# Design by Exploration: A Proposed CommonKADS Inference Structure

John K.C. Kingston

AIAI-PR-62

December 1994

**Keywords:** KADS, inference structures, iterative design, propose-critique-modify design, constraints, knowledge acquisition, repertory grid, card sort

Artificial Intelligence Applications Institute
University of Edinburgh
80 South Bridge
Edinburgh EH1 1HN
United Kingdom

**Abstract**

The KADS methodology for the development of knowledge based systems and its successor CommonKADS have had a major influence on the development of knowledge-based systems (KBS) in the last few years. KADS provides considerable assistance to KBS developers at the start of a project through its library of *generic inference structures*, which show the different processes which are typically carried out for a particular task type.

The library currently includes three structures which represent three different approaches to design at a generic level: *hierarchical* design, *transformational* design and *incremental* design. These approaches to design are all used in the commercial world; however, they rely on a sequential, non-iterative view of the overall design process. Many designers use a different approach, in which a solution is proposed, presented to the client for criticism, and then altered based on the client's comments (which effectively form further constraints on the design). This process is iterated until an acceptable design is reached. This process has been called *propose-critique-modify* design; for compatibility with KADS terminology, it is known in this paper as *exploration-based design.*

The process of exploration-based design is investigated in detail, with particular attention to understanding constraints, and to the re-use of previous designs. A possible generic inference structure for exploration-based design is then suggested, and the use of this inference structure on a knowledge-based design project is described. Finally, suitable knowledge acquisition techniques for exploration-based design are proposed.

# 1 Introduction

The KADS methodology for the development of knowledge based systems [Hickman *et al*, 1989] [Schreiber *et al*, 1993], and its successor CommonKADS [deHoog *et al*, 1993]), have had a major influence on the development of knowledge-based systems (KBS) in the last few years. KADS has provided a structured and documented framework for KBS development which is more acceptable to the commercial world than the traditional approach of iteratively developing a prototype KBS. It also provides considerable assistance to KBS developers at the start of a project through its library of *generic inference structures*[1]. These generic models are intended to provide guidance in knowledge acquisition and in knowledge analysis, by showing the different processes which are typically carried out for a particular *task type*. For example, if a KBS was being built to assess candidates for

[1]While this paper uses the term "KADS" to describe the approach taken by both the KADS methodology and the CommonKADS methodology, the terminology used in this paper is the terminology of CommonKADS. The generic inference structures were usually known by the term "interpretation models" in the original KADS methodology.

job suitability, the generic inference structure for *assessment* tasks could be selected from the library of inference structures. The generic model could then be adapted to the problem in hand by making small changes (e.g. the "case description" in the generic model could be instantiated to "curriculum vitae" in the adapted model) so that the final model reflects the actual inference processes which are carried out when a candidate is assessed by an experienced interviewer. Inference structures may also be used as a framework for the design and implementation of the resulting KBS (e.g. [Kingston, 1992a]).

The KADS methodology classifies task types into a taxonomy [Breuker., 1987]. The principal distinction in this taxonomy is between *system analysis* tasks and *system synthesis* tasks. Analytic tasks, such as diagnosis and assessment, have as their ultimate goal the establishment of unknown properties or behaviour of the system; synthetic tasks, such as configuration and planning, aim to define a structural description of a system in terms of some given set of elements. Certain tasks, such as repair or control, are considered to involve aspects of both analytic and synthetic tasks; these are known as *system modification* tasks. The majority of successful KBS systems have dealt with analytic tasks, such as diagnosis or selection, although several successful KBS have been developed for synthetic tasks, either using KADS (e.g. [Kingston, 1992b]) or without KADS (e.g. [McDermott, 1982] [Tate *et al*, 1994]). There are very few successful KBS systems which successfully handle modification tasks.

Design problems are classified by KADS as synthetic tasks, and are classified into three subtypes: design by hierarchical decomposition, design by gradual refinement, and design by transformation. Each of these subtypes has its own inference structure. The thesis of this paper is that the repertoire of inference structures for design tasks is incomplete. At least one more model needs to be added: a model which supports the process of *exploration-based* design (also known as *propose-critique-modify* design). This paper contains a justification for the addition of this model, a suggested framework for the model, and a discussion of knowledge acquisition techniques suitable for exploration-based design.

## 2 The design process

### 2.1 KADS modelling of the design process

There is considerable debate about the way in which design is, or should be, carried out. The underlying reason for this debate is that designers not only work in different ways, but actually think in different ways. Many textbooks on design encourage designers to think divergently, deliberately **not** restricting themselves to a fixed "design process", in order to stimulate the emergence of "creativity" which is seen as the key to many successful designs. Others argue that a design process should be used because, in some situations, creativity is less important than productivity, reusability, or ensuring that a design meets safety standards.

While the arguments continue, attempts have been made to categorise the ways in which design is actually performed (e.g. [Maher, 1990]). KADS offers the following categorisation [Breuker., 1987]:

1. **Hierarchical design**. In this process, a design task is broken down into a number of smaller design tasks, which are tackled separately, and then the results are recombined. Ideally, each subtask would be further decomposed until it reaches the stage where there is a well-understood solution: for example, in software design, a low-level subtask might be to design an ordered set of elements. This task could be solved by writing a sorting algorithm and applying it to the elements.

   Hierarchical design is used in cases where independent subproblems can be defined, such as software design, where modules only interact via their inputs and outputs. However, such independence is often impossible to achieve in design tasks; for example, the construction of a house cannot be broken down into an independent consideration of the design of each room in the house, because the chosen shape and the location of utilities in each room affects the design of the other rooms.

   An example of the use of hierarchical design can be found in [Kruger & Wielinga, 1993], which records an empirical study which aimed to identify the approaches taken by industrial designers to designing a garbage disposal system for a train.

