or whether it is just a way of reducing the complexity of the design process. Whichever the case,[2] this example provides a good illustration of the way heuristic design knowledge can be used to narrow down the number of design choices available for a parameter.

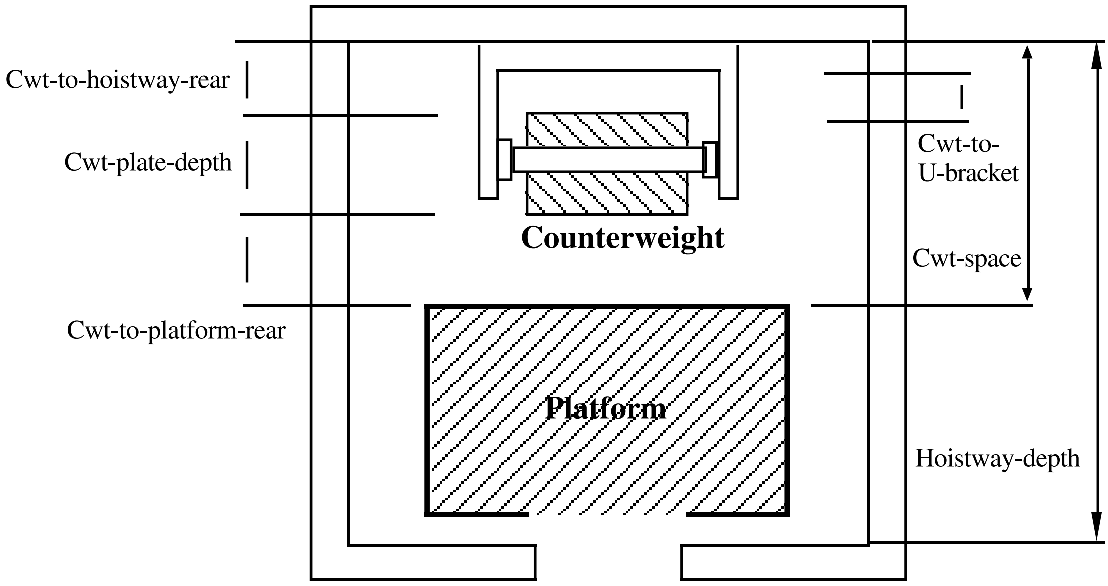

Figure 2. The elevator example

3.2. Design operators

A *design operator*, which is represented in figure 1 as a directed link between two states in the design space, defines a transition between design models. Here we assume that a design operator modifies at least a key design parameter, i.e. we restrict our design space only to 'meaningful' moves in the space of key design models. An example of design operator is provided by the procedure discussed in the previous section, which locates the counterweight half way between the platform and the U-bracket. In general, a design operator modifies a number of key parameters. In the simplest case a design operator reformulates in a functional way the available task knowledge about value ranges. In more complex cases, a design operator expresses additional heuristic domain knowledge, either brought in to narrow down large value ranges, see example in previous section, or to manage complex interactions between parameters. An example of this second kind of heuristic knowledge can be found in the VT domain, where concurrent application of fixes - called *fix combination* - is used to modify multiple parameter values in those cases where the complex interrelationships between parameters and constraints cannot be handled by the application of individual fixes. Our notion of design operator covers both individual fixes and fix combinations.

When two design states, say S_i and S_j, are linked by a design operator pointing to S_j, we say that S_i is the *predecessor* of S_j and that S_j is the *successor* of S_i.

---

[2] Our hunch is that this heuristic value range narrowing is just a way of managing complexity during problem solving.

Although we associate cost with design models, it is also useful to generalise such a notion to operators. We can do this by defining the cost of an operator as the difference between the cost of the successor state and that of the predecessor state. In the rest of the paper we will use the notation cf (S_i -> S_j) to denote the cost of an operator connecting S_i to S_j.

4. PROBLEM SOLVING METHODS FOR PARAMETRIC DESIGN

In this section we briefly illustrate five problem solving methods for parametric design. These methods provide alternative instantiations of the generic Propose & Revise class of problem solving methods (Chandrasekaran, 1990; Zdrahal and Motta, 1995). Introducing these methods serves a dual purpose in the context of this paper. On the one hand they provide us with a set of data points which will be useful to exemplify various aspects of our generic framework. On the other hand, we need them in order to validate our generic model, i.e. to show i) that our model is good enough to allow a re-description of these methods, and also ii) that such a re-description provides us with a better understanding of their behaviour.

4.1. Extend-Model-then-Revise (EMR)

This method is the one used in the original solution to the VT problem (Marcus et al., 1988). Starting from an initial design state, which typically expresses the given design requirements, the EMR algorithm iteratively extends the current design model by calculating values for unassigned parameters. After each assignement step all applicable constraints are evaluated. If some constraints are violated then the current design model is revised by means of the appropriate fixes. This must be done before any further design extension can be carried out. Thus, EMR always tries to maintain a consistent design and all violated constraints are fixed prior to a new assignment step. When one or more constraints need to be fixed, EMR applies the following revision strategy: one of the violated constraints is selected and the relevant design operators, i.e. operators which change values of relevant key design parameters, are applied. The operators are ordered in accordance with the increasing cost of the modification. An operator is rejected either if it does not fix the selected constraint or if it produces a new constraint violation. We can characterise the design strategy used by EMR as 'consistency-first'.

4.2. Complete-Model-then-Revise (CMR)

CMR uses the same revision strategy as EMR. The difference between EMR and CMR is that the latter begins the revision process only after all parameters have been assigned values (Zdrahal & Motta, 1995). Hence, we can characterize the CMR strategy as 'completeness-first'.

