Task-Oriented Knowledge Acquisition and Reasoning for Design Support Systems *

Dietmar Janetzko[1], Katy Börner[2], Oliver Jäschke[1], and Gerhard Strube[1]

[1] University of Freiburg
Institute of Computer Science
and Social Research
79098 Freiburg, FRG
dietmar | oliver | strube @cognition.iig.uni-freiburg.de

[2] HTWK Leipzig
Department of Informatics
P. O. Box 66
04251 Leipzig, FRG
katy@informatik.th-leipzig.de

Abstract

We present a framework for task-driven knowledge acquisition in the development of design support systems. Different types of knowledge that enter the knowledge base of a design support system are defined and illustrated both from a formal and from a knowledge acquisition vantage point. Special emphasis is placed on the task-structure, which is used to guide both acquisition and application of knowledge. Starting with knowledge for planning steps in design and augmenting this with problem-solving knowledge that supports design, a formal integrated model of knowledge for design is constructed. Based on the notion of knowledge acquisition as an incremental process we give an account of possibilities for problem solving depending on the knowledge that is at the disposal of the system. Finally, we depict how different kinds of knowledge interact in a design support system.

*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.

1 Introduction

Knowledge acquisition refers to analyzing requirements and knowledge of the system to be built, elicitating of knowledge, and developing models related to problem solving behavior. In acquiring knowledge that can be employed to build design support systems, a number of problems must be addressed: Knowledge brought to bear in design has to be identified; the particular contributions of different kinds of knowledge (e.g., tasks, cases, rules, models) to the design task have to be assessed; interrelations and interactions between different types of knowledge have to be exploited such that knowledge acquisition can proceed in an incremental way. Once these issues are addressed in a systematically way it is possible to move from knowledge acquisition to system design. This may be done by giving a specification of the data flow of a system that may be used as a design support system [JBCH93].

The goal of this paper is to delineate a methodology of task-oriented knowledge acquisition that may be used to build design support systems. Our presentation of a methodology of task-oriented knowledge acquisition proceeds in three sections. First, we introduce the kinds of knowledge to be used in design. Tasks, cases, rules, and their integration into a comprehensive model of knowledge are described both from a formal and from a knowledge acquisition perspective. Second, we discuss the rationale of incremental knowledge acquisition and the knowledge-dependent capabilities of a design support system. Third, we point out examples of applying different kinds of knowledge when specifying data-flow in design support systems. Additionally, we illustrate the interactive use of different kinds of knowledge in planning and design.

2 The Application Domain: Building Design

The application domain used to spell out the approach of task-oriented knowledge acquisition is building design. In particular, the focus is on the installation 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, transport of chemical substances, computer networks, phone cables, etc. Such a design project involves thousands of often incompatible objects in different stages of design, at different levels of abstraction, planned at different places, and by different engineers. The knowledge representation scheme we use throughout this paper is A4, which has been developed by L. Hovestadt [Hov93]. A4 allows to represent objects used in building design (e.g., rooms, paths, pipes) in a multidimensional design room. A4 may be described on a graphical level and on the code level.

On the graphical level, the most striking and irritating aspect of A4 is its use of ellipses. Utilizing ellipses instead of rectangles is a useful trick: Ellipses overlap only in a few points. Thus, ellipses permit a more condensed graphical representation of objects than rectangles do. The ellipses or circles refer to objects. Different states of our world correspond to different configurations of objects.

On the code level, each object (circle or ellipse) are represented by its spatial dimensions and nine further attributes like the time, at which the object was created, the subsystem, (e.g. used air, fresh air, rooms, paths, electricity, etc.) and morphology (e.g. development, connection etc.). This representation scheme will be used to produce graphics, the main basis for man-machine interaction in building design [GS88].

A complex domain like building design poses a number of requirements: A first global understanding of the domain and its peculiar problems has to be achieved, tasks to be tackled must be picked out, and the knowledge needed has to be identified. In addition, the part of the application domain, which is to be modeled has to be selected, the degree of support offered by the knowledge-based system ought to be specified, and the system has to be situated into the work-flow of the expert.

3 Knowledge Used In Building Design

In attempt to clarify the notions of "knowledge" and of "representation" the knowledge level has been introduced as the appropriate level for modelling the competence of a knowledge-based system [New82]. However, once an efficient rationale is required to relate knowledge configuration to knowledge application a more specific structure imposed on the knowledge-level is needed [VdV93, WVdVSA93]. While the knowledge-level principle is the common denominator of all approaches in knowledge acquisition there is to date no general agreement concerning the structuring of knowledge [WVdVSA93].

Apparently, problem solving in building design draws on different kinds of knowledge like tasks, cases, rules, and models. In our approach of task-driven knowledge acquisition, these kinds of knowledge are taken to structure the knowledge level. A clear-cut account of tasks, cases, rules, and models is required both in knowledge acquisition and system design. In this section, we are concerned with a description of tasks, cases, and rules, integrating them into a model of design knowledge. We will start out with an informal description of these kinds of knowledge and proceed by giving a knowledge acquisition perspective, then provide examples and elaborate the knowledge representation formally.

Our application domain, the construction of supply nets (e.g., air conditioning) in office buildings, confronts us with complex design tasks. Supporting architects during those tasks is the objective of the project FABEL, which started in 1992. We applied techniques of knowledge elicitation, acquisition, and modeling. The results showed the tremendous importance of workflow analyses, and how they can be used to structure the knowledge acquisition process. While we ignore in the present paper the important issue of the social embeddedness of both expertise and knowledge engineering (but see [SJKss]), we want to focus on the central role of task-structures in complex design. We believe that the objective of knowledge acquisition for knowledge-based system should be a competence model with respect to the tasks to be supported. Although two of the authors share a background in psychology, we do not aim at modeling architects, but their task-relevant knowledge. We will show how task analyses can guide knowledge acquisition and serve to construct a model of task-relevant knowledge. This model also provides a basis for system design and the assignment of subtasks to either the system or to the user in a flexible way that makes design support systems cognitively adequate [Str92].

According to our experience and to the literature as well [Kol93], design experts rely to a strong degree on cases; therefore, case-based reasoning (CBR) plays a major role in knowledge acquisition. There is evidence that cases are employed when complex problems are embarked that apparently have been solved before. In addition to cases rules also provide knowledge used to accomplish design problem solving. Rules, in turn, seem to be preferred when routine tasks are tackled. These two kinds of design knowledge are integrated with the task-structure, task interrelations, and task dependencies into a model of the domain.

3.1 Tasks

Tasks are the building blocks of model-based knowledge engineering. Often, quite general definitions are employed, like 'a task is something that needs to be accomplished' [Ste93, CJS92], or 'a task is a specification of a given problem in terms of input-output information at the computational level' [Goe89]. These definitions need to be linked to cognitive theories of problem solving, which usually characterize problem solving in humans and machines in terms of goal structures, e.g. [And83, NS72, Van89]. According to those and similar accounts from cognitive psychology, a problem consists of 'an initial situation, the actions that can be taken in seeking a solution, and the desired state of affairs, or goal' [Hol84]. Here, *goal* means the goal state, which could also be called the solution. Authors differ in their usage of 'solution', whether it is taken to be the result of successful problem solving alone, or together with the sequence of solution steps. In the following, we restrict ourselves to representations of the result. Is the goal then identical to the solution? This seems to be true only for a very restricted class of problems. Often, a goal is nothing more than a partial description of the solution. In chess, for example, the definition of a checkmate holds for an enormous number of possible mating positions. Design problems usually have a multitude of solutions.

Analyzing the work in complex design by even elementary knowledge engineering techniques (e.g., observation) yields the insight that design proper is only one side of expertise in this domain; its complement is strategic knowledge about how to break down large tasks into a non-arbitrary sequence of subtasks. The importance of this strategic knowledge has been acknowledged in the literature on human expertise [KS91, Van89]. Its objective, i.e., to define appropriate design task sequences, will be called *planning* in the following, in contrast to the term *design*, which we reserve for the elementary steps in the design process. We distinguish between tasks appearing in the domain (*application tasks* or and tasks the system has to fulfill (*system tasks*). Application tasks comprise both planning and design, and at least some of them can be delegated to the system. Issues of system design will be treated in the last section. Our objective is to present a framework for the construction of an abstract model of problem solving in the domain of complex design that is valid for both human and machine agents.

Tasks may be represented by a flat list or tangled hierarchies. The latter is also called *task-structure*. Tasks represented by a flat list just point out what to do. An unstructured list of errands is an example of a flat list of tasks. In addition, task structures represent dependencies between tasks. A structured list of errands represents dependencies between tasks is an example of a task-structure. Tasks structures allows for deriving the sequence

between different tasks to be achieved.

3.1.1 Tasks: A Knowledge Acquisition Perspective

Given a real world domain and aiming to support users in an task-oriented way we need to extract some *task-structure*. It can be used to acquire *case*-based and *rule*-based knowledge in an task-oriented way. This sets the stage for developing a *model of the domain*. The general idea of this methodology of knowledge acquisition is outlined in Fig. 1.

To establish a *task-structure*, firstly, the overall domain task (e.g., planning and design of supply nets in a building) is factored into manageable subtasks to be accomplished. This can be done along the physical components involved in the overall domain task. This task-structure reflects the structure of the physical components that occur in planning or design.

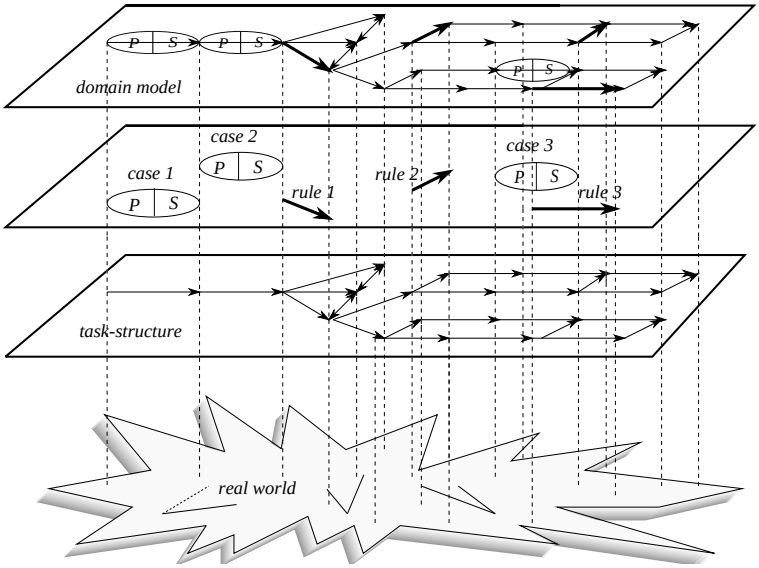

Figure 1: Task-driven knowledge acquisition

First, the domain and user support dependent task-structure is acquired. Second, cases and rules are determined corresponding to the tasks to be tackled. Third, the task-structure, cases and rules can be combined to constitute a domain model. – P denotes the problem part, S the solution part of a case.