2. **Transformational design**. This is a version of design in which a full specification of the artifact is available at an early stage of the design process, but is formulated in a different manner from the elements of the solution domain. A good example is VLSI design in which an algorithm is input to the design process (a formal specification) and the layout of the actual chip is the required output [Breuker., 1987].

   It is likely that the main knowledge-based components of transformational design will be problem-specific.

3. **Incremental design**. This occurs when there is no straightforward transformation of the conceptual design to a detailed design model; instead, the conceptual model is separated into design elements and constraints. Both of these are then transformed (perhaps in several stages) to a form where they can be amalgamated into a final design model.

In order to understand the above categorisation, it is important to note two points, which are sometimes not appreciated by people who are unfamiliar with KADS:

1. The categories above represent *generic* frameworks for performing a design task. These frameworks need to be instantiated to particular design tasks, which may involve the addition or removal of some inference steps in order to reflect the actual inferences which are performed for a particular task. The knowledge engineer is therefore asked to determine the most appropriate generic inference structure, rather than the only appropriate inference structure.

2. The modelling of expert tasks may require more than one level of decomposition or refinement: taking hierarchical design as an example, each sub-part of the overall design may need to be modelled individually in order to produce a fully detailed model of the design. It is important to note that tasks specified at a more detailed level will not necessarily use the same approach to problem solving (and hence the same generic inference structure) as the top level task; to continue the example, an approach which uses hierarchical design as the overall approach to problem solving level may use transformational design or incremental design to produce certain sub-parts of the design.

## 2.2 Higher level frameworks for design: data flow vs iteration

The different approaches to design suggested by KADS provide a fairly comprehensive classification of approaches to design – *if* it is assumed that design is a sequential, non-iterative process. This can be seen in the KADS "generic design model" (reproduced in Figure 1), in which an informal problem statement is transformed into a detailed design wit no significant iteration between the various stages of transformation. The different approaches to design suggested by KADS are essentially special cases of the generic design model, with emphasis on different inference steps; for example, incremental design emphasises the **transform/expand/refine** inference step [Tansley & Hayball, 1993].

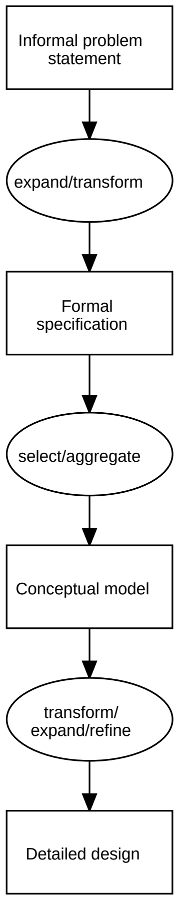


**Figure 1**: The sequential "generic design model" suggested by KADS

This sequential approach to design has been recommended by several sources (e.g. [Asimow, 1962] [Hubka, 1982]), including the influential Royal Institution of British Architects ([R.I.B.A., 1965]). It has been proposed as a suitable model for software engineering, where it corresponds to the 'waterfall' model of software development (see [Royce, 1970]). However, more recent writers have criticised the sequential approach to design. It has been claimed that this approach over-emphasises the need for the communication of data and under-emphasises the need to integrate the knowledge and information used in design [Smithers *et al*, 1990]; that the development of sequential models fails to represent the true nature of the design process [Visser, 1988]; and that the sequential approach has more to do with the job of managing the people employed in design, rather than with what designers actually do (cf. pp. 25-26 of [Lawson, 1980]). The second and third criticisms certainly seem to be valid in software design, for very few software projects actually adhere to a strict waterfall model of development.

So, if designers do not work according to a sequential model of design, how do they operate? The alternative to a sequential approach is an iterative approach. In such an approach, designers do not work through a problem step by step, first analysing and then synthesising. Instead, designers propose a solution at an early stage, and then iterate towards a final solution by presenting the early solution for criticism; this may involve elicitation of further constraints. In software engineering, this approach to design design corresponds to "rapid prototyping" which has commonly been used for the development of knowledge based systems, and is sometimes used to aid requirements specification in large software projects. Rapid prototyping involves preparing and implementing a software design quickly before showing it to the client for criticism; the implementation is then altered to take account of any changes which are suggested, and the process is repeated. This iteration normally continues until an acceptable design is reached.

The following experiment [Lawson, 1972] illustrates the use of iterative design by architectural designers. Two groups of students - postgraduate science students and final year architectural students - were given a set of blocks which had some faces coloured blue and some coloured red. The students were told to build a structure which had as few external blue faces as possible. The students were also told that there was another rule which limited their freedom of choice, but they would not be told what that rule was. Instead, they could present possible designs for criticism. They were, however, to present as few intermediate designs as possible.

The experiment revealed that the engineering students tended to focus on determining the unknown rule. Once they had presented enough attempts to deduce the rule, they calculated the optimum configuration of blocks. The design students, however, tended to propose a fairly good solution as a first step; if it was declared to be incorrect, they altered the design slightly, and continued to make slight alterations until they had produced the best design possible which was not declared to be incorrect. Analysis of the results showed that the design students performed as well as the enginering students in reaching an optimum design, and produced a significantly lower number of intermediate designs in the process.

There is documented support for the use of iterative design by architectural designers [Lawson, 1980], bridge designers [Reich, 1991] and user interface designers [Gould & Lewis, 1985] [Gould *et al*, 1987], as well as support from the AI community, with its inherent interest in identifying and modelling human cognitive processes [Bicard-Mandel & Tong, 1992] [Smithers *et al*, 1990] [Chandrasekaran, 1990]. Indeed, Chandrasekaran [Chandrasekaran, 1990] discusses iterative design, which he calls **propose-critique-modify** design, in detail. While Chandrasekaran's preferred name is an accurate description of the processes involved in iterative design, it does not correspond with KADS' terminology for inference functions (see [Breuker., 1987]), nor with the level of abstraction at which KADS names its inference structures (e.g. KADS uses the term *hierarchical* design instead of *decompose-*

*design-reconstitute* design). For compatibility with KADS, therefore, this paper follows Smithers *et al* [Smithers *et al*, 1990] in using the term "exploration-based design" to describe this approach to design.