4.3. Hill Climbing (CMR-HC)

CMR-HC also follows a 'completeness-first' strategy, which means that constraint violations are rectified only after an initial complete model has been produced. The difference between CMR and CMR-HC is in the revision strategy. CMR applies the operators sequentially, until one is found which fixes the current constraint violation. If no operator is found which can fix the current constraint violation, then the method fails. CMR-HC applies the operators one by one, and then selects the cheapest state which either fixes the constraint violation or comes closer to fix it. This means that CMR-HC accepts also fixes which only contribute to the solution i.e. which only decrease the value by which the constraint is violated even if they do not remove the violation completely. As a consequence, CMR-HC can be applied only to constraints for which the concept “the amount by which the constraint is violated” makes sense. A more detailed analysis of CMR-HC can be found in (Zdrahal & Motta, 1995).

4.4. CMR-A*

This is a version of the CMR method, which uses an A*-style search during the revision phase (Zdrahal & Motta, 1995). Thus, the design model is completed and constraints are evaluated as in a 'normal' CMR approach. Initially the design state associated with this complete design model is the only state which can be revised. Typically this design model will violate some constraints. In general, at each stage of the design process, the explored search space will include a number of design states from which the algorithm selects the most 'promising' one. Here we assume that the cost of a state, say S_j, is *cumulative*, i.e. it can be computed as the sum of the cost of its predecessor, say S_i, and the cost of the path from S_i to S_j, which is assumed to be positive. A state, say S_i, whose successor is S_j, is regarded as the most promising if it minimizes the sum $cf(S_i) + cf(S_i \text{->} S_j) + ecf(S_j \text{->} S_{sol\text{-}k})$. We use the notation $ecf(S_j \text{->} S_{sol\text{-}k})$ to refer to the estimated cost of the path leading from S_j to a solution state, $S_{sol\text{-}k}$. The algorithm stops when the selected design state is a solution state. If the algorithm stops the selected state guarantees an optimal solution.

4.5. Case-based design (CBR)

The starting point here is a design model retrieved from a case library (according to some selection criterion) which contains previous solution states. The retrieved case is evaluated and the values of the relevant parameters are modified so that the resulting design model satisfies the design requirements on the application in hand. This modification normally results in new constraint violations which are fixed by an adaptation mechanism. This can be one of the three revision strategies discussed above, i.e. CMR, CMR-HC, and CMR-A* (Zdrahal & Motta, 1996).

4.6. Pros and cons of sample problem solving methods

The problem solving methods described here have all been tested on the VT problem and exhibit very different behaviours, both in terms of competence and in terms of performance. In particular, experimental tests (Zdrahal & Motta, 1996) showed that when applied to the VT domain, CMR and EMR can solve less than half of the input specifications which can be solved by CMR-A*. Unfortunately, the latter is a much more expensive method. In the aforementioned paper we speculate that the reason for such a difference in competence has to be found in the restricted search strategy used by EMR and CMR, in comparison with CMR-A*. However, without a common framework it is difficult to compare alternative problem solving methods in a meaningful way. We address this problem in the next section where we introduce a framework describing the main generic tasks characterizing parametric design problem solving.

5. GENERIC TASKS IN PARAMETRIC DESIGN PROBLEM SOLVING

The discussion in the previous sections has essentially characterised our 'battleground'. In section 2 we specified the problem area by defining the structure of a task specification in parametric design applications. Having defined the types of applications we are dealing with, we turned our attention to the design process and we characterised it as efficient search in a design space. Moreover, we discussed the relation between a task specification and a design space, in particular highlighting the heuristic nature of the process by which a finite (or limited) design space is produced from a task specification. Finally, we introduced a number of problem solving methods, which we (and others) have used to solve parametric design applications, and we underlined the need for a generic model of parametric design problem solving, which could be used to describe these methods in a uniform way, and to compare and contrast them more effectively. Such a generic model should identify the important knowledge-based tasks required for parametric design problem solving and their knowledge requirements.

The approach we take here to identify the generic subtasks of parametric design problem solving is basically top-down. Given the design space representation we have adopted, there are essentially only four actions which can be carried out: selecting a design state, selecting a design operator, applying a design operator to the selected state, and evaluating the resulting design model. The latter is needed to assess its properties, e.g. whether it provides a solution, its cost, etc. Although these four subtasks are adequate to describe the process of searching the design space, such a framework is not rich enough to model design methods which exhibit some flexibility in their behaviour. For example, an intelligent parametric design problem solver could employ a number of different state selection strategies and use them in different moments of the design process (Zdrahal & Motta, 1996). Moreover, the selection of a design operator goes normally through a number of 'important' decision-making activities, which include the selection of a *design context* (e.g. extend-design vs revise-design) and the selection of a *design focus*. This can be seen as an abstraction mechanism which extracts from the current design state the relevant information which is used to restrict the field of possible operators. An example of a design focus is the constraint violation which a Propose & Revise problem solver attempts to resolve during a particular fix appplication.

