Design Synthesis:

A Model of Hierarchical Case-Based Reasoning

Barry Smyth, Donal Finn
Hitachi Dublin Laboratory
Trinity College,
University of Dublin,
Dublin,
IRELAND.

Phone: +353-1-6798911
Fax: +353-1-6798926
EMail: barry@hdl.ie donal@hdl.ie

•

Mark T. Keane
Computer Science Department,
Trinity College,
University of Dublin,
Dublin,
IRELAND.

Phone: +353-1-7021534
EMail: mkeane@cs.tcd.ie

ABSTRACT

A variety of artificial intelligence techniques have been used in attempts to automate design synthesis tasks. Two common approaches are case-based and decompositional design. While powerful techniques in their own right, their integration has lead to a new generation of design synthesis systems capable of tackling a larger range of problems with greater effectiveness. In this paper previous attempts at integrating these approaches are examined in a number of design systems. Although significant advances have been made, important shortcomings still exist. The main focus of this paper is to address the limitations of these design synthesis models. To this end Déjà Vu, a new hybrid model of design synthesis, has been developed and is described. Déjà Vu integrates decompositional and case-based approaches in a framework that exploits the power of experiential knowledge, and benefits from far greater domain applicability when compared to existing design techniques. Two implementations of this model, that are targeted at real-world software design tasks, have yielded encouraging preliminary results and are also described in this paper.

Design Synthesis:

A Model of Hierarchical Case-Based Reasoning

ABSTRACT

A variety of artificial intelligence techniques have been used in attempts to automate design synthesis tasks. Two common approaches are case-based and decompositional design. While powerful techniques in their own right, their integration has lead to a new generation of design synthesis systems capable of tackling a larger range of problems with greater effectiveness. In this paper previous attempts at integrating these approaches are examined in a number of design systems. Although significant advances have been made, important shortcomings still exist. The main focus of this paper is to address the limitations of these design synthesis models. To this end Déjà Vu, a new hybrid model of design synthesis, has been developed and is described. Déjà Vu integrates decompositional and case-based approaches in a framework that exploits the power of experiential knowledge, and benefits from far greater domain applicability when compared to existing design techniques. Two implementations of this model, that are targeted at real-world software design tasks, have yielded encouraging preliminary results and are also described in this paper.

1 Introduction

Case-based reasoning (CBR) and decomposition are two distinct techniques that have been used to model design synthesis. Decomposition has found application in design domains that are well understood, especially where generalised decomposition strategies can be easily applied to break down complex design problems into more manageable sub-problems. However, decomposition techniques are difficult to apply in less tractable domains where domain knowledge is incomplete or where generalised decomposition strategies are difficult to formulate (Chandrasekaran, 1990). On the other hand, case-based reasoning has found application in such ill-structured domains where cases provide a mechanism for representing incomplete or poorly defined knowledge (Maher, 1990). Unfortunately, case-based reasoning suffers disadvantages also, especially when dealing with large complex design problems. These shortcomings have prompted researchers to develop hybrid design synthesis models that combine decomposition and CBR techniques (Hinrichs, 1991; Maher and Zhang, 1993; Domeshek and Kolodner, 1993). The main approach in hybrid decompositional case-based reasoning is to initially apply a decomposition approach to simplify complex design problems, and then to exploit case-based reasoning techniques to solve for each of the synthesis sub-problems. Although this approach has allowed case-based reasoning to be applied to more complex design problems, it is still only applicable to the class of design problems where decomposition techniques are effective. Therefore hybrid decomposition-CBR remains ineffective for design synthesis tasks in domains where decomposition techniques are fraught with difficulties.

In this paper we propose a new model for design synthesis called Déjà Vu. Déjà Vu carries out design synthesis using a process model that we call *hierarchical case-based reasoning*. In common with other hybrid approaches, a design problem is first decomposed into simpler sub-problems and is then solved by applying case-based reasoning. However, what distinguishes Déjà Vu from other hybrid systems, is that, decomposition is carried out in a recursive, hierarchical manner using case-based reasoning methods. Hierarchical case-based reasoning has a number of significant advantages over other hybrid synthesis models developed to date and these include; design synthesis can be carried out in ill-structured domains where generalised decomposition techniques are difficult to apply, secondly, episodic-driven decomposition is more effective in strongly coupled domains where traditional decomposition techniques have failed in the past, and finally case-based decomposition facilitates learning thereby leading to more powerful design synthesis tools. Thus, we believe that the application of this novel approach to design synthesis allows the solution of a certain class of design problems which hitherto have been outside the realm of other hybrid models.

The paper is divided as follows. Section 2 examines design synthesis from a number of perspectives, including; knowledge engineering requirements, generalised models of synthesis and implemented models to date. We argue that there is a correlation between the synthesis model chosen and the knowledge characteristics of a design domain. Section 2 concludes by identifying a certain class of design synthesis problems that current models do not address and proposes the new

synthesis model of hierarchical cased-based reasoning. In Section 3, two domains in which this model has been applied are briefly described. Section 4 describes Déjà Vu in detail, focusing on the model's knowledge requirements and design processes, and closes with a description of the model's beneficial implications. In concluding, we sum up our results and briefly contrast our model with other hybrid design systems.

2 Design Synthesis

Gero argues that design exists because the world around us does not suit us, and the goal of designers is to change the world through the creation of artefacts. Designers design by positioning functions to be achieved and producing descriptions of artefacts capable of generating these functions (Gero, 1990). Considering the process of design itself, Maher subdivides it into three phases; formulation, synthesis and evaluation. In this proposal, a process model for design, Maher defines formulation as the identification of the goals of the design process; synthesis as the process of developing a design solution by considering the design space that contains the design knowledge; and evaluation as the task of assessing whether the design goals have been satisfied subject to design constraints (Maher, 1990). Chandrasekaran defines a task oriented structure for design in terms of a general class of methods that he describes as *propose-critique-modify* methods. In this definition, design is viewed as a series of process tasks that ultimately results in the creation of an artefact with some desired design functionality. Functionality is described in terms of a set of specifications that a design is expected to deliver within a set of prescribed design constraints, and the design process is described as the task of constructing the design artefact by using a repertoire of design components (Chandrasekaran, 1990). In these models, Gero, Maher and Chandrasekaran agree that synthesis is the process by which one or more design solutions, consistent with the requirements defined during formulation, are identified.

