J. S. Gero and F. Sudweeks (eds), Artificial Intelligence in Design '96, 269-288.

CONSTRAINT-BASED RETRIEVAL OF ENGINEERING DESIGN CASES

Context as constraints

TANER BILGIC AND MARK S. FOX
Enterprise Integration Laboratory
University of Toronto
Toronto, Ontario M5S 1A4, Canada

Abstract. The case-based retrieval is frequently reported as a valuable tool for engineering design. We discuss similarity based retrieval in the engineering design domain when the context is given as a set of constraints. This approach comprises the lowest level with which we support case-based retrieval from our Integrated Knowledge-Base. The characterization of the retrieval process yields a robust compliance measure and a similarity measure for the cases in a given context. The problematic concept of context is taken up front by making it an explicit part of the query.

1. Introduction

Engineering design involves usage of domain specific technical knowledge together with creative problem solving skills to come up with a properly functioning artifact that complies with a set of requirements — performance goals, physical constraints, etc. It is a creative process that relies heavily upon associations to past experiences and similar designs (Goel, 1994). Consequently case-based reasoning (CBR) (Kolodner, 1993) has been the focus of the design research community (Maher *et al.*, 1995).

Our interest in case-based design arises from the design of complex artifacts for the aerospace industry where design is requirements driven. It is initiated with a "high level" set of requirements, i.e., goals, functional requirements and constraints, that trigger the retrieval of one or more "high level" design cases. These cases are used by engineers to guide their construction of an abstract design that in turn provides a set of requirements for the next design level. Design is therefore a process of successive refinement, when each level iterates between requirements specification/analysis, design case retrieval and design decision-making.

Indexing, case retrieval and case modification are key issues in case-based design. In many case-based reasoning systems, case retrieval is performed based on the *similarity* between the new problem *context* and cases represented in the

case memory. An *indexing* scheme which defines the situations under which the new context is similar to the ones in the case-base drives the retrieval process. A case-based retrieval system is effective to the extent its indexing scheme covers *all possible contexts* since similarity is known to change from context to context.

Our work focuses on the indexing/case retrieval problem and leaves the problem of adaptation to the engineer. In particular, given that design is requirements driven, we are interested in how requirements, i.e., goals and constraints, can be used to dynamically retrieve relevant cases from a case library, and how cases in the library should be represented to support this style of dynamic indexing.

The rest of the paper is organized as follows: we review the relevant literature in Section 2 and then propose a view of the design process that is consistent with Fox and Salustri (1994) for one-off, high-tech artifacts. We then discuss what needs to be represented to support this particular view of the design for the *whole design life cycle* in Section 3. Particularly, we mention the framework we are working in, which is a broad scope project to support the concurrent and collaborative engineering design projects using knowledge-based technologies. In Section 4 we elaborate on the requirement-driven retrieval. Our retrieval strategy employs a dynamic indexing mechanism that is based on a compliance measure and resolves the problem of context dependency by providing an explicit representation for the context as a set of constraints. We also define a similarity measure between cases and briefly discuss the properties of the measures defined relevant to case-based retrieval. In Section 5, we give the implementation details and a small example to illustrate the system. We conclude the paper with a summary and further research directions.

2. Previous Related Research

Serrano and Gossard (1988) discuss a constraint-based approach to conceptual design. They build on Serrano's earlier work on constraint management in the context of computational design. Their constraint representation is parameters on nodes and constraints on the arcs of a graph. They discuss graph theoretic methods to handle constraints efficiently.

Sycara and Navinchandra (1992) consider retrieval strategies in a case-based design system. Their representation not only includes physical attributes but *function* and *behaviour* as well. They represent the behaviour as an influence graph where the nodes correspond to parts and the arcs to causal relations between them. The input to their case-based retrieval system is a similar graph which depicts the new design situation. The input is matched to other graphs or parts of graphs stored in the case-base.

Nakatani *et al.* (1992) describe a case-based engineering design support system called SUPPORT which is an interactive system for supporting various phases of engineering design. Their case-based retrieval module uses a three-level repres-

entation based on features, functions, and parts hierarchies. They use a constraint-based search to select parts from line-ups.

Maher and Zhang (1993) represent design cases using two indexes: design problem specifications and design solutions. They construct hierarchies of design cases in this manner. Retrieval is done by finding the closest match for given specifications. If a match does not occur at one level in the hierarchy the process is repeated for each of the lower levels.

Wood and Agogino (1993) discuss an architecture to support case-based conceptual design. Their architecture relies heavily on the emerging Internet protocols like WWW and WAIS. They propose to store the design cases in several different multimedia formats (hypertext, CAD drawings, audio, video etc.) and then to search the case-base as guided by the design engineer.

Domeshek *et al.* (1994) discuss MIDAS (a Memory for Initial Design of Aircraft Subsystems). The authors use a repository of *design stories* and discuss ways of creating and indexing those stories. They state that the most developed part of CBR technology is the retrieval. However, the major challenge in the retrieval is building a comprehensive *indexing vocabulary*. They suggest that creating a design story requires two types of information: *presentation* and *connections*.

Maher and Balachandran (1994) explicitly mention the *iterative* nature of the case-based retrieval in the engineering design process. They model the retrieval process as *exploration* rather than a one-shot search. They represent only function, behaviour, and structure for case-based retrieval and propose two index elaboration methods for iterative retrieval.

Kumar and Krishnamoorty (1995) argue that the indexing process is highly *context dependent* and must be carried out for each domain separately.

