Applying Recursive CBR for the Customization of Structured Products in an Electronic Shop

Armin Stahl and Ralph Bergmann

University of Kaiserslautern
Centre for Learning Systems and Applications (LSA)
PO-Box 3049
D-67653 Kaiserslautern, Germany
{stahl,bergmann}@informatik.uni-kl.de

Abstract. When applying CBR for Electronic Commerce, the adaptation capabilities of CBR can be used for product customization. Most adaptation techniques suffer from the problem that they require a large knowledge acquisition effort which leads to problems in the rapidly changing E-Commerce scenario. In this paper we present a new approach to adaptation that is particularly suited to Electronic Commerce applications. It assumes that products can be structured hierarchically into sub-components. Adaptation is achieved by incrementally replacing unsuitable sub-components through recursively applying CBR to find best-matching alternative sub-components. The presented approach avoids huge portions of the knowledge acquisition effort and is prototypically implemented as an extension of the CBR-Works tool.

1 Introduction

With the increasing number of companies and private persons who get fast access to the World Wide Web (WWW), Electronic Commerce is playing more and more a central role in economy. Case-Based Reasoning (CBR) has become a very important technique for realizing intelligent product recommendation agents for such E-Commerce applications [1, 10, 4]. While for fixed, unchangeable products a pure similarity-based retrieval approach is sufficient, product *customization* is important when complex products with a large number of possible variants must be supported during sales [8]. Examples of such products are technical products (e.g., personal computers) or service products (e.g., insurances or investment plans). Product customization can be realized by the known adaptation techniques from CBR. However, the applicability of the different techniques strongly depends on the kind of products to be supported, in particular on the number of different variants. A general problem in particular for the transformation-based approaches to adaptation is that they require a large effort for acquiring the adaptation knowledge, to be represented, for example, as rules or as adaptation operators. This turns into a real problem when products have many different variants or if the product spectrum changes rapidly so that the adaptation knowledge must be updated. On the other hand, derivational approaches to

adaptation require a complete problem solver for product recommendation and sufficient knowledge for this problem solver. This approach also suffers form intractability, both in terms of computational efficiency and knowledge acquisition effort. Hence, for customizing complex products with a large number of possible variants, the existing adaptation approaches from CBR are often not suitable in practice.

In this paper we present a new approach to adaptation that is particularly suited to E-Commerce applications. This approach avoids huge portions of the knowledge acquisition effort of the previous approaches. It assumes products that are structured into sub-components, possibly in a hierarchical manner. The knowledge required is knowledge about available pre-configured complete products as well as knowledge about available sub-components. Both kinds of knowledge are easily available in an E-Commerce setting. After retrieving the best pre-configured product with respect to the customers requirements, the product is customized by incrementally replacing sub-components by more suitable sub-components. These new sub-components are determined by recursively applying CBR, i.e. similarity-based retrieval, on the level of the sub-components.

In the remainder of this paper, this approach is described in more detail. We will first describe a typical E-Commerce scenario and analyze the shortcomings of existing approaches. Then the general idea of recursive CBR is introduced before it is specialized for the purpose of product customization. Finally, we report on the current state of the implementation.

2 Product Configuration in E-Commerce Applications

2.1 Case-Based Product Recommendation: The Scenario

We consider the scenario that a customer enters a virtual shop to buy a complex technical system. Complex technical systems (e.g., PCs) can usually be decomposed into a set of different components. We suppose that the shop offers a set of pre-configured standard systems as well as the different individual components.

When providing an intelligent product recommendation agent, the first step of the sales process is a demand acquisition phase in which the customer states his individual demands on the searched product. The result of this demand acquisition phase can be formulated in form of an incomplete product description (on a technical level) which we call *query*. This query is then used to start a similarity-based retrieval for determining one of the available pre-configured standard systems that fulfills the demands as well as possible. However, because of the large number of possible product variants, the retrieved product usually does not fulfill the demands exactly. Hence, to be able to present the customer a satisfying result, it is necessary to customize the product, e.g., by replacing some components by more suitable ones. This is the task of the adaptation phase of CBR. In the following we review several well known approaches to adaptation in CBR and discuss whether they are appropriate for the customization of complex products.