2.1 Design Synthesis - The Role of Domain Theory and Knowledge Structures

In his task oriented framework for design, Chandrasekaran identifies three broad methods in design synthesis; decomposition/transformation methods, case-based methods and global constraint-satisfaction methods (Chandrasekaran, 1990). Maher in her process model of design, identifies three almost similar approaches, namely; decomposition, case-based reasoning and transformation (Maher, 1990). In this paper we are primarily interested in decomposition and case-based reasoning, so we do not examine design by transformation or constraint satisfaction methods in any detail. It is our belief that the process model chosen for design synthesis tasks is strongly influenced by the design domain itself and the available domain knowledge, and these relationships are examined next.

2.1.1 Design Theory: Strong and Weak Domains

In this section we examine the relationship between domain theory, domain knowledge and the design synthesis process. We define domain theory as the underlying causal theory of the *real-world* design domain, whereas domain knowledge refers to the conceptual view of these theories as represented in knowledge-based design systems (see Figure 1). In our experience this is a more useful perspective than the usual view of *shallow/deep* knowledge. The question of whether the knowledge is shallow or deep is not really important. What *is* important is, firstly, how well the theory-based structures capture the design domain (strong *vs.* weak theory), and secondly, how well a system's knowledge representation approximates this theory (complete *vs.* incomplete domain knowledge).

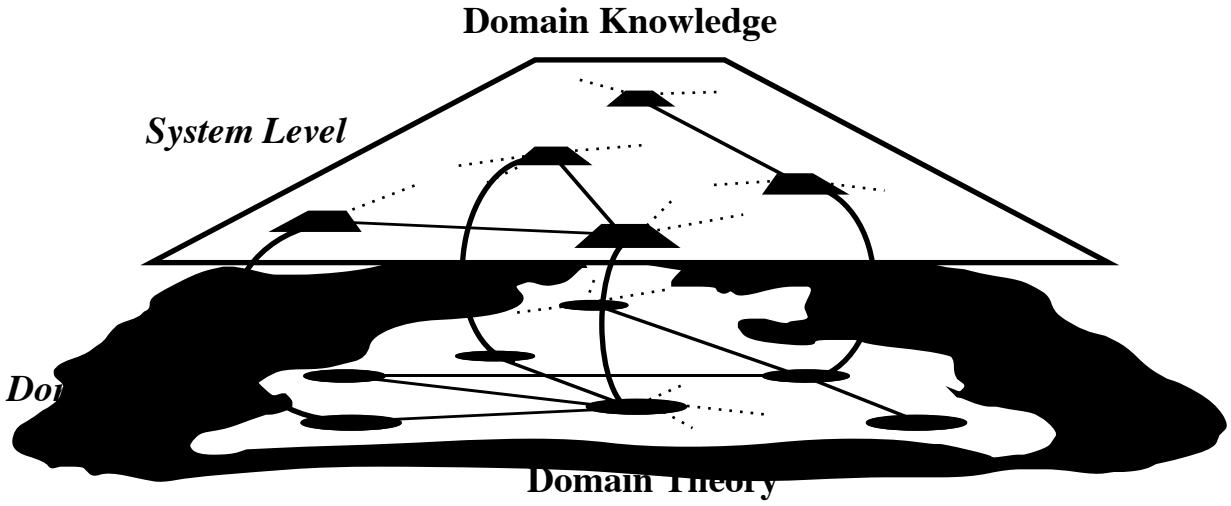

Figure 1. Domain theory vs. domain knowledge.

A strong domain theory suggests the existence of a causally coherent model of design within a given domain. Figure 2(a) shows a representation of such a strong domain model, where essentially all of the design domain is covered by a theory-based framework.

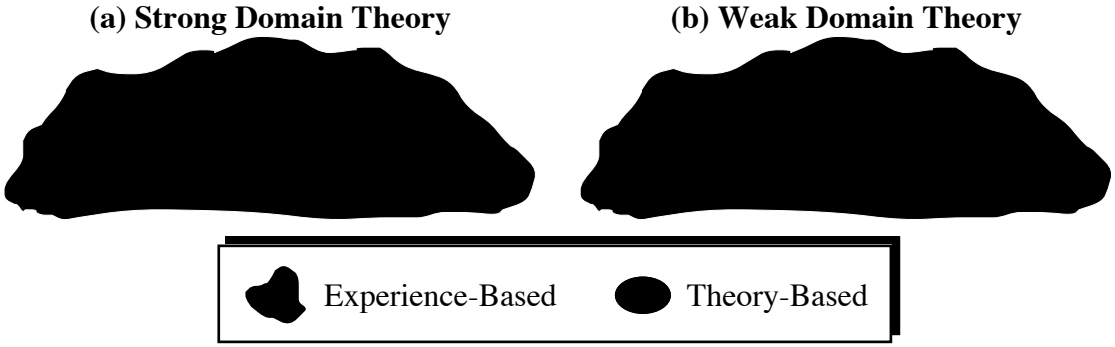

Figure 2. Strong *vs.* Weak Domain Theories.

However, in many design tasks such strong theoretical models do not exist and thus experience-based domains are common. So called weak domains lack a comprehensive model to explain all

phenomena; the causality behind many phenomena simply does not exist, no matter how hard one looks (Takeda et al. 1990). Another perspective is that there are holes in the theory-based structures and experiential knowledge acts as cementing agent, thereby consolidating a domain's theory-based model. Therefore, in weak domains (see Figure 2(b)), experiential knowledge is necessary in some form to bridge in the gaps that exist in the domain's theory-based structures. Such weak domains are typified by tasks such as architectural design, where many decisions are based on non-qualitative, aesthetic criteria. Device control software design is another weak domain task; designs often rely on empirical measurements (for example, timing considerations) that cannot be adequately captured or predicted by the domain theory.

2.1.2 Domain Knowledge: Complete and Incomplete Knowledge

Domain knowledge refers to a system's representation of a domain's theory-based structures. In this context, one can meaningfully talk about complete or incomplete domain knowledge. A complete domain knowledge representation is one that precisely models every detail of some underlying theory (weak or strong). Such complete representations only occur in routine design applications and can be observed in other AI domains such as planning. For example the domain knowledge of early planning systems such as STRIPS are good examples of complete representations. Within very restricted domains (Blocks World), a complete model of the underlying strong theory (domain rules about moving and stacking blocks) can be formulated and encoded as some symbolic representation (planning operators).

