Case-based Learning for Design*

Katy Börner
University of Freiburg
Centre for Cognitive Science
79098 Freiburg, FRG
katy@cognition.iig.uni-freiburg.de

Dietmar Janetzko
University of Freiburg
Centre for Cognitive Science
79098 Freiburg, FRG
dietmar@cognition.iig.uni-freiburg.de

We present a general framework for an adaptive, user-interactive design support system. Adaptability requires the acquisition and usage of huge amounts of highly structured knowledge. Interactiveness, i.e., the ability to perform and react in tight co-operation with the user, is of special importance in complex domains like industrial building design. Case-based learning will be applied to provide the knowledge necessary to support subgoaling, pre-organization of the case-base, similarity assessment, and adaptation. Prior work (Börner 1994a, Börner 1994b) focused on structural similarity assessment and adaptation. Our main intention here is case-based subgoaling and its interaction and relation to the former via some appropriate knowledge representation. The paper gives the underlying ideas and the basic algorithms of the framework. For its practical evaluation the framework has been partially implemented in *SynGraph*, a module of the knowledge-based system FABEL–IDEA.

1 Introduction

In knowledge acquisition, reasoning, and system development there are strong interactions between the tasks the system has to fulfil, the reasoning methods chosen, and the knowledge needed cf. (Janetzko, Börner, Jäschke and Strube 1994). The application domain used to delineate case-based learning for design is industrial building design. Given an actual problem solving state, we need to suggest: *subgoals to be tackled next*; *a set of previous experiences to give some hints about possible solutions*; and most ambitious to provide *adapted design solutions*. Frequently, architects browse through old blueprints reflecting similar designs in order to solve new problems. According to our experience and to the literature as well (Goel 1989, Hinrichs 1992, Hua and Faltings 1993, Kolodner 1993), case-based reasoning (CBR) seems to be the natural problem solving method.

The tasks and CBR as the main problem solving method force the acquisition of cases and appropriate similarity measures over case-sets. Arising questions are: What *grainsize* should cases have? How to *represent* cases? How to define *similarity* in a computational effective but at the same time structural selective manner? Finally, how to acquire and structure the huge amount of knowledge necessary to support building design? Given an appropriate knowledge representation scheme, the integration of learning is a promising way to lessen the burden of knowledge elicitation to system development. The intention of this paper is to present a framework for a system that starts with no knowledge at all and is able to learn incrementally the knowledge needed to support the design of buildings.

The sections of this paper may be summarized as follows. Section 2 provides an introduction into the application domain inclusive its formalization. Section 3 presents the outline of our framework. Section 4, which is the heart of this paper, provides the basic learning and reasoning algorithms applied. The final part of the paper discusses and relates the presented framework.

*This research was supported by the German Ministry for Research and Technology (BMFT) within the joint project FABEL under contract no. 413-4001-01IW104. Project partners in FABEL are German National Research Center of Computer Science (GMD), Sankt Augustin, BSR Consulting GmbH, München, Technical University of Dresden, HTWK Leipzig, University of Freiburg, and University of Karlsruhe.

2 Application Domain and its Formalization

The application domain used to delineate our rationale of incremental, adaptive learning and interactive case-based problem solving is industrial building design. Our focus is on installations in buildings with a complex infrastructure. Here, the main problem is how to layout subsystems for fresh and used air, electrical circuits, warm, cold, and used water, computer networks, phone cables, etc. In the following we will introduce the notions of *objects, states, task-structures,* and two kinds of *case* representations inclusive *similarity measures*. Formalizations are given in *slanted letters*. The formalizations are needed to introduce the algorithms for case-based learning and reasoning in the subsequent section.

2.1 Objects

To represent CAD-like drawings, we use the representation scheme A4 (Hovestadt 1993). A4 allows to represent *objects* (e.g., concrete objects like rooms, pipes, chairs etc., but also more abstract objects like areas for the entire house or the climate system, etc.) *graphically* and *attribute-based*. The former is used as the main basis for man-machine interaction. The latter constitutes the basis for machine learning and reasoning.

Graphically, A4-objects are represented by geometrical objects. Different states during the design of a building correspond to different configurations of objects[1]. In this paper, we restrict ourselves to two dimensional pictures for readability.