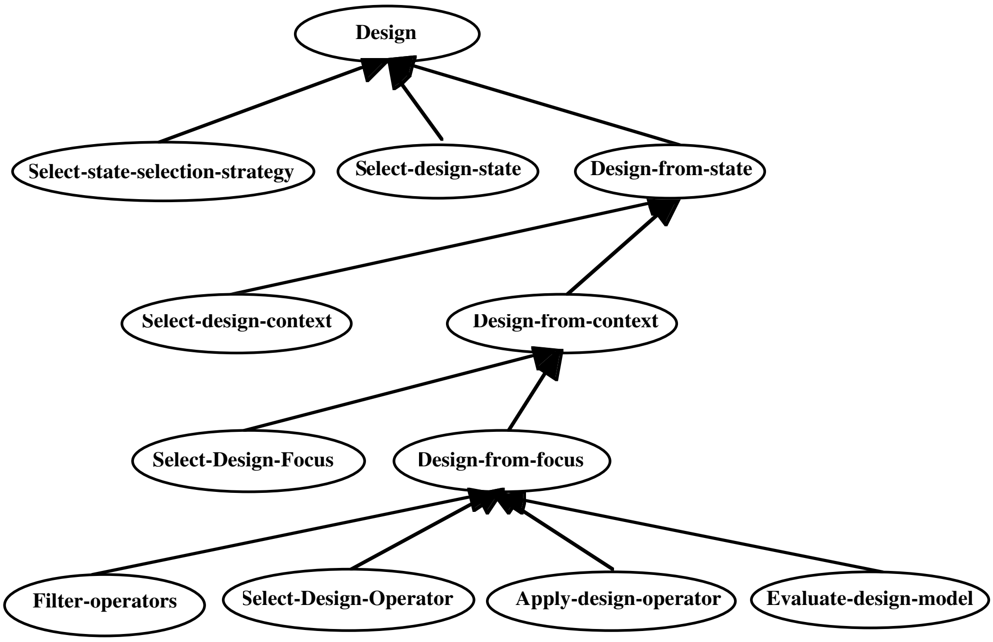

Figure 3. Main task-subtask hierarchy in parametric design problem solving.

The resulting task-subtask decomposition is shown in figure 3. It is important to emphasize that the goal of this structure is mainly analytical: we use it to identify the main tasks involved in parametric design. It does not specify a particular control architecture, neither it proposes modelling solutions. For instance, the task Select-state-selection-strategy could be modelled in different ways, as a meta-level control task, or 'hardwired' in alternative formulations of task Select-design-state, depending on the adopted modelling approach and/or language.

In figure 4 we show a KADS-style inference structure (Wielinga et al., 1992), showing the input/output behaviour of the subtasks in the first four layers of the task-subtask decomposition given in figure 3. Italicized labels indicate repeated occurrences of a *knowledge role*.

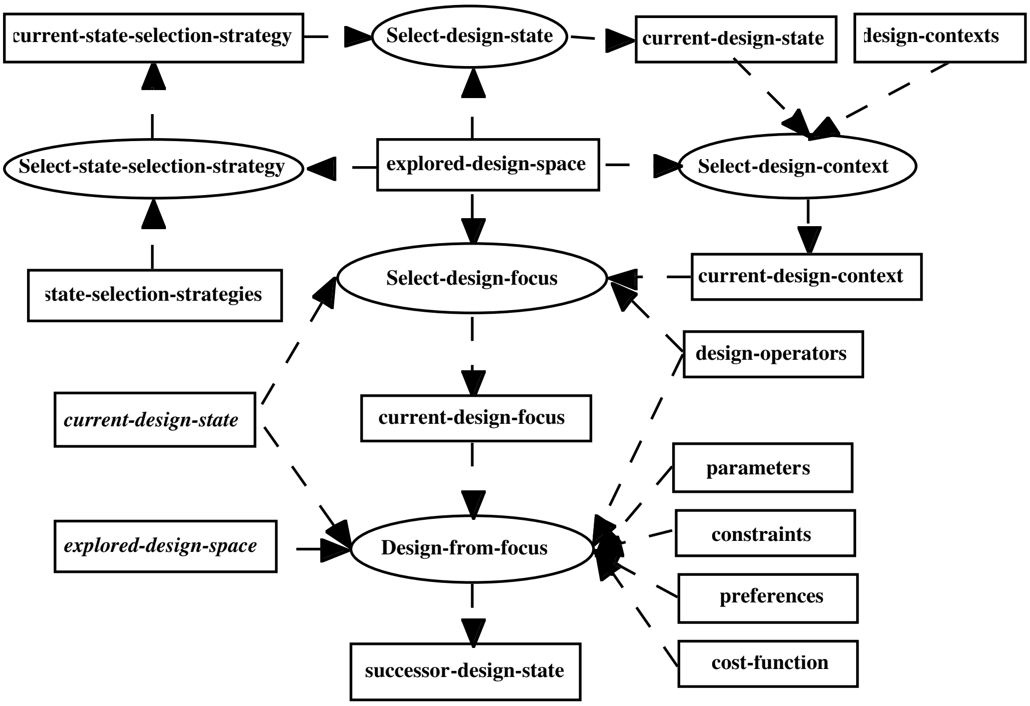

Figure 4. Inference structure in parametric design problem solving.

Finally in figure 5 we show the inference structure of task Design-from-focus.