2.2 Existing Approaches

Transformational Adaptation. One approach for realizing case adaptation is known under the term *transformational adaptation* [11]. Here, the knowledge how existing solutions can be adapted on actual problems is encoded explicitly, e.g., in form of rules. *Adaptation rules* [3] consist of a set of preconditions and a set of actions. Dependent on the evaluation of the preconditions the actions are able to modify a retrieved case to a new target case with respect to the given query. *Adaptation operators* [8] are very similar to adaptation rules. While the rules are executed after the retrieval automatically, the operators provide more control by the customer. Therefore, the retrieved case with an additional set of applicable operators is presented to the customer. If the retrieved product does not fulfill his demands, he can repeatedly choose operators to change the given product until he gets a satisfying result.

The general problem of both approaches is the necessity to define every possible case modification that may occur explicitly in the form of preconditions and actions. Therefore, even simple domains often require a large number of adaptation rules or operators to cover all adaptation possibilities. For really complex domains the huge amount of necessary rules or operators prevents the application in practice.

Configuration From Scratch. The configuration of a product can also be performed from-scratch by classic configuration systems [5] without using CBR. Applying such a system for the customization of products in an E-Commerce application often leads to some disadvantages. First, if it is impossible to configure a product that fulfills all customer demands exactly, such a system rarely finds a suitable alternative solution. Second, in classic configuration systems it is often difficult to handle optimality criteria. The system can only present any solution (if existing), but a high quality of this solution cannot be guaranteed in general. An additional problem of the from-scratch configuration is the time-critical aspect, i.e., the configuration process usually takes a lot of time. This is often not acceptable in E-Commerce applications.

Derivational Adaptation. The common property of adaptation rules and operators is that they are used to transform an existing solution to a new, hopefully better solution. Another adaptation approach is known under the term *derivational adaptation*. Here, the cases do not represent concrete solutions. Instead, they contain a description of the problem solving trace which describes how a concrete solution is generated. This known solution trace can then be used by a from-scratch problem solver to generate a solution very quickly. Generally, this approach is coupled with the same disadvantages as configuration from-scratch, except for better performance. A description of a configuration system using derivational adaptation can be found in [7].

In the next section we will introduce the basic theoretical idea for developing a new approach to realize efficient product configuration by using CBR.

3 Adaptation of Highly Structured Cases

3.1 Highly Structured Problems

Consider the situation that we have a domain in which it is possible to subdivide the arising problems and the related solutions into more or less independent sub-problems and related sub-solutions. This means, the complete problems, or we can also call them *complex problems*, can only be solved if all included sub-problems are solved and combined to an overall solution. When assuming such a problem structure, it is possible to represent the problems and the corresponding solutions in form of part-of hierarchies (see figure 1). The root-nodes of such part-of hierarchies represent the complex problems or solutions, respectively. Inner nodes represent *complex sub-problems (sub-solutions)* that can be divided again into less complex sub-problems (sub-solutions). Finally, the leaf nodes represent *atomic problems* and the related *atomic solutions*.

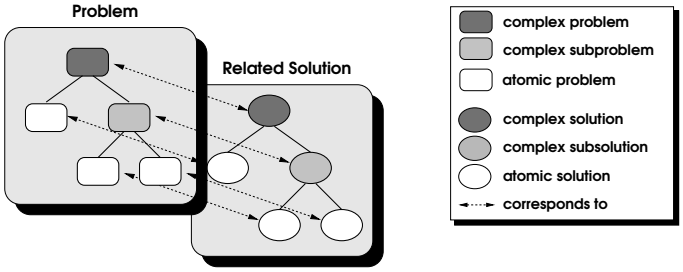

Fig. 1. Highly Structured Problems

Dependencies between Sub-Problems. Generally, the single sub-problems cannot be considered absolutely isolated because of dependencies between them. The solution of one sub-problem often depends on the solutions of some other sub-problems. Thus, it is unavoidable to handle these dependencies while solving the different sub-problems, or at least when combining them to the final solution.