In the case-based design systems mentioned above the retrieval is implemented as a memory search task and the system's ability is directly proportional to the "richness" of the indexing scheme. One has to foresee and provide indices for most of the query contexts that may arise (i.e. one should be able to abstract apples and oranges as similar in the context of edible items but be able to differentiate between them in the context of fruits). However, we observed that in the process of case-based retrieval:

- a person begins an information interaction with only a vague understanding of the design problem,
- her knowledge, constraints, and goals change over time.

This tends to suggest that, *(i)* case-base retrieval should be *iterative*, and *(ii)* instead of trying to foresee the context in which the retrieval is to be performed, the indexing mechanism has to be *dynamic* and similarity of one case to the context has to be computed on-the-fly.

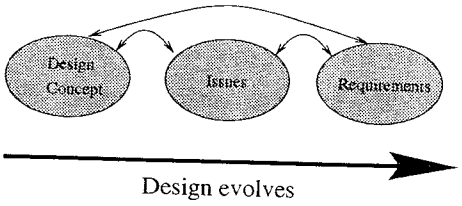

Figure 1. A particular view of design that emphasizes the iterative nature of the process.

3. Case Representation

At the heart of the case-based reasoning paradigm lies case *representation, indexing* and *matching*.

For engineering design purposes, comprehensive case libraries can be built in-house or distributed libraries available on the Internet can be used (e.g. PARTNET (http://part.net/), INDUSTRY NET (http://www.industry.net/)).

We have found the following three concepts crucial in the process of designing one-off, high-tech artifacts (cf. Figure 1):

- *Concepts*: The first thing that the design engineers come up with are concepts which provide a solution to the (design) problem at hand. These can either be competing or complementary design alternatives (e.g. Let's build a remote manipulator arm with six joints to solve the problem).
- *Issues*: Then the design team raises issues and deals with them until a compromise closure is attained. The issues can be from anywhere within the life cycle of the design. There can be issues of risk, cost, schedule, control, stability, manufacturability, quality, fit, form, function etc. (e.g. How are we going to stabilize the system? What is the power consumption of a particular joint? How did we handle the manufacturability issues for past designs? etc.)
- *Requirements*: The design objectives together with design concepts and issues yield functional, structural and performance requirements. Design is satisfaction of these requirements. (e.g. The remote manipulator should have six degrees of freedom. The remote manipulator should be able to handle payloads upto 1000 kg. The shoulder joint should provide inclination and travel to the arm, while elbow and wrist joints should provide travel to the end effector etc.). The requirements are iteratively decomposed and elaborated on (e.g. Providing travel decomposes to provide rotation for all joints.)

Although there seems to be a natural hierarchy between concepts, issues, and requirements, one should bear in mind that the concurrent engineering practices allow for a concept's requirements to be refined, while issues arising from another concept are investigated at the same time.

The knowledge to support the *full life cycle* of this particular view of the design is captured in the TOronto Virtual Enterprise (TOVE) (Fox, 1992; Fox *et al.*, 1993; Fox and Gruninger, 1994). Particularly TOVE *(i)* provides a shared terminology for the enterprise that each agent can jointly understand and use, *(ii)* defines the meaning of each term in a precise and as unambiguous manner as possible, *(iii)* implements the semantics in a set of axioms that will enable TOVE to perform *deductive* query processing to answer "common sense" questions about the enterprise. TOVE represents both generic concepts (time, causality, activity, and constraints) as well as enterprise specific entities (products, requirements, activities, organisation, cost, and quality).

The framework we operate in is a complex engineering design project which requires the services of many engineers and their efficient collaboration. Reusing existing designs, which we address in this paper, is *one of* the objectives of the system we are developing.

The product, parameter, requirement, constraint and function representations in TOVE are closely related to the case-based retrieval of engineering design cases (cf. Figure 2) since TOVE provides a sophisticated representation of the design. In this paper, we do not utilize TOVE's activity, organisation, and cost ontologies.

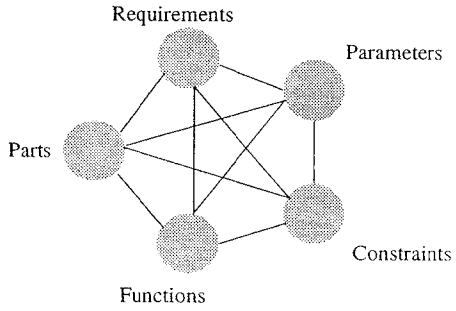

Figure 2. Representations in TOVE that are used to represent engineering design cases.

We adopt the *repository* view of TOVE and consider it as a repository of knowledge relevant for engineering design. We use the term "Integrated Knowledge-Base" to refer to the repository. This view induces a "monolithic" representation of design cases (as contrasted with the "snippet" representation) (Kolodner, 1993, Section 5.4.1) from which sub-cases need to be extracted. We make that extraction via functions. The functional representation is the higher level indexing mechanism of design cases and it complements the dynamic indexing based on requirements that will be discussed in detail in Section 4.

Furthermore, we assume that the case-based retrieval process is *iterative*, a view shared by others (Domeshek *et al.*, 1994; Maher and Balachandran, 1994),

which is consistent with the V-model of systems engineering view (Fox and Salustri, 1994). The user is not confined to any level of representation at any time and can ask a question of arbitrary generality at will.

We briefly identify what needs to be represented to support case-based retrieval for the particular view of design put forth in this section and then discuss the details of requirement-driven retrieval in Section 4.

3.1. DESIGN CONCEPTS

In order to reason about *design concepts* one has to consider notions of:

- *fit*: how do the parts of the design fit together?
- *form*: the structure of the design as captured by the parts hierarchy.
- *function*: the intended behaviour of the artifact that is designed as might be found in a functional classification of parts.
- *behaviour*: the causal relationships between different parts of the artifact.
- *working principle* hydraulic, electro-mechanic etc.