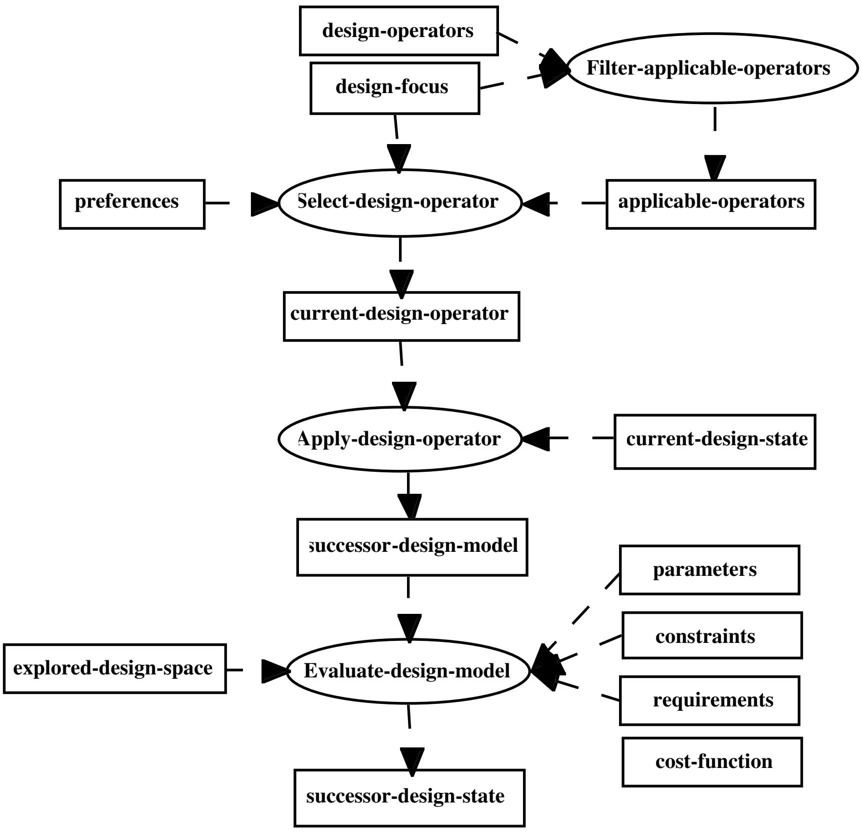

Figure 5. Inference structure of task Design-from-focus

Of course, not all subtasks shown in figures 3-5 exhibit the same level of complexity. For instance, no decision-making is involved in carrying out task Apply-design-operator, which is basically a function application call.

From a decision-making point of view, there are seven interesting tasks in our model: the six selection/filtering tasks and task Evaluate-design-model. These tasks provide a set of dimensions along which problem solving methods can be differentiated and compared. These subtasks and the associated roles are discussed in the next three sections.

5.1. Design model evaluation

The role of task Evaluate-design-model is to assess a design model by producing the relevant problem solving information. In general, see figure 6, there are four main criteria which can be used to evaluate a design model in a design problem solving context (i.e. a state in a design space): feasibility (i.e. whether a state can lead to a solution), cost, completeness (i.e. whether a state provides a complete solution to the design problem), and consistency (i.e. whether a state violates some requirements or constraints). Of course, it is not always trivial or even possible to extract all this information from a design model. For instance, typically only weak criteria for assessing the feasibility of a design model can be provided and in design problems in general, it can be very tricky to assess the completeness of a design model. However, if we restrict to parametric design, and we assume that a complete task specification exists as result of a task analysis, then we can also assume that the issues of completeness, consistency, and cost are not problematic during the problem solving process itself.

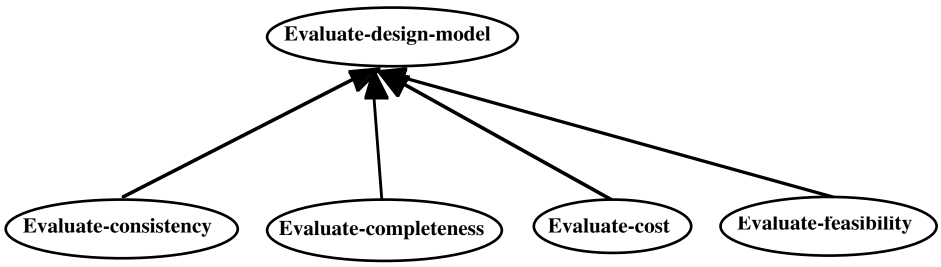

Figure 6. Task-subtask decomposition for task Evaluate-design-state

5.2. Design state selection

At any stage of the design process, a problem solver (be it human or artificial) knows about a number of design states which are relevant to the problem in hand. Typically, these are the states which have been explored during the current design process, i.e. the states included in the portion of the design space searched so far. However, human designers are of course capable of reusing past designs and the same applies to problem solvers which make use of case-based reasoning techniques when solving design applications (Zdrahal & Motta, 1996). Therefore, from a general point of view we can say that role explored-design-space refers to all the design states available to the problem solver, either because they have been explored during the current design process, or because the problem solver has access to other relevant design knowledge (e.g. a case-based library of design states).

Assuming that a rational problem solver would not normally select a design state known to be unfeasible, it follows that state selection is carried out in terms of the other three main criteria discussed in the previous section: contents of the design model, constraint violations, and cost. For example, both EMR and CMR use a strategy of selecting the state which maximizes the number of assigned parameters, minimizes the number of constraint violations, and minimizes the cost. These criteria are applied in order, i.e. the second criterion is used when the first one has not enough discrimination power, and the third one is used only when the application of the first and second criteria fail to produce a single candidate. CMR-A* uses instead a different strategy and selects the cheapest state among those that maximize the number of assigned parameters. As we'll see more in detail in section 6, this difference in the way states are selected accounts for the different competence patterns expressed by EMR and CMR-A*. In other words a sound state selection strategy is crucial for the competence of a problem solver. In practice, given the complexity of design applications, selecting the right state is also important for performance reasons, as efficient pruning of the search space is needed for all but the most trivial applications. Of course, the crucial aspect here is to be able to remove only the 'bad' states. Otherwise we could eliminate states leading to a solution, thus affecting the competence of the design method.