3.2 The Idea of Recursive Case-Based Reasoning

Assuming a highly structured domain as described in the previous section, we can now introduce a special procedure for solving such problems by applying CBR. The first step is equal to the retrieval phase of the classic CBR-cycle. The given (complex) problem is used as query and a most similar case is retrieved from a case-base. Due to the complexity of the problem structure, in general, it is unlikely that the complete solution of the retrieved case can be applied to the current problem directly. This means that the retrieved case can include sub-solutions which cannot be reused. Then it is necessary to adapt these unsuitable

sub-solutions. Such adaptation can become a very complex task if the unsuitable sub-solutions belong to complex sub-problems.

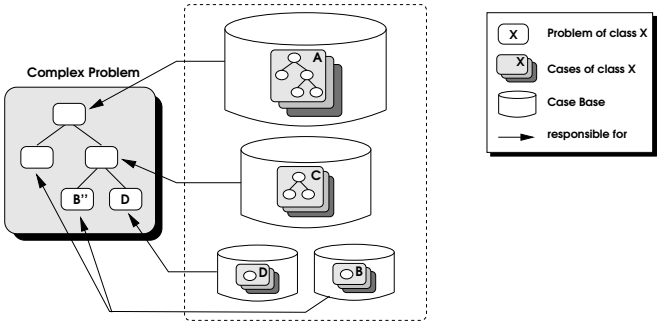

Fig. 2. The Use of Several Case-Bases

The idea of recursive CBR is to realize the necessary adaptation again in a case-based manner. This means that we will not try to adapt the retrieved but unsuitable sub-solutions by modifying them with traditional methods. Instead, we use still unsolved sub-problems as new queries for a retrieval in additional case-bases. These case-bases must contain cases that correspond to the classes of the current sub-problems, i.e., the cases are generally less complex than the cases in the top-level case-base (see figure 2). We now distinguish two situations, depending on the position of an unsolved sub-problem in the hierarchy:

- If the unsolved sub-problem is an atomic problem, the responsible case-base contains atomic solutions only. In this case we suppose that retrieving a case that can be reused directly is possible. Nevertheless, if adaptation is still necessary, it should be quite simple so that it can, for example, be realized by applying few adaptation rules.
- When having an unsolved complex sub-problem and retrieving a suitable case is impossible, we can repeat the described procedure in a recursive manner. This means that we start a retrieval to find a suitable sub-solution. If the retrieved complex sub-solution cannot be reused directly, necessary adaptation can be realized again in the described case-based manner.

To obtain a solution for the overall complex problem which was the starting point of the whole process, it is finally necessary to combine all found sub-solutions. An illustration of the suggested procedure is shown in figure 3.

The basic idea of recursive CBR is the principle of "Divide and Conquer", which can often be applied successfully in Computer Science. Generally, it is important to consider one basic limitation of this approach. If it is possible to divide a complex problem into sub-problems that can be solved separately in an easy way, it must also be guaranteed that the final combination of the found sub-solutions will not become too difficult. Otherwise, such a procedure will not be