Attribute-based, A4-objects are represented by values for a fixed set of *geometrical* attributes $G := \{a_1, \ldots, a_k\}$ and *type* attributes $D := \{a_{k+1}, \ldots, a_m\}$. Geometrical attributes that describe placement and extension are $\{x, dx, y, dy\}$. The type of some object is denoted by the attributes { aspect, morphology, resolution, size }. Aspect relates to the subsystem of the building (e.g., z=supply–air). Morphology denotes the general function of the area addressed (e.g., a=linkage, e=development, v=connection). The third attribute specifies the kind of resolution employed (e.g., b=area, h= bounding-box). Size relates to the part of the building that is envisaged (e.g., 4=hallway, 6=room, 8=areas within a room).

Objects are defined by their geometrical and type attribute values. We use some notions from database theory to denote an attribute value of an object o at some attribute a by $o.a$. Thus an object o_i can be represented by the union of the set of geometrical attribute values $o_i.G := \{o_i.a_1, \ldots, o_i.a_k\}$, and the set of type attribute values, defined by $o_i.D := \{o_i.a_{k+1}, \ldots, o_i.a_m\}$, i.e., $o := o_i.G \cup o_i.D$.

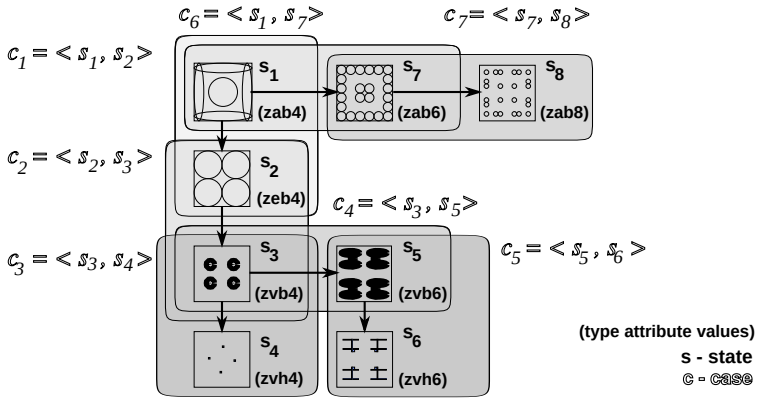

Figure 1: Objects, states, and cases

[1] Ellipses are a substitute for rectangles circumscribed by the ellipses. Using ellipses instead of rectangles is an unaccustomed but very useful trick: Ellipses overlap only in a few points. The readability of drawings becomes essentially improved.

2.2 States

CAD-like drawings correspond to snapshots of *states* in problem solving. We assume, that objects of identical type are grouped together to constitute a state.

Fig. 1 depicts the design of eight different states named $s_1, ..., s_8$ of some fresh air system. Each state is represented by a number of objects. The geometrical attributes of objects are represented graphically. Their types are represented by attribute values. As an example, state s_1 of type *zab4* shows five objects that represent the areas of supply–air linkage which cover the hallway.

States *s_j are a set of objects $o_{i_1}, \ldots, o_{i_{n_j}}$ of identical type $o_{i_k}.D$. We define $s_j.G$ as the set containing all sets of geometrical attribute values for the objects belonging to s_j, i.e., $s_j.G := \{o_{i_k}.G \mid k = 1, \ldots, n_j\}$. We may now represent a certain state s_j by the pair consisting of the set of geometrical attribute values for the set of objects belonging to s_j and their type attribute values. In our approach holds: $(o_{i_1}.D = o_{i_2}.D = \ldots = o_{i_{n_j}}.D)$. So without loss of generality we may represent some state s_j by $s_j := \langle s_j.G, o_{i_1}.D\rangle$. The set of all states is named S.*

2.3 Task-Structures

The task to be supported by case-based reasoning is the creation of new objects based on objects already designed. Given a certain design state, say s_1 in Fig. 1, the design of two states s_2 or s_7 may be attacked. Designing s_2 first, s_3 on the basis of s_2 or s_7 on the basis of s_1 may be tackled next etc. In Fig. 1 arrows are used to denote these predecessor relations between states. The state-transitions are seen as *tasks* to be tackled one after each other. The set of states together with their relations constitute the so called *task-structure*.