5.3. Design operator selection

By selecting a design state the problem solver decides which design model becomes the focus of the design process. Having focussed on a model, a designer has to decide how to modify it, which requires a decision on which operator to apply to the current model. In general, there are various levels of decision-making required in order to reach a decision on the selection of a design operator. At the highest level of abstraction operator selection is driven by the current design context. This can be seen as a high-level abstraction which is useful to label typically design scenarios. For instance a Propose & Revise problem solver, say EMR, distinguishes between two kinds of operators: procedures and fixes. If the selected design model is inconsistent, then EMR will choose one of the relevant fixes, otherwise it will choose one of the applicable procedures. To account for this behaviour we say that operator selection in EMR is carried out in two different contexts: one in which design extensions are carried out (extend-design context), and one in which inconsistent models are revised (revise-design context). Having selected the current design strategy, a problem solver then abstracts from the current design state the current *design focus*. This refers to the element of the design state which is going to drive the selection of the design operator. For example, in the context of the revision phase in EMR, the design focus is given by the particular constraint violation which EMR tries to resolve. If the context is one of model extension, then the design focus would be given by the parameter which is going to be assigned a value. In a nutshell, the tasks Select-design-context and Select-design-focus allows us to model the two levels of decision-making involved in the formulation of a set of candidate design operators. The more abstract level - Select-design-context - selects a generic design context, while the more specific level focusses on a specific design element within the generic context.

As we discuss extensively in other papers (Motta et al, 1996; Zdrahal & Motta, 1996), selecting the 'right' design focus, i.e. the right parameter or constraint violation, is important to solve the VT problem using a Propose & Revise approach. From a competence point of view this is a side-effect of the restricted search model used by EMR and CMR problem solvers - see section 6.1. However, even with more flexible search mechanisms, selecting the right focus is very important to minimize unnecessary backtracking. This problem has been extensively analysed in the constraint satisfaction literature and various heuristics have been devised to improve the efficiency of variable (in our case, focus) selection (Dechter and Meiri, 1994).

Because parametric design is about finding optimal or sub-optimal solutions, the relevant preference knowledge is used to drive the selection of a design operator. This is often called *local* preference knowledge (Poeck and Puppe, 1992) and it is defined as preference knowledge which can be used to make locally optimal decisons, i.e. to perform *best-first* search. For instance, a problem solver tackling the Sisyphus-I problem would normally use the preference criterion discussed in section 2.2.1 when trying to allocate secretaries.

6. MAPPING THE GENERIC MODEL TO INDIVIDUAL METHODS

In this section we apply the generic model presented in the previous section to the problem solving methods described in section 4. The purpose of this exercise is two-fold: i) to validate our model by showing that it adequately accounts for the problem solving behaviour of a number of methods, and ii) to provide useful insights in the strengths and limitations of these methods. Here, we follow the same approach we used in section 5, and we describe the problem solving methods in terms of the selection and filter tasks, which are the interesting ones from a decision-making point of view.

6.1. Analysis of state selection strategies in problem solving methods

All the methods included in our sample employ fixed strategies for design state selection and therefore there is not much to say about how these methods operationalise task Select-state-selection-strategy: they do not.

The input to task Select-design-state in figure 4 comprises two knowledge roles: current-state-selection-strategy and explored-design-space, which we can use to structure our analysis.

6.1.1. Comparison by role explored-design-space. The explored search space considered by all the methods except CBR consists of all design states generated within the current design process. It means that these problem solvers can only exploit design states produced when solving the current problem. In contrast with the other methods, the design space considered by CBR includes both the design states generated within the current design process as well as solutions to similar design problems generated in the past.

6.1.2. Comparison by state selection strategy. We will specify the state selection strategies used by different methods in terms of selection rules.

EMR, CMR. Let S1 be the set of states denoted by role explored-design-space with the maximum number of assigned parameters in the associated design model. Let S2 be the set of states in S1 with the minimum number of constraint violations. Select the state from S2 with the minimum cost.

CMR-A*. Let S1 be the set of states denoted by role explored-design-space with the maximum number of assigned parameters in the associated design model. Select the state from S1 with the minimum cost.

CMR-HC. Let S1 be the set of states denoted by role explored-design-space with the maximum number of assigned parameters in the associated design model. Denote as S2 the set of states from S1 with the minimum number of constraint violations. Select the state from S2 with the maximum cost.

CBR. There is no single strategy for design state selection which is applicable to all CBR design methods. Each method is characterised by its own technique. Some possibilities are presented in (Zdrahal & Motta, 1996).

The criteria used by the different methods for selecting design states are summarised in table 1.

	Design Model	Violated Constraints	Cost
CMR, EMR	max	min	min
CMR-HC	max	min	max
CMR-A*	max	not applied	min
CBR	method dependent	method dependent	method dependent

