Techniques and Knowledge used for Adaptation during Case-Based Problem Solving

Wolfgang Wilke and Ralph Bergmann

University of Kaiserslautern
Centre for Learning Systems and Applications (LSA)
Department of Computer Science
P.O. Box 3049, D-67653 Kaiserslautern, Germany
E-Mail: {wilke,bergmann}@informatik.uni-kl.de

Keywords: Case-Based Reasoning, Adaptation, Knowledge Modelling

Topics: 1.1 Knowledge Modelling, 6.2 Analogy and Case-Based learning, 6.3 Multistrategy and hybrid formulations

Preferred presentation: oral

Acknowledgements

The authors would like to thank: Padraig Cunningham, Harald Holz, Héctor Muñoz-Avila, Jürgen Schumacher, Barry Smith and Armin Stahl for their helpful discussions and comments on the topic of the paper. This work was partially funded by the Commission of the European Communities (ESPRIT contract P22196, the INRECA II project: Information and Knowledge Reengineering for Reasoning from Cases) with the partners: AcknoSoft (prime contractor, France), Daimler Benz (Germany), technno (Germany), Irish Medical Systems (Ireland) and the University of Kaiserslautern (Germany).
1 Introduction

Case-based reasoning (CBR) has recently become a rising star in knowledge-based decision support technology [Thompson, 1997]. It has been used to create numerous applications in a wide range of domains, including prediction, diagnosis, planning, process and quality control, monitoring, supervision, technical maintenance and software systems for decision or customer support. This results from the viewpoint that CBR is seen as a cognitively sound modelling approach for explaining human problem solving in domains where experience plays an important role and as a software engineering approach for how to model and implement decision support systems that are able to use past experiences for suggesting solutions or making predictions.

It is especially appealing to those professionals who solve problems by recalling what they did in similar situations that happened in the past. The main hypothesis behind CBR is quite simple: "similar problems have similar solutions" - or put it the other way round - "you can reuse the solution of a similar problem in order to solve your actual problem". Based on this hypothesis the adaptation of the solution from the most similar problems should be a very efficient and straightforward way to come to a suggested solution for the actual problem. But in reality it is not always so easy.

While the theory and practice of case-based reasoning has benefited greatly from recent advances in case representation, similarity assessment, and retrieval, adaptation is still considered as the most difficult step in CBR. The reason for this is that there is a large variety of different adaptation techniques, all having different requirements on available adaptation knowledge. While it is often easy to acquire the cases, the required adaptation knowledge is often very hard to get. Therefore, one strategy for building CBR applications is to bypass adaptation entirely [Leake, 1996] which can be appropriate for analytic tasks like classification, diagnosis, or decision support. Instead, one tries to blow-up the case base to ensure that for every problem a sufficiently similar case is available. However, for synthetic tasks (like configuration, design, or planning), the vast space of possible problems and solutions disables this approach and adaptation becomes essential.

This paper presents an overview of different adaptation methods which are common in today's systems. We introduce first the process model of CBR and the used knowledge according to the different knowledge containers. Next, we present current models of adaptation and illustrate them in an example domain and close with some remarks.

2 Knowledge Modelling for Case-Based Systems

Before we have a closer look at different adaptation techniques we have to understand the CBR process during problem solving and the used knowledge sources. This is illustrated in Figure 1.
Richer [Richer, 1995] describes four containers in which a CBR system store knowledge. The four containers are: the **vocabulary knowledge** used to describe the whole domain, the **cases knowledge** of the cases in the case base, the **retrieval knowledge** used for the retrieval of similar cases and the **adaptation knowledge** used during the transformation of retrieved solutions.

During problem solving, first a query (Q) is entered by the user. Next, the most **similar problem descriptions** (S) are retrieved from the **case base**. To determine similar problems and their solutions retrieval knowledge, like the indexing of the cases or the similarity measure is used. The case knowledge is represented in the case base, which consists of all known problems and their belonging solutions. The system takes the correlating solutions from the past and **adapts** them to solve the new problem. Therefore, adaptation knowledge is applied that describes how to transform the similar solutions for solving the query problem. Often, adaptation is a necessity and makes problem solving still tractable or even possible at all. This results from the possibility to broaden the coverage of past cases by adaptation knowledge for new problems. Which and how knowledge is represented constraints the coverage of the whole CBR system and establishes the vocabulary knowledge.

### 3 Current Models of Adaptation

In this section we have a closer look at several adaptation techniques which are used in CBR. To demonstrate these techniques, we will first introduce an example domain were personal computers (PC’s) are configured.