Task-structures *represent some semiordering relation $\prec$ over a set of states S that is defined by $s_i \prec s_j :\Longleftrightarrow$ iff objects of s_j are designed on the basis of objects constituting state s_i. States $s_1, s_2 \in S$ are called immediately subsequent iff there exists no other state $s \in S$ with $s_1 \prec s \prec s_2$. Weights $w_{s_i \prec s_j}$ may be used to state the frequency, i.e. preference, of possible state transitions.*

2.4 Cases

Aiming at a task-oriented user support (Janetzko et al. 1994), the grainsize of *cases* is identical to the grainsize of tasks. Different design steps, i.e. tasks, require different design strategies. For example, while return air accesses will be connected by using the shortest path, connections for fresh air accesses take curved tracks to achieve noise reduction. Each task requires separate cases, i.e., a solution state dependent case-base. For an illustration see Fig. 1. Given the problem objects of type *zab4*, case $c_1 = \langle s_1, s_2\rangle$ supports the design of *zeb4*-objects. Next, these *zeb4*-objects of s_2 become the problem and are solved by designing objects of type *zvb4*, i.e. s_3, applying $c_2 = \langle s_2, s_3\rangle$. Depending on the stage in problem solving a state can either take the role of a problem or of a solution.

Attribute-based, cases *are defined by the geometrical and type attribute values of immediately subsequent states. Let the superscripts p and s denote the problem and solution state of some case c. The set $CB_{s.D}$ of cases with identical solution type $s.D$ is defined by*

$$CB_{s.D} := \{\langle s^p, s^s\rangle \in S \times S \mid s^p.D \prec s^s.D \wedge s^s.D = s.D\}.$$

There is an important difference between $s^s.G$ and $s^s.D$ describing the solution. While $s^s.D$ represents its global *intentional* solution, i.e., the next subgoal, $s^s.G$ represents the specific *extensional* solution, i.e., the design solution. Thus, $s^s.D$ refers to peculiarities of states with its different case-bases and similarity relations. Contrary, $s^s.G$ represents specific geometrical attribute values of objects.

To determine similarity between cases represented in an attribute-based way we may use standard similarity measures of CBR. See (Voß 1994) for different approaches.

Attribute-based Similarity *of two problem states s_j^p, s_k^p may here be simply defined via the distance of their geometrical attribute values $s_j^p.G$ and $s_k^p.G$. Weights w^δ may be used to denote the relevance*

of attribute values, e.g., to prefer the geometrical extension of objects as opposed to their placement. Assuming $s_j^p.G$ and $s_k^p.G$ have the identical number n of objects their HAMMING *distance δ is defined by:*

$$\delta(s_j^p, s_k^p) := \sum_{i=1}^{n} w_i^\delta \cdot \delta(o_{j_i}^p.G, o_{k_i}^p.G).$$

The distance may be transformed into a similarity measure σ^a by:

$$\sigma^a(s_j^p, s_k^p) := \frac{1}{1 + \delta(s_j^p, s_k^p)}.$$

Due to the application domain, attribute based object descriptions are not sufficient to determine similarity in terms of adaptability. For illustration, have again a look at Fig. 1, state s_5. Given the geometrical attribute values of the four partial object arrangements, their identity after rotation and reflection is hard to determine. Not the geometrical attribute values of single objects count. Instead the relations between objects belonging to one state have to be considered. Reasoning has to proceed over object relations instead over attribute values of single objects. Necessary are topological, structural representations of states inclusive appropriate similarity concepts. Using an algebraic approach we represent problems and solutions by ground terms (i.e. terms containing no variable like *above(a,b)*) of some appropriate signature.