Incomplete domain knowledge, on the other hand, does not precisely model every aspect of a domain theory, but is merely an approximation; that is, even though there may exist some domain theory it cannot be accurately modelled, perhaps due to its complexity. Of course irrespective of the strength of the domain theory, an incomplete knowledge representation requires the use of experiential, case-type knowledge. In fact, the only situation where experiential knowledge is *not* required is in a system that has a complete representation of a strong domain.

2.2 Design Synthesis: Decomposition and Case-based Reasoning

Decomposition and case-based reasoning are two commonly used process models of design synthesis. In this section, we examine the appropriateness of each model in the context of domain theory and domain knowledge.

2.2.1 Decomposition

Maher describes decomposition as the most ubiquitous model of design. The approach of dividing a large complex problem into several smaller less complex problems is well accepted and practised by designers (Maher, 1990). Maher argues that decomposition is usually carried out along two lines, either by decomposing according to structure (an object-centred approach) or decomposing according to function (a function-oriented approach). These ideas are borne out in the literature; for instance in CELIA (Redmond, 1992) and PRODIGY (Velosa, 1991) decomposition is applied

along functional lines, whereas partonomic decomposition is applied in JULIA (Hinrichs, 1991) and CLAVIER (Hennessy 1992). Chandrasekaran and Maher both suggest that decomposition strategies are easiest to apply to well understood problems (Chandrasekaran, 1990; Maher, 1990), in this paper we consider such problems to be synonymous where a strong domain theory exists and complete knowledge is available.

Decomposition can offer significant advantages to designers in design situations where familiar and well understood decomposition strategies are known. Chandrasekaran notes that such situations are commonplace in routine design (Chandrasekaran 1990). Routine design is usually carried out against a background of complete domain knowledge, where effective decompositions are well known, can be easily applied and involve little search in the space of available decomposition strategies. Another domain characteristic that favours the use of decomposition is where weak coupling exists between the components or sub-functions of the decomposed problem. This effectively facilitates the independent solution of the component parts and favours a simple model of recomposition to give an overall design solution.

As decomposition is generally more effective in routine design applications, its use in domains where only incomplete knowledge is available has proven to be more problematic, e.g., non-routine design. Chandrasekaran notes that in weak domains where difficulty exists in expressing and representing domain knowledge, the specification and application of decomposition strategies is often a non-trivial task. In addition, in domains where a strong interdependency exists between components or functions, there is no guarantee, after decomposition and solution, that recomposition will be possible (Chandrasekaran, 1990). Maher expresses the difficulties associated with decomposition in non-routine design in terms of the following issues; deciding what is to be decomposed, how the decomposition is carried out, how decomposition strategies are found, how such strategies are applied, how component solutions are recomposed to give overall solutions, etc. (Maher, 1990).

Considering these general comments on decomposition, it is interesting to note that they are also borne out by practitioners who have been involved in the development of application systems. Domeshek and Kolodner report that their experience has been that decomposition in design synthesis is a difficult task, and they argue that this is especially true where the domain theory is weak and therefore forcing a relatively conservative approach to decomposition to be adopted (Domeshek and Kolodner, 1993).

2.2.2 Case-Based Reasoning

Maher describes case-based reasoning in design as the generation of a design solution using the solution or the solution process from previous design problems. She stresses that this model for design synthesis requires design episodes in the form of cases rather than generalisations about a design domain as in decomposition (Maher, 1990). Case-based reasoning involves several operations and these have been well documented in the literature (Hammond, 1989; Kolodner,

1991; Reisbeck and Schank, 1989). Briefly, they include, retrieving the relevant cases from case memory (retrieval), selecting the most promising case (mapping), modifying its solution for use in the new problem situation (adaptation), testing the solution, evaluating the results, and finally updating the case memory by storing the new case(learning). Design synthesis, in particular, encompasses the operations of case retrieval, case mapping and case adaptation.

Compared to other paradigms for design synthesis, case-based reasoning is thought to offer a number of advantages. Hua notes that a case-based design system does not require complete domain knowledge, but can produce complete and complex designs with even a small or incomplete knowledge-base (Hua and Faltings, 1993). Hinrichs notes that for design domains where incomplete knowledge exists, case-based reasoning offers a powerful paradigm by which isolated knowledge structures can be linked (Hinrichs, 1991). In addition, case-based reasoning can be quite effective where strong interdependencies exist in a solution as it explicitly maps known good solutions within the problem domain, this makes the paradigm particularly suitable for weak domains where only incomplete knowledge is available.

However, there are some disadvantages associated with case-based reasoning, particularly for design tasks. Domeshek notes that as design problems grow large, it becomes difficult to see how retrieving a single monolithic case from memory will advance the design process very much; the retrieval of a such huge case is just the start of another complex search, this time through the welter of details stored in the case (Doemshek and Kolodner, 1993). Pu emphasises this point by noting that one of the differences between traditional CBR systems and case-based design systems, is that in non-design applications it is usually feasible to store complete solutions in a single case, however, in design this approach is generally considered problematic because of the complexity of design problems (Pu, 1993). Thus, in many design applications design cases are often represented as autonomous solutions which are indexed independently of the complex design to which they physically belong. This approach has worked well for design tasks as is evident from the array of design systems that have adopted it.

2.3 Towards an Integrated Model of Design Synthesis

In the previous section we reviewed two models that have been used extensively in design synthesis. However, they have shortcomings and these have prompted researchers to consider developing hybrid design systems that incorporate both paradigms. The main approach has been to use decomposition techniques to split large design problems into smaller more manageable sub-problems and then to apply case-based reasoning to these simpler design tasks. Three systems, JULIA, CADSYN and ARCHIE demonstrate this hybrid approach and they are reviewed briefly here.

2.3.1 CADSYN

CADSYN provides a process model for design where CBR is combined with generalised decomposition techniques (Maher and Zhang, 1993). CADSYN has been developed in the application area of building structural design which is considered to be a weak domain but, because of building codes and guidelines, it can be considered to have relatively complete domain knowledge available (see Figure 3). In CADSYN, case knowledge and decomposition knowledge are represented separately. Cases are used to describe specific design episodes and are recorded as descriptions of the problem and its solution. Decomposition strategies are represented as generalised decomposition knowledge and are applied as domain knowledge. Given a new design problem, CADSYN retrieves a set of relevant cases, analyses their relevance to the current problem and determines whether case-based reasoning can be applied directly or whether decompositional techniques should be employed instead. If a decomposition approach is taken, then it is assumed that ultimately a sufficiently simplified set of problem specifications will be produced so that CBR techniques can be applied.