Therefore any case-based design tool should be able to represent and *reason on* the above items. TOVE provides explicit representations for fit, form, and function. Behaviour and working principle is simply implemented as a classification of design parameters. A simplified schematic representation of an engineering design case is shown in Figure 3.

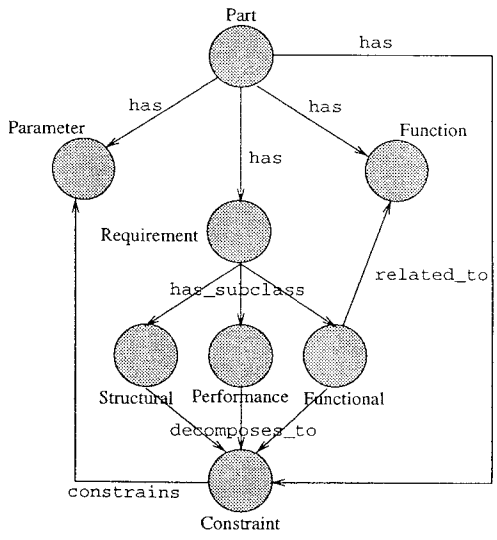

Figure 3. Representation of engineering design cases using TOVE.

Furthermore, each category has its internal classification (e.g. allocated parameters, estimated parameters, actual parameters, basic functions, non-basic functions, unary functions, binary functions etc.).

We briefly mention the representation of functions since that is usually the starting point of case-based retrieval in engineering design domains (Goel and Chandraskaran, 1989; Sycara and Navinchandra, 1989). The functional representation is as described in (Pahl and Beitz, 1988) who identifies five *generally valid* functions (change, vary, connect, channel, and store). These functions take energy, materials, and signals as their arguments. We distinguish user-defined non-basic functions from the generally valid basic functions (e.g. the non-basic function "provide rotation" is related to joints of the manipulator arm as well as to the basic function "connect(energy,matter)"). Generally valid functions are useful when the system cannot retrieve any prototype for a given non-basic function. In that case, if there is another level, the retrieval algorithm moves one level up in the non-basic function hierarchy. If there is still not one item retrieved, the algorithm moves to the related generally valid basic function and retrieves prototypes related to that function with the hope of retrieving something relevant. The retrieved cases are pruned by the designer with respect to their relevance.

3.2. ISSUES

In a sense, issues define the solution context for which more detailed questions can be asked. What issues have been dealt with in the previous cases and how they were dealt with is an important piece of knowledge. Eventually one can discover recurring issues in "similar" situations (e.g. how risk was reduced in a previous project when stabilization issue was raised can be a valuable piece of information in the current context.)

Hence a case-based design tool should be able to represent and reason on issues. Issue-based retrieval is used to retrieve those cases in which the same issues have been dealt with. In the current prototype, issues are simply indexed by their names and no further abstraction is available. The extension of issue-management and categorizing issues lie in out further research agenda.

3.3. REQUIREMENTS

Design is highly requirement-driven in the engineering design domain we are concentrating on. Usually the customer comes in with a set of higher level requirements which get decomposed and elaborated on during the design process and find their way to every detail of the design, usually in the form of a constraint on design parameters.

TOVE supports the requirement management process in various ways. The requirements are elaborated on and decomposed into sub-requirements until they are represented (internally) as *constraints* in the knowledge network (cf. Figure 3).

We do not impose a way of managing requirements but provide a rich representation which can support many requirement management schemes.

We differentiate between functional, structural, and performance requirements. Functional requirements dictate "how" to achieve a desired behaviour whereas performance requirements dictate "how well" a behaviour must be achieved. Structural requirements are usually physical laws that are required for the design to achieve its goals.

Requirements are the basic means to describe the design to our system. The higher level requirements (which are usually functional) retrieve candidate design prototypes which come with their own requirements and constraints. The designer modifies and prioritize the new requirements and continue retrieving in an iterative manner.

The requirement-driven retrieval is elaborated fully in Section 4. An example is given is Section 5.

4. A Characterization of the Requirement Driven Retrieval

In this section we outline the retrieval mechanism in the presence of constraints. The formal treatment in this section should not turn the reader off. What we are saying is really simple: when context is given as a set of constraints, individual cases comply with the context to the degree they satisfy the constraints. The case that satisfies most of the constraints (or more than a predetermined number of constraints) is retrieved. To yield more flexibility, the constraints can be weighed as to their importance, in which case our compliance measure is the weighted average of the number of constraints satisfied. Furthermore, when a case does not contain some attribute mentioned in the constraints, we propose to solve for it with the purpose of aiding the designer in *selecting* an appropriate design case. We do that by allowing to solve for multiple, weighted objectives.

The formalism will allow us to discuss the properties of the compliance measure and the similarity of cases which, we believe, are too important to be overlooked.

Basically we define individual *cases* with finite number of attribute-value pairs and a *retrieval context* with finite number of constraints on the attributes. Then a case satisfies the retrieval context to the degree it satisfies the constraints.

Formally, the situation is as follows: an individual case, S, is assumed to be comprised of a finite list of attribute-value pairs:

$$S = \{\langle a_1, v_1\rangle, \langle a_2, v_2\rangle, \cdots, \langle a_k, v_k\rangle\}.$$

Then the *case-base*, *CB*, is a finite collection of individual cases:

$$CB = \{S_1, S_2, \ldots, S_\ell\}.$$

The context of retrieval is explicitly defined by a set of constraints [1] :

$$X = \{\langle a_1, C_1\rangle, \langle a_2, C_2\rangle, \ldots, \langle a_{m'}, C_{m'}\rangle\}.$$

These can either be explicit constraints on the particular design or constraints related to functional or structural requirements as well as constraints from anywhere in the life cycle of the design.