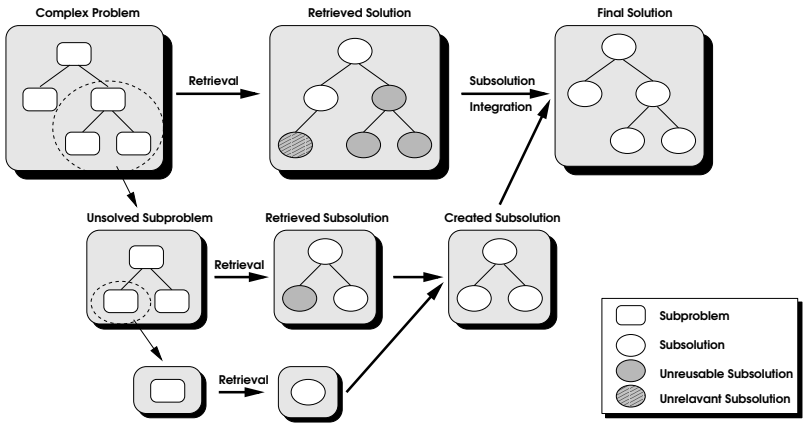

Fig. 3. The Idea of Recursive CBR

very efficient. Therefore, we assume a limited number of dependencies between the different sub-problems in the application domain to apply our approach.

Another work that applies incremental case adaptation to compose a solution for an actual design task out of the solutions of several cases is the system EADOCS [6]. However, this approach does not follow a problem decompositional hierarchy.

The advantage of a divide and conquer approach for solving problems with large complex cases is described in [9]. It is shown, that dividing a large problem into several smaller sub-problems leads to smaller solution spaces that have to be covered by the different case bases. Therefore, the CBR-System is able to find solutions with a significantly higher accuracy.

4 Applying Recursive CBR for Product Configuration

4.1 Domain Representation

In the previous section we have presented the idea of recursive CBR. Now, we will show how this idea can be applied for the customization of technical systems. To illustrate the functionality of this approach we consider the configuration of personal computers. However, the presented approach can be applied for the customization of any structured products.

Structuring Product Cases. In general, it is possible to represent complex structured products by an object-oriented domain model with an additional aggregation hierarchy [2, 3]. In the following we assume a special form of such a compositional structure like shown in figure 4.

In this structure, the root node represents the whole technical system to be configured, i.e., in our example a complete personal computer. The leaf nodes

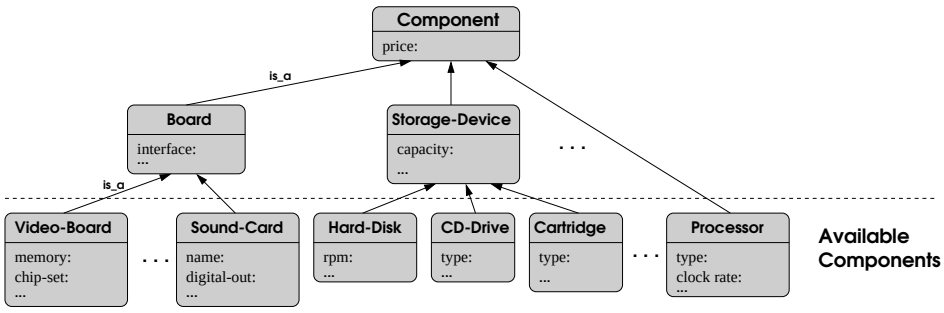

Fig. 4. An Example Compositional Structure

represent the concrete parts that a PC consists of, like the hard-disk, the processor, the CD-Drive, and so on. In a concrete PC these parts are realized by concrete technical components. For example, the processor part can be realized by a Pentium-III with 500MHz. The inner nodes of the compositional structure (if existing) do not correspond to concrete parts of the technical system. They rather represent abstract nodes used for aggregating related parts into kinds of abstract subsystems (e.g. Base-System or Storage-System).

Structured Query. The starting point of the configuration process is the query represented by an incomplete instantiation of the compositional structure. In section 3.1 we assumed that a complex problem can be decomposed to apply the idea of recursive CBR. When looking at the example query shown in figure 5, we can interpret the root note as our complex problem, i.e., we are searching a PC with a set of special properties. To reach this goal it is necessary to select appropriate components that fulfill the properties of the respective part-queries. In our example, one part-query states that the PC should have a hard-disk with 12GB of capacity. To fulfill this demand we can, e.g., integrate the concrete hard-disk "Maxtor91303D6" for the hard-disk part in the PC. Thus, we can interpret the different leaf nodes of the query as the sub-problems that must be solved to solve the overall problem, i.e., the configuration of the required PC.