The thesis of this paper is that exploration-based design is a commonly used approach to design, and is worthy of being included in the KADS library of generic inference structures, because it is sufficiently different from the approaches already specified in the library. The structure of the paper is as follows:

- The next two sections discuss two key aspects of exploration-based design: the role of constraints in design, and the use of previous models as a basis for a design;
- The following two sections bring the conclusions together into a suggested inference structure, and show how that inference structure was applied to a particular project;
- The final section looks at how knowledge acquisition might be performed for a task which uses exploration-based design.

## 3 The role of constraints in exploration-based design

In any design task, the key elements of the design problem are the constraints placed on the designer. Designers must identify these constraints, and then work within them to produce an acceptable design. If a design cannot be produced which fully satisfies all constraints, then one or more constraints must be relaxed, or abandoned entirely, in order to to produce a feasible design.

In exploration-based design, a client's criticisms of a possible design effectively place more constraints on the design. However, since criticism requires communication, the designer may take the opportunity to negotiate with the client on which constraints can be relaxed, and how far. It is therefore crucial for the designer to understand each constraint, and the consequences of relaxing it, thoroughly. This is particularly important if there are time restrictions on the design process, which reduce the number of explorative iterations which can be performed.

### 3.1 Understanding constraints

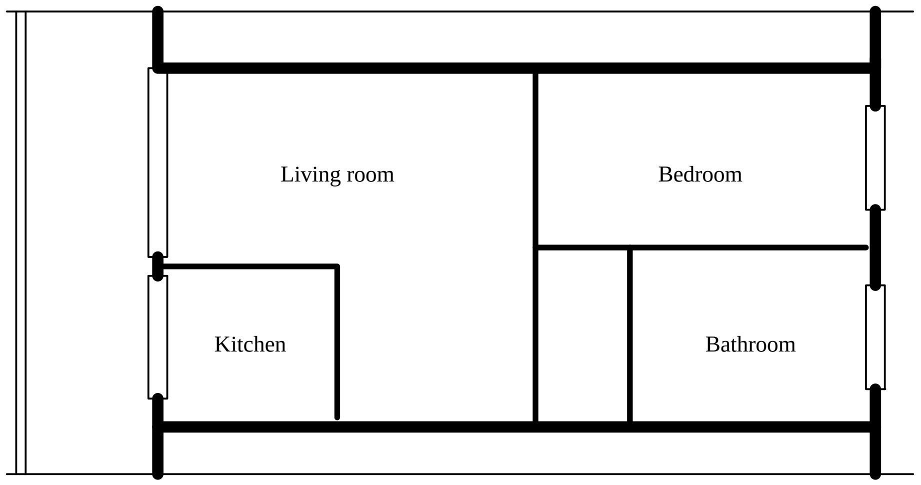
In order to understand constraints fully, Lawson [Lawson, 1980] suggests that constraints should be analysed on three dimensions:

- Is the constraint imposed *internally* or *externally*? An internal constraint is one imposed by a decision of an interested party; for example, an architectural design may be required to include ramps throughout for use by disabled people, or a graphic design may be required to make use of the colours associated with the client company's corporate image. An external constraint is one which cannot be altered by any decision of the project team; the points of the compass (and hence the position of the sun) is an important external constraint on the design of housing.

- Who imposes the constraint? Is it the designer, the client, the user (if different from the client), or legislators? A graphic designer might decide that a better effect would be achieved if he limits his design to soft pastel colours only, which is an example of a designer-imposed constraint. The width of corridors and the number of doors in a building is affected by fire regulations, which is an example of a legislative constraint. (As an aside, Lawson notes that legislative constraints tend to be biased towards factors that can easily be measured. This has often led to designer dissatisfaction with legislation which is seen as overly restrictive, or failing to take account of special features of the particular design problem).

- What function does the constraint fulfil? Is it a *radical* constraint, affecting the fundamental purpose of the design, a *practical* constraint imposed by the limitations of technology or nature, a constraint on *form*, affecting the style and visual impact of the design or a *symbolic* constraint, affecting the visual symbolism of the design? A radical constraint might be that a school building requires rooms suitable for teaching classes; a practical constraint might be that the site for a building has a certain load-bearing capacity; a constraint on form might be that a graphic designer is required to make an advertisement striking, unusual and memorable; and an example of a symbolic constraint is that the roof of the Sydney Opera House was designed to be parabolic in shape because it is intended to symbolise the surrrounding marine environment. Practical constraints can usefully be subdivided into constraints on the parameters of the design and its environment, and constraints on the process of making, testing or assembling the artifact [Brown & Chandrasekaran, 1989].

Gaining an understanding of constraints also requires designers to recognise that they themselves sometimes place implicit constraints on the design process which are non-essential. Lawson [Lawson, 1980] reports an exercise in which novice designers (architectural students) were asked to design the floorplan for a block of flats (see Figure 2). The students were unable to produce a design which allowed sufficient light into the living room of each flat until they relaxed the constraint *which they had unconsciously imposed upon themselves* that no part of one flat should overlap with a neighbouring flat. The floorplan shown in the lower half of Figure 2 allows plenty of light into both living room and kitchen, makes each flat slightly narrower, and also provides a recessed "entrance area" for each flat. The lesson to draw from this example is that the students did not recognise the constraint on overlapping, and therefore did not realise that this constraint could be relaxed.