We use the characteristic function, χ, to denote satisfaction of a constraint by a case: for any case S_j and constraint C_i:

$$\chi_i(S_j) \begin{cases} 1 & \text{if } S_j \text{ satisfies } C_i \\ 0 & \text{otherwise} \end{cases}$$

Then we define another relation for a case which satisfies a given *context*:

$$sat(S, X) \text{ iff } \forall C_i \in X, \exists S \in CB, \chi_i(S) = 1.$$

However, this is not flexible enough for retrieval purposes: a case either satisfies a context or not! We are interested in cases which *almost* satisfy the context as well. To achieve this flexibility we can define a measure which shows *how much* a case satisfies a given context:

$$\mu_X(S) = \frac{\sum_{i=1}^{m} \chi_i(S)}{m}.$$

Clearly, $\mu_X(S) \in [0, 1]$ with $\mu_X(S) = 0$ showing no compliance with the given context, $\mu_X(S) = 1$ denoting full compliance and $\mu_X(S) \in (0, 1)$ denoting partial compliance.

Consider the situation where one objective dominates all the others in the sense that if it is not fulfilled then the satisfaction of the rest is not that important (when an envelope objective is not satisfied it really may not matter whether a weight objective is fulfilled or not). This situation is typical in engineering design.

To be able to provide more flexibility to the user in terms of retrieval the constraints can be *weighed* by the user as to their importance.

Therefore the context, X, is now given as:

$$X = \{\langle a_1, w_1 C_1\rangle, \cdots, \langle a_{m'}, w_{m'} C_{m'}\rangle\}$$

[1] Since a single constraint can apply to several attributes the list contains more tuples than the number of constraints. We assume that there are m constraints but the context is given by m' attribute-constraint pairs where $m' \geq m$.

where w_i are the weights which denote importance of each constraint[2]. We assume that weights are positive real numbers. The situation is as depicted in Figure 4.

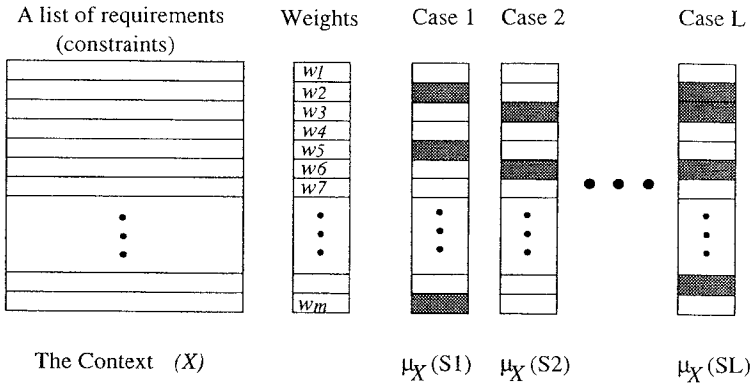

Figure 4. Constraint-based retrieval: case of weighted compliance measure. The shaded areas of the cases represent unconforming parameters.

The retrieval with weighted constraints is based on the extended compliance measure, $\mu_X(S)$, which is given as:

$$\mu_X(S) = \frac{\sum_{i=1}^{m} w_i \chi_i(S)}{\sum_{i=1}^{m} w_i}.$$

Some other properties of μ_X are as follows:

- Two $\mu_X(\cdot)$ values are *commensurate* as long as they denote the same context X. On the other hand, μ_X and μ_Y are incommensurate if the relation of X to Y is not known.
- $\mu_X(S)$ is a *summary* measure in the sense that it gives an average compliance measure. It does not tell anything about the *similarity* of one case to another (i.e., two cases that have the same compliance measure can be totally dissimilar simply because they satisfy different constraints but end up satisfying the same number of constraints!).

As far as the retrieval is concerned for a given context X, one can choose the case(s) with:

$$\max_i \{\mu_X(S_i)\}$$

[2]There is a major assumption here about the weights from a measurement-theoretic point of view. It has to be the case that the weights attributed by the user must be on a ratio scale (Krantz *et al.*, 1971) (i.e., if the user is assigning 10 and 20 to two different constraints she means not only the latter is a more important constraint but it is *twice* as important.). This cognitive task is usually fulfilled when aided with proper visual tools like sliding scales. This issue is should not be overlooked. If the designer is not able to fulfill the cognitive requirement that the weights are on a ratio scale, the averaging operation (and the retrieval based on it) is simply meaningless!

or cases with compliance measure greater than a user defined threshold:

$$\mu_X(S_i) \geq \tau.$$

However, μ_X is not monotonic as are most of the retrieval measures on which retrieval is based (i.e., if one adds a constraint to context X to construct context Y the relation between μ_X and μ_Y is undetermined).

4.1. RETRIEVAL BY SOLVING CONSTRAINTS

Although the representation of the context by weighted constraints and using the weighted compliance measure for retrieval of cases is a flexible way of retrieving design cases, it is not sufficient to retrieve a case which does not have a particular attribute that the constraint requires. In such a case, the unknown parameter required by the constraint(s) must be *solved for*.

Furthermore, the engineering design usually has *performance requirements* set as goals to achieve.