2.3.2 JULIA

JULIA's problem domain is in the area of meal planning. Each of JULIA's cases describe a complete meal structure and consists of sub-cases, where each sub-case describes a particular meal course (Hinrichs, 1991). JULIA tries to transfer a complete case directly, but if this fails, it then breaks the problem down under the guidance of generalised decomposition design plans and recursively solves for each sub-case, by retrieving and using different cases. An important distinguishing factor in JULIA is that, decomposition techniques represent fixed decomposition plans that operate on rigid, predetermined structures (i.e., meal courses) and therefore decomposition can be considered to be applied in a context where complete knowledge is available (see Figure 3).

2.3.3 ARCHIE

ARCHIE's problem domain is conceptual design in architecture (Domeshek and Kolodner; 1993). The authors note that in architecture, cases usually exist as large monolithic structures, and consequently decomposition is used to break these large cases into more meaningful sub problems that are useful to the user. ARCHIE uses architectural heuristics to guide decomposition according to either functional or form requirements. ARCHIE can be considered to be an application in a weak domain (conceptual design in architecture) where decomposition is guided by weak methods (heuristics) which are based on incomplete knowledge (see Figure 3).

2.4 Positioning Hybrid Design Systems

CADSYN, JULIA and ARCHIE demonstrate how decomposition and case-based reasoning techniques have been combined in knowledge-based design systems to date. Figure 3 illustrates our view of the relative position of these systems in the context of a *knowledge completeness* spectrum.

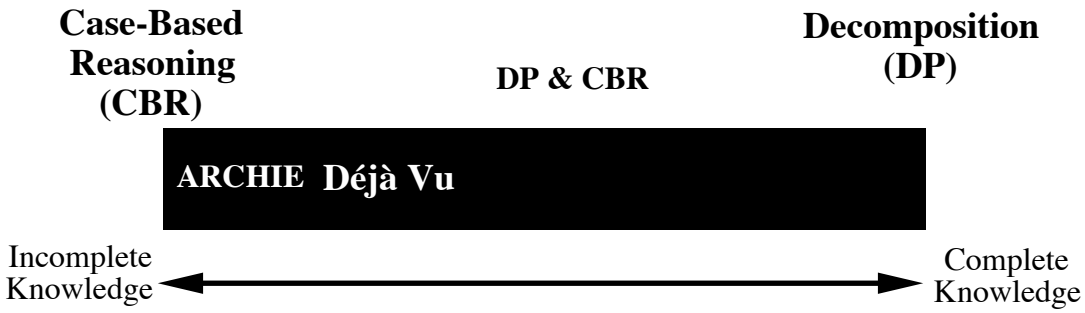

Figure 3. Hybrid decomposition and case-based reasoning systems.

A number of points can be made about this categorisation. Firstly, we consider the systems discussed to address weak domain tasks. This is true for most design applications, as design is generally considered to be an ill-structured problem (Simon, 1973). Conceptual design in domains such as architecture is further hindered by the availability of only incomplete knowledge and decomposition is carried out using a combination of heuristics and user intervention. In other weak domains such as structural design, typified by CADSYN, domain knowledge is more complete and therefore allows fixed generalised decomposition strategies to be formulated and applied. These hybrid systems do address some of the shortcomings associated with CBR by allowing complex design problems to be represented as a series of smaller more manageable cases; this is possible assuming that decomposition techniques are powerful enough to effectively decompose any presented problems. Nevertheless, the disadvantages associated with decomposition continue to persist, particularly with domains where generalised decomposition techniques are not effective and cannot be easily applied. This is especially true in context sensitive domains where interdependencies exist between the various components and functions.

In this paper, we propose a new design synthesis model called Déjà Vu, which can be conceptually described as hierarchical case-based design. This model captures specific decomposition strategies as (decomposition) cases, and is therefore particularly suitable for applying decomposition techniques in context sensitive domains where only incomplete knowledge is available. Thus this model can be viewed as occupying a niche for design problems (see figure 3) where domain decomposition knowledge is incomplete and decomposition strategies are both difficult to generalise and apply.

3 Two Systems for Software Design

In this section two software design systems are briefly introduced, Déjà Vu^{PCS} (Smyth & Cunningham, 1992) and Déjà Vu^{GUI}. Both have been developed in terms of the Déjà Vu model and examples from each will be used to illustrate a more detailed discussion of this model in Section 4. For now let us briefly examine their application domains and look at some characteristics that render traditional methods of design inappropriate. In Section 4 we will argue that the Déjà Vu model has been developed with domains exhibiting these problematic characteristics in mind.

3.1 Déjà Vu^{PCS} : Plant-Control Software Design

The design of Plant-Control software (PCS) is the application domain of Déjà Vu^{PCS}. Plant-control software is concerned with controlling robotic vehicles within a factory plant environment. For example, Figure 4, illustrates one particular class of problems that Déjà Vu^{PCS} is designed to handle; vehicles (coil-cars) are controlled during the loading and unloading of, in this case, spools or coils of steel in a steel-mill.

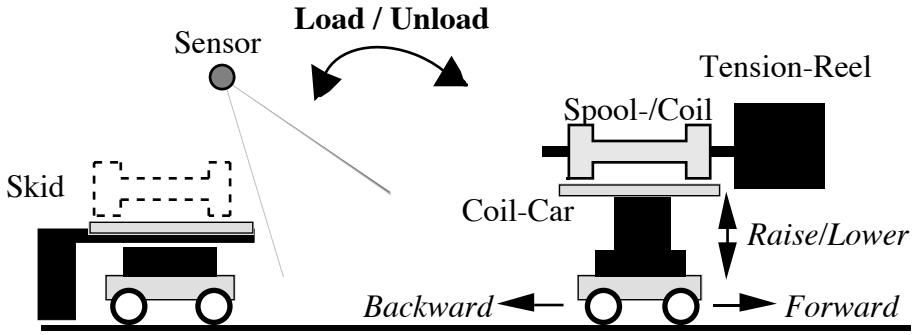

Figure 4. Plant-Control Domain

An important consideration in automating the design of plant-control software is that the formality of the software domain[1] is offset by the presence of unpredictable features in the domain, features that lack any identifiable causal model. For example, plant-control software is highly dependent of certain timing considerations that are known to the designer only from experience. Furthermore, plant-control design is fraught with interactions problems between domain elements making pure decomposition approaches impractical; generalised (plan-based) decomposition strategies cannot be formulated without running into difficulties during recomposition.