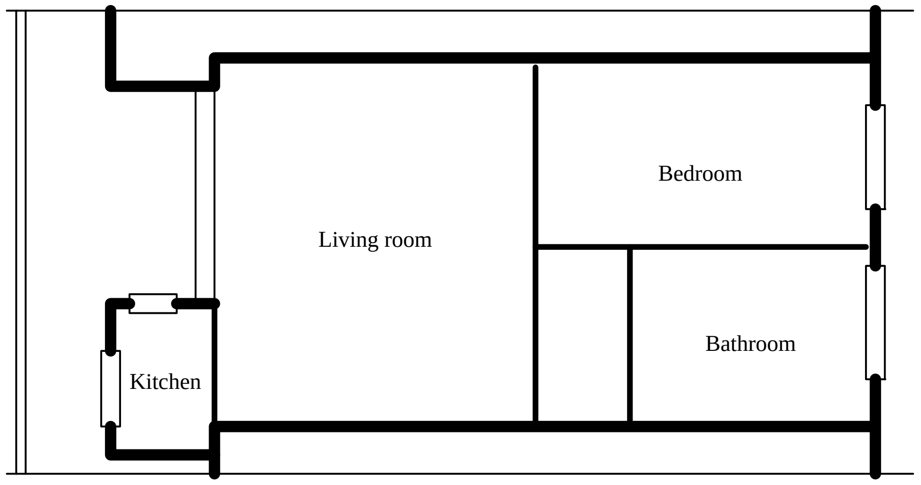


**Figure 2**: Proposed designs for single-bedroom deck-access flats (a) with a rectangular floorplan (b) with one flat overlapping the next

## 3.2 Prioritising constraints

A second key factor in exploration-based design is that designers who are presented with a large number of constraints to fulfil tend to focus on fulfilling a small number of constraints which are perceived to be important. In another experiment on students of architectural design [Lawson, 1980], three groups of students were asked to design an office building for a design competition. They were told that the building would be sited between two major roads, across the line of an existing public footpath, and that it should not present a remote or forbidding image to local ratepayers. The students all appeared to focus on one aspect of the problem, and to design their whole solution around that one aspect. One group focussed on the office environment, and designed an office layout with careful attention to the provision of service ducts and flexibility of partitioning. Another group focussed on making the building visitor-friendly, and so designed a building with different departments in different blocks leading off a central court. The third group, however, focussed on the image presented to ratepayers, and particularly on the public footpath. They proceeded to design an arch-shaped building with a covered mall in the centre doubling as the footpath!

It is important that this prioritisation of constraints is made explicit, so that a reasoned decision can be made on the relative advantages of one constraint against another.

## 4 The use of previous designs as a basis for current designs

An obvious possibility for reducing the time required to perform exploration-based design is to start with a design used in a previous similar situation, which therefore ought to satisfy most of the constraints. The issue of whether it is wise to use a previous design as a basis for a current design has been a subject of considerable debate within the design community. On the positive side, the main advantage of using an existing design is that this design (presumably) satisfies all the constraints which were imposed on it, and so is likely to satisfy many of the constraints which will be imposed in a similar situation. Some would also claim that it is well-nigh impossible for a designer to ignore his previous experience of similar designs when producing a new design, and so the process might as well be explicit. On the negative side, it is claimed that re-use of existing designs stifles creativity in design; the experiment cited in section 3.1 showed how the unconscious effects of previous experience hindered the students from arriving at an acceptable solution to their design problem. It is accepted that innovative design is largely dependent on improvement of a feature of an existing design, but it is argued that truly creative design is crucially dependent on freedom from such restrictions. This argument is at the heart of much criticism of designs (from both sides), and it is unlikely that designers will ever agree completely on this matter.

In the AI community, the recent successes of case-based reasoning technology for design tasks [Various, 1989] have swung the pendulum towards favouring re-use of existing designs. Case-based reasoning attempts to match the key features of the current design task against the key features of previous design tasks. If it finds a previous task which closely matched the current task, it retrieves the solution to that previous task, and then presents that solution to the user for minor modifications, or possibly attempts to make modifications itself.

Given the potential of case-based reasoning, and a degree of suspicion about whether AI is appropriate for "creative" design, it seems pragmatic to assume that any AI-based approach to exploration-based design is likely to make use of an existing design as a basis for the current design.

## 5 The inference structure

Based on the above analyses of the process of exploration-based design, a generic inference structure should include:

- acquisition of constraints
- ordering of constraints
- creation of a possible solution
- verification of that solution
- feedback from verification to an earlier stage in the process, thus creating an iterative loop
- potential input from previous design models

In some cases, it is possible that attempts at producing a design may prove to be dead-ends, because constraints cannot be relaxed sufficiently to produce an acceptable design. In these cases, the designer has to choose another initial model, and re-start the reasoning process. The inference structure should therefore also represent the possibility of selecting a new initial model from the model library.

The suggested inference structure is shown in Figure 3. As described in section 2.1, KADS allows inference structure diagrams to be hierarchically decomposed. In this case, it is convenient to decompose the *transform-2* inference function in the top level model. This inference function, which represents the process of assigning importance to constraints, is shown in Figure 4.