Similarity Measure. To retrieve appropriate PCs from a case-base, the assumed CBR-system uses special domain dependent similarity measures. To compute the similarity between a query and a given product we assume a bottom-up strategy using the defined compositional structure. First, it is necessary to compute the similarities between the part-queries and the corresponding components. These *local similarities* are used for determining the *global similarity* between the query and the product. A more detailed description of such a similarity computation can be found in [2].

Dependencies between Components. In section 3.1, we have mentioned the problem of existing dependencies between sub-problems. In the described appli-

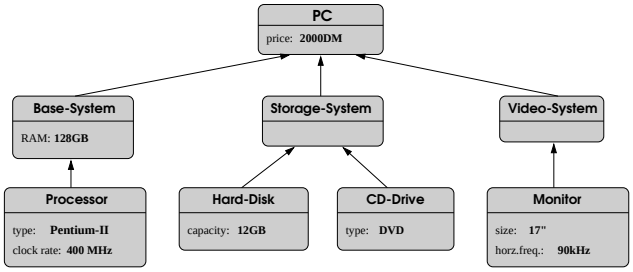

Fig. 5. An Example Query

cation domain such dependencies occur in form of technical *constraints* between the different components. For example, it is impossible to combine every kind of processor with any mainboard model because the respective interfaces must fit together. To be able to handle these technical restrictions during the configuration process, the domain model must include a formal description of the existing constraints. For our approach we suppose the existence of a special *constraint system* that is able to check whether the constraints of a given configuration are fulfilled.

4.2 The Configuration Framework

We now describe the different steps during product customization. Figure 6 shows the respective process model.

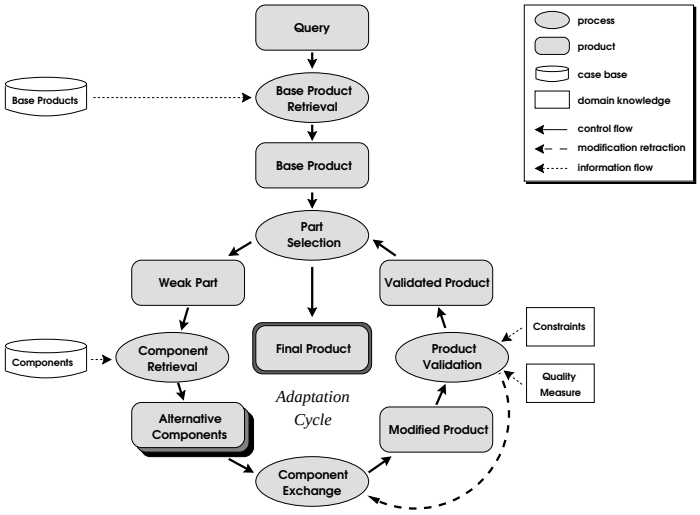

Fig. 6. The Configuration Process

Basically, the whole configuration process can be subdivided into two major steps, called *base product retrieval* and *adaptation cycle*. The task of the first step is the similarity-based retrieval of the best available base product from the respective case-base. The second step is an iterative procedure which performs the necessary adaptation of the retrieved base-product if it does not fulfill all customer demands.

At the beginning of the adaptation cycle, it has to be determined if the base product includes parts that do not fulfill the requirements of the query. This can be done by an examination of the local similarities between the different product parts and the related part-queries. Product parts with a low similarity to the respective part-query are called *weak parts*. The determination of these parts is the task of the *part selection* process. If the product includes no weak parts at all, the adaptation process is finished and the given product is presented to the customer as suggested customization result.

Generally, we can distinguish between two causes for weak parts. The first one occurs if the retrieved standard product does not include components for required product parts. For example, if a customer wants a PC with a modem, but the base-product includes no modem at all. Here, we can speak of a *missing component*. The second cause is given if installed components do not fulfill the technical requirements of the customer, e.g., if the PC still contains a modem but this modem provides only an insufficient data transmission rate. Then, we can speak of a *weak component*.