To account for the two concepts above we extend the definition of a context to include *objectives* to be optimized as well as constraints. Hence, the context, X, is now given as:

$$X = \{\langle a_1, w_1^o O_1\rangle, \cdots, \langle a_{n'}, w_{n'}^o O_{n'}\rangle, \langle a_1, w_1^c C_1\rangle, \cdots, \langle a_{m'}, w_{m'}^c C_{m'}\rangle\}$$

where the tuple $\langle a_i, w_i^o O_i\rangle$ (footnote 1 applies here as well) denotes a weighted objective on the attribute (e.g. maximize torque, minimize risk, maximize power output etc.) and w^o and w^c denote weights on objectives and constraints, respectively. Note that we allow for *multiple* objective functions and both the objectives and constraints can be assigned weights by the user.

During the interaction of the designer with the system we are assuming that she defines the context using equations to be optimized with respect to constraints to be satisfied. This is not an unreasonable assumption, since the design proceeds by posting requirements (of which performance requirements are the goals, and structural and functional requirements are constraints) and trying to fulfill them.

This complicates the problem particularly when the constraints are not linear. From an implementation point of view such a system of constraints can be handled either by constraint logic programming techniques (e.g. CLP($\mathcal{R}$) (Jaffar *et al.*, 1992; Holzbaur, 1995)) or mathematical programming techniques (e.g. many implementations of the Simplex algorithm or interior point methods for linear constraints or non-linear search algorithms). In this paper we tackle the case where the objective functions and constraints are linear.

When the the objective function and the constraints are linear the problem can be solved by using numerous Multiple Objective Linear Programming (MOLP) techniques. In fact, multiple criteria optimization techniques have been employed in *detailed* engineering design (Statnikov and Matusov, 1995). We are employing

similar techniques for the full life cycle of the design process. The situation with unknowns in constraints and objectives to be optimized is depicted in Figure 5.

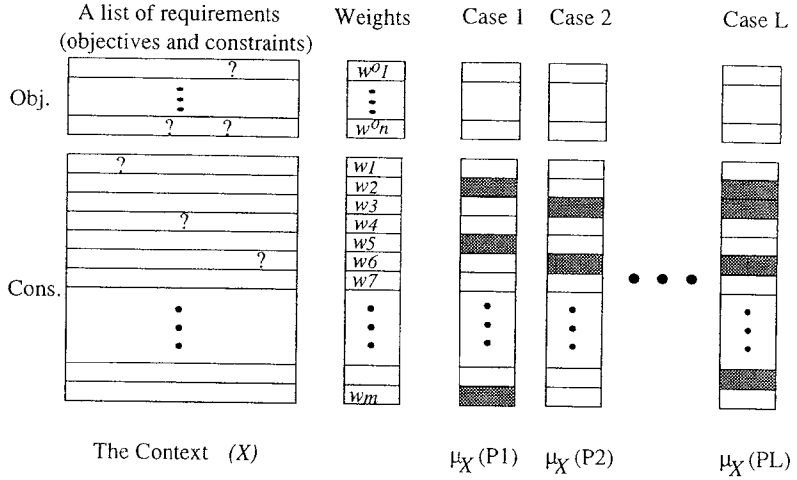

Figure 5. Constraint-based retrieval: the general case. There are multiple objectives to be optimized with respect to the given constraints.

The measure of compliance can be extended to the multiple objective case in a natural manner retaining all the desired properties of the measure for retrieval process:

$$\mu_X(S) = \frac{\sum_{i=1}^{m} w_i^c \chi_i(S) + \sum_{i=1}^{n} w_i^o \chi_i(S)}{\sum_{i=1}^{m} w_i^c + \sum_{i=1}^{n} w_i^o}.$$

The characteristic function for the objectives is evaluated by the designer and only then the compliance measure can be calculated (cf. Section 5).

The retrieval is again based on $\mu_X(S)$ but the process of retrieval requires *solving* a MOLP.

In fact the cognitive task of coming up with weights for objectives and constraints can be quite a hard task in large domains. In order to eliminate the need for an *intrusive* acquisition of weights from the user we assume further generality in the sense that the weights w_i are given as *intervals* rather than a single number: $w_i \in [\ell_i, u_i]$.

When this is the case the problem is still tractable as long as the objectives and the constraints are linear. This formulation gives rise to the family of weighted sum problems (Steuer, 1986).

4.2. SIMILARITY OF CASES

If one is interested in the *similarity* of one case to the other the compliance measure μ_X is useless. For similarity one can define another measure, s_X, which meas-

ures the similarity of two cases for a given context X as (for the most general case with weights):

$$s_X(S_1, S_2) = \frac{\sum_{i=1}^{m}[w_i^c \chi_i(S_1)\chi_i(S_2)] + \sum_{i=1}^{n}[w_i^o \chi_i(O_1)\chi_i(O_2)]}{\sum_{i=1}^{m} w_i^c + \sum_{i=1}^{n} w_i^o}.$$

$s_X(S_1, S_2)$ measures the similarity of case S_1 to case S_2 in context X (It simply counts the occurrences where the two cases satisfy the same constraints and normalizes it using the weights). This is a particularly novel definition of similarity since context is explicitly taken care of. Cases that are similar in one context may be totally dissimilar in others. This effect of context on the similarity measure is a well known problem in the research and practice of similarity measures (Tversky, 1977).