Secondly, dependencies viz. interactions between tasks are specified. There are dependencies between two or more tasks if one task cannot be achieved without considering the realization of one or more other tasks.

Thirdly, input and output of each of the resulting subtasks is specified. In so doing, the flat list of subtasks is replaced by a structure made up of tasks and enabling relations. The task-structure is constructed in collaboration between domain experts and knowledge engineers using knowledge elicitation techniques that range from observation and interviews to highly structured methods (cf. [SJKss]).

Once the task-structure is specified it provides a useful platform for incremental knowledge acquisition, problem solving, and system design. In *knowledge acquisition*, elicitation of cases and rules may be advanced in a focused manner. Elicitating cases is supported by referring to instances of tasks as cases; eliciting rules by referring to knowledge that realizes task transitions as rules. A structured assembly of all types of knowledge together represents a model of the domain. In *problem solving*, the task-structure provides hints as to the subsequent tasks to be tackled. Thus, the task structure serves as coordinating knowledge to be used, e.g., for planning. In *system design*, the task-structure may be taken to select tasks, that can be supported by the system.

3.1.2 Tasks: An Example

Fig. 2 exemplifies the derivation of the task-structure in planning and design of supply nets in a building. The example is taken from the domain of building design. The arrows

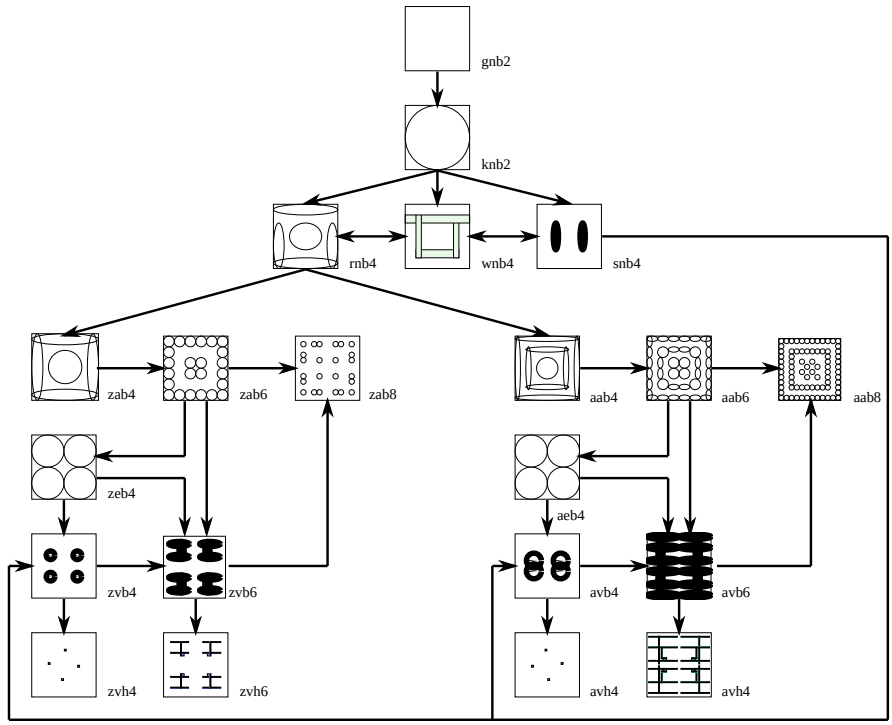

Figure 2: Task decomposition in building design

Tasks are described by expressions made up of three letters and a number. The first letter relates to the subsystem of the building (e.g., a=used air, g=building, k=climate, r=rooms, w=paths, s=shaft, z=supply–air). The second letter denotes the general function of the area addressed by the task (e.g., a=linkage, e= development, n=usage, v=connection). The third letter specifies the kind of resolution that is employed (e.g., b=area, h=bounding-box). The number relates to the part of the building that is envisaged (e.g., 2=building, 4=floor, 6=room, 8=areas within a room). For example, knb2 is used as an abbreviation of the task to plan or design the area of climate use that covers the whole building. The grey shaded part will be used in the next section to illustrate the acquisition of case-based knowledge.

in Fig. 2 indicate the standard sequence of tasks that are addressed while tackling the overall problem. The arrows neither point out the necessary requirements that have to be fulfilled to tackle a particular task nor do they exclude other sequences an architect may possibly choose when addressing these tasks. Simple arrows refer to a transition between two adjacent tasks with the former providing input to the latter. For example, the ground plan *gnb2* provides input for the task of designing the areas of a building that are intended to have a homogeneous climate *knb2*. The double arrows connect two adjacent tasks that interact strongly (loop), i.e., they provide input to each other. For example, the layout of the rooms *rnb4* and the layout of the paths *wnb4* in a building have to be compatible with each other.

3.2 Cases

During design architects frequently browse through old drawings. Case-based reasoning seems to be an appropriate reasoning method to support design tasks (see [Goe89, NC91, Hin92, HF93, DK92]). *Cases* are specified as instances of tasks. They provide episodic or specific knowledge about state transitions. Applying CBR, the domain of discourse is represented by a finite set of already solved cases (stored in the case base CB) and a similarity (often equivalence) relation σ over them. Case-based reasoning proceeds as follows. Given a new problem, cases with (modulo the similarity relation) similar problems are selected from a case-base. The solution of the most similar case is transferred to the new problem and adapted if necessary. Storing the new problem including its solution and updating the similarity relation can be seen as a kind of learning [Aam90] Detailed introductions into CBR may be found in [RS89, Kol93].

Until now, CBR has mainly been applied to analytic tasks. Here properties or symptoms and corresponding concepts or diagnoses are quite ready at hand. The former are represented by the problem, the latter constitutes the solution of a case. There is an ongoing debate how to cut cases in synthesis tasks. Issues at the stake in this debate are the *method to cut cases* and as a consequence the *grainsize of cases*. Imagine the design of buildings. Concerning the method, cases may be cut ad lib from an overall architectural plan. In a word, no particular method is applied. Cases may, on the other side, be derived by using a task decomposition. The task decomposition, in turn, rests upon an analysis of the problems that occur in the domain [DPvC$^+$87]. Concerning the grainsize, in an application domain like architecture complete buildings have been taken as cases [Goe89, DK92, Hin92]. Again, a method like a task- decomposition may be employed to accomplish units which are better suited for problem solving.

Note, that the main reason for employing a well-founded method for cutting cases is not the size of cases as such but the need to come to cases that are usable in case-based reasoning. Cases should reflect and support the way how architects design buildings. That is, they should allow tackling specific problems that are known to reoccur in the application domain. Embarking on specific problems becomes extremely hard if the grainsize of the problem to be solved, e.g., the design of the supply air system, and the grainsize of the case, e.g., a complete building does not match. Such a mismatch is likely to occur if

no particular method for cutting cases is employed or oversized units in the domain are viewed as cases.

Case problems should represent an initial state, case solutions its goal state. Thus, cases relate *immediately subsequent* states for means of reasoning. Additionally, in domains like building design, not only knowledge about attributes of case objects is important but also their inherent relations. Reasoning in design proceeds over complex structures. Knowledge representation schemas have to meet this fact. To formalize this kind of knowledge we use an algebraic approach, i.e., case problems and solutions are represented by terms or graphs over some underlying algebra. Cases may be represented in an *attribute-based* or in a *structural* way. The first representation scheme provides the object attribute values with respect to a given finite number of attributes as usual in CBR (cf. [RS89, Kol93]). The structural representation gives the relations between objects of cases. While the former permits fast preselection of candidate cases out of large case-bases, the latter is suitable to guide adaptation.

3.2.1 Formalization

Knowledge representation. All these considerations lead to the following definitions of cases and similarity relations. Let A denote an *attribute-based* description of a case problem or solution, g denotes a *structural* description (e.g., graph-based description), and the superscripts p and s denote *problem* and *solution* descriptions. We assume the existence of a transformation function ϕ and its inverse that transforms attribute-based representations into structural representations and vice versa:

$$\exists \phi \mid \phi(A^p) = g^p \wedge \phi(A^s) = g^s \wedge \phi^{-1}(g^p) = A^p \wedge \phi^{-1}(g^s) = A^s$$

A case c is represented by

$$c := \langle A^p, g^p, A^s, g^s \rangle.$$

We denote the set of all cases by

$$CB := \{c_1, ..., c_{m_{CB}}\}.$$

To define similarity of prior and actual case problem descriptions we use two different similarity relations, namely *surface similarity* and *structural similarity*.

Surface Similarity. To determine the so called *surface similarity* of attribute-based case representations we need a similarity function σ_a. The similarity function (often a metric) returns a single number between zero and one, i.e.,

$$\sigma_a(A_1, A_2) \in [0, 1],$$

which is meant to reflect all aspects of similarity. The similarity function based on some distance measure holds the following axioms:

$$\begin{array}{ll} \sigma_a(A_1, A_1) = 1 & \text{(reflexive)} \\ \sigma_a(A_1, A_2) = \sigma_a(A_2, A_1) & \text{(symmetric)} \\ \sigma_a(A_1, A_2) = 1 \wedge \sigma_a(A_2, A_3) = 1 \rightarrow \sigma_a(A_1, A_3) = 1 & \text{(transitive)} \end{array}$$

and therefore $\sigma_a(-,-) = 1$ is an equivalence relation.

Structural Similarity. To determine *structural similarity* σ_g of a given set of structurally represented problems we determine their most specific common structure $MSCS^p$. Modifications which lead to this structure are called *proper* and are stored in σ_g. Given an actual problem we apply *proper* modifications given by the similarity relation to it and compare the result with $MSCS^p$. Given equality we transfer the prior solution to the actual problem. Thus, σ_g is defined as the union of proper modifications r (e.g., modification rules like generalization, abstraction, or geometrical transformation), which relate a set of structural problem descriptions $g_i^p \in G,\ i = 1, ..., n$ to each other. That is

$$\sigma_g := \cup_{i=1}^{n}\ r_i \text{ satisfying } r_i(g_i^p) = MSCS^p,\ i = 1, ..., n.$$

The unique term $MSCS^p$ is called the *most specific common structure*. For any other specific common structure g of G^p there exists a modification r s.t. $r(MSCS^p) = g$. Modifications are formalized by *inverse substitutions*, i.e., mappings from constants into variables. A detailed description can be found in (cf. [Bör94b]).

Reasoning. These kinds of knowledge representation enable two-stage similarity assessment [GF91] and reasoning. We are able to combine computationally cheap, fast preselection by *surface similarity assessment* and the computationally expensive process of *structural similarity assessment*. Reasoning proceeds as follows:

First, we determine a set of so called *candidate* cases using the attribute-based representation of prior cases, the actual problem description and the similarity measure σ_a. Second, we use the structural case representation to determine structural similarity which enables to guide adaptation. Therefore, as in standard CBR, we need first to know about the similarity relation σ_g over a set of *candidate* cases. Thus, we determine σ_g and $MSCS$ out of the problem descriptions of the candidate cases. Now we transfer the actual, attribute-based problem description into its structural representation using the transformation function ϕ. Next, we apply modification rules (e.g., rotate about 90 degrees or substitute some concrete number of objects by variables) to the structural description of the actual problem.

Rules in σ_g are applied until the modified problem is syntactically identical with the *most specific common structure* of the set of candidate cases or all rules are tried. Given identity, the solution of this case set (which may or may be not modified) can be transferred to the new problem. After that, inverse modification rules (e.g, rotate about minus 90 degrees or replace the variable by a definite number of objects) are applied. In this way, we get the concrete structural representation of the new solution. Adaptation is done implicitly by representing the solution operationally and transferring it to the actual problem. Using the inverse transformation function ϕ^{-1} we derive the attribute-based representation of the actual solution. The full formalism can be found in ([Bör94a, Voß94]).

3.2.2 Cases: A Knowledge Acquisition Perspective

Having specified a task-structure, a kind of template sheet is introduced into the domain that may be employed for discerning and acquiring cases. Projecting such a template

sheet or grid onto a domain is nothing more than an attempt at making up for natural units that are ambiguous, hidden, or even missing. This is an indispensable requirement for using case-based reasoning in domains that have no units suitable to be referred to as problem or solution. As with any other structure that is used to discriminate cases within a complex domain like building design, the task-structure makes strong commitments concerning the representation, content, and possible use of cases acquired in this way.

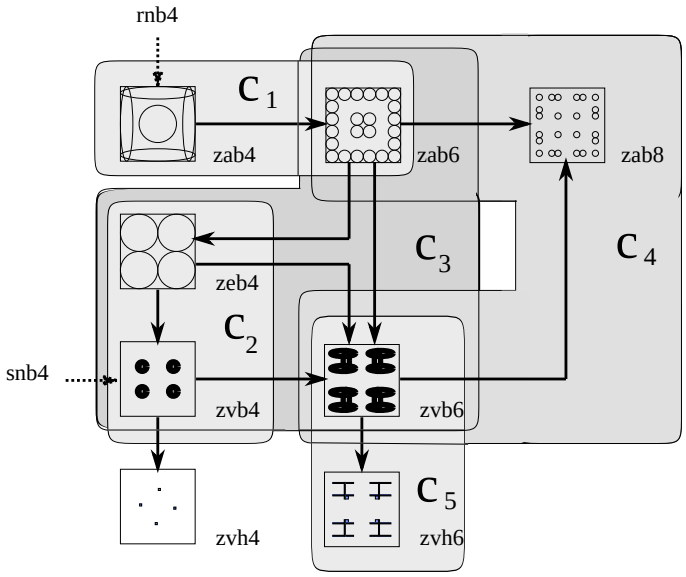

Figure 3: Cutting cases for case-based reasoning

Fig. 3 illustrates the acquisition or cutting of cases by using some part of the task decomposition introduced in Fig. 2. Dotted arrows represent dependencies among of states not depicted here. In synthesis tasks, parts of a solution of a task constitute the problem description for a subsequent task. For instance, in Fig. 3 the case c_1 consists of the problem *zab4* and its solution *zab6*. Given *zab6* we may start to design *zeb4*. The solution of *zeb4* is *zvb4*, see case c_2. Next, given *zab6, zeb4* and zvb4 representing the new problem we are able to design *zvb6* recorded by case c_3. Case c_4 represents the design step to derive *zab8* out of *zab6* and *zvb6*. Case c_5 depicts the use of *zvb6* to design *zvh6* etc.

There is no hardwired part of a case that may be referred to as problem or solution. This tricky issue may be resolved by referring to problem and solution as "roles". A considerable part of problem solving is figuring out dynamically which part of a situation or a case may be assigned to the role of the problem.

To represent cases in an attribute-based way we use the A4-model of organizing data in a multidimensional dataspace [Hov93]. Each object is represented by its spatial dimensions (i.e., x, dx, y, dy, z, dz) and further attributes like the time, at which the object was created, aspect, (e.g., return–air, supply–air, electricity, construction) and morphology, (e.g., development, connection) etc. Similarity relations σ_a over attribute-based case descriptions are usually handcoded.

As an example for structural case representations we will show one which is graph-based. Arrangements of objects and their relations are represented in the following way. Objects are represented by nodes with labels corresponding to the concrete object attribute values. Relations between objects are represented by edges labeled with the concrete relations. For communication nodes are labeled with arabic letters, edges are labeled with roman numbers. As an example, case c_2 of Fig. 3, may be represented by:

```
;;; case c2
;;; problem - a4 position zeb4
;;; data
(project/home/user/FABEL task-deco.hdraw)
(viewPosition 46.350723 24.8507234 0.155312 0.155312)
;;; A^p
a       (298 1440 160 1440 0 420 0.01 supply–air development)
b       (298 1440 1600 1440 0 420 0.01 supply–air development)
c       (1738 1440 1600 1440 0 420 0.01 supply–air development)
d       (1738 1440 160 1440 0 420 0.01 supply–air development)
;;; g^p
I p1(a,b,c,d)
II p3(a,b,c,d)
;;; eof
;;; solution - a4 position zvb4
(project/home/user/FABEL task-deco.hdraw)
(viewPosition 46.350723 24.850723 0.155312 0.155312)
;;; data
;;; A^s
e       (1018 240 1000 240 0 420 0.03 supply–air connection)
f       (1018 240 1960 240 0 420 0.03 supply–air connection)
g       (2218 240 1960 240 0 420 0.03 supply–air connection)
h       (2218 240 1000 240 0 420 0.03 supply–air connection)
;;; g^s
III     p3(e,f,g,h)
IV      p2(a,e)                 VI      p2(c,g)
V       p2(b,f)                 VII     p2(d,h)
;;; eof
```

The case problem is represented by attribute values characterizing four objects (labeled a to d) and their inherent relations (labeled I and II). The solution is represented by attribute values for another four objects (e to h) and their inherent relations (III). Relations IV to VII relate solution and problem objects. There are different kinds of relations. In this example we used:

$p_1(x_i)$:=objects $x_i, i = 1, ..., n$ touch each other,

$p_2(x, y)$:=object x covers object y, and
$p_3(x_i)$:=objects $x_i, i = 1, ..., n$ are disjunct to each other.

The corresponding graph of problem and solution is represented in Fig. 4. Double arrows denote edges, which represent relations combined of p_1 and p_3. Dotted double arrows represent the relation p_3. Relation p_2 is represented by simple arrows. Thick arrows denote the transformations ϕ and ϕ^{-1} which transform the attribute-based representation into the structural one and opposite.

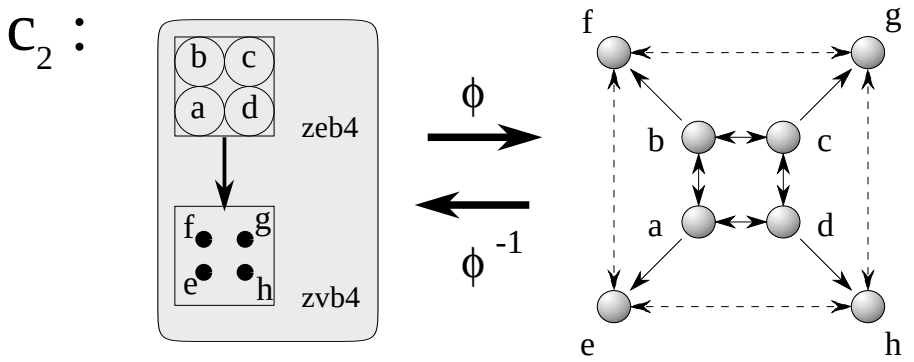

Figure 4: Graph-based representation of case c_2

To realize structural similarity assessment we use proper case modifications and their inverses (e.g., generalizations, abstractions, or in the domain of building design: geometrical transformations). They are handcoded or extracted from the structural case representation. The latter can be done by determining proper modifications, which lead to the common structure of cases with equal solutions. The common structure of a case set will be represented by graphs labeled with variables instead of constants.

3.2.3 Cases: An Example

As an example we use the design of bounding boxes out of areas of supply–air connections. Given *zvb6* the problem is the design of *zvh6* (see case c_5 in Fig. 3). As a starting point knowledge about prior cases is required. This is provided in the knowledge-base together with a set of proper modification rules used to derive their common structure (see Fig. 5). Now, given an actual new problem situation *zvb6*, in which way is the actual solution derived by using the knowledge stored in the knowledge-base?

We start by determining possible candidate cases dealing with the same design step. That is, we look for prior problem descriptions containing the attribute values {*supply–air connection, area*} using *surface similarity assessment*. The result is $c_{5,1}, ..., c_{5,4}$. Additionally, we need corresponding proper modification rules, e.g., *generalization, rotation* and *reflection*.

Now we are able to do *structural similarity assessment* and *adaptation*. By comparing object relations of the previous and the new problem we determine the most similar

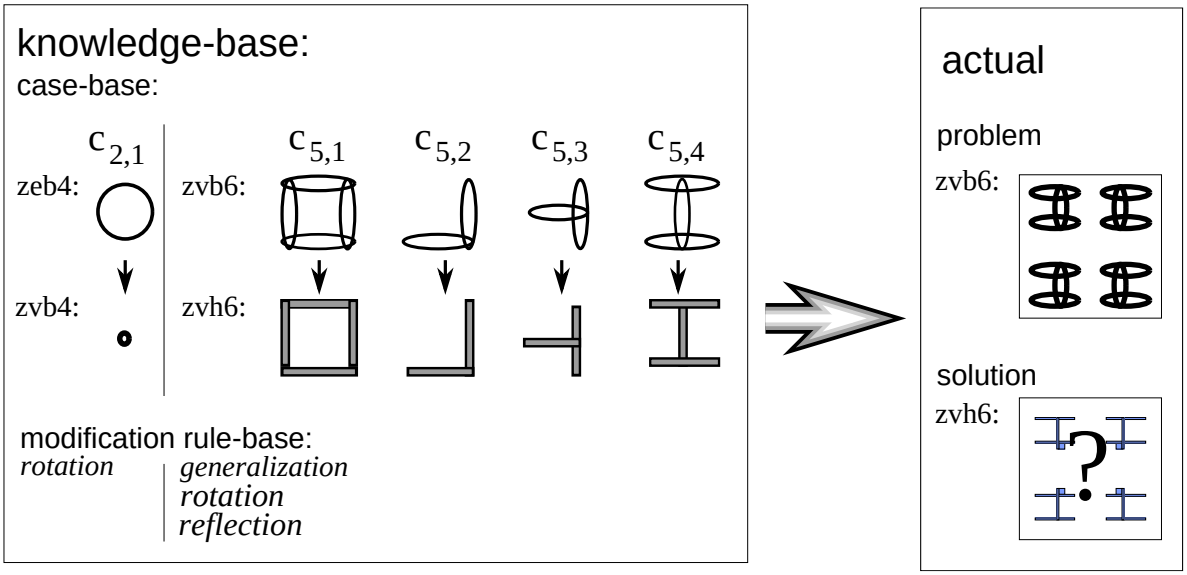

Figure 5: The problem

case out of the case-base, i.e. $c_{5,4}$. But the problem description of this case is far from being identical with the actual problem. Using the proper modification *generalization* of attribute values for x, dx and y we determine the common structure of the prior problem of $c_{5,4}$ and the upper left part of the actual problem description (see Fig. 6). To illustrate the most specific common structure of both problems we used dotted lines for variable x, dx and y values. Note, that structural relations remain identical here. Given structural similarity, we transfer the prior solution to the actual problem part. Given this solution, we solve the remaining three problem parts by using *rotation* and *reflection*. Finally, the actual problem including its solution will be stored in *CB*.

3.3 Rules

Apart from *cases*, *rules* are used to accomplish tasks. Rules are derived by generalizing transitions between tasks. Thus, rules provide generic knowledge about task transitions. At the most general level, once a set of conditions on the left hand side of a rule is met, the right-hand-side of the rule will produce a certain effect. This effect can be employed to drive reasoning that spans across different types of problem solving like planning, design, or verification. For example, when designing the network for air supply in a certain part of the building, rules are employed to make sure that the task is attained, i.e., the network for supply air in a certain part of a building is actually realized.

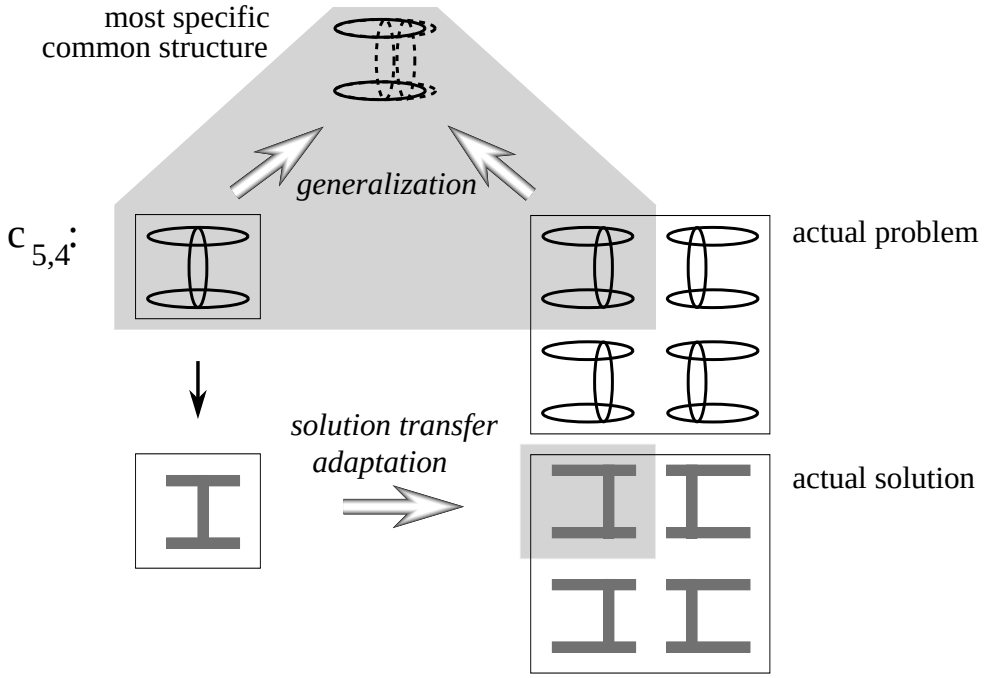

Figure 6: Structural similarity assessment and adaptation

3.3.1 Formalization

In general, there are certain requirements for accomplishing a task t, i.e., relations between a realization of t, denoted by t^*, and realizations of some other tasks [JJ94]. Let $t_1, \ldots, t_n$ be the tasks relevant for the verification, and let $t_1^*, \ldots, t_n^*$ denote some realization of them. The requirements may be represented by predicates, e.g., denoted by $P_0, \ldots, P_n$, with each of them having a certain arity. For example, P_c may represent the condition of *covering*, i.e., P_c may be binary and $(t_{i_1}, t) \in P_c$ may refer to the requirement that the realization of task t_{i_1} should cover the realization of task t.

A rule may be referred to as a mechanism yielding a realization for some task t, such that t^* meets the predicates $P_0, \ldots, P_n$ corresponding to a certain t_i. Let us denote this process of arriving at a realization by $G^{(P_0 \ldots P_n)}$. This process may be split up into two steps: A *generation* G and a *selection* S. The process of unconstrained generation of all possible sets of objects could be formalized by $G^{\emptyset}$. We use the notation $S_{P_i^T}$ for a selection process regarding the predicate P_i^T. Thus, we could model the rule $G^{(P_0 \ldots P_n)}$ as $S_{(P_0 \ldots P_n)} \circ G^{\emptyset}$. In most cases, however, the unconstrained generation of all possible sets of objects ($G^{\emptyset}$) is not feasible. Instead, we prefer a constrained generation of solutions that integrates as many predicates as possible leaving the remaining predicates for the selection.

Hence, our rules are of the type $S_{(P_{j_1} \ldots P_{j_k})} \circ G^{(P_{i_1} \ldots P_{i_m})}$

3.3.2 Rules: A Knowledge Acquisition Perspective

The formal description of rules presented above provides a rationale to be used in knowledge acquisition directed at rules. The approach of deriving rules relies heavily upon the task-structure introduced above. Tasks involved in a rule are taken to be free of interactions. This of course limits the applicability of rules in our system severely. Nevertheless, rules can be used and formulated for quite a range of design tasks, which are more or less 'routine'. Accordingly, when observing and interviewing our domain experts we found that routine construction tasks are those where they indeed apply rules and prefer rule application to the use of cases. With respect to rules these tasks are nothing more than input states and output states. Basically, there are three steps to be taken in order to realize knowledge acquisition directed at rules.

- Determine the task transition to be represented by a rule.
- Assign the tasks involved in the chosen task transition to the input tasks or the output tasks of the rule.
- Specify on the right hand side the conditions, i.e., predicates $P_0, \ldots, P_n$ to be met by a possible solution.

3.3.3 Rules: An Example

We present two examples for *rule application* that relate to the design of a supply air system. In particular, the generation of *zab6* and *zvb4*, viz., the design of areas for the installation of pipes for this kind of system are concerned. As a kind of basic rule we realize the partition of an area into a number of subareas. This is abbreviated by G^Z (cf. Fig. 7). The rule that realizes the task *zab6* (zab6-rule) is nothing else than G^Z applied to the number of objects that represent zab4.

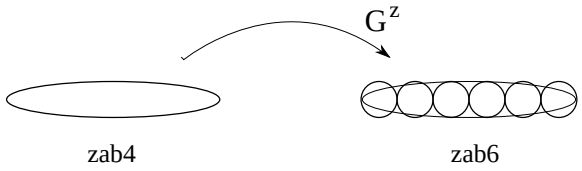

Figure 7: Application of a rule for partition

The zab6-rule - as any other rule - may be reemployed by other rules. For example, the zvb4-rule is a combination of the G^Z-rule and a selection of certain predicates, in particular distance and number, which may be noted by S_P. Thus, our rule can be abbreviated by $S_P \circ G^Z$ (cf. Fig. 8).

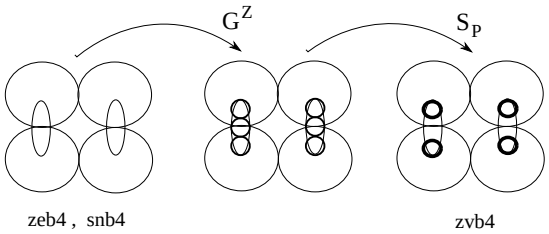

Figure 8: The *zvb4*-rule modeled as a combination of partition and selection

3.4 Integrated Models of Knowledge

The most elaborated kind of knowledge to be acquired is knowledge encapsulated into domain models. Models may be distinguished according to the aspects they preserve. There is the widespread distinction between structural and behavioral models [Gen84]. An integrated circuit, for example, may be represented by focusing on the structure or on the functional behavior. The structural model encompasses the physical components and their interrelations. The behavioral model captures the stimuli and responses and their interrelations. Each of those models whether they be structural, behavioral or whatsoever is not a purpose in itself but a means to support a particular function. Recently, this idea has attracted considerably attention both in cognitive science and in artificial intelligence under the heading of change of representation and problem solving as modelling (e.g. [BR90, Ind92, VdV93]). For example, preferring a causal view of the functioning of a car implies an assumption that this causal view leads to an adequate approximation of the functioning of the car, which allows for further problem solving, e.g., diagnosis. (cf. [VdV93]). Hence, the type of aspects described by a model has to allow for fulfilling the function, which the model is intended to support.

The model we develop in knowledge acquisition is a structural model of a real-world phenomenon like a building. It describes physical components, their structural features, and the relations between them that are required to realize, i.e., to plan and to design, such a building. Such a model represents knowledge about sequences of design steps like tasks do. It also comprises methods suitable to embark on concrete design tasks, viz., cases and rules. Additionally, it contains meta-knowledge about how to combine all knowledge in a consistent and reasonable way. This kind of knowledge may be represented by descriptive interrelations and dependencies between the aforementioned kinds of knowledge. Its purpose is to guide the process of problem solving and the selection of appropriate problem solving methods. The model of the domain, defined on the set of tasks will consist of following parts:

- the structure of descriptive interrelations
- the task-structure, which can be divided into
  - the structure of decomposition
  - the structure of dependencies

- the structure of cases
- the structure of rules

We use T to denote the set of tasks. As a certain state of a concrete project we mean a substitution of each task $t \in T$ with a set of certain domain-objects (in our case objects). The realization of a task t is termed t^*. Thus, $t^* = \emptyset$ means that task t has not been designed yet. If V is a set of tasks we denote the set of realizations of these tasks by V^*.

3.4.1 The Structure of Descriptive Interrelations

We presume that relations between task-realizations may be specified on the computational level, i.e., there are algorithms $\alpha_1, \ldots, \alpha_m$, with certain arities (e.g., n_j for α_j) that decide for any n_j-tupel of sets of domain-objects (especially task-instances) $(t^*_{i_1}, \ldots, t^*_{i_{n_j}})$ whether or not $\alpha(t^*_{i_1}, \ldots, t^*_{i_{n_j}}) = \text{TRUE}$ holds. Let $(P_i^T)_I := (P_1, \ldots, P_m)$ be predicate symbols, one for every α_j, with the same arity as α_j. So if we have $(t_{i_1}, \ldots, t_{i_{n_j}}) \in P_j^T$ we want $\alpha_j(t^*_{i_1}, \ldots, t^*_{i_{n_j}}) = \text{TRUE}$ to hold. Thus, we can define the *structure of descriptive interrelations* by the pair $(T, (P_i^T)_I)$ In the domain of building design the following predicates proved to be very useful [JJ94]:

(i) Let $t \in P_D$ denote that the objects realizing t are disjoint

(ii) Let $(t_1, t) \in P_C$ denote that the realization of t_1 covers the realization of task t

(iii) Let $(t_1, t) \in P_N$ denote that the objects realizing t_1 are equal in number with the objects realizing t

We could thus collect all requirements on the solution of a task t that the model represents by

$$\bigcup_{j=1}^{m} \{(t_{i_1}, \ldots, t_{i_{n_j}}) \in P_j \mid \exists k \in \{1, \ldots, n_j\} : t = t_{i_k}\}$$

3.4.2 The Structure of Decomposition

Let $S(t) := \{t_{i_1}, \ldots, t_{i_{n(t)}}\} \subset T$ be a set of tasks that can be regarded as a decomposition of the task t. That means a solution for the sub-tasks is a solution for the top-task t. On this level of description, the problem of tackling the top-level task t by tackling the sub-tasks in $S(t)$ can be represented by the pair $(S(t), t)$. Establishing $S(t)$ is quite straightforward if the task renders itself into a partition, e.g., according to a set of physical components denoted by $p_1, \ldots, p_{n(t)}$. Each P_i^T may or may not be specified in different degrees of specification. We may associate with each physical component P_i^T a task t_i. There may be various ways of decomposing t. Let S_t be the set of possible decompositions of t. Thus, we can define the *structure of decomposition* by the pair $(T, S^T := \bigcup_{t \in T} S_t)$, with the relation S^T representing the sub-task-decomposition of tasks.

3.4.3 The Structure of Dependencies

As a possible *dependency* concerning a task t we understand a set V of tasks t_i the solutions of which are sufficient for the user to design t. t may be accomplished in more than one ways. Each alternative way of accomplishing a task t will yield a different pair (V, t). So we collect in D_t all pairs $(V, t) \in \mathcal{P}(T) \times T$, such that t can be viewed as dependent on V. Thus, we can define the *structure of dependencies* as the pair $(T, D^T := \bigcup_{t \in T} D_t)$, with the relation D^T representing dependencies between tasks and sets of tasks.

Figure 9 shows a part of the relation D^T in the domain of building design, concerning the supply-air (abbreviated by the letter z). The diagram illustrates that the design of *zab6* depends on the realization of *zab4*, the design of *zeb4* depends on the realization of *knb2* and *zab6*, the design of *zvb6* depends on the realization of *zab6*, *zeb4*, *zvb4*, and so forth.

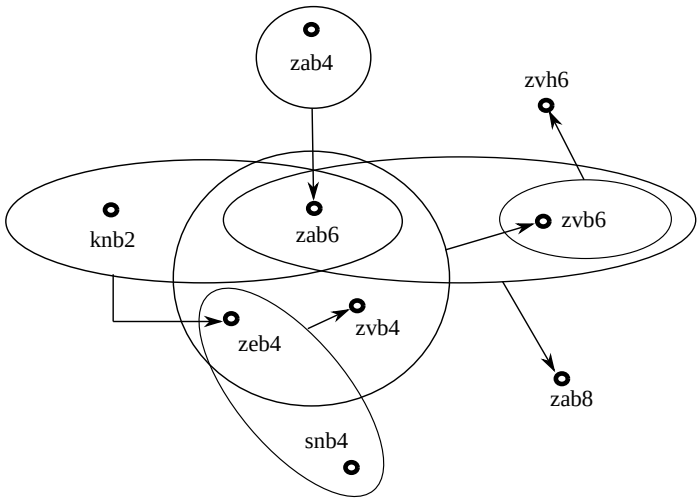

Figure 9: Graphical representation of dependencies in the domain of building concerning the supply air

Specification of dependencies. If a task t can be viewed as dependent on the tasks in V we could specify the dependency represented by $(V, t) \in D^T$ according to the relations $P_1, \ldots P_m$ by collecting the set

$$\bigcup_{j=1}^{m} \{(t_{i_1}, \ldots, t_{i_{n_j}}) \in P_j \mid \forall k \in \{1, \ldots, n_j\} : t_{i_k} = t \vee t_{i_k} \in V\} \ .$$

As an example we take a look at the task *zvb4*. We have $(\{zeb4, snb4\}, zvb4) \in D$, i.e., the realization of the *zvb4* depends on the realization of the *zeb4* as well as on the *snb4*. The dependency described by $(\{zeb4, snb4\}, zvb4) \in D$ could thus be specified by $(snb4, zvb4) \in P_C$ and $(zeb4, zvb4) \in P_N$. In addition, we have the requirement $(zvb4) \in P_D$.

The t_i in V with $(V, t) \in D^T$ represent tasks that are (if realized) sufficient for the user to

find a solution for t. The specifications concerning those $t_i \in V$ and t need *not* be sufficient to define a realization for t, i.e., the specifications represent only necessary conditions.

3.4.4 The Structure of Cases

In CBR, the domain of discourse is represented by a set of cases stored in CB together with a similarity measure σ over these cases. The attribute-based problem state A^p and its structural representation g^p correspond to states $t \in V$ which are necessary to consider in order to solve a some task t. The solution of a case represented by A^s and g^s corresponds to the realization of the task t denoted by $t^* \in V^*$. Thus a case may be represented by a tuple (V, V^*). A new problem corresponds to a tuple $(V^*, \emptyset)$ meaning that $t^* \in V^*$ has not been designed yet. Our aim is to get a function $f_{[CB,\sigma]}$ representing the knowledge given by the CB and σ which applied to a case-problem, yields its solution, i.e.:

$$f_{[CB,\sigma]}(V) = V^*.$$

Considering the knowledge encoded into the set of tasks T the *structure of cases* may be represented by

$$(T, f_{[CB,\sigma]}).$$

Case-based knowledge represented by $f_{[CB,\sigma]}$ is suitable to design the solution to a current problem.

3.4.5 The Structure of Rules

A rule r_k with arity n_k yields for any n_k-tupel of finite sets of domain-objects $(s_1, \ldots, s_{n_k})$ a finite set of domain-objects s, so that certain relations P_j hold between the input and the output. Let be $r_k(s_1, \ldots, s_{n_k}) \equiv s$, so we demand $\alpha_j(s_{i_1}, \ldots, s_{i_{n_j}})$ to be TRUE for certain α_j and $(s_{i_1}, \ldots, s_{i_{n_j}}) \in \{s_1, \ldots, s_{n_k}, s\}^{n_j}$. Thus, rules could be represented by strings of the following kind:

$$r_k(s_1, \ldots, s_{n_k}) \equiv s \rightarrow \bigwedge \alpha_j(s_{i_1}, \ldots, s_{i_{n_j}}) = \text{TRUE}$$

in which the elements of the conjunction depend on the rule r_k.

We collect in R the rules and define the *structure of rules* by (T, R).

3.4.6 The Model of the Domain

The model of the domain can now be summoned by collecting all former defined structures. The model may thus be denoted by

$$\mathcal{M} := (T, (P_i^T)_I, S^T, D^T, R, f_{[CB,\sigma]}).$$

We would like to point out that a complete domain model is an ideal. It would certainly not be feasible as a realistic objective for knowledge acquisition, since a complete model is dynamically extended in itself whenever new knowledge - be it additional cases, or

whatever - is acquired. But for the very same reason, this model provides the guideline for incremental knowledge acquisition. Its very incompleteness *because* of its extensibility, makes it superior to static models in terms of cognitive adequacy. This will be shown in the following section.

4 Interrelations in Knowledge Acquisition and Problem Solving

In the previous section, we have introduced different kinds of knowledge that enter the knowledge base of a design support system. We will now investigate how these kinds of knowledge interrelate in knowledge acquisition and problem solving. In particular, our attention is attracted to three aspects of interrelations in knowledge acquisition and problem solving. First, the focus is on the starting point of task-driven knowledge acquisition, i.e., the concept of a task. Second, the notion of knowledge acquisition as an incremental process is addressed. Third, interrelations between knowledge configuration and knowledge application [WVdVSA93] in problem solving given a particular state in knowledge acquisition are outlined and discussed.

4.1 Task-driven Knowledge Acquisition

Broadly speaking, a task is an objective or goal relating to thing or states to be attained. [Ste93]. Issues start to get tricky once this concept is used in a more specific sense. In general, there are various usages of the notion of a task. First, the notion of a task may be used exclusively with respect to human problem solving or with respect to mapping of aspects of human expertise to knowledge-based systems. Second, the notion of a task may have a general flavor with generic, i.e., reusable tasks reappearing allegedly across different domains [Cha83], or they may be very domain specific. There is a trade-off between highly general tasks on the one side and highly specific tasks on the other side: While general tasks may be re-used in top-down knowledge acquisition specific tasks allow - by definition - for a closely tailored fit to the domain and provide guidance for incremental knowledge elicitation in particular to case-based knowledge. Third, tasks may be described on different levels of specification, i.e., by referring to the input and output of tasks [Goe89], the goal to be accomplished in tasks, or the method, strategy, or inference structure taken to realize the task. Sometimes tasks are identified with goals or problem types [CJS92] or with a goal and a method how to achieve that goal [BWvS$^+$87]. Finally, tasks may be represented by an unstructured set, or by a task-structure, which organizes tasks and subtasks. Task structures may or may be not specified with respect to methods used to realize tasks or interactions between tasks.

Our usage of the notion of task is concerned with mapping of aspects of human expertise to knowledge-based systems. Since our concept of task has accrued from a bottom-up approach it is closely tailored to the requirements of our domain. Specifications of input and output, the goal to be accomplished, methods taken to realize the task are also included. Task structures made-up of task-subtask links are also provided. These are

enriched by methods, i.e., cases or rules and by a specification of interactions between tasks. Enlarging a set of task to a full-blown task-structure is accomplished in incremental knowledge acquisition.

There are various approaches to *identifying and elicitating tasks*. For instance, orientating at a physical decomposition, analysing requirements, using a particular problem solving method the application of which enforces a particular task decomposition [Ste83] or adopting task decomposition schemes traditionally used in the domain are alternative heuristics used to identify tasks. Various heuristics to task decomposition may or may not yield the same task-structure. If a domain may be decomposed into physical components like, e.g., in architecture it is a natural way to take this partition as a rational to identify tasks. Thus, "pipes to transport supply air" may be referred to as physical components or as the task to plan and design this very physical components. Fixing the *grainsize of tasks* is usually realized once a particular heuristic to task identification is chosen and applied. The decision concerning the choice of the rational for a task decomposition influences the identification and elicitation of other kinds of knowledge. For example, rules are specified as procedural knowledge used to accomplish transitions between tasks. Thus, the choice of a particular grainsize for tasks influences the kind of rules that are candidates for knowledge acquisition [JB93].

4.2 Incremental Knowledge Acquisition

In what follows, we address the notion of knowledge acquisition as an incremental process. This orientation marks a contrast to previous research in knowledge acquisition since hitherto not very much attention has been paid to the fact that knowledge acquisition is a process that develops in time. There is no doubt that in reality knowledge acquisition can hardly be conducted without dead-ends or revisions. However, each step in knowledge acquisition should gain maximal profit from what has already been acquired in the preceding steps. This is not always possible. Sometimes, there are incremental extensions of knowledge to be acquired, which are supported by the knowledge already acquired, and sometimes there are none. We draw a distinction between different kinds to proceed in knowledge acquisition: In true *incremental knowledge acquisition* a preceding step supports a subsequent step. For example, elicitating a set of tasks followed by an elicitation of cases that are instances of tasks is an example of incremental knowledge acquisition; elicitating tasks supports elicitating and representing cases since tasks provide a guidance how to partition cases. In mere *additional knowledge acquisition* a subsequent step is taken without any support by a preceding step. Cast in these notions an important requirement to meet in knowledge acquisition is to maximize the steps of incremental knowledge acquisition while the steps of additive knowledge acquisition should be kept at a low level.

This distinction between incremental and additive knowledge acquisition is also true for task-driven knowledge acquisition. If a set of tasks T is taken to be the foundation and beginning of knowledge acquisition, there is a number of alternative possibilities to proceed and to enlarge the knowledge base (cf. Fig. 10).

Note, that we do *not* advocate for a deterministic sequence of steps in knowledge acqui-

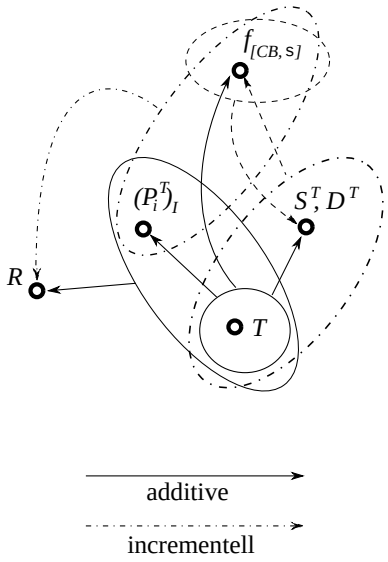

Figure 10: Dependencies in task-driven knowledge acquisition

sition and do not exclude other possible tracks. However, once a set of tasks T is taken as a starting point in knowledge acquisition and system design there are a number of tracks that are advantageous since using them permits an incremental knowledge acquisition. The acquisition of the task-structure S^T and D^T as well as the relational structure $(P_i^T)_I$ is based on a set of tasks T. Expanding the knowledge acquisition by case-based knowledge represented by $f_{[CB,\sigma]}$ is supported by having a task-structure (T, S^T, D^T) that points out important areas of a domain to be covered by cases and also the grainsize of cases. Adding rules R to the knowledge-base rests upon corresponding predicates $(P_i^T)_I$ on T (cf. Fig. 10).

4.3 Incremental Problem Solving

Taking an incremental approach, knowledge acquisition can be used for incremental system development such that the capabilities of the knowledge based system to be built are gradually enlarged. There are different types of knowledge within the space of all states of knowledge that may be acquired incrementally. In more general terms, these kinds of knowledge may be referred to as coordinating or realizing knowledge. Knowledge used to coordinate problem solving - e.g., tasks T and the dependency-structure thereof (T, S^T, D^T) - is used in planning. Knowledge used to realize problem solving - e.g., cases $f_{[CB,\sigma]}$ and rules R - are employed in design. Knowledge used to coordinate problem solving and knowledge used to realize problem solving have to be well balanced. Only concentrating on knowledge that realizes problem solving, which is not counterbalanced by knowledge that coordinates it will not enhance the overall problem solving capabilities of the system. Whether or not both kinds of knowledge are balanced well depends on the track taken in incremental knowledge acquisition and system development. In the remainder of this section we give an exemplary account about the kinds of problem solving

that is possible given a particular kind of knowledge.

Providing a Task Control List. Incremental task-driven knowledge acquisition sets out with a set of tasks T provided for problem solving and further knowledge acquisition. Which type of problem solving may be successfully realized by a system in this stage? Seen from the viewpoint of a simple single-task problem (e.g., simple diagnosis) there does not seem to be any need for using tasks in problem solving at all. However, a more complex real world problem like design is a different cup of tea. Addressing such multi-task problems requires devices for coordinating problem solving, e.g., decomposing an overall task into subtasks, finding a track through the space of sub-tasks, administering tasks, which still have to be tackled or tasks, which are already solved. Tasks provide knowledge suitable for coordination in problem solving. Even a set of tasks T may prove to be useful. As a matter of fact, the benefits for problem solving grow if in addition to a mere enumeration of tasks, i.e., a set of tasks, dependencies between them or relational descriptions are also represented. For example, a set of tasks T may be useful in computer-supported design as a check-list with the active part of problem solving still taken by the user. By pointing at tasks still to be done or at tasks already done, such a list provides a kind of administrational support in problem solving. Apart from its usefulness in problem solving, the set of tasks T is the kernel of the overall knowledge acquisition described in this paper.

Checking Realizations of Tasks. The predicates $(P_i^T)_I$ represent knowledge about relations between realizations of tasks. Predicates $(P_i^T)_I$ have a description-like flavor. They state what a task looks like that is embedded into a network of other tasks and that is realized well according to the judgement of an expert. Predicates $(P_i^T)_I$ are the foundation of checking design solutions in our system [JJ94]. Given a particular stage in design problem solving descriptive interrelations $(P_i^T)_I$ between tasks may be employed for tracking errors in design. Due to the interrelations between tasks an error in a given task spreads out and affects other tasks, too. Knowledge about dependencies may be used by the system for localizing the fan-out area affected by an error in a particular task.

Proposing Task Sequences to Support Planning. A set of tasks T may be also extended by knowledge about their structure of decomposition S^T and dependencies D^T may be provided for problem solving and further knowledge acquisition. S^T and D^T represent *structural* knowledge especially useful for coordination of planning. Which type of problem solving may be successfully realized by a system in this stage? When solving a multi-task problem it is - by definition - not sufficient to limit the attention just to single tasks. There has to be a device to plan a sequence of tasks to be tackled. Advantageous sequences of tasks have to be selected and disadvantageous ones or dead ends have to be avoided. To realize this objective the system has to be equipped with additional coordinating knowledge. Knowledge about dependencies fulfills this requirement. Dependencies are relations between single tasks and certain sets of tasks. If there is a dependency relation between a certain task and other tasks a solution for the task can not be attained without considering the other tasks. Once knowledge about dependencies between tasks is acquired in addition to a set of tasks T, the system may exploit knowledge appropriate for coordinating problem solving.

Realizing Tasks to Support Design. Given a set of tasks T, knowledge about their structure of decomposition S^T and dependencies D^T, the knowledge-base may be incrementally enlarged by case-based knowledge $(T, f_{[CB,\sigma]})$ or rule-based knowledge (T, R) or both. This means, the knowledge already given allows for generating planning. Which type of problem solving may be successfully realized by adding cases or rules? If the knowledge-base encompasses cases or rules the system will be able to design single, predefined tasks. Note, that design may be used to work out the plan derived by applying a set of tasks, knowledge about their structure of decomposition and dependencies. Designing tasks proceeds by applying cases to a task t given by T to accomplish its realization t^*. This is done in an analogous manner to prior solutions of this specific task. Alternatively, rules may be applied. The result will be the realization of the t again. Given given both kinds of knowledge the user has to choose the advantageous one.

Full Support. Finally, there are all types of knowledge mentioned so far provided for problem solving and system design. This means there is a set of tasks T, structured by S^T and D^T, descriptive interrelations $(P_i^T)_I$, case-based knowledge $f_{[CB,\sigma]}$, and rules R. Which type of problem solving may be successfully realized by a system equipped with this kind of knowledge? Since all types of knowledge described above are given all problem solving capabilities discussed so far are possible in a system that uses all these kinds of knowledge.

Objective	**Selection Criteria**
Test	Using both cases and rules and comparing the results
Success	Giving preference to the kind of knowledge that has been applied successfully in the past
Speed	Giving preference to the kind of knowledge the application of which proceeds fastest
Transparency	Using the kind of knowledge the application of which is most transparent to the user
Specificity	Giving preference to cases if and only if there is a very close match between the problem and prior cases and choosing rules otherwise

Figure 11: Criteria for selecting cases and rules

The system may for example use the coordination knowledge for tracking the user's problem and may offer both planning or design proposal, which are derived from knowledge appropriate for realizing problem solving. Such a system comprises both cases and rules for tackling a task. As a consequence, there has to be a device to control whether cases, rules, or both may be applied. Different selection criteria have been set up to cope with this problem [JS91]. Which of these is chosen depends very much on the objective of using cases or rules (cf. Fig. 11). While there are many possible approaches to this problem there is, however, no selection criterion that can be applied successfully across all possible applications. Taken together, when used in a design support system the full

bag of different kinds of knowledge mentioned so far allow for a systematic coaching of the expert's problem solving:

- The system is in a position to scan the experts problem solving behavior by localizing the tasks the expert is concerned with.
- As far as the tasks are concerned, which the user has already tackled the system makes an attempt at checking them. The result is communicated to the user.
- Knowledge about dependencies may be used by the system for localizing the fan-out area affected by an erroneous task.
- Concerning tasks not yet tackled by the user, the system is able to make a planning proposal about the next step to take. The user may accept or reject the planning proposal.
- The system may work out the planning proposal by making a design proposal, which the user may accept or reject.

The system outlined at the end of this section allows for the most far reaching systematic support of a user. Services offered by the system are embedded into the work-flow of the user and may be called upon by the user on demand. The system is in a position to offer support unobtrusively which the user may accept or reject.

5 Knowledge Interaction

The last section spelled out interrelations between different kinds of knowledge. In order to exploit them, we will now illustrate the interactions of different kinds of knowledge. Therefore we present two prototypical knowledge level and data-flow specifications for a system aiming at a case-based and rule-based support of the climate system. Effects of different, but interacting kinds of knowledge are pointed out by drawing on tasks and cases, and tasks and rules. We will illustrate the kind of support, which was referred as *realizing tasks to support design* in the preceding section. An instance of design of supply nets is taken to exemplify the interactions between tasks, cases, and rules. This example is meant to show how *full support* of a design support system looks like.

5.1 Interactions and System Design

We use MoMo [WVL+92] a language designed to specify and operationalize KADS conceptual models. In MoMo, the control flow is defined on the system layer using *system–tasks*. The data flow is specified on the inference layer in terms of *places*, *actions*, and *types*. Actions refer to inferences directed toward a particular system–task. Places are "containers" holding the knowledge, i.e., input and output of an action. The kind of knowledge used by an action has to be specified by assigning it to a particular type. To represent MoMo

models graphically, e.g, as an inference structure, boxes are used to represent actions, and ovals are used to represent places.

In section 2, Fig. 2 we gave a survey of the steps required when planning and designing a complex technical device like a climate system. When watching an expert planning and designing supply nets in buildings the whole process may be broken down into a sequence of snapshots or states of problem solving. The type of knowledge that enters each state of problem solving may be specified by a MoMo place. We extended this set of MoMo places to a full-blown MoMo inference structure by inserting MoMo actions that draw on the precedent place to generate the subsequent place.

5.1.1 Tasks and Cases

The first inference structure in Fig. 12 makes use of the task decomposition and the case-based approach for structural similarity assessment. A new problem is represented by the `situation description`. The task decomposition provides `known subgoals` and `state-subgoal-pairs`. Both are the input to the MoMo action `identify problem`. Output of this *planning phase* is a concrete `problem description`, which is worked out

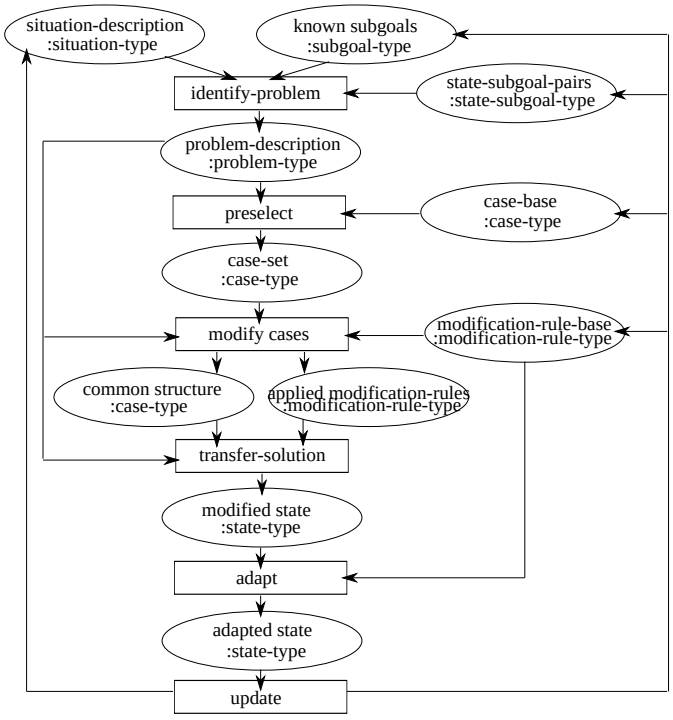

Figure 12: Inference structure to integrate tasks and cases

in the *design-phase* using a case-based approach. We use an attribute-based similarity assessment to `preselect` appropriate candidate cases `case-set` out of the `case-base`. The next action is the application of rule-based background knowledge stored in the `modification-rule-base` called `modify cases`. These modification rules are applied

to the problem description and the candidate cases until a most specific common structure of both can be determined. This `common structure` together with the `applied modification rules` is taken to define the structural similarity. Now, the modified solution can be transferred. Applying the inverse rules to the solution results in the concrete, adapted solution of the new problem. Additionally, the action `update` is used to add the new case to the `case-base`. Finally, the `case-base`, `modification-rule-base`, `state-subgoal-pairs` and `known-subgoals` will be enlarged by knowledge derived from the lessons learned when solving the new problem.

5.1.2 Tasks and Rules

The second inference structure (cf. Fig. 13) uses rules instead of cases. Again the task decomposition provides knowledge needed to determine the subsequent planning steps. For the *design phase* the decision to use rules instead of cases propagates through the rest of the inference structure. The action `identify problem` provides the input for `select` rules out of a finite set of rules.

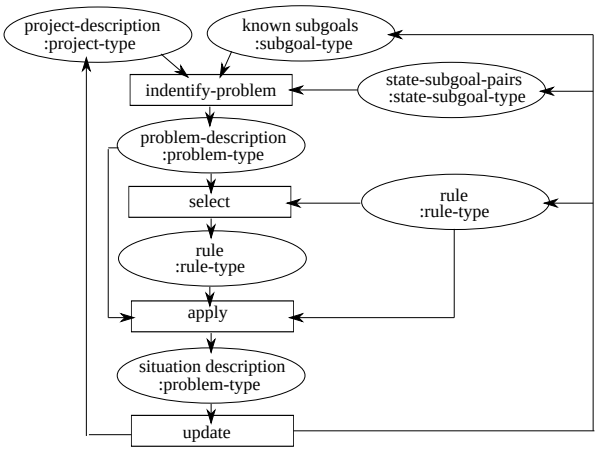

Figure 13: Inference structure to integrate tasks and rules

These `rules` are applied to the `problem description` resulting in the `situation description` of the correct solution. Finally, the `known subgoals`, the `state-subgoal-pairs` and the `rules` are updated. Obviously, there is no need any more to entertain system–tasks for adaptation or testing purposes.

5.2 Interactions: An Example

To exemplify the knowledge interactions we integrate the knowledge represented by tasks, cases, and rules to design a supply air network in an office building. This will be illustrated by interactively solving tasks of the given task-structure using different approaches. Fig. 14 depicts a part of the task decomposition introduced in section 2, Fig. 2. Shaded areas mark the tasks that are solved in a case-based or a rule-based way. The area marked I

refers to the task solved by applying the *zab4* rule (introduced in section 3.3.3). The task marked II can be solved by applying the *zvb4* rule (cf. section 3.3.3). The rectangularly shaped areas III and IV depict the tasks solved in a case-based manner applying case c_4 and c_5 respectively (see section 3.2.3). Note, that it would also be possible to solve task I and II in a case-based manner using the cases c_1 and c_2 see section 3.2.3, Fig. 3.

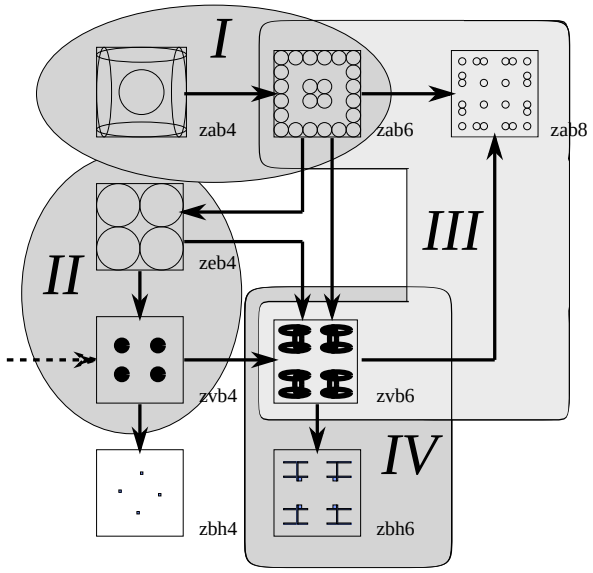

Figure 14: Solving tasks integrating different problem solving approaches

As discussed in the preceding section, a description of the task decomposition can be employed as coordinating knowledge, which can be used to select and integrate different problem solving approaches. The task decomposition provides information as to "when" and "what" should be solved. Cases and rules provide knowledge about "how" it should be solved.

6 Discussion

Our intention has been to give a systematic account of knowledge acquisition for design support systems. To that end, we employed the dependencies between tasks in order to drive the knowledge acquisition. Task-oriented knowledge acquisition of cases, rules, and models is apparently an effective approach to system design and seems to be especially well-suited to design tasks like the design of buildings and their supply nets. Depending on the knowledge available the system is able to realize particular degrees of support. Providing the right task assignment between user and system is a challenge to enable useful and realizable computer-aided support in design. Only those tasks should be assigned to the system that can be tackled successfully by the system on the basis of the knowledge represented. This sounds quite straightforward and simple but requires careful analyses of the application domain and the workflow of human-computer interactions

Although our results are experimental in nature they might pave the way to a more principled account of knowledge acquisition and system design in CBR-systems. Parts of this work have already been implemented along the rationale presented [JJ94]. As an outlook on future work we investigated and exemplified interactions of different kinds of knowledge by generating and using variants of inference structures of the system to be built. In sum, task-oriented knowledge acquisition has turned out to be a viable approach.

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 owe thanks to Brona Collins for correcting a previous version of this paper, and to many of our colleagues for fruitful discussions about the ideas introduced in this paper. Nonetheless, the paper reflects our personal view on specific questions and tasks involved in knowledge acquisition, reasoning, and system development.

References

[Aam90] A. Aamodt. *A computational model of knowledge-intensive problem-solving and learning.* EKAW-90, Amsterdam, 1990.

[And83] J. R. Anderson. *The architecture of cognition.* Cambridge, MA: Harvard University Press, 1983.

[Bör94a] K. Börner. Structural similarity as guidance in case-based design. In *First European Workshop on Case-Based Reasoning (EWCBR-93)*, LNAI, page in press. Springer Verlag, 1994.

[Bör94b] K. Börner. Towards formalizations in case-based reasoning for synthesis. In *accepted for the AAAI-94 Workshop on Case-Based Reasoning*, page in press, 1994.

[BR90] M. I. Bauer and B. Reiser. Incremental envisioning: The flexible use of multiple representations in complex problem solving. In *Proceedings of the 12th Conference of the Cognitive Science Society*, pages 317–324, 1990.

[BWvS+87] J. Breuker, B. Wielinga, M. van Someren, R. de Hoog, G. Schreiber, P. de Greef, B. Bredeweg, J Wielemaker, and P. Billaut. Model-driven knowledge acquisition: Interpretation models. *Technical Report Esprit Projet 1098, Deliverable task AI, Amsterdam, University of Amsterdam, Social Sciene Informatics*, 1987.

[Cha83] B. Chandrasekaran. Towards a taxonomy of problem solving types. *AI Magazine*, 4:9–17, 1983.

[CJS92] B. Chandrasekaran, T. R. Johnson, and J. W. Smith. Task-structure analysis for modeling domain knowledge and problem solving for knowledge system construction. *Paper presented on a Workshop on Problem-Solving Methods, Stanford, July 9-11*, 1992.

[DK92] E. A. Domeshek and J. L. Kolodner. A case-based design aid for architecture. In *Proc. Second International Conference on Artificial Intelligence in Design*, pages 497–516. Kluwer Academic Publishers, June 1992.

[DPvC+87] C. G. Drury, B. Paramore, H. P. van Cott, S. M. Grey, and E. N. Corlett. Task analysis. In Salvendy G., editor, *Handbook of human factors*, pages 370–401. John Wiley & Sons, 1987.

[Gen84] M. R. Genesereth. The use of design descriptions in automated diagnosis. *Artificial Intelligence*, 24:411–436, 1984.

[GF91] D. Gentner and K. D. Forbus. MAC/FAC: A model of similarity-based retrieval. In *Proceedings of the Cognitive Science Conference*, pages 504–509, 1991.

[Goe89] A. K. Goel. *Integration of case-based reasoning and model-based reasoning for adaptive design problem solving.* PhD thesis, Ohio State University, 1989.

[GS88] P. Goumain and J. Sharit. Human-computer interaction in architectural design. In *Handbook of human-computer interaction*, pages 709–727. Elsevier Science Publishers B.V. (North-Holland, 1988.

[HF93] K. Hua and B. Faltings. Exploring case-based building design – CADRE. *AI EDAM*, 7(2):135–143, 1993.

[Hin92] Th. R. Hinrichs. *Problem solving in open worlds: A case study in design.* Lawrence Erlbaum Associates, 1992.

[Hol84] K. J. Holyoak. Mental models in problem solving. In J. R. Anderson and S. M. Kosslyn, editors, *Tutorials in learning and memory*, pages 193–218. San Francisco: Freeman, 1984.

[Hov93] L. Hovestadt. A4 – digital building – extensive computer support for the design, construction, and management of buildings. In *CAAD Futures '93, Proceedings of the Fifth International Conference on Computer-Aided Architectural Design Futures*, pages 405–422, Pittsburgh, June 1993. North-Holland, Amsterdam.

[Ind92] B. Indurkhya. *Cognition and metaphor.* Kluwer Academic Publishers, Dordrecht, Netherlands, 1992.

[JB93] D. Janetzko and K. Börner. Tasks-methods-knowledge interdependencies in CBR-systems. FABEL–Report 09, Gesellschaft für Mathematik und Datenverarbeitung mbH (GMD), 1993.

[JBCH93] D. Janetzko, K. Börner, C.-H. Coulon, and L. Hovestadt. Toward a task-oriented methodology in knowledge acquisition and system design in CBR. In M.M. Richter, S. Wess, K.-D. Althoff, and F. Maurer, editors, *First European Workshop on Case-Based Reasoning (EWCBR-93), Posters and Presentations*, volume 2, pages 360–365, 1-5 November 1993.

[JJ94] O. Jäschke and D. Janetzko. On verification in design. In *submitted to the First European Conference on Cognitive Science in Industry*, 1994.

[JS91] D. Janetzko and G. Strube. Case-based reasoning and model-based knowledge acquisition. In *Contemporary knowledge engineering and cognition*, pages 99–114. Springer, Berlin, 1991.

[Kol93] J. L. Kolodner. *Case-based reasoning.* Morgan Kaufmann, 1993.

[KS91] A. E. Karl and J. Smith. *Toward a general theory of expertise: Prospects and limits.* Cambridge: Cambridge University Press, 1991.

[NC91] D. Navin Chandra. *Exploration and innovation in design: Towards a computational model.* Springer, 1991.

[New82] A. Newell. The knowledge level. *Artificial Intelligence*, 18:87–127, 1982.

[NS72] A. L. Newell and H. A. Simon. *Human problem solving.* Englewood Cliffs, NJ: Prentice-Hall, 1972.

[RS89] Ch. K. Riesbeck and R. C. Schank. *Inside Case-Based Reasoning.* Lawrence Erlbaum Associates, Hillsdale, 1989.

[SJKss] Gerhard Strube, Dietmar Janetzko, and Markus Knauff. Cooperative construction of expert knowledge: The case of knowledge engineering. In *Interactive Minds.* Cambridge: Cambridge University Press, in press.

[Ste83] L. Steels. Components of expertise. *AI Magazine*, 11:28–49, 1983.

[Ste93] L. Steels. The componential framework and its use in reusability. In J.-M. David, J.-P. Krivine, and R. Simmons, editors, *Second Generation Expert Systems*, pages 273–298. Springer, 1993.

[Str92] Gerhard Strube. The role of cognitive science in knowledge engineering. In *Contemporary knowledge engineering and cognition*, pages 161–174. Berlin: Springer (Lecture Notes in AI, 622), 1992.

[Van89] K. VanLehn. Problem solving and cognitive skill acquisition. In M. I. Posner, editor, *Foundations of Cognitive Science*, pages 527–579. Cambridge MA: MIT Press, 1989.

[VdV93] W. Van de Velde. Issues in knowledge level modelling. In J.-M. David, J.-P. Krivine, and R. Simmons, editors, *Second Generation Expert Systems*, pages 211–231. Springer, 1993.

[Voß94] A. Voß. Similarity concepts and retrieval methods. FABEL–Report 13, Gesellschaft für Mathematik und Datenverarbeitung mbH, Forschungsbereich Künstliche Intelligenz, 1994.

[WVdVSA93] B. Wielinga, W. Van de Velde, G. Schreiber, and H. Akkermans. Towards a unification of knowledge modelling approaches. In J.-M. David, J.-P. Krivine, and R. Simmons, editors, *Second Generation Expert Systems*, pages 299–335. Springer, 1993.

[WVL+92] J. Walther, A. Voß, M. Linster, Th. Hemmann, H. Voß, and W. Karbach. MoMo. Technical report, Gesellschaft für Mathematik und Datenverarbeitung (GMD), 1992.