During the adaptation cycle, a selected weak part leads to a new similarity-based retrieval, called *component retrieval*. In this process, the part-query corresponding to the selected weak part is used to retrieve a collection of alternative components from the component case-base. The elements of this collection are arranged by their similarity to the query in a decreasing order.

The task of the next process is then the replacement of the weak component (if existing) by the first element of the collection of alternative components. If the weak part corresponds to a missing component, the first alternative component is introduced as a new component for this part. Because of the mentioned order, this alternative component represents the most suitable component that is available within the component case-base. The result of the described *component exchange* is a *modified product*.

To judge the effects of the component exchange, the modified product must finally be inspected during the *product validation* process. The task of this process is the validation of the modified product with respect to two different aspects. On the one hand, it must be guaranteed that the modification leads to a consistent product, i.e., to a product fulfilling the technical *constraints* of the respective application domain. Therefore, the assumed constraint system has to check the modified product. On the other hand, the modified product should naturally represent a better solution as the product before the modification. Therefore, we have to compare the similarity to the query of these two products. Either if the modified product is inconsistent or if it has a lower similarity to the query

as the product before the modification, retracting the last modification becomes necessary.

The *retraction* of the last modification is realized by the repeated application of the component exchange process. This means, the just now installed component that had led to the failure of the product validation is replaced by the next element of the alternative components collection. If no available alternative component is suitable to fulfill the mentioned criteria, it becomes necessary to restore the situation at the beginning of the current iteration of the adaptation cycle. In this case, an adaptation of the respective weak product part is currently impossible.

If the product validation could be passed successfully, one iteration of the adaptation cycle is finished. The result of this adaptation cycle is the *validated product*. In the next step, it is again the task of the part selection process to determine a weak part within this validated product which shall be used as the starting point for a new iteration. If it is impossible to determine a weak part whose adaptation could perhaps improve the product, the complete configuration process is succeeded and the final product can be presented to the customer. Optionally, it can be useful, e.g., if the actual configuration result is dissatisfying, to try an additional configuration attempt with another retrieved base product.

4.3 Controlling the Adaptation Cycle

After giving an overview of our configuration framework, we will now show that the part selection process (see figure 6) is the most important step of the adaptation cycle. This process is mainly responsible for the overall success of the adaptation procedure.

Generally, we can notice that the adaptation cycle has to solve a combination of a *constraint satisfaction problem (CSP)* and an *optimization problem*. On the one hand, it has to find a combination of different components representing a working product, i.e., a product that fulfills all constraints of the application domain. On the other hand, it has to find an optimal combination fulfilling the demands of the customer as well as possible. To solve this special task it is not sufficient to determine weak product parts during the part selection process in any order. It is rather necessary to find adaptation orders which will lead to an optimal configuration result. Generally, we can distinguish between two different approaches to determine adaptation orders:

Preservation of Consistency Up to now we have assumed that a product modification during one iteration of the adaptation cycle must even lead to a consistent product. This means that only adaptation orders that preserve the consistency of the product are allowed.

Temporary Loss of Consistency In contrast to the previously described approach it is also possible to allow the temporary loss of consistency during the adaptation of a base product. That means, if the product validation process determines the violation of constraints after the component exchange process, this must not necessarily lead to an immediately retraction procedure.

It is rather possible only to notice the loss of consistency and to continue the whole adaptation process without the retraction of the last modification. But, to find a final solution that can be presented to the customer it is of course necessary to re-establish the consistency during the further adaptation.

The advantage of the first approach is the quite simple implementation. Besides that, it represents an *any time algorithm*, i.e., it allows the interrupt of the adaptation cycle at any time to obtain the best consistent solution that could be formed so far. Nevertheless, with preservation of consistency it is often impossible to get optimal configuration results. The reason are the different constraints between the components. For example, if we want to install a SCSI-hard-disk we have perhaps also to replace the IO-controller. Therefore, the integration of the SCSI-hard-disk will lead to an inconsistent PC before the necessary IO-controller is installed, too. If we allow a temporary loss of consistency during the adaptation we are able to handle such problems. But, this requires more complex algorithms to guarantee the final consistency of the configured product.