As defined here s_X is a *valued relation*[3] (Ovchinnikov, 1991; Bilgiç, 1995). It has the following desired properties:

- $s_X \in [0, 1]$,
- s_X is reflexive (i.e., $\forall S, s_X(S, S) = 1$), a design case is totally similar to itself.
- s_X is symmetric (i.e., $\forall S_i, S_j, s_X(S_i, S_j) = s_X(S_j, S_i)$), if a design case, S_i, is similar to another, S_j, to some extent then S_j must also be similar to S_i to the *same* extent.
- a more robust definition of transitivity holds:

$$\forall S_i, S_j, S_k, s_X(S_i, S_k) \geq \min\{s_X(S_i, S_j), s_X(S_j, S_k)\}.$$

  If a design case S_i is similar to S_j to an extent and S_j is similar to S_k then S_i must also be similar to S_k to *some* extent. Note that this reduces to the usual definition of transitivity when $s_X \in \{0, 1\}$.

Therefore s_X is a *bona fide* valued similarity relation. It has a robust transitivity condition which avoids heap paradoxes[4] The transitivity condition of the similarity measure we define will make the retrieval algorithm stop at a point when the similarities of the two items diminish to zero (or when it is under a predefined threshold).

Similarity of cases is important if one needs to index cases according to their similarity and *store it that way*. However this can result in inadvertent effects since similarity is dependent on the context. Therefore computing similarity on-the-fly for a particular context at the time of query seems to be the superior alternative. However note that for retrieval purposes the compliance measure μ_X is sufficient.

[3] A valued relation, R, is an extension of the concept of classical relation which takes on either 0 or 1 as its values. Valued relations take on values in the unit interval $[0, 1]$.

[4] A cup of coffee without any sugar and another one with just one grain sugar added are similar in terms of sweetness. The transitivity condition entails that the first cup and a cup with thousands of grains of sugar added are still similar in terms of sweetness.

The valued similarity relation can be a basis for *soft classification* of cases in which every case belongs to a cluster to a degree (for a given context). Such a framework provides flexible retrieval strategies and it reduces to crisp clustering methods similarities are an all-or-nothing matter.

5. Implementation and an Example

In this section, we briefly discuss the implementation of the techniques we described and give a small example.

The implementation of the design (and design case) representation has been carried out in Prolog in an object-oriented manner. An object-oriented layer built on top of Prolog (together with inheritance mechanisms) contains the design representations.

The case retrieval module is also implemented in Prolog. We use ECRC's constraint logic programming system ECLiPSe (Wallace and Veron, 1993) and the CLP(Q,R) constraint solver (Holzbaur, 1995) that comes with it on a Sun SPARC workstation running SunOS version 4.1. The user interface to the system is via World Wide Web (WWW) and it requires a web browser.

We have the representations of several manipulator arms in the system complete with their part, parameter, requirement, constraint, and function hierarchies. We assume that we are faced with the situation of designing another manipulator arm for a totally different task. The aim is to be able to reuse some of the past designs to reduce development costs.

For purposes of illustration we assume that the design has come to a stage where it is decided that another arm with three joints is going to be designed. The designer is faced with the problem of selecting brakes for each joint. Each joint has its own requirements from the brakes to be used (i.e, each joint is a different context for brake selection) and there are several types of brakes used in the past designs.

The functional requirement for a particular joint (context) is represented in the system as shown in Figure 6.

This particular functional requirement (REQ202) is related to two functions that are already defined in the system (FUN30 and FUN31). The representation of FUN30 is shown in Figure 7.

Since the functional requirement is related to the function "stop-motion", a first retrieval on the basis of this requirement retrieves eight parts (design prototypes) that are functionally related to that requirement (cf. Figure 7). In this example, all the prototypes that are retrieved are brakes. It could have been the case that there were other prototypes (e.g. motors with inverse drive capabilities) retrieved for the same function. Pruning of design prototypes can be done by the designer at this moment or the designer may choose to continue, with everything retrieved so far, to requirement-based retrieval where prototypes are evaluated for

Class Frame	REQ202
Subclass of:	REQ190
Instances:	
Relations:	related-to-function [FUN30,FUN31]
Attributes:	token [slow-stop] name [rms.slow-stop.req] req-type [functional-req] req-title [Functional requirement for slowing and stopping] req-id [3.2.7.11b] req-documentation [RMS-SG-1944A] req-short-description [The artifact should slow down and stop the motion produced by the joint motors]
Messages:	

Figure 6. Representation of the functional requirement in the system.

Class Frame	FUN30
Subclass of:	
Instance of:	non-basic-function
Relations:	related-to-part [PRT138,PRT139,PRT140,PRT141 PRT142,PRT143,PRT144,PRT145] generalizes-to [FUN3]
Attributes:	name [stop-motion]
Messages:	

Figure 7. Representation of a function in the system.

their compliance (e.g. motors would have been eliminated from further consideration because they would violate weight and envelope constraints of the context).

In our example, the designer continues with the eight brakes retrieved and evaluates their compliance for the given context. Since the design solution seems to be the concept of a brake, the detailed requirements for the brakes can either be entered explicitly at this point or the requirement tree of one of the retrieved brakes can be adopted and modified.

There are fourteen requirements for the brake in the current context and the representation of one of the requirements (REQ142) is shown in Figure 8 as an example.

The designer weighs each of the fourteen requirements as to their import-

Class Frame	REQ142
Subclass of:	REQ90
Instances:	
Relations:	requirement-of [PRT133,PRT134,PRT135] has-expression [CON53] has-document ['slip-torque.req.html']
Attributes:	name [brake.slip-torque.req] req-derivation-type [derived-req] req-title [Breakaway torque of the brake] req-id [3.2.1.2.3] req-documentation [RMS-SG-1954A] req-short-description [The peak torque level required to induce brake slip shall not exceed 12- oz-inches]
Messages:	

Figure 8. Representation of a derived requirement in the system.

ance and starts the requirement-based retrieval. The results of such a transaction is shown in Figure 9.