Table 1. Criteria for state selection in problem solving methods

We have already mentioned that our experimental results with the VT domain show that CMR-A* is able to find solution designs in cases where EMR, CMR and CMR-HC fail (Zdrahal & Motta, 1996). Our analysis of the state selection strategies employed by these methods enables us to explain why they exhibit different competence patterns. All four methods focus on maximal design states - i.e. states whose design model comprise the highest number of assigned parameters. The difference is that while EMR, CMR, and CMR-HC use the number of constraint violations as the next discrimination criterion, CMR-A* uses instead cost. More importantly, the cost criterion we use when solving VT by means of CMR-A* is based on an estimate of the cost of the revision process, and is a cumulative, monotonic function. These properties guarantee that the method will eventually converge to an optimal solution, if this exists. The assumption underlying the state selection strategy used by EMR and CMR is instead that solutions can eventually be found by selecting maximal states with the minimum number of constraint violations and minimum cost. The problem here is that the number of constraint violations of a successor state may increase, remain the same, or decrease compared to its predecessor - i.e. counting constraint violations is not a monotonic, cumulative function. Therefore there is no evidence that the VT domain satisfies the assumption - which is held in EMR, CMR, and CMR-HC - that the state with a higher number of constraint violations cannot result in a solution. Nevertheless, this is the basic principle of revising inconsistent solutions both in EMR and CMR, and, in our view, it is the reason for the limited competence exhibited by these methods. In contrast with these methods, the cost criterion used by CMR-A* is consistent with our intuition that extensions and corrections to the design model cannot decrease its cost. By extending the design model we include additional components with non-negative cost and by correcting the design model we replace a cheaper solution with a more expensive one. Consequently, selecting states with minimal cost does not affect the competence of the method. However, if we break our implicit assumption and the cost of the successor state is allowed to decrease, i.e. if it is possible that the revised solution is cheaper than the original one, then the CMR-A* method would display the same unpredictable pattern of behaviour as EMR and CMR[3].

6.2. Analysis of successor state generation in problem solving methods

A design problem solver generates a new design model by i) selecting a design context, ii) selecting a design focus, iii) filtering the applicable operators, iv) selecting a design operator and v) applying this to the selected design state. The latter is not a very interesting task and therefore we will focus our analysis on the other four tasks and show how they are operationalised in different problem solving methods. In the descriptions below we assume that the selected design state is S_i, characterised by design model D_i, with associated constraint violations CV_i, and cost cf_i. All the problem solving methods analysed here follow the Propose & Revise approach, and therefore they all employ two types of design operators: procedures and fixes. In this approach, only one procedure exists for each key design parameter.

6.2.1. Successor state generation in problem solving methods: task analysis

EMR

Select-design-context. If CV_i is empty and D_i is incomplete, the context is extend-design. If CV_i is not empty, then the context is revise-design.

Select-design-focus. If the context is extend-design, then the design focus is one of the unassigned key design parameters, say p_i. If the context is revise-design, then the focus is one of the violated constraint violations, say cv_{ij}.

Filter-applicable-operators. If the current context is extend-design, then the procedure which provides a value for parameter p_i is selected. If the current context is revise-design, then the set of applicable design operators is given by the set of all fixes which 'affect' cv_{ij}.

Select-design-operator. If the current context is extend-design, this task is trivial. If the current context is revise-design, then the design operator - i.e. fix or fix combination - with lowest cost is selected (i.e. a best-first strategy is used).

---

3 An anonymous reviewer has suggested that, given the right assumptions about the application domain, one could envisage sound, converging criteria based on monotonic decreasing of constraint violations. Of course, this is true in principle. However, in practice it is much more difficult to guarantee that design operator applications will monotonically reduce the number of constraint violations. This is the same as guaranteeing that no antagonistic constraints can be found in the application in hand. Such assumption only applies to relatively simple applications chracterised by low *connectivity* in the constraint network.

CMR

Select-design-context. If D_i is incomplete, then the context is extend-design. Otherwise, the context is revise-design.

Select-design-focus. The same as in the case of EMR.

Filter-applicable-operators. The same as in the case of EMR.

Select-design-operator. The same as in the case of EMR.

CMR-HC

Select-design-context. The same as in the case of CMR.

Select-design-focus. The same as in the case of EMR and CMR.

Filter-applicable-operators. The same as in the case of EMR and CMR.

Select-design-operator. If the current context is extend-design, this task is trivial. Otherwise all operators (i.e. fixes and fix combinations) with the lowest cost are tentatively applied. If one of the operators solves cv_{ij}, then this operator is selected. Otherwise the operator which results in the maximum decrease of the amount by which cv_{ij} is violated is selected. If no such operator exists, i.e. all operators with the lowest cost increase the amount by which cv_{ij} is violated, then the same procedure is applied to the operators with the next highest cost, and so on. If this procedure fails even for the operators with highest cost, then the task fails.

CMR-A*

Select-design-context. The same as in the case of CMR.

Select-design-focus. The same as in the case of EMR and CMR.

Filter-applicable-operators. The same as in the case of EMR and CMR.

Select-design-operator. The same as in the case of EMR and CMR.

CBR

The task specifications for these tasks differ depending on the particular revision strategy used, i.e. CMR, CMR-HC or CMR-A*, and are the same as those used by the relevant method.

6.2.2. Successor state generation in problem solving methods: summing up. The main result of the comparison carried out in this section is that there is no much difference in the way successor design models are generated by these methods. More specifically, there is no difference between the approach used by CMR-A* and the others. We can therefore conclude that the superior competence of CMR-A* is only a result of the state selection strategy it employs.

7. DISCUSSION OF RELATED WORK

Parametric design is formally analysed in (Wielinga et al., 1995). In this paper the authors define a parametric design task specification and refine the associated *competence theory* (Van de Velde, 1988) to derive properties of a CMR problem solving method. Such an approach succeeds in highlighting a number of assumptions underlying CMR, but it only produces limited results with respect to assessing its competence. Our impression is that the framework adopted by Wielinga et al. is not powerful enough to account for important aspects of parametric design problem solving in general and CMR in particular. The reason for this inadequacy is that, as discussed in this paper, the competence of CMR is directly related to its state selection strategy. Hence, it is not possible to characterize it without a framework which explicitly operates with these concepts. In particular, in the conclusion to their paper, Wielinga et al. point out that "the non-monotonic nature of the Propose & Revise method is difficult to capture in intuitively understandable theories". In our view this is not necessarily the case. While we agree that the revision mechanism in Propose &

Revise is difficult to capture in first-order theories, it does not necessarily follow that its behaviour cannot be 'intuitively understood'. But in order to understand it, one needs to adopt the ‘right framework’ for parametric design, which takes into account both the decision making, as well as the search-related aspects of design problem solving. In this paper we have tried to provide such a framework.