3.2 Déjà Vu^{GUI} : Graphical User Interface Software Design

Déjà Vu^{GUI} is a prototype system for the automated design of MOTIF-based graphical user interfaces. MOTIF is one of many GUI development languages and runs under the X window

[1]We acknowledge that many software design tasks have the advantage of a strong domain theory with a well defined semantics.

system. GUI components, or *widgets* are defined in terms of a set of generic types (buttons, text areas, lists etc.) and are combined to provide a graphical context for the manipulation of data (Berlage, 1991). The design of any particular interface is driven by the data set in question and the types of operations that are to be performed on it. For example, to design an interface allowing the manipulation of some list of data, widgets must be created that facilitate the display of the data list, as well as widgets to which functions can be attached that allow elements to be added or deleted to and from the list. In general, target interfaces are more complex with many different types of data sets and a variety of manipulation functions.

One of the most important issues in graphical user interface design relates to the "look & feel" of the interface. The aim of any graphical interface is to provide a novice user with *intuitive* access to a range of complex functions. Therefore, during the design of an interface it is important that the designer considers what the user will find intuitive, useful, and aesthetically pleasing. Thus, like plant-control software design, GUI design can be classified as a weak theory domain and so suggests the use of a case-based approach. Again, the highly interactive nature of the GUI domain causes problems for conventional design methods which further suggest case-based methods.

4 Déjà Vu: A Model of Hierarchical, Case-Based Design

The objective of this section is to advance, in the Déjà Vu architecture, a significant new model of design synthesis. The model, which integrates decompositional design with case-based design, addresses the shortcomings of other design methodologies. In describing this model we will first outline its design knowledge requirements. Secondly, the case-based design synthesis process will be examined. And finally it will be argued that this approach to design exhibits a number of advantages, over existing methodologies, that may be exploited in future design systems.

4.1 Case-Based Design Knowledge

The Déjà Vu model integrates two distinct types of design knowledge: specific design episodes (*design cases*) and specific decomposition episodes (*decomposition cases*). Design cases capture actual design solutions as well as some descriptive representation of the behaviour, function, and structure of these solutions[2]. The description of a case is a feature-based structure that is used during retrieval to determine the applicability of a case to a new target situation. A similar representation scheme is used for decomposition cases. These capture more abstract information but are nevertheless firmly grounded in specific design episodes, and again each case contains a solution component and some descriptive component. In fact by organising experiential design knowledge in this way, complex designs are represented as design hierarchies (Figure 5).

---

[2]Cases may also include causal information related to the design in question and perhaps may even contain the design rational (the justifications and evaluations of design steps) that lead to the development of the particular design; however this approach was not taken in our prototypes.

Each design hierarchy captures a specific complex design at a number of levels of abstraction. Design cases are the terminal nodes of these hierarchies as these contain the most specific design details. Decomposition cases are the internal nodes, with the level of abstraction increasing as one moves up through the hierarchy. A design hierarchy captures the decomposition structure of a

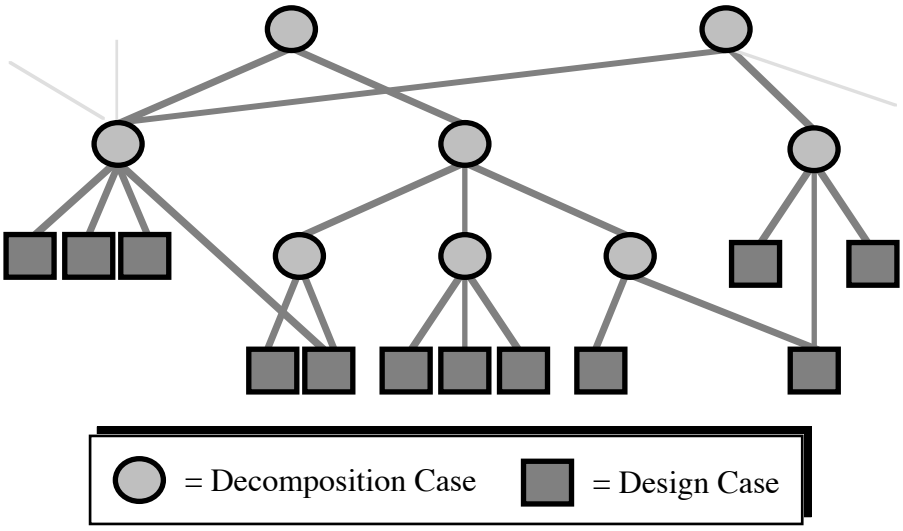

Figure 5. Design hierarchies.

particular design, from the most abstract level, the specification, to the most specific level, the actual design solution.

For example, in Déjà Vu^{PCS} design cases are actual plant-control software solutions, while decomposition cases correspond to the "rough plans" of more complex design solutions. Figure 6. depicts a portion of a design hierarchy. The top-level node, a decomposition case whose solution is a rough plan for a design that uses a particular vehicle (coil-car*1) to unload a coil from a tension-reel and load it onto a skid. Part of this plan (as highlighted) is fulfilled by a design that deposits the coil on the vehicle from the tension-reel. Another decomposition case is used to further break down this deposition task, the components of which are solved by two design cases; the design case for aligning the coil-car with the tension-reel is shown, and after aligning the vehicle the second design case (not shown) releases the coil from the tension-reel onto the coil-car.

Déjà Vu^{GUI}'s design hierarchies are similar in nature, but a greater variety of decomposition perspectives are utilised. Whereas the above functional decomposition is adequate for plant-control software, GUI software can be decomposed in a number of ways. For example, by the structure of the data to be manipulated, or by the manipulation functions themselves, or by the desired interface layout and geometry.