5 Conclusions and State of Implementation

In our opinion, the most important benefit of the presented approach is the capability to realize case-based configuration without requiring a lot of additional domain knowledge. In contrast to the application of adaptation rules/operators our approach needs only the knowledge about existing components and the constraints between them. It is not necessary to describe every thinkable adaptation-step. The selection of adaptation-steps is controlled by the knowledge that is encoded in already existing similarity measures. The additional case information, i.e., the description of the available components, is not critical because it is usually accessible in existing product databases.

Because the presented approach can obviously be classified as transformational adaptation, it is also coupled with the respective advantages: It is able to configure products if the customer demands are not satisfiable and it can handle implicit optimality criteria through the cases that are assumed to be good pre-configured standard products.

To evaluate the functionality of our configuration approach we have implemented a generic prototype for the described configuration process. This prototype consists of an extension of the commercial CBR tool CBR-Works which has been developed jointly by the University of Kaiserlautern and tec:inno GmbH. To be accessible over the World Wide Web the prototype also provides two respective interfaces for the demand acquisition and the result presentation.

In the current version we have modeled the already mentioned PC-domain as a first test domain. However, in the future it could be useful to apply the system on other application domains to get more concrete experiences about the power of the approach. Because of the structure of the selected PC-domain, currently we have only implemented a simple adaptation control algorithm preserving the

consistency of the retrieved product after every adaptation step. Therefore, the next step should be the implementation and the evaluation of different more or less sophisticated adaptation control strategies to improve the adaptation process.

References

1. R. Bergmann, S. Breen, M. Göker, M. Manago, and S. Wess. *Developing Industrial Case-Based Reasoning Applications: The INRECA Methodology.* State-of-the-Art-Survey, LNAI 1612. Springer Verlag, 1999.
2. R. Bergmann and A. Stahl. Similarity measures for object-oriented case representations. In B. Smyth and P. Cunningham, editors, *Case-Based Reasoning Research and Development*, volume 1488 of *LNAI*, pages 25–36. Springer, 1998.
3. R. Bergmann, W. Wilke, I. Vollrath, and S. Wess. Integrating general knowledge with object-oriented case representation and reasoning. In *4th German Workshop: Case-Based Reasoning - System Development and Evaluation 1996*, 1996.
4. R. Burke. The wasabi personal shopper: A case-based recommender system. In *11th International Conference on Innovative Applications of Artificial Intelligence*, 1999.
5. R. Cunis, A. Günther, and H. Strecker, editors. *Das PLAKON-Buch.* Number 266 in Informatik-Fachberichte. Springer, Berlin, 1991.
6. B.D. Netten and R.A. Vingerhoeds. Incremental adaptation for conceptual design in eadocs. In *European Conference on Artificial Intelligence (ECAI'98) Workshop on Adaptation in Case-Based Reasoning*, 1996.
7. K. Pfitzner. *Fallbasiertes Konfigurieren technischer Systeme.* Dissertation, Universität Hamburg, 1993.
8. S. Schmitt and R. Bergmann. Product customization in an electronic commerce environment using adaptation operators. In *7th German Workshop on Case-Based Reasoning*, 1999.
9. I. Watson and S. Perera. Context guided retrieval. In David B. Leake and Enric Plaza, editors, *Case-Based Reasoning Research and Development*, volume 1266 of *Lecture Notes in Artificial Intelligence*, pages 359–370. Springer, July 1997. International Conference on Case-Based Reasoning ICCBR-97.
10. W. Wilke. *Knowledge Management for Intelligent Sales Support in Electronic Commerce.* DISKI 213. Infix Verlag, 1999.
11. W. Wilke and R. Bergmann. Techniques and knowledge used for adaptation during case-based problem solving. In *11th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems IEA-98*, 1998.