Another important difference between the analysis of parametric design carried out by Wielinga et al. and ours concerns the approach taken to move from a task specification to a generic problem solving model. In their paper, Wielinga et al. use the Generate & Test problem solving paradigm to refine the task specification into an initial problem solving model. In this paper, we have instead adopted a view of design as search in a problem space. While search and Generate & Test have typically been considered as equivalent paradigms, see for instance (Newell, 1990), it is interesting to note that a procedural approach leads to a quite different analysis of parametric design problem solving.

The DIDS system (Balkany et al., 1994; Runkel and Birmingham, 1994) provides domain-independent support for building design applications. The system is based on a generic model of configuration design problem solving, which defines the generic data structures and tasks, called *mechanisms* in the DIDS terminology, required for building design applications. Our work has a number of similarities with the DIDS approach. We also characterise design as search through a design space, and one of our goals was also the development of a generic problem solving model for design. A difference between our analysis and theirs concerns the application area, which is narrower in our case, as we concentrate on parametric, rather than configuration design problems. This wider focus allows the DIDS researchers to define a richer set of data structures, than it is possible within the parametric design framework used in this paper. On the other hand, the DIDS framework is very method-oriented in the sense that it only focuses on the generic method ontology required for building configuration design applications. On the contrary, in this paper we have also analysed, in a method-independent way, the structure of parametric design applications, and discussed the relationship between this structure and the generic problem solving model.

In a separate paper (Balkany et al., 1993) the DIDS researchers analyse a number of configuration design systems, and try to classify the various mechanisms used by these systems into a number of generic categories: select design extension, make design extension, detect constraint violation, select fix mechanisms, make fix mechanisms, and test if-done. For each of these generic tasks a number of mechanisms drawn from the various systems are identified - e.g. 41 "make design extension" mechanisms are listed in the paper. The granularity of the mechanisms uncovered by this analysis varies significantly. Some mechanisms are low-level actions such as "do-step", which performs "one step of a task", others are fairly high-level mechanisms such as "create-goal-for-constraint". The problem with such a bottom-up approach is that each system uses a different terminology, solves a different class of tasks - for instance arrangement vs parametric design - and employs control mechanisms at very different levels of abstraction. Therefore it is difficult to compare them. The approach we have taken here is diffferent. We have specified exactly the class of tasks we are dealing with - parametric design - and then we have formulated the generic structure of the problem solving required for this class of tasks. This means that specific parametric design problem solvers can then be described by analysing how they carry out the generic tasks introduced by our framework.

The paper by Gero (1990) provides a comprehensive analysis of the design process in general, not just parametric or configuration design, and identifies the general classes of knowledge which are needed for design. This analysis leads to the formulation of a representation framework which can be used to support design activities. In comparison to this paper our scope here is much narrower. However, by restricting our analysis to parametric design we can of course analyse the relevant concepts in much more detail than it is possible in papers with a wider scope. Moreover, we believe that in-depth studies of parametric design problems are also very important for the broader design area, for two main reasons. First, all design activities include parametric design sub-activities. Second, frameworks for more complex types of design can be devised as extensions of parametric design task and method ontologies.

The analysis by Chandrasekaran (1990) has also a broader scope than ours. In this paper he decomposes the design task in four subtasks: propose, verify, critique, and modify, and then discusses the different methods which can be used to tackle these subtasks. For instance verification can be carried out by means of simulation methods. For each method, Chandrasekaran briefly describes its knowledge requirements and computational properties. We share Chandrasekaran's philosophy and this paper is an attempt to instantiate some of his ideas - e.g. the use of task analysis to understand problem solving - in a precisely defined subset of the space of design applications.

8. CONCLUSIONS

In a number of related papers we have investigated i) the optimisation aspects of design applications (Motta & Zdrahal, 1995; Zdrahal & Motta, 1995), ii) several architectures for Propose & Revise (Zdrahal & Motta, 1995), iii) the ontological issues associated with design problems (Motta & Zdrahal, 1995), and iv) the different competence patterns expressed by different methods (Zdrahal & Motta, 1996). In this paper we have tried to pull together the various threads of our recent research and answer questions concerning the structure of a parametric design task specification and the knowledge-level organisation of a generic parametric design problem solver. Our task specification emphasises the optimisation aspects of parametric design applications, while the proposed model of parametric design problem solving capitalises on a view of design as search, which enables us to generalise from Propose & Revise architectures. Moreover, we have also tried to clarify the relationship between task knowledge and parametric design problem solving, in particular emphasising the use of heuristic domain knowledge to circumscribe the set of applicable design operators. By applying the proposed generic model to a number of methods we achieved a dual goal of validating our model and explaining the reasons for the superior competence shown by CMR-A*.