4.2 Case-Based Design Synthesis

During a typical design session an initial specification is transformed, through a set of intermediate design abstractions, into the actual design solution; solutions are developed in a top-down, hierarchical fashion. An iterative case-based reasoning cycle controls the processes at each level of synthesis. The basic cycle is the retrieval and adaptation of a case. The fundamental issues that must be resolved include, the retrieval of a suitable case and the adaptation of this case for the current design context. In addition, after solving the component tasks of a given specification the integration (or recomposition) of the resulting solution components must be addressed; it is often the case that these solutions will need further modification if they are to provide a complete solution to the target problem.

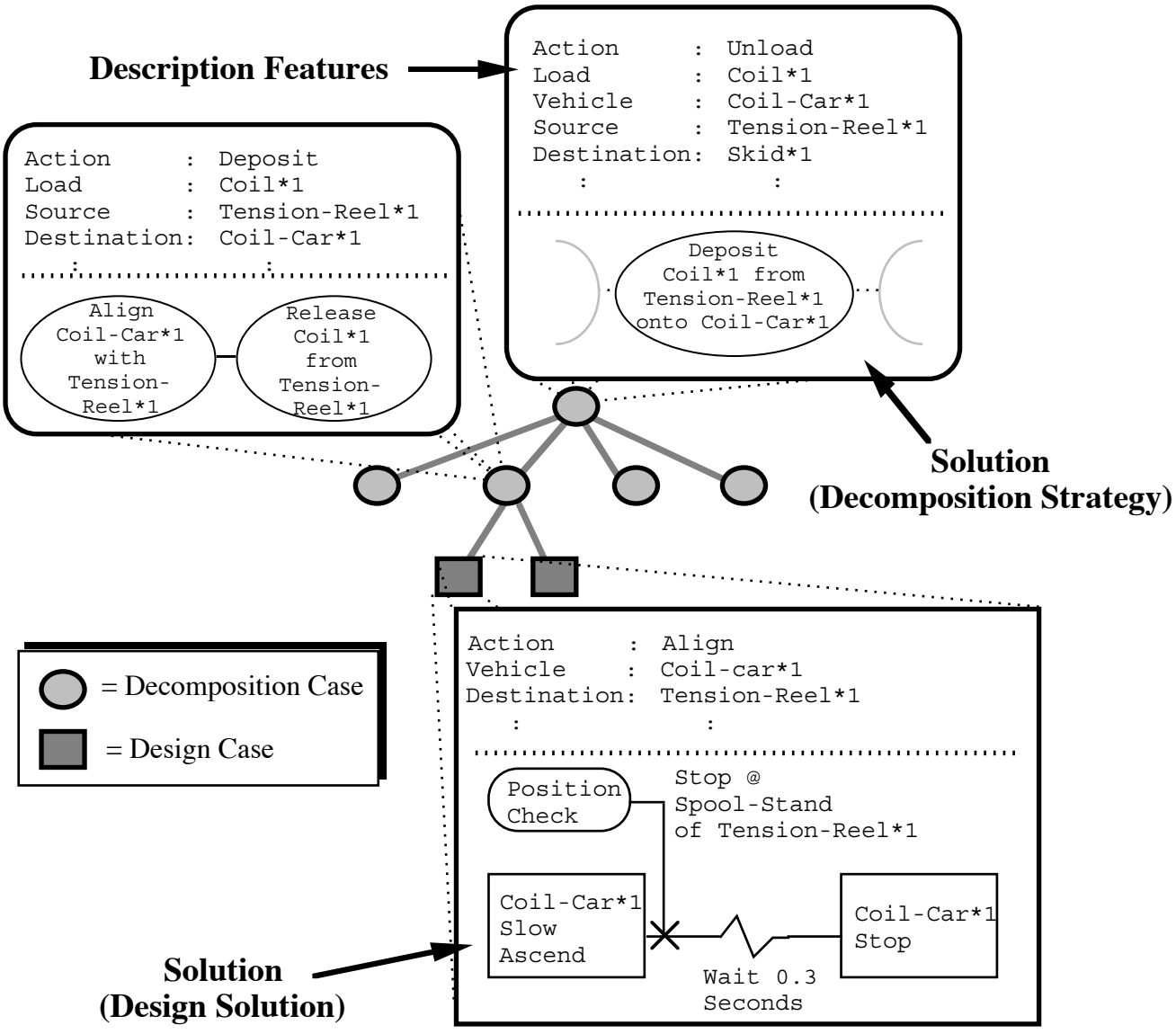

Figure 6. A plant-control design hierarchy.

4.2.1 Case Retrieval

During a design session the retrieval of a design case means that a detailed design solution can be found by adapting its solution. However, in contrast to traditional case-based approaches, a decomposition case may be retrieved. This indicates that the specification was too complex for design case to be of use, but that it may be decomposed into simpler sub-specifications; the solution of the decomposition case, when adapted, being used as the appropriate decomposition strategy for the specification at hand. In this way the decomposition process is fully integrated into the design model, and is used if and only if it is really necessary.

The retrieval of a suitable case is critical to the success of case-based reasoning (Cain, Pazzani and Silverstein, 1991; Kolodner, 1989). In design problem solving the salient features for determining a good case depend very much on the target context and so cannot be predetermined. This problem is compounded when using complex design cases, part or all of which could be suitable to a variety of target situations; predetermining these different relevant contexts is clearly not a viable option. In addition, the importance of each feature as part of a fixed similarity metric cannot be predetermined either (Alberts, Wognum and Mars, 1992). Déjà Vu employs a novel approach to similarity measurement that does not rely on a fixed similarity metric or predetermine feature salience. Very briefly, the similarity of a candidate with respect to the target is measured on the basis of the adaptation requirements of the candidate. That is, the modifications that must be made to the candidate are predicted during retrieval by analysing the mappings generated between the candidate and target. Adaptation knowledge (knowledge about possible adaptations) is used by the retrieval process to make these predictions. Thus feature weightings reflect their "adaptability" with respect to the current target and are not based on some predetermined estimate of semantic salience. Although beyond the scope of this work this retrieval approach offers many advantages, improving the accuracy and flexibility of the case-based approach (see Smyth and Keane, 1993).

4.2.2 Case Adaptation

The retrieved case represents a specific design grounded in a specific design context. The adaptation task is concerned with transferring this design to the current, target design context. During adaptation the important issues are the identification of those features of the retrieved design that can be transferred unchanged and the identification of those features that cannot. In the Déjà Vu model both design cases and decomposition cases can be modified.

It is our experience that, in many domains, design adaptation knowledge can be most readily captured as a set of task specific adaptation models. Each model or *specialist* is charged with modifying a particular part of a retrieved design corresponding to some feature of the design. So in this way design adaptation knowledge can be organised as feature-to-feature adaptation models or specialists. For example, Déjà Vu^{GUI} contains adaptation specialists designed to make modifications to the graphical elements (widgets) of an interface. To design an interface for displaying a body of

text, the system might retrieve a design for displaying a list of text elements. During adaptation the list widget of the retrieved design must be changed to a text widget required in the target. Therefore one type of transformation specialist is concerned with transforming list widgets into text widgets.

Unfortunately, design domains are usually highly interactive in nature; that is, dependencies exists between elements of design domains and consequently between elements of design solutions. These dependencies constrain the modifications that can be made on a particular design, and so the adaptation task can be viewed from the perspective of a constraint satisfaction problem; retrieved designs can only be transformed in a manner that preserves the integrity of the design with respect to the target context. The "blind" application of adaptation specialists is bound to introduce conflicts into the design that must be resolved; a specialist that alters one feature of a design may effect another feature. To this end, we propose a second type of adaptation model, *strategies*. Each strategy corresponds to some generalised conflict configuration that can occur; Hammond (Hammond, 1989) outlines a number of such failure configurations that are applicable to a range of planning and design tasks. Each strategy also contains a number of repair methods that can resolve the conflict in question. Returning to the GUI design domain one common conflict configuration occurs when the result of some specialist's action blocks the pre-condition of some solution feature. For instance, transforming a list widget into a text widget may cause a failure because the list accessor functions cannot be applied to text widgets; that is, a precondition of the list accessor functions is that they can only be used on a list widget. One repair that the strategy might suggest is to change the blocked feature to one with a compatible pre-condition; that is, change the list accessors to text accessors (for further details with particular emphasis on the adaptation of plant control software designs see Smyth and Keane, 1993).

4.2.3 Case Integration

During integration the generated solutions must be combined to provide a coherent solution spanning the current specification. In fact, integration (or recomposition) is more a consequence of the decomposition process than an externally motivated design stage in its own right; the decomposition of a problem into sub-problems implicitly assumes that recomposition will occur. Its recognition here reflects the fact that, for design tasks it is by no means a simple process. In general dependencies will exist between design components and these dependencies must be represented and dealt with during integration. Déjà Vu uses knowledge, similar in nature to its adaptation knowledge, to carry out the task of integration. Integration models act on solution components in an effort to combine them into a conflict free design.

4.3 Beneficial Implications

This model has a number of advantages over other approaches to design, including existing hybrid models. Domain applicability is certainly improved and in particular Déjà Vu facilitates design in a relatively untouched class of design tasks. In addition the model provides tractable answers to many of the problems associated with decompositional design methods, including the selection and

application of appropriate decomposition strategies. Learning is also facilitated, not just of new design cases but also of new decomposition strategies.

4.3.1 Domain Applicability

From the perspective of "real-world" design one of the most damaging disadvantages of decompositional design is its restricted applicability. Decompositional design methods are unlikely to facilitate design in domains with a weak domain theory or an incomplete design knowledge-base. In contrast, an important advantage of case-based methods is their use of case knowledge, filling gaps in domain theories and knowledge, and hence facilitating design in weak domains or with incomplete knowledge.

In weak domains it may not possible to develop plan-based decomposition strategies as many systems such as CADSYN propose. In contrast Déjà Vu's reliance on case-based strategies is a more reasonable request from the knowledge acquisition viewpoint. The needed strategies can be abstracted from design episodes, even in weak domains, and they do not need to be generally applicable.

Decompositional methods are also plagued by interaction problems in context sensitive domains where a high degree of dependencies exist between domain elements; problems cannot be decomposed, using generalised strategies, into independent sub-problems and so recomposition becomes problematic. Cases on the other hand constitute "good" and correct design solutions. Conflicting dependencies between design variables have been satisfied within the scope of a particular case. The hope is that in applying these cases to similar target problems, adaptation will not result in the emergence of any new conflicts. That is, once cases are sufficiently similar to the target they should lead to the target design in a fairly straightforward fashion. Consequently, during decomposition the prevention of conflicts between the decomposed structures leads to a far simpler integration stage.

So, as an integration of case-based and decompositional design, Déjà Vu benefits from greater domain applicability. Of course it can be used in strong domains with complete knowledge (to enhance performance). But more importantly it can be used in weak theory domains or in systems with incomplete domain knowledge. And in particular, it is best utilised for weak theory domains *with* incomplete knowledge where the representation of decomposition strategies is more readily achieved as episodic structures.

4.3.2 Locating and Applying Decomposition Strategies

There are many difficulties associated with decomposition in all but the simplest of domains; deciding what has to be decomposed, how the decomposition is to be carried out, how to find the appropriate decomposition strategy, and how to apply this strategy.

A common approach to decomposition is to use a fixed decomposition plan. However in many domains this is not feasible; for example, it has been argued that in weak domains such as design, there is often no causal basis for decomposition and so general decomposition strategies cannot be easily formulated. In addition, fixing a small number of general strategies seriously limits the capabilities of the approach; in general a wide array of strategies might be desirable, the choice of one in particular depending on the current target situation. The answer provided by Déjà Vu is to make use of design experience as a framework for decomposition. This affords Déjà Vu with a tractable mechanism for selecting the appropriate strategy. A large number of decomposition cases can be maintained within the case-base and case retrieval methods provide a means for selecting the appropriate strategy on the basis of their suitability to the target problem

In addition the use of plan-based strategies may cause problems when they are applied to specifications for which they are not quite suited. Déjà Vu has the facility to adapt decomposition cases, and hence decomposition strategies, so that they fit the target problem in a precise manner. This results in less problems during the application of strategies, since they are tailored to the specifics of the target problem.

A further advantage that Déjà Vu has to offer is that during design a number of strategies may be combined at varying levels of abstraction. Again this is a benefit that many traditional approaches don't offer. Such approaches are restricted to using one particular strategy during the decomposition of a given problem. At each stage of design Déjà Vu can choose a decomposition strategy that is best suited to the task at hand. This is important since as a design solution is elaborated on, previously hidden details will emerge. A premature commitment to a particular decomposition strategy without reference to these details may prove problematic.