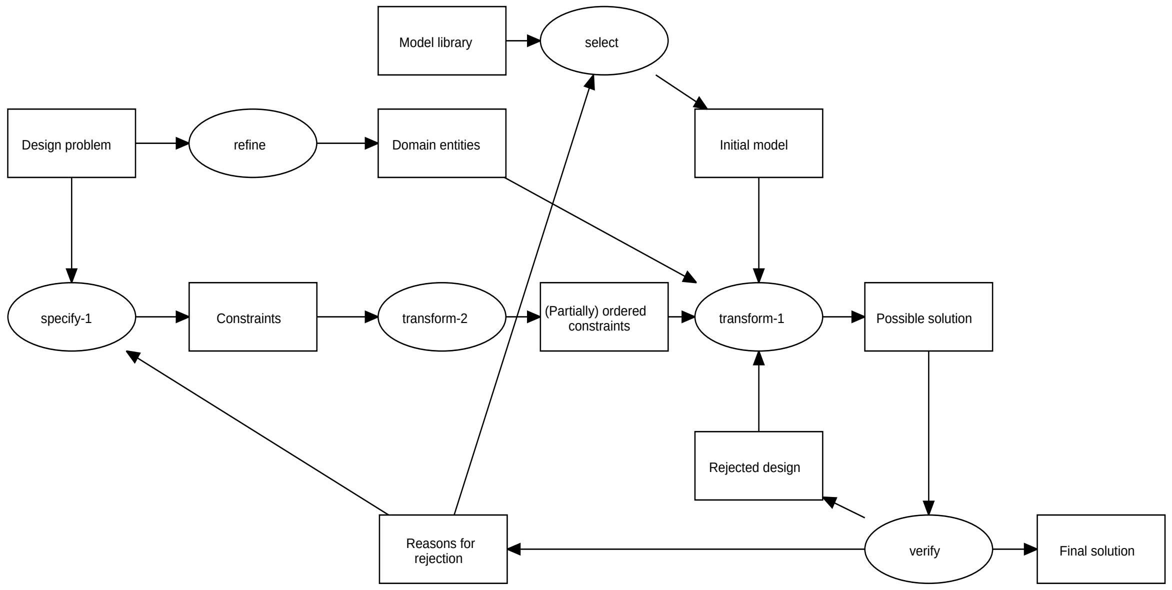


**Figure 3: Top level inference structure for exploration-based design**

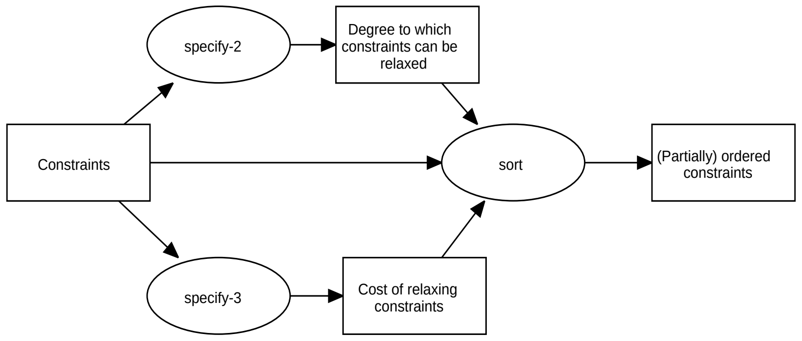

specify-2
Degree to which constraints can be relaxed
Constraints
sort
(Partially) ordered constraints
specify-3
Cost of relaxing constraints

**Figure 4: Expansion of *transform-2***

The inference structure shown above is not intended to make every aspect of the design process explicit; it only shows the typical processes in exploration-based design. Certain information which is specific to the problem domain must be added for the model to be complete. For example, the model does not indicate which constraints should be relaxed or abandoned if it proves impossible to produce a design which fully satisfies all constraints; nor does it provide any information about how an appropriate initial model is selected from the model library. Both of these factors form a significant component of design expertise, and should be specified as part of the process of instantiating the generic inference structure to a particular task.

## Task structure for the top level inference structure

```
task design
    goal to synthesise a solution to a design problem
    task structure
        refine(design problem → domain entities)
        specify(design problem → constraints)
        select(model library → initial model)
        for all constraint ∈ constraints do
          specify(constraint → degree to which constraint can be relaxed)
          specify(constraint → cost of relaxing constraint)
        sort(constraints & cost of relaxing constraints & degree to which constraints can be relaxed → (partially) ordered constraints)
        transform(domain entities & initial model & (partially) ordered constraints → possible solution)
        loop
          verify(possible solution → rejected design & reasons for rejection OR final solution)
          specify(reasons for rejection → further constraints OR select new model from model library)
          transform(rejected design & constraints → possible solution)
```

## 6 Validation of the inference structure

The suggested inference structure has been validated by applying it to a real-life knowledge-based design problem. The problem chosen was that of a consultant or subcontractor negotiating an acceptable workplan for a commercial contract. The tasks to be done, the skills required for each task, and the overall cost of the task must all be defined by the consultant and agreed by the client. Previous workplans may be used as a basis for a current workplan, especially in companies which have well-defined “packages” of work which are sold as a whole.

The application of an inference structure requires the knowledge roles to be instantiated to entities from the domain. If necessary, inference functions and knowledge roles may be added or deleted. In this case, no alterations were required; the mapping was as follows:

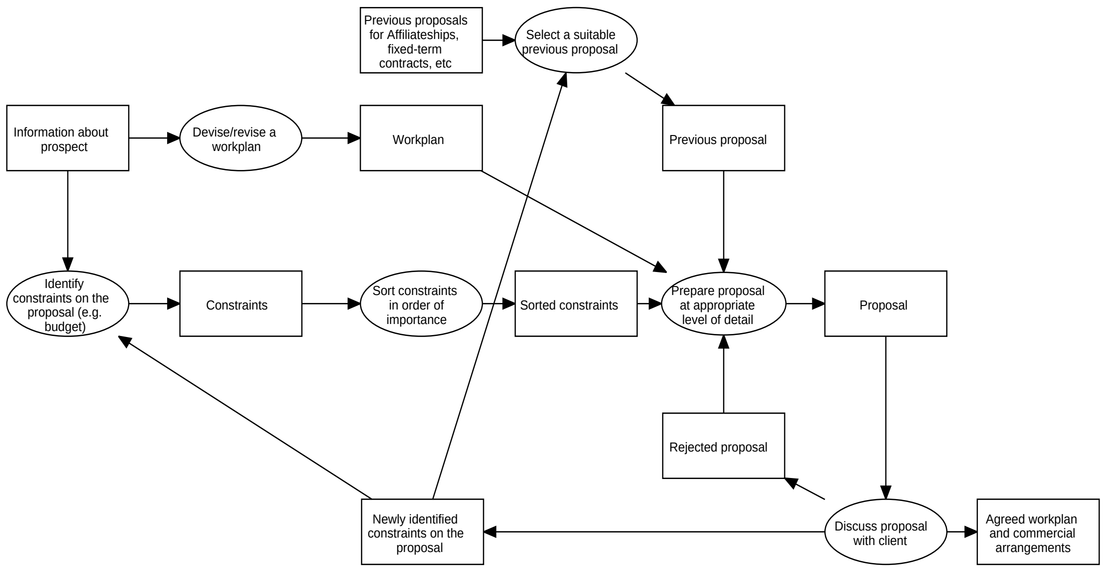

Previous proposals for Affiliateships, fixed-term contracts, etc
Select a suitable previous proposal
Information about prospect
Devise/revise a workplan
Workplan
Previous proposal
Identify constraints on the proposal (e.g. budget)
Constraints
Sort constraints in order of importance
Sorted constraints
Prepare proposal at appropriate level of detail
Proposal
Rejected proposal
Newly identified constraints on the proposal
Discuss proposal with client
Agreed workplan and commercial arrangements

**Figure 5: The processes involved in discussing a proposal with a client**

This study indicates that the proposed generic inference structure for exploration-based design can indeed be instantiated to a real-world design problem.

## 7 Knowledge Acquisition for exploration-based design

The process of instantiating a generic inference structure requires that knowledge acquisition is carried out which identifies the information required for each knowledge role. The information which must be acquired includes the domain entities and initial constraint which form the design problem, the contents of the model library, and the format of knowledge generated from these initial inputs.

The most obvious technique for acquiring knowledge for exploration-based design is to perform exploration-based design; this quickly provides a lot of useful knowledge, particularly about constraints. This technique has been used successfully in a number of knowledge based projects, by using "rapid prototyping" as a basis for knowledge acquisition. However, some information has to be gathered before an initial design can be produced; there may also be some benefit in reducing the number of times that solutions are presented to the client/expert, to avoid causing irritation, or to reduce the total time required for design. It therefore seems wise to devise techniques which can acquire as much knowledge as possible before presenting a solution to a client, and on each iterative loop thereafter.

This section presents some suggested techniques for knowledge acquisition, looking particularly at acquisition of constraints.

## 7.1 Knowledge Acquisition for constraints

According to the inference structure, a design problem can be described in terms of the constraints which are placed on the design, and the domain entities. It is rarely difficult to determine what the domain entities are (although the relationships between them may require a little more thought); many of the difficulties in design revolve around undetermined or underspecified constraints. Knowledge acquisition for a design problem is therefore primarily concerned with the acquisition of constraints.

### 7.1.1 Interviews

So how can constraints be acquired? An obvious method for acquiring knowledge of any sort is to perform interviews with experts in the field. While interviews have advantages, particularly at early stages of a knowledge engineering project, they also have considerable disadvantages, particularly in the elicitation of tacit knowledge. Many designers find it easier to work on refining an actual design rather than attempting to analyse every constraint, and many design faults are due to unidentified constraints; it follows that many constraints on design problems are either within the designer's mind but unexpressed, or within the problem but unnoticed. These constraints can therefore be classified as tacit knowledge. While it is possible that structured interviews may have value at later stages of the knowledge acquisition process (for example, a designer may be asked to critique a written list of constraints), it seems that knowledge engineers will need to rely on techniques other than interviews in order to acquire constraints successfully.

### 7.1.2 The Problem Identification Game

For the early stages of constraint acquisition, Lawson [Lawson, 1980] suggests the "Problem Identification Game", which was devised at the Open University as an aid to identifying constraints. The 'game' requires designers to start by making a short and simple statement of the design problem as a contrasting pair; an example might be "slum clearance – aged slum dwellers". Next, designers are asked to amplify this statement by considering the following principles:

- Conflict - convert the statement into interested parties who might be viewed as in conflict. For example, "Town planners see a need for change and renewal which is not necessarily appreciated by the aged who have lived in the area all their lives";

- Contradiction - trying to contradict an earlier statement by taking an opposing viewpoint (e.g. "slum clearance – old people need safety & hygiene");
- Complication - identify any factors which should really have been considered when making a previous statement e.g. "Old folk need modern housing because they need safety & hygiene" is subject to the complication "But modernisation usually means increased rent charges";
- Similarity - try to think of, and then think through, an analogous situation (e.g. slum clearance is to housing as a plough is to a field; the process of slum clearance destroys previous street patterns, but opens up the area for new growth);
- Chance - pick a word from a dictionary and see if it sparks any new ideas. For example, the word "softly" might suggest soft music, which in turn leads to a consideration of the difficulties of moving grand pianos and other accumulated furniture into modern housing.

It is usually a simple task to extract constraints from these statements, although further analysis may have be done on the degree to which the constraints can be relaxed and the cost of relaxing them. For example, two constraints which can be extracted from the example given above are "Old folk prefer large housing to accommodate their possessions" and "Old folk prefer low-rent housing". However, these two constraints are (usually) in opposition, and further analysis is needed to determine how resistant old people would be to giving up possessions in order to live in smaller housing, or how heavy the financial burden of large housing would be.