Structurally, cases *are represented by ground terms t in some term algebra $T(\Sigma, \emptyset)$. See (Börner 1994a, Börner 1994b) for a detailed explanation. Assuming the existence of some transformation function ϕ with its inverse automatically enables to transform attribute-based representations into structural representations and vice versa. The structural representation of problem and solution state of some case $c = \langle s^p, s^s \rangle$ is defined by: $t^p := \phi(s_i^p.G) \quad \wedge \quad t^s := \phi(s_j^s.G)$.*

Structural similarity assessment proceeds by comparing some actual problem with the common structure or *prototype* of a set of prior problem experiences. For that reason we need to derive the prototype of prior problems first. Storing the modifications which lead to the common prototype we know about proper modifications to change the actual problem to fit into the prototype. Contrary to standard approaches to CBR these modifications are represented by the structural similarity measure[2].

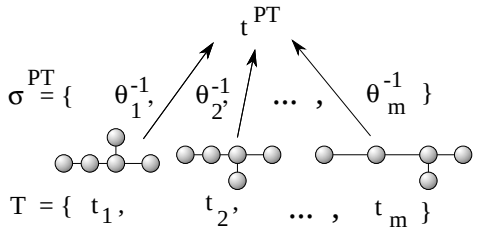

Figure 2: Prototypes

Structural Similarity *σ^{PT} is defined as the union of proper modifications which relate a set of problems $t_i^p \in T^p,\ i = 1, ..., n$ to each other (see Fig. 2). That is*

$$\sigma^{PT} := \cup_{i=1}^{n} \theta_i^{-1}$$

satisfying $t_i^p \theta_i^{-1} = t^{PT,p},\ i = 1, ..., n$. The unique term $t^{PT,p}$ is called the prototype of this term set[3]. For any other prototype t of T^p there exists an inverse substitution θ^{-1} s.t. $t^{PT,p}\theta^{-1} = t$.

[2] As an example of this view to similarity, imagine a prototypical room. This room, which may or may not exist, is a representative for all rooms you have ever seen. Remembering prior rooms (cases) of specific types, forms, usages, etc. you are able to determine proper modification rules (σ^{PT}) which lead to the room-prototype. Entering an unseen space, you will be easily able to recognize if it is a room. You simply apply proper modifications and compare the result with the corresponding room prototype.

[3] This definition of syntactic similarity is close to the concept of *syntactic antiunification*. Given two terms t_1, t_2 one is searching for some term t, called anti-unifier, and corresponding substitutions θ_1, θ_2 satisfying both $t\theta_1 = t_1$ and $t\theta_2 = t_2$. The term t is called *least general anti-unifier* (Muggleton 1992) or *most specific generalization* (Plotkin 1970) iff there is no other anti-unifier $t\prime$ with $t\prime = t\theta$.

3 Outline of the Framework

Taken from (Börner and Voß 1994) Fig.3 provides the general outline of the framework. We use a trace of the human-system interaction, which feeds into four algorithms all of which incrementally learn knowledge needed in design support. The following convention is used here: arrows are employed to denote knowledge processing, squared boxes sketch knowledge examples.

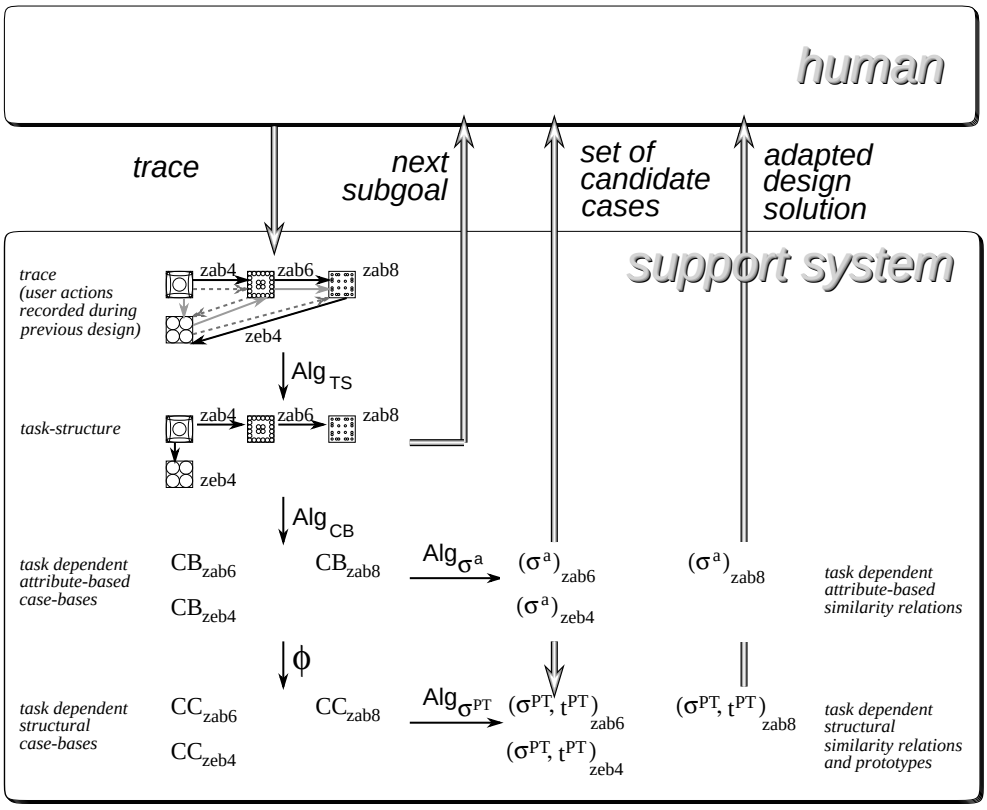

Figure 3: Outline of the framework

As a graphical surface we employ a knowledge-based CAD-system called DANCER (Hovestadt 1993). The user while employing the system like a standard CAD drawing tool designs objects, assigns labels, etc. The design steps and the objects designed are recorded in a *trace* that provides the input to the support system.

The actual support by the system may be divided into three parts: First, providing guidance to select the *next subgoal*, i.e., the next state to be tackled in design. Second, suggesting a set of *candidate cases* able to give some hint to solve the actual problem. Third, generating an *adapted design solution* which fits the selected subgoal. In all three settings the user is each time in a position to accept, modify, or reject the solution. If the problem actually given can't be solved by the system it is allocated to the user. Each successful solution and each user interaction are inputs to the algorithm and influence learning and reasoning immediately.

Let's have a closer look at the algorithms. Algorithm ALG_{TS} extracts preferred state sequences, i.e. task-structures, out of several traces. Corresponding to their type attribute values objects are grouped to states. In the task-structure consecutive states are recorded together with weights of their frequency. The task-structure itself is used to support subgoaling and to elicit cases in a task-oriented way. Algorithm ALG_{CB} uses it like a cookie-cutter to stencil cases out of the original trace. The cases (stored in *CB*) are given in A4 representation, i.e., by attribute-value pairs. Given some appropriate distance or similarity measure, algorithm ALG_{σ^a} learns about attribute relevance. Applying this measure enables to

determine a set of candidate cases CC out of the case-base CB. For adaptation, relations between objects have to be taken into account. Structural representations are needed. To transform the geometrical object representation into some topological, algebraic representation we assume the existence of some transformation function ϕ. Given this knowledge representation algorithm $\text{ALG}_{\sigma^{PT}}$ extracts prototypes t^{PT} from sets of prior problems automatically. New problems will be matched against these prototypes by applying the structural similarity measure σ^{PT}. Given structural similarity, the prototypical solution is transferred. Applying inverse modifications and the inverse transformation function ϕ^{-1} leads to the *adapted design solution.*

4 Algorithms

Given the introduced knowledge representation and appropriate reasoning methods the algorithms are straightforward. For readability we introduce ALG_{TD} responsible for *subgoaling* and $\text{ALG}_{\sigma^{CB}}$, ALG_{σ^a}, $\text{ALG}_{\sigma^{PT}}$ responsible for *design* separately.

4.1 Task-Structures and Subgoaling

Algorithm ALG_{TD} learns preferred state sequences, i.e., the task-structure out of user traces. Traces are primarily a sequence of designed objects o_{i_k}, $k = 1, ..., k_{n_j}$ constituting states s_j. The objects itself are represented by their geometrical and type attribute values. Traces may be seen as never ending. After the complete design of a building, the user may design the next. Starting with state sequences for specific buildings we derive complex structures called task-structures (see subsection 2.3) which correspond to the alternatives and peculiarities when designing different buildings.

Subgoaling refers to the capability, of knowing what to do subsequently at any state in problem solving. In order to show this capability, states to aim at, i.e., subgoals have to be created, selected, and linked. Excluding interacting and parallel ways we want to learn proper (alternative) state sequences. In building design it is difficult to acquire preconditions and effects of certain design steps. Thus, we learn alternative state sequences, i.e., sequences of states traversed one after each other.

Algorithm ALG_{TD}:

begin
input $o_{i_k} \in s_j$, $k = 1, ..., k_{n_j}$; $s^p := s_j$;
repeat
  if subgoal-support **then**
    begin
    if $(s^p \in S)$ **and** $(\forall(s_k, s_l \in S)\ (w_{s^p \prec s_k} > w_{s^p \prec s_l}))$ **then**
      begin
      $s^s.D := s_k$; **output** $s^s.D$
      end else begin output no-subgoal; **input** $s^s.D$ **end**
    end;
  [call ALG_{σ^a}, $\text{ALG}_{\sigma^{PT}}$, and $\text{ALG}_{\sigma^{CB}}$];
  $w_{s^p \prec s^s} := w_{s^p \prec s^s} + 1$;
  $s^p := s^s$
until abort
end.

Figure 4: Algorithm to support subgoaling

Algorithm ALG_{TD} (see Fig. 4) may be seen as the main, all other algorithms supporting and activating one. If *subgoal-support* is required by some human user ALG_{TD} takes objects o_{i_k} as input, determines

the next subgoal, calls subsequent algorithms for the actual design solution, and learns weights for state transitions in order to deal with multiple choices. Obviously, at the first run with empty knowledge base, the algorithm provides not hint what to do. Given some object o_{i_k}, which is supposed to belong to some problem state s^p with $o_{i_k}.G \in s^p.G$ and $o_i.D = s^p.D$, the actual solution consisting of $s^s.G$ and $s^s.D$ with $s^p.D \prec s^s.D$ must be provided by the user. As long as $abort = FALSE$ the algorithm repeats with the actual solution state as the new problem, i.e. $s^p := s^s$.

4.2 Cases, Similarity Measures, and Design Solutions

For the actual design solution we need to derive the geometrical attribute values $s^s.G$ of type $s^s.D$. In CBR a two-stage similarity assessment (Gentner and Forbus 1991) has been widely accepted as an efficient way to case-selection. Taking this stance we advocate for a combination of fast preselection to determine a set of candidate cases CC, followed by structural similarity assessment to derive adapted solutions. Preselection proceeds by comparing weighted attribute values using σ^a. For structural similarity assessment and adaptation prior and actual geometrical attribute-based descriptions are transformed into term-based (or graph-based) ones. Using the underlying term algebra (or graph algebra) and the concept of *antiunification* (Muggleton 1992) we are able to learn inductively about proper substitutions of $t_i^p \in CC$ represented by σ^{PT} which lead to their common prototype t^{PT}. The modification rules (e.g., proper substitutions, deletions of objects, geometrical transformations, etc.) will be used to determine the set of cases to which a new problem belongs. Given structural similarity the prototypical solution of this case set is transferred to the actual problem. Adaptation proceeds by applying the substitutions, which lead the common prototype. This has been discussed in detail elsewhere (cf. (Börner 1994b)).

Algorithm ALG$_{CB}$, ALG$_{\sigma^a}$, and ALG$_{\sigma^{PT}}$:

begin
if design-support **then**
  begin
  $CC_{s^s.D} := \{c_i \mid c_i \in CB_{s^s.D} \wedge \sigma^a(c_i^p.G, s^p.G) \geq \text{threshold}^{\sigma^a}\}$; **output** $CC_{s^s.D}$;
  if (adaptation-support) **and** ($\mid CC_{s^s.D} \mid > 1$) **then**
    begin
    $t^p := \phi(s^p.G)$; $T^p := \{t_k^p \mid \forall\ (c_k^p \in CC_{s^s.D})\ (t_k^p := \phi(c_k^p.G))\}$;
    $\sigma^{PT} := \cup_{k=1}^{n} \theta_k^{-1}$ with $\forall\ (t_k^p \in T^p)\ \ (t^{PT,p} := t_k^p\ \theta_k^{-1})$;
    if $(t^p\ \sigma^{PT} = t^{PT,p})$ **then**
      begin
      $t^s := t^{PT,s}\theta$ with $\forall\ (c_k^s \in CC_{s^s.D})((t_k^s := \phi(c_k^s.G) \wedge (t^{PT,s} = t_k^s\ \theta_k^{-1}))$;
      $s^s.G := \phi^{-1}(t^s)$; **output** $s^s.G$
      end
    end else input $s^s.G$
  end else input $s^s.G$;
  $CB_{s^s.D} := CB_{s^s.D} \cup \langle s^p, s^s \rangle$
  end
end;

Figure 5: Algorithms to support design solutions

Fig. 5 provides the general algorithm to support design. It is called by algorithm ALG$_{TD}$. Input are the geometrical problem description $s^p.G$ and the next subgoal $s^s.D$ to be addressed. If the user wants *design support*, a set of candidate cases $CC_{s^s.D}$ is selected and presented. The appropriate threshold$^{\sigma^a}$ is strongly user, domain, and task dependent. If the user asks for *adaptation-support* and there are more than one candidate case structural similarity assessment and adaptation proceeds. That is, the actual problem $(t^p := \phi(s^p.G))$ and the problems of the candidate cases $(T^p := \{t_k^p \mid \forall\ (c_k^p \in CC_{s^s.D})\ (t_k^p := \phi(c_k^p.G))\})$ are transferred into their structural descriptions. The prototype $t^{PT,p}$ and the structural similarity measure

σ^{PT} of T^p is derived. Given structural similarity ($t^p\ \sigma^{PT} = t^{PT,p}$) the prototypical solution is transferred and adapted ($t^s := t^{PT,s}\theta$). Applying the inverse transformation function ϕ^{-1} leads to the geometrical attribute values of the design solution $s^s.G$. If knowledge is missing or the user wants to solve a certain task on his own, he has to determine the actual design solution $s^s.G$. Finally, the actual case is added to the appropriate case-base. The algorithm continues in Fig. 4.

4.3 Implementation

The algorithm ALG$_{TD}$ is partially implemented and generates task-structures of 85% accuracy out of 12 user traces using several heuristics. ALG$_{CB}$ was implemented separately in order to cut cases of unique grainsize automatically and in a task-oriented way. Both algorithms constitute the backbone of a program called *SynGraph* (*Syn*thesis using *Graph*-based knowledge representations) which fully implements the algorithms ALG$_{\sigma^a}$ and ALG$_{\sigma^{PT}}$ to *similarity assessment* and *adaptation*. *SynGraph* is a module of the design support system FABEL-IDEA.

5 Discussion

When CBR is applied in real world domains, problems that relate to the amount and the structural complexity of the knowledge needed have to be addressed. Systems that require an immense effort for knowledge elicitation will hardly succeed. Filling in large forms to label and save each possibly useful experience, as suggested e.g., for the ARCHIE system (Domeshek and Kolodner 1992) will not do. There is simply no time to feed in all knowledge required. We believe that highly user-interactive frameworks, which manage knowledge elicitation during the system usage, are a real challenge to enable useful and realizable computer-aided support in design. The close linkage of knowledge acquisition, problem solving, and learning guaranties support that is adaptive to the user and the peculiarities of the chosen domain.

There are three basic approaches our framework and therefore the intense integration of learning is based on. First, the task-oriented approach to knowledge acquisition and reasoning. It enables not only the automatical acquisition of task-structures and cases but also determines knowledge exchange points between user and system. Their reasoning strategies may differ, their solution exchange via states is exactly defined. Second, the introduced conjunction of subgoaling and design via the representation of states and cases. Third, the two level case representation as basis for computationally effective and structurally selective reasoning.