If we take a broader perspective, it seems to us that the main result of this paper is that our generic model of parametric design problem solving clearly shows the need for flexible problem solving models in design. In particular, the analysis of the various instances of Propose & Revise architectures shows important trade-offs between competence and performance. There are no good methods and bad methods. EMR and CMR do perfectly well for most parts of the VT domain. However, the assumptions on which these methods are based do not hold for the machinery component of the VT model - see also (Zdrahal and Motta, 1996). From a problem solving point of view this means that research into problem solving methods should produce control architectures which can opportunistically switch to a different problem solving regime when the current one fails. From a knowledge acquisition point of view, the implication is that we need better techniques and tools to verify that the assumptions underlying a problem solving method actually hold for a particular domain. Recent papers by Fensel (1995) and Benjamins and Pierret-Golbrich (1996) on specifying and verifying assumptions of problem solving methods are initial steps in this direction.

ACKNOWLEDGEMENTS

Many thanks to the (not-so-)anonymous referees of KAW '96 for their extensive and stimulating comments.

REFERENCES

Balkany A., Birmingham W.P. and Tommelein J. (1993). An Analysis of Several Configuration Design Systems. *Artificial Intelligence in Engineering, Design, and Manufacturing*, 7(1), pp. 1-17.

Balkany A., Birmingham W.P. and Runkel J. (1994). Solving Sisyphus by Design. *Int.J.Human-Computer Studies (1994)* ***40.*** *pp. 221-241*

Benjamins, R. and Pierret-Golbreich, C. (1996). Assumptions of Problem Solving Methods. *Proceedings of the 6th Workshop on Knowledge Engineering Methods and Languages*. Gif-sur-Yvette, France, January 15-16.

Chandrasekaran, B. (1990). Design Problem Solving: A Task Analysis. *AI Magazine*, 11(4), pp. 59-71.

Dechter, R., and Meiri, I. (1994). Experimental Evaluation of Preprocessing Algorithms in Constraint Satisfaction Problems. *Artificial Intelligence Journal*, Vol. 68, pp. 211-241.

Fensel, D. (1995). Assumptions and Limitations of a Problem Solving Method: a Case Study. In Gaines and Musen (Eds.), *Proceedings of the 9th Banff Knowledge Acquisition Workshop*.

Gero, J. S. (1990). Design Prototypes: A Knowledge Representation Schema for Design. *AI Magazine*, 11(4), pp. 26-36.

Linster M (1994). Problem statement for Sisyphus: models of problem solving. *Int.J.Human-Computer Studies (1994)* ***40.*** *pp. 187-192*

Marcus, S, Stout, J., and McDermott, J. (1988). VT: An Expert Elevator Designer that uses Knowledge-Based Backtracking. *AI Magazine*, 9(1), pp. 95-112, Spring 1988.

Motta, E. (1995). KBS Modelling in OCML. Proceedings on the Workshop on Modelling Languages for Knowledge-Based Systems, Vrije Universiteit Amsterdam, January 30-31, 1995.

Motta E., Zdrahal Z. (1995). The Trouble with What: Issues in method-independent task specifications. In *Proceedings of the 9th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop* (B.R.Gaines and M.Musen eds.). pp. 30-1 - 30- 17.

Motta E., Stutt A., Zdrahal Z., O'Hara K.and Shadbolt N.(1996). Solving VT in VITAL: a study in model construction and knowledge reuse. *International Journal of Human-Computer Studies*, 44(3/4), pp. 333-371.

Newell A. (1980). Reasoning, Problem Solving, and Decision Processes: The Problem Space as a Fundamental Category. In R.S. Nickerson (Ed.), *Attention and Performance VIII*, Lawrence Erlbaum Associates, Hillsdale, New Jersey.

Newell A. (1982). The knowledge level. *Artificial Intelligence.* Vol. 18, 1. pp. 87-127

Newell A. (1990). *Unified Theories of Cognition*. Harvard University Press.

Poeck, K. and Puppe, F. (1992). COKE: Efficient Solving of Complex Assignment Problems with the Propose-and-Exchange Method. *5th International Conference on Tools with Artificial Intelligence*. Arlington, Virginia.

Runkel, J. T. and Birmingham, W. B. (1994). Solving VT by Reuse. Proceedings of the 8th Banff Knowledge Acquisition Workshop, Banff, Canada, 1994.

Stefik M.(1995). Introduction to Knowledge Systems. *Morgan Kaufmann.* 1995.

Van de Velde, W. (1994). Issues in Knowledge Level Modelling. *Proceedings of the 8th Banff Knowledge Acquisition Workshop*, Vol 2, 1994a. 38-1 - 38-11.

Van de Velde, W. (1988). Inference structure as a basis for problem solving. *Proceedings of the 8th European Conference on Artificial Intelligence*, pp 202-207, London, Pitman.

Wielinga B. J., Schreiber A.Th. and Breuker J., (1992) KADS: A Modelling Approach to Knowledge Engineering. *Knowledge Acquisition* 4(1) pp.5-53.

Wielinga B.J., Akkermans J.M. and Schreiber A.Th. (1995). A Formal Analysis of Parametric Design Problem Solving. In Gaines and Musen (Eds.), *Proceedings of the 9th Banff Knowledge Acquisition Workshop*, pp. 37-1 - 37- 15.

Yost G.R., Rothenfluh T.R. (1996). Configuring elevator systems. *International Journal of Human-Computer Studies*, 44(3/4), pp 521-568.

Zdrahal Z., Motta E. (1995). An In-Depth Analysis of Propose & Revise Problem Solving Methods. In Gaines and Musen (Eds.), *Proceedings of the 9th Banff Knowledge Acquisition Workshop*, pp. 38-1 - 38- 20.

Zdrahal Z., Motta E. (1996). Improving Competence by Integrating Case-Based Reasoning and Heuristic Search. Proceedings of the 10th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop.