ON ADAPTATION IN CASE-BASED DESIGN

Benno Stein Marcus Hoffmann

Department of Mathematics and Computer Science / Knowledge-based Systems
University of Paderborn, D–33095 Paderborn, Germany
Email: stein@uni-paderborn.de

Abstract: Case-based reasoning (CBR), sometimes also called "reasoning by remembering", has shown success—especially in fields where human problem solving mechanisms are either partly understood or cannot be resembled properly.
The design of technical systems is such a field; here human designers outclass the computer as well as traditional AI concepts. CBR can play two roles in this connection: Creating a starting position for existing design approaches to draw up, or forming a frame for problem solving approaches to be embedded.

Case adaptation plays the key role in case-based design. The paper in hand investigates case adaptation theoretically and exemplary. Its main contribution is the identification and formalization of premises that must be fulfilled if case adaptation shall be a successful concept for solving design problems.

Keywords: case-based design, design problem solving, adaptation in CBR, CBR.

1. INTRODUCTION

1.1 *Design Problem Solving*

Solving a design problem means to transform a set of demands, wishes, or expectations at a non-existing system towards a description from which the desired system can be constructed in a definite manner. Speaking formally, a set of demands D is transformed towards a system description S.

Human designers develop this transformation often within a cyclic (and evolutionary) process, the so-called design process, which comprises synthesis, analysis, and evaluation tasks (see Figure 1).

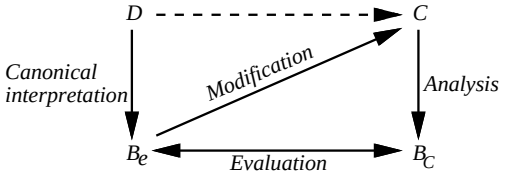

Figure 1: *Generic design process: The expected behavior B_e controls the synthesis of a system S, whose analysis reveals the system behavior B_S. Within the evaluation phase, the two behavior sets are compared to each other, leading to new information for the adaptation of S (Gero, 1990; Stein, 1995).*

Moreover, human designers fall back on design problems previously solved. Consequently, design problem solving must not start from scratch, and the synthesis step in the process above will correspond to a retrieve-and-adapt step, if a suited design pattern can be found.

Automating design means to operationalize the transformation $D \longrightarrow S$ on a computer—either directly or by emulating the human design process. However, for demanding engineering domains, the synthesis and adaptation tasks can only be automated partly, and, by now, user support concentrates on demand formulation and analysis automation (Stein, 1995).

1.2 *Case-based Reasoning*

Let a case combine a description of a problem along with a solution. Basic idea of case-based reasoning (CBR) is to exploit previously solved cases when solving a new problem. I. e., a collection of cases is browsed for the most similar case, which then is adapted to the new situation. The commonly accepted CBR cycle shown in Figure 2 goes back to (Aamodt and Plaza, 1994) and is comprised of four steps:

(1) *Retrieve.* A case relevant for the problem is retrieved.
(2) *Reuse.* Having performed more or less adaptations, the retrieved case may be reused.
(3) *Revise.* Having evaluated the adapted case, additional repair adaptations may be applied.

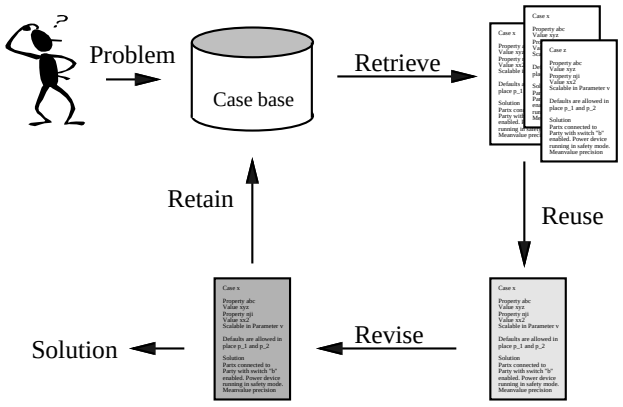

Figure 2: *The classical CBR cycle.*

(4) *Retain.* The new case, consisting of the problem along with a solution, is stored.

1.3 *Design Problem Solving and CBR*

Configuration, design, synthesis—these terms stand for problem classes where the AI paradigm "generate and test" has been applied rather successfully (Brown and Chandrasekaran, 1989; Cunis *et al.*, 1989; Stein and Weiner, 1991). CBR, however, follows the paradigm "retrieve and adapt" (Leake, 1995). Both concepts can work fine together to solve design problems.

A previously solved design problem that contributes a good deal to the desired solution may bound difficult synthesis and adaptation tasks to a tractable rest problem. Following this idea, the starting position of a design problem should be created with CBR methods, while for the heuristic and search-intensive adaptation tasks other AI paradigms come into play.

As mentioned at the outset, a design problem is stated by a set of *user demands*, D; a solution to a design problem is a *system*, S, which can be understood as a collection of objects or as some kind of construction plan. S is a solution of the design problem D, if the behavior of the system, B_S, complies with D.

Remarks. (*i*) With respect to a concrete domain, a design problem can be formalized more precisely. Nevertheless, in connection with the considerations and conclusions of this paper, an abstracted view is more adequate here. (*ii*) There exist two concepts of how a problem's solution can be defined: One of them codes the problem solving *process*, the other codes the *result* of a problem solving process, for example in the form of a system description S. From this distinction result two analogy concepts in CBR, namely that of derivational analogy (belonging to the former) and that of transformational analogy (belonging to the latter) (Carbonell, 1986; Goel and Chandrasekaran, 1989; Hinrichs and Kolodner, 1991). For reasons of clearness, the considerations of this paper are oriented at the latter, i. e., at the system description view, but they could be reformulated to the process-centered view as well.

Definition 1.1 (Case, Case base, Query). Let $\mathbf{D}$ be a set of demand sets, and let $\mathbf{S}$ be a set of systems. A *case* C is a tuple $C = \langle D, S \rangle$, $D \in \mathbf{D}, S \in \mathbf{S}$, where S constitutes a solution for D. A set CB consisting of cases is called a *case base*. A case of the form $p = \langle D, \cdot \rangle$ is called *query* or problem definition to a case base.

When given a query $p = \langle D, \cdot \rangle$ to a case base CB, two jobs must be done to obtain a solution to p. (*i*) Retrieval of a similar case c, and (*ii*) adaptation of c such that D is fulfilled.

In (Weß, 1995) three approaches to define similarity are mentioned: Similarity based on predicates, similarity based on a preference relation, and the most generic concept, similarity based on a measure. In connection with design problem solving, only the last is powerful enough, and the following definition will formalize a similarity measure for design case bases.

Definition 1.2 (Case Similarity). Given is a symmetric function $\sigma : \mathbf{D} \times \mathbf{D} \rightarrow [0;1]$, which additionally has the reflexivity property, $\sigma(D_1, D_2) = 1 \Leftrightarrow D_1 = D_2$.
Moreover, let $c_1 = \langle D_1, S_1 \rangle$ and $c_2 = \langle D_2, S_2 \rangle$, c_1, $c_2 \in CB$, be two cases. Then the *case similarity sim* : $CB \times CB \rightarrow [0;1]$ is defined by means of σ in the following way: $sim(c_1, c_2) = \sigma(D_1, D_2)$.

Remarks. (*i*) The semantics of σ shall be as follows. The more similar two demand sets D_1 and D_2 are, the larger shall be their value $\sigma(D_1, D_2)$. (*ii*) The symmetry property guarantees that $sim(c_1, c_2) = sim(c_2, c_1)$; the reflexivity property defines the self-similarity of a case.

2. ADAPTATION IN CASE-BASED DESIGN

The identification of similar cases is a prerequisite for solving a design problem by means of CBR. However, case *adaptation* plays the key role. As a consequence, the similarity between two cases c_1 and c_2 should be defined in relation to the adaptation effort that is necessary to transform c_1 towards c_2.

This section discusses case adaptation in greater detail. It investigates at which places in the CBR cycle adaptation happens and defines premises that must

be fulfilled when case adaptation shall be a successful concept.

- Adaptation can occur in the reuse step.
  A retrieved case is modified to better fulfill the demands. The adaptation is not evaluated respecting efficacy.
- Adaptation can occur in the revise step.
  An already modified case is evaluated and eventually further modified to better fulfill the demands.

Each adaptation of a case $c = \langle D, S\rangle$ is a modification of c; nevertheless, not every modification yields an adaptation: An adaptation has teleological character—it serves the purpose to modify S towards S' in such a way that a demand, which has been under-satisfied by S, is fulfilled to a higher degree by S'. Formally:

Definition 2.1 (Modification, Adaptation). Let $c = \langle D, S\rangle \in CB$ be a case, and let $p = \langle D_p, \cdot\rangle$ be a query. A *modification* of c respecting p is a function $\mu : \mathbf{D} \times CB \to \mathbf{D} \times \mathbf{S}$, with $\mu(D_p, c) = \langle D', S'\rangle$ for all $D_p \in \mathbf{D}$ and $c \in CB$.
A modification is called an *adaptation* of c if the following condition holds:

$$sim(\langle D', S'\rangle, p) > sim(\langle D, S\rangle, p)$$

Adaptation poses several requirements to a domain and a design problem respecting feasibility and evaluability. These requirements can be quantified. The next subsections develop a hierarchy of adaptations, which is ordered by their complexity.

2.1 *Level 0—No Adaptation*

Design problems that can be solved without a modification (Level 0 adaptation) form the basis of the hierarchy. Given a case base CB and a query p, the most similar case is used as a solution for p. In (Watson, 1997; Weß, 1995) this situation is called "null adaptation".

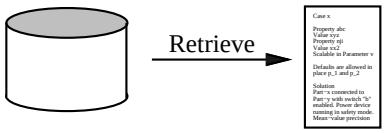

Figure 3: *The null adaptation case (Level 0 adaptation).*

Solving a design problem by such a table lookup procedure is rarely possible. If case adaptation is abandoned, the size of the case base must compensate this deficit. As a result, the similarity measure must be of a simple form to guarantee an efficient retrieval. On the other hand, the similarity measure needs not to encode a case's adaptability.

A (nearly) null adaptation approach has been employed successfully within CLAVIER (Barletta and Hennessy, 1989; Hennessy and Hinkle, 1991), a CBR system that guides autoclave loading for graphite-thread composites.

2.2 *Level 1—Automatic Adaptation*

Having retrieved a case from the case base, adaptation is usually necessary. If the adaptation can be performed automatically, and if the adapted case does not require an evaluation, an adaptation of Level 1 is given.

Automatic adaptation can be performed in several ways. Two important concepts in this connection are *scalability* and *composibility*.

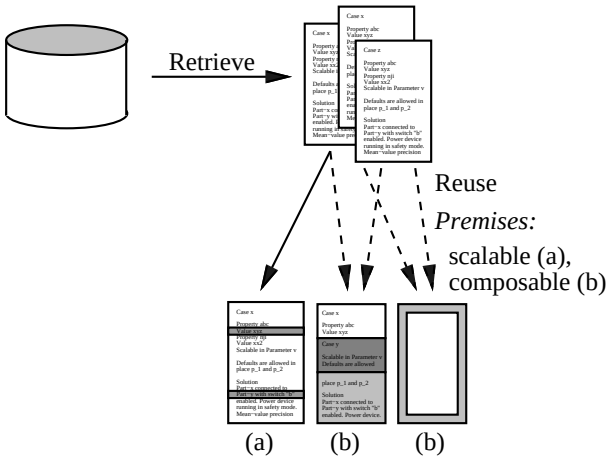

Figure 4: *Automatic adaptation is possible without a subsequent evaluation (Level 1 adaptation). The Figure hints three adaptation variants.*

Definition 2.2 (Scale Function, Scalable, Scaling). Given is a query $p = \langle D_p, \cdot\rangle$ and a demand subset $D'_p \subseteq D_p$. A function $scale : \mathcal{P}(\mathbf{D}) \times CB \to \mathbf{D} \times \mathbf{S}$ is called *scale function* of a case respecting D'_p, if the following conditions hold:

(i) $scale(D'_p, c) = c' = \langle D', S'\rangle$, where $D'_p \subseteq D', c \in CB$, and
(ii) $sim(c', p) > sim(c, p)$

c is called *scalable* with respect to D_p, c' is called *scaling* of c.

In other words, with respect to a demand subset D'_p there is a case $c = \langle D, S\rangle$ in the case base whose system S can be modified—scaled—towards S' in such a way that S' complies with D'_p and c' is more similar to p than is c.

Definition 2.3 (Composable). Given is a query $p = \langle D_p, \cdot \rangle$ and two cases $c_1 = \langle D_1, S_1 \rangle$ and $c_2 = \langle D_2, S_2 \rangle$, $c_1, c_2 \in CB$. Moreover let the sets $D_1' \subseteq D_1$, $S_1' \subseteq S_1$, $D_2' \subseteq D_2$, and $S_2' \subseteq S_2$ be given. c_1 and c_2 are called *composable* respecting a query p, if a function $comp : \mathbf{D} \times CB \times CB \rightarrow \mathbf{D} \times \mathbf{S}$ can be stated such that the following conditions hold:

(*i*) $comp(D_p, c_1, c_2) = c_3 = \langle D_3, S_3 \rangle$ where $D_1', D_2' \subseteq D_3$ and $S_1', S_2' \subseteq S_3$, and

(*ii*) $sim(c_3, p) > sim(c_1, p) \wedge sim(c_3, p) > sim(c_2, p)$

Adaptation by scaling is realized among others in WAYLAND, a CBR system that advises on the setup of aluminum pressure die-casting machines (Price and Peglar, 1995). Adaptation by composing requires the analysis of several pattern cases each of which contributing a particular aspect to the new case. This approach is pursued by the systems FABEL (Voss, 1997). A special composition variant is the frame transformation (Maher and de Silva Garza, 1997) where a "master" case defines the basic structure of the new case, which then is completed with other cases. COMPOSER (Purvis and Pu, 1996) is a system of this type; it has been used to plan the assembly sequence of electric motor assemblies.

Remarks. Note that Level 1 adaptation gets by without an extra evaluation step. I. e., the effects of an adaptation can completely be foreseen, or, at least, they can be estimated within narrow bounds.

2.3 *Level 2—Automatic Adaptation Plus Revise*

Exact the last point of the previous subsection cannot be guaranteed for Level 2 adaptations. Here, an adaptation's effect cannot be predicted at a sufficient accuracy, and consequently an additional evaluation becomes necessary.

Definition 2.4 (Evaluable). Given is a case $c = \langle D, S \rangle \in CB$. c is called *evaluable* if a function $\varepsilon : \mathbf{S} \rightarrow \mathbf{D}$ with $\varepsilon(S) = D$ can be stated.

Evaluability forms the basis for further adaptations. Using CBR terminology, these adaptations are called "repair" or—along with a preceding evaluation—"revise". Obviously adaptation plus evaluation can be performed several times, leading to a cycle that renders the design cycle presented at the outset in Figure 2. Here CBR forms a frame where an approach for design problem solving is embedded.

The CBR application JULIA (composition of menus) operationalizes an explicit evaluation/repair step (Hinrichs and Kolodner, 1991). Note that automatic

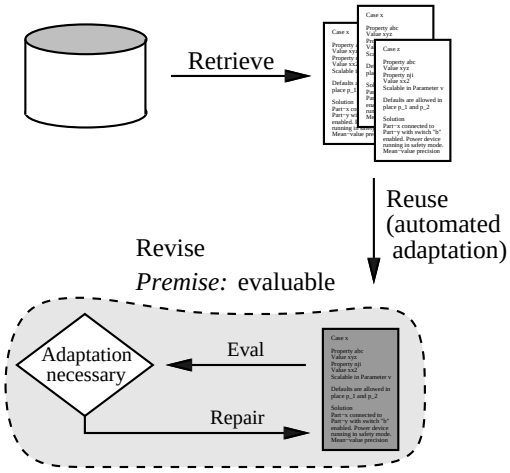

Figure 5: *Automatic adaptation is possible, but must be evaluated and applied more than once (Level 2 adaptation).*

evaluation can turn out to be a complex job involving demanding reasoning and simulation tasks (Goel *et al.*, 1997; Stein, 1998).

2.4 *Level 3—User Adaptation*

A common feature of all preceding adaptation types is automation: Computable functions and tractable algorithms can be stated, rendering a supervision by the user superfluous. However, adaptation and evaluation is left to the user in the following cases:

- The application domain is weakly structured.
- An automated adaptation is too expensive.
- Human designers can perform necessary adaptations or evaluations easily.
- Creativity is essential for getting the knack of the design problem.

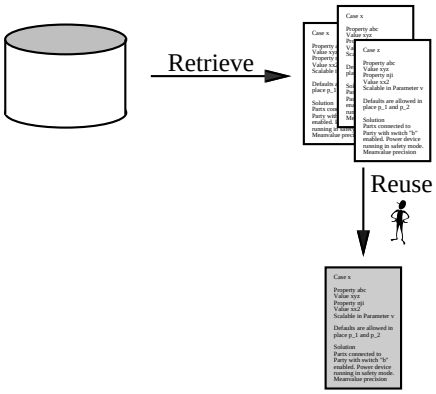

Figure 6: *Adaptation must be carried out by a human (Level 3 adaptation).*

According to the paradigm that a poorly adapted case is of less use than a null-adapted one (Riesbeck, 1996), the following tools bet on user adaptation: SEED in the architectural domain (U. Flemming et

al., 1997), and CADET (Narashiman *et al.*, 1997), which supports the design of mechanical devices.

Table 7 puts together the different types of adaptation. Note that the complexity of adaptation within case-based design problem solving must not be of a unique level. The next section presents a design problem, where adaptation jobs vary from level 1 to level 2.

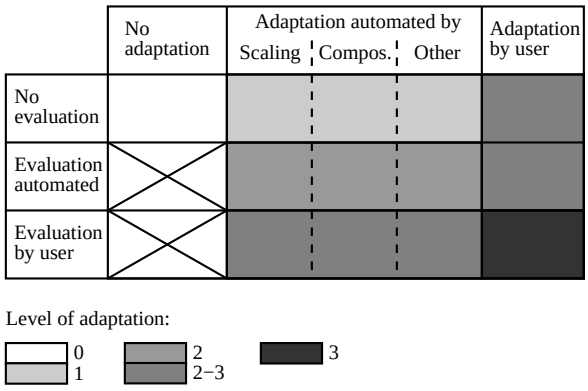

Figure 7: *An overview of adaptations and their complexities.*

3. CASE ADAPTATION IN HYDRAULICS

This section provides examples for adaptation, which stem from the field of fluid engineering (= hydraulics and pneumatics). Based on research and experiences in fluid engineering, we have realized tools for drawing, simulating, and structure visualization of electro-fluidic circuits (Stein *et al.*, 1998). At present, research concentrates on design support within hydraulics.

Fluidic manipulation jobs vary from simple lifting problems up to the realization of complex robot kinematics, and, given a demand description D for such a manipulation task, the design of an appropriate drive is a truly creative job (Gero, 1990; Brown and Chandrasekaran, 1983). Thus our working hypothesis is that we still *have* a preliminary design S of a system which has been retrieved by CBR methods, and which can be adapted to comply with D.

The next subsections illustrate that various adaptation jobs in hydraulics can be tackled by scaling and composition techniques.

3.1 *Adaptation by Scaling*

Consider the simple example of designing a lifting hoist. Moreover, let's assume that $c = \langle D, S\rangle$, the most similar case found respecting the query $p = \langle D_p, \cdot\rangle$, does not fulfill the maximum force constraint $(F, v_{F_p}) \in D_p$. Given this situation, c can be scaled up to fulfill D_p if the force difference between the existing and the desired system is of the same order of magnitude (see Figure 8).

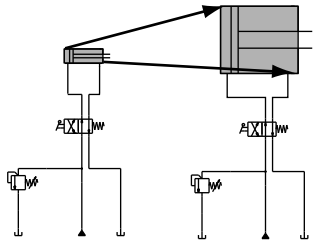

Figure 8: *Scaling a cylinder respecting a desired force.*

Notice that the scaling of the force is possible since the responsible underlying physical connections can be quantified: $F = P \cdot A$. [1]

A reasonable scale function applies this law as follows. It adapts the force value v_F of c according to the required value v_{F_p} by scaling the piston area to a new value $v_{A'}$ with respect to the maximum pressure $p_{\max}$:

$$v_{A'} = \frac{v_{F_p}}{p_{\max}}$$

Formally, the scale function takes two arguments (recall Definition 2.2); the first of which defines the subset of D to be scaled, the second is the case to be modified:

$$scale\big(\{(F, v_{F_p})\}, c\big) = c' = \langle D', S'\rangle \in \mathbf{D} \times \mathbf{S},$$

where

$$D' = D \setminus \{(F, v_F)\} \cup \{(F, v_{F_p})\},$$
$$S' = S \setminus \{(A, v_A)\} \cup \{(A, v_{A'})\}, \quad \text{with } v_{A'} = \tfrac{v_{F_p}}{p_{\max}}$$

D' and S' result from the demand set D and the system description S respectively by simply substituting the new parameter-value-pairs for the old ones.

Note that it remains to be shown that condition (*ii*) of Definition 2.2,

$$sim(c', p) > sim(c, p),$$

is fulfilled when applying the above function *scale*.

For a typical similarity function, which is based on the Euclidean distance measure, this is easily understood. Observe that the difference (and hence the Euclidean distance) between the desired maximum force value, v_{F_p}, and the maximum force value produced by c', $v_{A'} \cdot p_{\max}$, is zero:

$$v_{F_p} - v_{A'} \cdot p_{\max} = v_{F_p} - \frac{v_{F_p}}{p_{\max}} \cdot p_{\max} = 0$$

[1] The cylinder force equals the pressure times the piston area.

As a consequence, the similarity between the scaled case c' and the query p is strictly larger than the similarity between the original case c and p.

To formulate and to operationalize such type of scaling knowledge, we have developed a prototypic design language, which is tailored to the fluidic domain (Stein and Vier, 1998; Schlotmann, 1998).

3.2 *Adaptation by Case Composition*

Case composition in hydraulics is more sophisticated. Note that a prerequisite for applying the composition paradigm is a decomposition of existing cases into functional units.

In fluidic systems the functional level is reflected by so-called hydraulic or pneumatic axes, which are responsible to fulfill a particular function. Figure 9 shows three cases contributing a supply unit and two hydraulic axes to a new system.

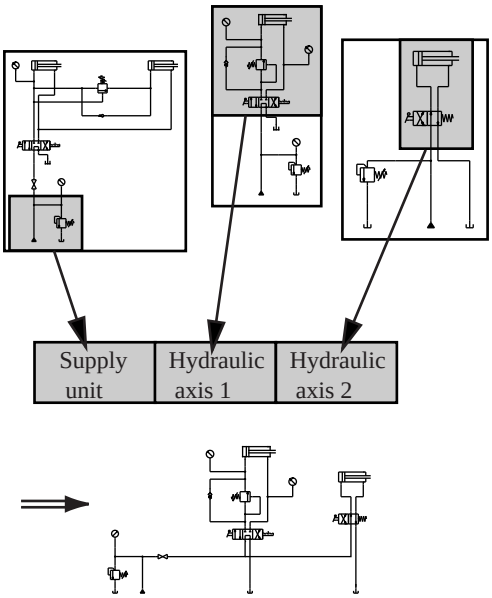

Figure 9: *Creating a solution by composition.*

SCHEMEBUILDER pursues this composition approach (Oh *et al.*, 1994; da Silva and Dawson, 1997): A demand set D is interactively decomposed into subtasks, assuming that each of the subtasks is realized by means of a single hydraulic axes. The axes in turn are retrieved from a database containing carefully selected design prototypes.

At present, the SCHEMEBUILDER approach lacks in the following respects:

(*i*) Axes can only be connected in parallel, which restricts the possible designs to an uniform circuit type.

(*ii*) The evaluation of a composed circuit is not integrated.

(*iii*) The case base must be maintained by a human engineer.

Remarks. The problem of automatically analyzing hydraulic systems respecting their functional units has been addressed in (Stein and Schulz, 1998). The authors developed graph-theoretical concepts to identify hydraulic axes—a concept which may also be extended to solve the case retaining problem mentioned under (*iii*).

4. SUMMARY AND FURTHER WORK

The design of technical systems is a field where human problem solving mechanisms cannot be resembled properly. Case-based reasoning may show a way out, for instance by creating a starting position for existing problem solving approaches to draw up: A previously solved design problem that contributes a good deal to the desired solution can bound difficult synthesis and adaptation tasks to a tractable rest problem.

The adaptation of cases plays the key role in case-based design. The paper in hand investigated case adaptation and formulated premises to discriminate between different levels of adaptation. These premises have been formalized and illustrated at a complex design problem, the design of hydraulic systems.

Current work is concerned with the operationalization of case-based design in fluidics. Based on our developments that include a prototypic design language, algorithms that break up complex circuits into useful pieces, and efficient simulation algorithms, case adaptation in fluidics shall be automated and tested in particular design scenarios.

Future work shall concentrate on the improvement of criteria (theoretically and exemplary) that help to decide a-priori whether or not a retrieved case can be upgraded to the solution of a given design problem.

References

Aamodt, Agnar and Enric Plaza (1994). Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches. *AICOM* pp. 39–59.

Barletta, R. and D. Hennessy (1989). Case adaptation in autoclave layout design. In: *Proceedings: Case-Based Reasoning Workshop* (K. J. Hammond, Ed.). Morgan Kaufmann Publishers. pp. 203–207.

Brown, D. C. and B. Chandrasekaran (1983). An Approach to Expert Systems for Mechanical Design. In: *Trends*

and Applications '83. IEEE Computer Society, NBS, Gaithersburg, MD.

Brown, David C. and B. Chandrasekaran (1989). *Design Problem Solving*. Morgan Kaufmann Publishers.

Carbonell, J. G. (1986). Derivational anaology: a theory of reconstructive problem solving and expertise acquisition. In: *Machine Learning: an Artificial Intelligence Aproach* (R. Michalski, J. Carbonnel and T. Mitchell, Eds.). Vol. 2. Morgan Kaufmann. Los Altos, CA. pp. 371–392.

Cunis, R., A. Günter, I. Syska, H. Peters and H. Bode (1989). PLAKON—An Approach to Domain-independent Construction. Technical Report 21. BMFT Verbundsprojekt. Universität Hamburg, FB Informatik.

da Silva, Jonny Carlos and David Dawson (1997). The development of an expert system for hydraulic systems design focusing concurrent engineering aspects. In: *Proceedings: International Conference on Enginieering Design (ICED 97) Tampere*.

Gero, John S. (1990). Design Prototypes: A Knowledge Representation Scheme for Design. *AI Magazine* **11**, 26–36.

Goel, A. K. and B. Chandrasekaran (1989). Use of device models in adaption of design cases. In: *Proceedings: Case-Based Reasoning Workshop* (K. J. Hammond, Ed.). Morgan Kaufmann Publishers. pp. 203–207.

Goel, A. K., S. Bhatta and E. Stroulia (1997). Kritik: An early case-based design system. In: *Issues and Applications of Case-Based Reasoning in Design* (M. L. Maher and P. Pu, Eds.). Lawrence Erlbaum Associates. Hillsdale, N.J. pp. 87–132.

Hennessy, D. and D. Hinkle (1991). Initial results from clavier: A case-based autoclave loading assistent. In: *Proceedings: Case-Based Reasoning Workshop* (R. Bareiss, Ed.). Morgan Kaufmann Publishers. pp. 225–232.

Hinrichs, K. and J. Kolodner (1991). The roles of adaptation in case-based design. In: *Proceedings of AAAI-91*. Cambridge, MA: AAAI Press / MIT Press.

Leake, David B. (1995). Case-based reasoning: Issues, methods, and technology.

Maher, Mary Lou and Andres Gomez de Silva Garza (1997). Case-based reasoning in design. *IEEE Expert*.

Narashiman, S., K. Sycara and D. Navin-Chandra (1997). Representation and synthesis of non-monotonic mechanical devices. In: *Issues and Applications of Case-Based Reasoning in Design* (M. L. Maher and P. Pu, Eds.). Lawrence Erlbaum Associates. Hillsdale, N.J. pp. 187–220.

Oh, V., P. Langdon and J. Sharpe (1994). Schemebuilder: an integrated computer environment for product design. In: *Computer Aided Conceptual Design*. Lancaster International Workshop on Engineering Design.

Price, C. J. and I. S. Peglar (1995). Deciding Parameter Values with Case-Based Reasoning. In: *Progress in Case-Based Reasoning* (I. Watson, Ed.). pp. 122–133. Lecture Notes in Artificial Intelligence 1020. Berlin: Springer-Verlag.

Purvis, L. and P. Pu (1996). An approach to case combination. In: *Proceedings of the Adaptation in Case Based Reasoning Workshop of the European Conference on Artificial Intelligence (ECAI96)*. Budapest, Hungary.

Riesbeck, C. K. (1996). What next? the future of case-based reasoning in post modern ai. In: *Case-Based Reasoning: Experience, Lessons, & Future Directions* (D. B. Leake, Ed.). Cambridge, MA: AAAI Press / MIT Press. pp. 371–388.

Schlotmann, Thomas (1998). Formulierung und Verarbeitung von Ingenieurwissen zur Verbesserung hydraulischer Systeme. Diploma thesis. Universität-GH Paderborn, FB 17 Mathematik / Informatik.

Stein, Benno (1995). Functional Models in Configuration Systems. Dissertation. University of Paderborn, Department of Mathematics and Computer Science.

Stein, Benno (1998). Supporting Hydraulic Circuit Design by Efficiently Solving the Model Synthesis Problem. *Proc. EIS 98, International ICSC Symposium on Engineering of Intelligent Systems, University of La Laguna, Tenerife, Spain*.

Stein, Benno and André Schulz (1998). Topological Analysis of Hydraulic Systems. Technical Report tr-ri-98-197. University of Paderborn, Department of Mathematics and Computer Science.

Stein, Benno and Elmar Vier (1998). An approach to formulate and to process design knowledge in fluidics. In: *Recent Advances in Information Science and Technology*. Second Part of the Proceedings of the 2nd IMACS International Conference on Circuits, Systems and Computers (CSC '98).

Stein, Benno and Jürgen Weiner (1991). Model-based Configuration. In: *OEGAI '91, Workshop for Model-based Reasoning*.

Stein, Benno, Daniel Curatolo and Marcus Hoffmann (1998). Simulation in FluidSIM. In: *SiWis '98. Workshop "Simulation in wissensbasierten Systemen"* (Fachausschuß 4.5 der GI ASIM Arbeitsgemeinschaft Simulation, Ed.). 61. Fachgruppe 4.5.3 "Simulation und Künstliche Intelligenz" der Gesellschaft für Informatik. Paderborn.

U. Flemming et al. (1997). Case-based design in a software environment that supports the early phases. In: *Issues and Applications of Case-Based Reasoning in Design* (M. L. Maher and P. Pu, Eds.). Lawrence Erlbaum Associates. Hillsdale, N.J. pp. 61–86.

Voss, A. (1997). Case design specialists in fabel. In: *Issues and Applications of Case-Based Reasoning in Design* (M. L. Maher and P. Pu, Eds.). Lawrence Erlbaum Associates. Hillsdale, N.J. pp. 187–220.

Watson, Ian (1997). *Applying Case-Based Reasoning: Techniques for Enterprise Systems*. Morgan Kaufmann Publishers, Inc.

Weß, Stefan (1995). Fallbasiertes Problemlösen in wissensbasierten Systemen zur Entscheidungsunterstützung und Diagnostik / Grundlagen, Systeme und Entscheidungen. Dissertation. Universität Kaiserslautern.