4.3.3 Learning

A major advantage of the case-based reasoning approach is that new design knowledge can be incrementally learned during problem solving. In Déjà Vu this facility extends, not only to the accumulation of new design cases, but also new decomposition strategies. That is, as target problems are solved, new strategies will be devised and learned that are specifically tuned to these problem situations. Newly developed solutions are packaged as a case structures and indexed into the case-base where they are available for future design sessions.. In using hierarchical case-based reasoning, complex design hierarchies of interconnecting cases will be learned during the design process, rather than just a single case as in conventional CBR systems.

5 Conclusions

This paper has been concerned with investigating hybrid approaches to design synthesis and examining their shortcomings. In particular, with these shortcomings in mind, a new model of design has been advanced. This model, Déjà Vu, has tackled the problems of traditional design

methods in a direct fashion, and provides tractable answers to many open questions. The result is a model of design that relies on case-type knowledge to decompose and generate complex design solutions. The technique can be applied to a wider range of domains than other methods. In particular, the use of decomposition cases provides answers to many of the problems with decompositional methods for design tasks with a weak domain theory and incomplete knowledge.

Déjà Vu is substantially different from other hybrid models of design synthesis, most notably in its approach to decomposition. CADSYN for example, uses an "either-or" approach to design, choosing either a generalised decomposition strategy or a case-based approach to solve the current task. JULIA, on the other hand, does use experience-based strategies, but these represent fixed decomposition plans. ARCHIE is also different, using a set of weak decomposition heuristics to suggest possible ways of dividing complex situations. In contrast to these systems, Déjà Vu uses cases to provide decomposition strategies which can be more readily acquired than generalised plan-based strategies, in weak theory domains. Furthermore, decomposition cases are not fixed and can be adapted to fit the target situation. In addition, Déjà Vu is capable of accumulating new design knowledge during problem solving. The learning of new design hierarchies means that, not only are actual design cases learned (as in many CBR systems), but new decomposition strategies are also acquired. The overall result, is a more complete architecture in which the CBR paradigm guides and controls all stages of the synthesis process.

Finally, this model of design has been evaluated through two implementations of software design systems. In both, a complex design task was addressed, one that was based on a domain without a strong causal theory and where complete knowledge was not available. Preliminary results have been encouraging, and future work will continue to examine the applicability of the model and extend its design capabilities to accommodate more and more complex design synthesis tasks.

References

Alberts, L.K., Wognum, P.M. and Mars, N.J.I. (1992). Structuring Design Knowledge on the Basis of Generic Components. In *Artificial Intelligence and Design '92*, Netherlands: Kluwer Academic Publishers, pp 639-656.

Berlage, T. (1991). OSF/Motif: Concepts and Programming. Addison-Wesley.

Cain T., Pazzani M.J., and Silverstein, G. (1991). Using Domain Knowledge to Influence Similarity Judgements. In *Proceedings of the Case-Based Reasoning Workshop*. Morgan Kaufmann, pp. 191-198.

Chandrasekaran, B. (1990). Design Problem Solving: A Task Analysis. *AI Magazine,* Winter 1990, Vol. 11, No. 4, 59-71.

Domeshek, E. and Kolodner, J. (1993). Using Points in Large Cases. *AI Engineering Design Analysis and Manufacturing Journal*, Vol 7, No. 2, 87-96.

Gero, J.S. (1990). Design Prototypes: A Knowledge Representation Schema for Design. *AI Magazine,* Winter 1990, Vol. 11, No. 4, 26-36.

Hammond, K.J. (1989). *Case-Based Planning*, Academic Press.

Hennessy, D. and Hinkle, D. (1992). Applying Case-based reasoniong to Autoclave Loading. *IEEE Expert*, 7(5), 21-26.

Hinrichs, T. (1991). *Problem Solving in Open Worlds: A Case Study in Design,*. Hillsdale, New Jersey: Lawrence Erlbaum.

Hua, K. and Faltings, B. (1993) Exploring Case-based Building Design - CADRE. *AI Engineering Design Analysis and Manufacturing Journal*, Vol 7, No. 2, 135-143.

Kolodner, J. (1989). Judging Which is the "Best" Case for a Case-Based Reasoner. In *Proceedings of the Case-Based Reasoning Workshop*. Morgan Kaufmann , pp 77 - 84.

Kolodner, J. (1991). Improving Human Decision Making through Case-Based Decision Aiding. *AI Magazine*. Summer 199, 52-68

Maher, M.L. (1990). Process Models for Design Synthesis. *AI Magazine,* Winter 1990, Vol. 11, No. 4, 49-58.

Maher, M.L. and Zhang, D.M. (1993). CADSYN: A Case-based Design Process Model. *AI Engineering Design Analysis and Manufacturing Journal*, Vol 7, No. 2, 97-110.

Pu, P. (1993). Issues in Case-based Design Systems. *AI Engineering Design Analysis and Manufacturing Journal*, Vol 7, No. 2, 79-85.

Redmond, M. (1992). Learning by observing and understanding expert problem solving. Ph.D Dissertation, Georgia Institute of Technology, College of Computing. TRGIT-CC-92/43.

Reisbeck, C. K., and Schank, R. C. (1989). *Inside Case-Based Reasoning*. Lawrence Erlbaum Associates.

Simon, H.A. (1973). The structure of ill structured problems. *Artificial Intelligence*, 4 181-201

Smyth, B. and Cunningham P. (1992). Déjà Vu: A Hierarchical Case-Based Reasoning System for Software Design. In *Proceedings of the 10th European Conference on Artificial Intelligence*. Wiley, pp 587-589.

Smyth, B. and Keane, M.T. (1993). Retrieving Adaptable Cases: The Role of Adaptation Knowledge in Case Retrieval. Working

papers of *First European Workshop on Case-Based Reasoning*, Kaiserslautern, Germany.

Velosa, M. and Carbonell, J. (1991) Automating case generation, storage, and retrieval in PRODIGY. In *the Proceedings of the First International Workshop on MultiStrategy Learning*. George Mason University, Fairfax, VA, pp 363-377.

Takeda, H., Veerkamp. P., Tomiyama, T. and Yoshikawa, H. (1990) Modelling Design Processes. *AI Magazine,* Winter 1990, Vol. 11, No. 4, 37-48.