### 7.1.3 Multi-dimensional techniques

Once some constraints have been identified, it is also possible to elicit constraints using multi-dimensional knowledge elicitation techniques such as the *card sort* [Shadbolt & Burton, 1990] and the *repertory grid* [Kidd, 1987]. To use the card sort technique for acquiring constraints, the name of each constraint is written on an individual index card, and the designer is asked to sort the cards into piles, in any way which seems sensible. This technique is repeated several times, until the constraints have been classified in several different ways. The key step in eliciting constraints is to ask the designer, after each sort has been completed, if there are any other constraints which belong in the categories he has created, but which are not represented on cards. Despite its simplicity, card sorting has proved to be an effective technique in commercial projects.

The repertory grid is used in a similar fashion. The repertory grid technique identifies problem *elements* which have *constructs* (attributes).For constraint elicitation, constraints form the elements of the grid, and the designer is then presented with three constraints (chosen at random) and asked to state how two of them differ from the third. The designer's answer (e.g. "Two of these have a low impact on cost of the design, while one has a high impact") is taken to be an attribute of all constraints, and is defined as a construct. Each constraint is then assigned a value for this construct on a continuous scale. If the scale used for constructs is the same throughout the grid, and is numerical, then repertory grids can be subjected to statistical analysis which produces an implicit clustering of elements. This clustering can be discussed with the expert designer, with emphasis on unexpected assignment to clusters and the nature of the clusters themselves. As with the card sort, it is possible to enquire if any constraints which are not yet represented belong in the clusters.

#### 7.1.4 Eliciting constraints by identifying incompatibilities in possible solutions

It is possible to use a variation of the repertory grid knowledge acquisition technique to analyse constraints, if possible solutions to the design problem (or parts of the design problem) can be defined. Bradshaw [Bradshaw *et al*, 1989] shows how "possibility grids" can be defined, in which possible solutions are assigned "goodness" values on a range of constraints. The grid is then analysed in terms of the "goodness values"; incompatible combinations of values are ruled out, and all other possible combinations are generated. If there is a combination of constraint values which does not match an existing design solution, then either a new possible solution has been found (if this combination is permissible), or a new constraint is elicited (if this combination is deemed unacceptable).

## 7.2 Knowledge Acquisition of model library

At first sight, it might appear that obtaining examples of previous designs would not be difficult. In practice, the situation is more complex. The problem lies in deciding how to represent designs within a library. Either the library will contain a large number of previous designs, which must be indexed by some key design features in order to allow for efficient search through the library, or it will contain an abstracted set of "typical" designs, in which certain specific features of real-life designs are not represented.

In either case, it is crucial that the key factors which differentiate designs are defined carefully. The existence of differentiating factors implies an underlying classification scheme; however, there is no agreed classification for design tasks in general. As a result, key differentiating factors must be defined for each domain. This is a significant task in knowledge acquisition. It is possible that machine learning techniques, such as rule induction or neural networks, may be of assistance here, but little empirical work has been done to verify this.

### 7.3 Knowledge Acquisition of inference functions

The best way of acquiring knowledge about the various inference processes in a design task is likely to be highly domain-dependent. If there are a considerable number of procedural steps to be followed, however, then certain knowledge acquisition techniques such as protocol analysis, the laddered grid [Shadbolt & Burton, 1990] or the "20 Questions" technique [Burton *et al*, 1988] may be useful. For an example of the use of these techniques, see [Kingston, 1992a].

## 8 Conclusion

This paper has demonstrated that an iterative approach to design, which is termed "exploration-based design", is used by real-life designers. It is not used by all designers – sequential approaches to design are often used where they are feasible. The existing KADS library of generic inference structures provides a useful classification of techniques for sequential design. However, some designers clearly do use exploration-based design, and the library lacks an inference structure for exploration-based design. A suitable inference structure is therefore proposed, and tested in the field. Techniques for acquiring the knowledge required for exploration-based design are also suggested.

In order to introduce the subject of exploration-based design, this paper has given considerable space to discussion of the nature of design tasks. A key conclusion of this discussion is that design tasks can be classified at two levels of abstraction. At the higher level, design tasks can be classified as either sequential or iterative. At a lower level, sequential design tasks can be classified as hierarchical design, transformational design or incremental design. Iterative design tasks currently include exploration-based design only; it is possible that further research may produce more categories of iterative design, which would help to expand the KADS library of inference structures even further.

## Acknowledgements

I would like to thank Gail Anderson and Jim Doheny of AIAI for their comments on drafts of this paper.

## References

[Asimow, 1962] Asimow, M. (1962). *Introduction to Design.* Prentice Hall.

[Bicard-Mandel & Tong, 1992] Bicard-Mandel, J. and Tong, X. (17-18 Feb 1992). ICT: Integrity Checking Task - A Generic Task for Design under Constraints. In *Proceedings of the 2nd KADS User Meeting*, Siemens AG, Munich. European KADS User Group.

[Bradshaw *et al*, 1989] Bradshaw, J.M., Boose, J.H., Covington, S.P. and Russo, P.J. (1989). How To Do With Grids What People Say You Can't. In *Proceedings of Knowledge Acquisition Workshop.*

[Breuker., 1987] Breuker., J. A. (1987). *Model-driven Knowledge Acquisition.* University of Amsterdam and STL, ESPRIT project 1098, Deliverable A1.

[Brown & Chandrasekaran, 1989] Brown, D.C. and Chandrasekaran, B. (1989). *Design Problem Solving: Knowledge Structures and Control Strategies.* Research Notes in Artificial Intelligence. Morgan Kaufman.