We want to contrast our work to CBC (Avila, Paulokat and Weß 1994) the CBR component of the planning system CAPLAN. In order to do so, we compare the *choice points*, i.e. the ways how *subgoals, cases / operators*, and their right *adaptations / instantiations* are determined. In CBC already solved plans are used to control and restrict the search space. In our approach the *goal-choice* is based on the task-structure, i.e. the *entire set* of prior states inclusive their predecessor relations. CAPLAN applies operators to derive concrete solutions. In our domain most state transitions can not be represented by operators. Accessible knowledge are concrete prior experiences (cases). Thus the *operator choice* in CBC is comparable to our *selection of appropriate candidate cases*. Last but not least the *choice of variable instantiation* for a selected goal may be compared to our concrete solution instantiation, i.e., *adaptation*. Again, and contrary to work done, e.g. in case-based explanation (Kass, Leake and Owens 1986), structural similarity assessment and adaptation is done on the basis of the *entire set* of relevant cases. Taking indexes to retrieve prior experiences and applying adaptation rules separately will not do. We have to deal with structures to determine similarity in terms of adaptability.

6 Acknowledgements

This research has been strongly inspired by work done in the project FABEL the general objective of which is the integration of case-based and model-based approaches in knowledge-based systems. We

would like to acknowledge detailed comments of Jürgen Paulokat and Oliver Jäschke on a prior draft of this paper. Nonetheless, the paper reflects our personal view. We thank one anonymous reviewer for detailed comments. We would like to encourage others to think about the need and special importance of formalizations in CBR.

References

Avila, H. M., Paulokat, J. and Weß, S. (1994). Controlling a nonlinear hierarchical planner using case-based reasoning, *2nd European Workshop on Case-Based Reasoning*.

Börner, K. (1994a). Structural similarity as guidance in case-based design, *in* S. Wess, K.-D. Althoff and M. M. Richter (eds), *Topics in Case-Based Reasoning Selected Papers from the First European Workshop on Case-Based Reasoning (EWCBR-93)*, Vol. 837 of *Lecture Notes in Artificial Intelligence*, Springer, pp. 197–208.

Börner, K. (1994b). Towards formalizations in case-based reasoning for synthesis, *AAAI-94 Workshop on Case-Based Reasoning*, pp. 177–181. also FABEL Report 22.

Börner, K. and Voß, A. (1994). Applying machine learning to improve innovative design support systems, *AID-94 Workshop Machine Learning in Design*.

Domeshek, E. A. and Kolodner, J. L. (1992). A case-based design aid for architecture, *Proc. Second International Conference on Artificial Intelligence in Design*, Kluwer Academic Publishers, pp. 497–516.

Gentner, D. and Forbus, K. D. (1991). MAC/FAC: A model of similarity-based retrieval, *Proceedings of the Cognitive Science Conference*, pp. 504–509.

Goel, A. K. (1989). *Integration of case-based reasoning and model-based reasoning for adaptive design problem solving*, PhD thesis, Ohio State University.

Hinrichs, T. R. (1992). *Problem solving in open worlds: A case study in design*, Lawrence Erlbaum Associates.

Hovestadt, L. (1993). A4 – digital building – extensive computer support for the design, construction, and management of buildings, *CAAD Futures '93, Proceedings of the Fifth International Conference on Computer-Aided Architectural Design Futures*, North-Holland, pp. 405–422.

Hua, K. and Faltings, B. (1993). Exploring case-based building design – CADRE, *AI EDAM* **7(2)**: 135–143.

Janetzko, D., Börner, K., Jäschke, O. and Strube, G. (1994). Task-oriented knowledge acquisition for design support systems, *First European Conference on Cognitive Science in Industry*.

Kass, A. M., Leake, D. B. and Owens, C. C. (1986). SWALE: A program that explains, *in* R. C. Schank (ed.), *Explanation patterns: Understanding mechanically and creatively*, Lawrence Erlbaum Associates, pp. 232–254.

Kolodner, J. L. (1993). *Case-based reasoning*, Morgan Kaufmann.

Muggleton, S. (1992). *Inductive logic programming*, Academic Press.

Plotkin, G. D. (1970). A note on inductive generalization, *in* B. Meltzer and D. Michie (eds), *Machine Intelligence 5*, American Elsevier, pp. 153–163.

Voß, A. (1994). Similarity concepts and retrieval methods, *FABEL–Report 13*, Gesellschaft für Mathematik und Datenverarbeitung mbH, Forschungsbereich Künstliche Intelligenz.