The results indicate that PRT138 has the maximum compliance for this particular context. However, the designer selects the top four of the retrieved prototypes (PRT138, PRT139, PRT144, PRT143) and adds the following goals (performance requirements) to the system:

- maximize brake actuation life (10), and
- minimize cost (7).

The numbers by the objectives denote their relative importance. The designer submits this new query to the system which solves for actuation life and cost parameters of three parts (PRT144 did not yield a solution to the required parameters due to insufficient information). Then, the designer evaluates the objectives for each part as to their acceptance and the system returns the new compliance measures: (PRT138:0.88, PRT139:0.89, PRT143:0.92).

The designer selects PRT143 as the new brake of the joint with the knowledge that it has to be modified to meet requirements REQ125, REQ128, and REQ142.

The designer also has the option of classifying retrieved items on the basis of their similarity. To illustrate that, assume that the designer wanted to classify the eight brakes retrieved before the introduction of the objectives. Figure 10 shows the clustering of the eight brakes at two levels of similarity, 0.55 and 0.70.

Note that this clustering is only valid for the particular context (joint).

PRT138 (0.971429) satisfies 13 out of 14 requirements. Requirements that are NOT satisfied: REQ128	PRT139 (0.914286) satisfies 11 out of 14 requirements. Requirements that are NOT satisfied: REQ125 REQ128 REQ130
PRT140 (0.6) satisfies 8 out of 14 requirements. Requirements that are NOT satisfied: REQ123 REQ125 REQ130 REQ137 REQ140 REQ142	PRT141 (0.619048) satisfies 8 out of 14 requirements. Requirements that are NOT satisfied: REQ125 REQ128 REQ130 REQ136 REQ137 REQ142
PRT142 (0.752381) satisfies 9 out of 14 requirements. Requirements that are NOT satisfied: REQ123 REQ125 REQ128 REQ136 REQ140	PRT143 (0.819048) satisfies 11 out of 14 requirements. Requirements that are NOT satisfied: REQ125 REQ128 REQ142
PRT144 (0.914286) satisfies 11 out of 14 requirements. Requirements that are NOT satisfied: REQ125 REQ128 REQ130	PRT145 (0.628571) satisfies 9 out of 14 requirements. Requirements that are NOT satisfied: REQ125 REQ128 REQ138 REQ139 REQ141

Figure 9. Results of requirement-based retrieval.

6. Summary

In this paper we describe a certain view of design for one-off, high-tech artifacts and outline how that design process can be supported in concurrent engineering environments. Our aim is to be able to support all phases of the design life cycle. We briefly mention the type of information we have in TOVE to represent knowledge necessary for the design task. We suggest that fit, form, function, behaviour, working principle, issues and requirements need to be explicitly rep-

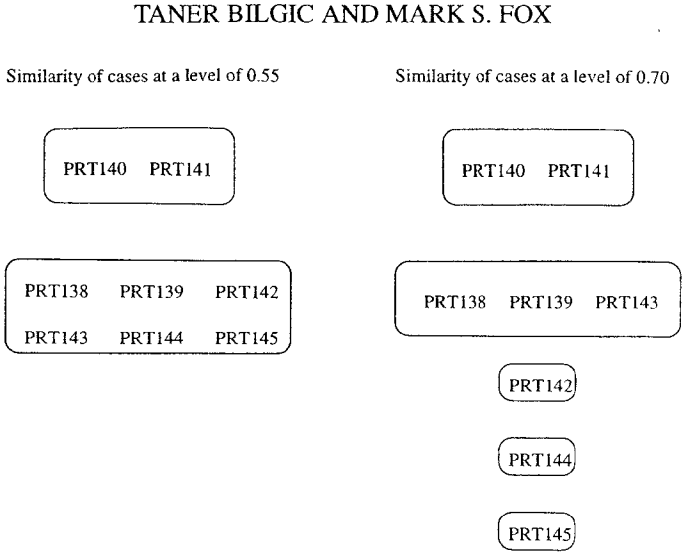

Figure 10. Clustering of the retrieved cases for two similarity levels.

resented. We formalize the retrieval process on the basis of constraints in which we make the constraints, goals, and their appropriate weights an explicit part of the query rather than part of the knowledge-base. This results in a flexible way of retrieving and selecting design cases. This formalization leads to:

- A definition of *context* in terms of constraints on the design. Therefore context becomes an explicit part of the case-based query itself.
- A concept of *compliance* measure, $\mu_X(S)$ for the given context. Each case, S, can be evaluated on the basis of this measure as to its compliance with the context, X.
- An extension to the context such that it not only contains constraints but goals (objectives) to be satisfied as well. This approach is more realistic in the engineering design domain where constraints are decompositions of structural and functional requirements and objectives stem from performance requirements.
- A further extension to the context in the sense that not all objectives and constraints are weighed equal. The designer has the flexibility to choose weights for each objective which denote the importance of that particular objective. The weights can be given as *intervals* if there is any doubt about their validity.
- A definition of similarity of two cases, $s_X(S_i, S_j)$. This measure is based on the context X and can change from one context to the next. (e.g. apples and oranges are not similar in the context of fruits that contain starch but they are similar in the context of edible things).

We are planning on extending the approach provided here in several respects. One immediate concern is the measurement units used in parameters and constraints. A retrieval mechanism should be able to distinguish between different units that are used and should be able to convert from one unit system to another for correct retrieval.

Solving for non-linear objectives and constraints are computationally expensive. Case-based design systems must have well defined protocols to communicate with commercially available symbolic mathematics and detailed engineering design software to care for non-linear objectives and constraints.