[Burton *et al*, 1988] Burton, A.M., Shadbolt, N.R., Rugg, G. and Hedgecock, A.P. (1988). Knowledge Elicitation Techniques in Classification Domains. In *Proceedings of ECAI-88: The 8th European Conference on Artificial Intelligence.*

[Chandrasekaran, 1990] Chandrasekaran, B. (1990). Design Problem Solving: A Task Analysis. *AI Magazine*, 11(4).

[deHoog *et al*, 1993] de Hoog, R., Martil, R., Wielinga, B., Taylor, R., Bright, C. and van de Velde, W. (1993). The Common KADS model set. ESPRIT Project P5248 KADS-II KADS-II/M1/DM1.1b/UvA/018/ 6.0, University of Amsterdam and others, http://swi.psy.uva.nl/ projects/CommonKADS/Reports.html.

[Gould & Lewis, 1985] Gould, J. D. and Lewis, C. (1985). Designing for usability: key principles and what designers think. *Communications of the ACM*, 28:300–311.

[Gould *et al*, 1987] Gould, J. D., Boies, S., Levy, S., Richards, J. T. and Schoonard, J. (1987). The 1984 Olympic message system: A test of behavioural principles of system design. *Communications of the ACM*, 30:758–769.

[Hickman *et al*, 1989] Hickman, F., Killin, J., Land, L. *et al.* (1989). *Analysis for knowledge-based systems: A practical introduction to the KADS methodology.* Ellis Horwood, Chichester.

[Hubka, 1982] Hubka, V. (1982). *Principles of Engineering Design.* Butterworth Scientific, Guildford, trans. W. E. Eder.

[Kidd, 1987] Kidd, A., (ed.). (1987). *Knowledge Acquisition for Expert Systems: A Practical Handbook.* Plenum Press.

[Kingston, 1992a] Kingston, J. K. C. (1992a). KBS Methodology as a framework for Co-operative Working. In *Research and Development in Expert Systems IX.* British Computer Society, Cambridge University Press. Also available from AIAI as AIAI-TR-130.

[Kingston, 1992b] Kingston, J.K.C. (November 1992). Pragmatic KADS: A methodological approach to a small KBS project. *Expert Systems: The International Journal of Knowledge Engineering*, 9(4). This paper is also available as AIAI Technical report AIAI-TR-110.

[Kruger & Wielinga, 1993] Kruger, C. and Wielinga, B. (1993). A KADS model for the industrial design task. In Löckenhoff, C., (ed.), *Proceedings of the 3rd KADS Meeting*, pages 131–141, ZFGE BT SE 21, Otto-Hahn-Ring 6, D8000 Munich 83, Germany. Siemens AG.

[Lawson, 1972] Lawson, B.R. (1972). *Problem solving in architectural design.* Unpublished Ph.D. thesis, University of Aston, Birmingham.

[Lawson, 1980] Lawson, B. (1980). *How Designers Think.* The Architectural Press Ltd: London.

[Maher, 1990] Maher, M. L. (1990). Process Models for Design Synthesis. *AI Magazine*, 11(4):49–58.

[McDermott, 1982] McDermott, J. (1982). R1: A rule-based configurer of computer systems. *Artificial Intelligence*, 19, 1:39–88.

[Reich, 1991] Reich, Y. (September 1991). Design knowledge acquisition: task analysis and a partial implementation. *Knowledge Acquisition*, 3:237–254.

[R.I.B.A., 1965] R.I.B.A. (1965). *Royal Institution of British Architects' Architectural Practice and Management Handbook*. RIBA Publications, London.

[Royce, 1970] Royce, W. W. (1970). Managing the Development of Large Systems: Concepts and Techniques. In *1970 WESCON Technical Papers, v. 14, Western Electronic Show and Convention*, pages A/1–1 – A/1–9, Los Angeles. WESCON. Reprinted in Proceedings of the Ninth International Conference on Software Engineering, Pittsburgh, PA, USA, ACM Press, pp.328–338.

[Schreiber *et al*, 1993] Schreiber, A. Th., Wielinga, B. J. and Breuker, J. A., (eds.). (1993). *KADS: A Principled Approach to Knowledge-Based System Development*. Academic Press, London.

[Shadbolt & Burton, 1990] Shadbolt, N. and Burton, A.M. (1990). Knowledge elicitation. In J. Wilson and N. Corlett, (ed.), *Evaluation of Human Work: A Practical Ergonomics Methodology*, pages 321–346. Taylor and Francis.

[Smithers *et al*, 1990] Smithers, T., Conkie, A., Doheny, J., Logan, B., Millington, K. and Tang, M. X. (1990). Design as intelligent behaviour: an AI in design research programme. *Artificial Intelligence in Engineering*, 5(2):78–109.

[Tansley & Hayball, 1993] Tansley, D. S. W. and Hayball, C. C. (1993). *Knowledge-Based Systems Analysis and Design: A KADS Developers Handbook*. Prentice Hall.

[Tate *et al*, 1994] Tate, A., Drabble, B. and Kirby, R. (1994). O-plan2: An open architecture for command, planning and control. In Fox, M. and Zweben, M., (eds.), *Knowledge Based Scheduling.* Morgan Kaufmann., Palo Alto, California, 94303, USA.

[Various, 1989] Various. (1989). *Proceedings of the Third DARPA Workshop on Case-based Reasoning.* Morgan Kaufmann, Articles in this proceedings include "A Case-based tool for Conceptual Design Problem Solving" and "Case Adaptation in Autoclave Layout Design".

[Visser, 1988] Visser, W. (March 1988). Giving up a hierarchical plan in a design activity. Technical Report 814, Institut Nationale de Recherche en Informatique et en Automatique, Domaine de Voluceau Rocquencourt B.P. 105, 78153 Le Chesnay Cedex, France, An English version of a paper presented in French at the COGNITIVA87 conference, Cesta, Paris 1987.