Acknowledgments

We would like to thank JinXin Lin for fruitful discussions on an earlier version of this paper. Anonymous referee reports contributed to the clarity of the presentation.

References

Bilgiç, T.: 1995, *Measurement-Theoretic Frameworks for Fuzzy Set Theory with Applications to Preference Modelling*. PhD Thesis, University of Toronto, Department of Industrial Engineering Toronto Ontario M5S 1A4 Canada.

Domeshek, E. A., Herndon, M. F., Bennett, A. W. and Kolodner, J. L.: 1994, Case-based design aid for conceptual design of aircraft subsystems, *Proceedings of the 10th Conference on Artificial Intelligence for Applications*, IEEE, Piscataway, NJ, pp. 63–69.

Fox, M. S. and Gruninger, M.: 1994, Ontologies for enterprise integration, *Proceedings of the 2nd Conference on Cooperative Information Systems*, Toronto, Ontario.

Fox, M. S. and Salustri, F.: 1994, A model for one-off systems engineering, *Proceedings of the AI and Systems Engineering Workshop, AAAI '94*, Seattle, Washington.

Fox, M. S., Chionglo, J. F. and Fadel, F. G.: 1993, A common sense model of the enterprise, *Proceedings of the 2nd Industrial Engineering Research Conference*, Institute for Industrial Engineers, Norcsross, GA, pp. 425–429.

Fox, M. S.: 1992, The TOVE project: Towards a common sense model of the enterprise, *in* C. Petrie (ed.), *Enterprise Integration*. MIT Press, Cambridge, MA.

Goel, A. and Chandraskaran, B.: 1989, Use of device models in adaptation of design cases, In *Proceedingsof the DARPA Workshops on Case-Based Reasoning*, Morgan Kaufmann, New York, pp. 109-120.

Goel, V.: 1994, A comparison of design and nondesign problem spaces, *Artificial Intelligence in Engineering*, **9**(1), 53–72.

Holzbaur, C.: 1995, OFAI clp(q,r), manual, edn 1.3.3, *Technical Report TR-95-09*, Austrian Research Institute for Artificial Intelligence, Vienna.

Jaffar, J., Michayov, S., Stuckey, P. and Yap, R.: 1992, The CLP($\mathcal{R}$) language and system, *ACM Transactions on Programming Languages and Systems*, **14**(3), 339–395.

Kolodner, J.: 1993, *Case-Based Reasoning*. Morgan Kaufmann, New York.

Krantz, D. H., Luce, R. D., Suppes, P. and Tversky, A.: 1971, *Foundations of Measurement*, Vol. 1, Academic Press, San Diego.

Kumar, H. S. and Krishnamoorty, C. S.: 1995, A framework for case-based reasoning in engineering design. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing (AI EDAM)*, **9**, 161–182.

Maher, M. L. and Balachandran, M. B.: 1994, Flexible retrieval strategies for case-based design, *in* J. S. Gero and F. Sudweeks (eds), *Artificial Intelligence in Design '94*, Kluwer, Dordrecht,

pp. 163–180.
Maher, M. L. and Zhang, D. M.: 1993, CADSYN: A case-based design process model, *Artificial Intelligence for Engineering Design, Analysis and Manufacturing (AI EDAM)*, 7(2), 97–110.
Maher, M. L., Balachandran, M. B. and Zhang, D. M.: 1995, *Case-Based Reasoning in Design*, Lawrence Erlbaum, Hillsdale, New Jersey, USA.
Nakatani, Y., Tsukiyama, M. and Fukuda, T.: 1992, Engineering design support framework by case-based reasoning, *ISA Transactions*, **31**(2), 165–180.
Ovchinnikov, S.: 1991, Similarity relations, fuzzy partitions, and fuzzy orderings, *Fuzzy Sets and Systems*, **40**, 107–126
Pahl, G. and Beitz, W.: 1988, *Engineering Design: A systematic approach*, Springer-Verlag, Berlin, trans. A. Pomerans and K. Wallace.
Serrano, D. and Gossard, D.: 1988, Constraint management in MCAE, *in* J. S. Gero (ed.), *Artificial Intelligence in Engineering: Design*, Elsevier/CMP, Boston/Southampton, pp. 217–240.
Statnikov, R. B. and Matusov, J. B.: 1995, *Multicriteria Optimization and Engineering*, Chapman and Hall, New York.
Steuer, R. E.: 1986, *Multiple Criteria Optimization: Theory, Computation, and Application*, John Wiley, New York.
Sycara, K. P. and Navinchandra, D.: 1992, Retrieval strategies in a case-based design system, *in* C. Tong and D. Sriram (eds), *Artificial Intelligence in Engineering Design. Vol. II*, Academic Press, New York, NY, pp. 145-164.
Sycara, K. P. and Navinchandra, D.: 1989, Integrated case-based reasoning and qualitative reasoning in engineering design, *in* J. S. Gero (ed.), *Artificial Intelligence in Design*, CMP/Springer-Verlag, Berlin, pp. 231–250.
Tversky, A.: 1977, Features of similarity, *Psychological Review*, **84**(4), 327–352.
Wallace, M. and Veron, A.: 1993, Two problems – two solutions: One system – ECLiPSe. *Proceedings IEE Colloquium on Advanced Software Technologies for Scheduling*, London.
Wood III, W. H. and Agogino, A. M. 1993, A case-based conceptual design information server for concurrent engineering, *Technical Report 93-1104-1*, University of California, Berkeley, Department of Mechanical Engineering, BEST Laboratory, Berkeley , CA 94706, USA.