#### 3.1 The Example Domain: PC-Configuration

The task of PC configuration is to arrange a personal computer for the special customer needs. A related CBR application can support shop assistants to sell optimal configurations as well as possibly retail computers via the Internet. The customer specifies his requirements by denoting which applications he would like to use the PC for. For capturing this requirements, we introduce a set of attributes for application areas like **word-processing**, **database**, **music**, **programming** or **games**.
Each of these attributes has a certain value denoting the importance of this application area for the user. The goal is to configure a PC which can be used for the desired mixture of applications. The solution itself, i.e. the final PC configuration, is represented using an object-oriented model of the PC and its components.

3.2 The Continuum of Adaptation Models

From a case-based reasoning perspective there is a natural relationship between the complexity of the problem solving task and the complexity of the CBR adaptation process. Even adaptation is a special kind of problem solving for the desired task with the helpful input of a problem description, similar problems and a belonging similar solutions. The underlying hope is, that these useful input together with a special optimised problem solving algorithm for the given information shorten the problem solving process against problem solving from scratch. An overview of several basic adaptation approaches which are exploited in recent research is given in Figure 2. We observe an increasing complexity from null adaptation on the left, over transformational approaches to generative approaches. Further using multiple similar cases for adaptation is called compositional adaptation. We will now describe these different approaches and illustrate them with an example in our problem domain.

3.3 Null Adaptation

The simplest kind of adaptation is null adaptation. The case base is searched for a similar case containing the most similar user requirements and the solution from this case is taken to solve the
current problem without any modification. Null adaptation means: the adaptation is left to the user or it is even not necessary to adapt at all. Most commercial CBR systems work with null adaptation.

### 3.4 Transformational Adaptation

Transformational adaptation means that the old solution of the similar case is transformed into a new solution for the new problem. Transformational adaptation [Carbonell, 1983] supports the reorganisation of solution elements and permits the modification, addition and deletion of these elements under certain conditions. Typically, transformational adaptation systems employ a fixed set of adaptation operators and/or transformation rules. Depending on the difference of problem attributes in the query and the problem attributes in the similar case, the actual solution is modified. Thus, transformational adaptation requires domain knowledge on how certain differences in the problem lead to differences in the solution. Depending on the degree of modification we distinguish between substitutional adaptation and structural adaptation. Transformational adaptation is for example used in: CLAVIER [Barletta and Hennessy, 1989, Hennessy and Hinkle, 1991] or INRECA [Wilke and Bergmann, 1996].

**Substitutional Adaptation** For CBR systems concerned with simple problem solving tasks simple substitutional adaptation is sufficient. It is promising when the retrieved case will typically be very close to the target and consequently will require only minor modifications.

In this adaptation model it is only possible to change the values of attributes. The structure of the new solution remains unchanged. Of course, the values of the attributes are reused from the old case, too, but it is possible to change some of them to get a solution of a better quality. Here adaptation means a recalculation of several parameters depending on the relation of the attributes of the problem description of the query and the similar case.

In our example domain, a possible substitutional adaptation is the modification of the hard disk space depending on the usage of database systems with the configured PC. If the value for database usage in the problem descriptions differs there is the necessity to add hard disk space. This could be coded into simple rules, like:

\[
\text{query.database} - \text{similar.database} > 5 \Rightarrow \text{solution.diskspace} = \text{solution.diskspace} + 2\text{GB}
\]

Detailed descriptions of substitutional adaptation can be found in [Bain, 1986].

**Structural Adaptation** With routine problem solving tasks, the retrieved case, will probably require more substantial modifications. If we allow to change the structure of the solution during adaptation we speak of structural adaptation. Structural adaptation [Goel and Chandrasekaran, 1989,
Smyth and Keane, 1996] supports the reorganisation of solution elements and permits the addition and deletion of such elements under certain conditions. Also structural adaptation systems employ a fixed set of adaptation operators and/or transformation rules which modify the structure of the solution, depending on relations between the problem description of the query and the similar case. Structural changes can occur when complete components are added or deleted or when another object replace an existing one.

An example: The similar case being retrieved was not intended for playing computer games. One of the components which is needed only for games is a joystick. Therefore the solution in the old case does not contain a joystick. If the customer demands computer games in the query, during adaptation an instance of the object joystick must be added and configured to make the old case suitable for the new query. An adaptation rule which realises such a structural modification would be:

\[ \text{query.games} > 0 \text{ and similar.games} = 0 \Rightarrow \text{AddObject : Joystick} \]

In contrast to the first example, joystick is a new complex object in the object-oriented model of the solution and not an already existing part of the solution which is modified only.

### 3.5 Generative Adaptation

Generative adaptation is radically different from transformational adaptation. Generative adaptation requires a generative from-scratch problem solver that is tightly integrated into the case-based reasoner for the purpose of adaptation. Typically, such a problem solver (e.g., a configuration system in the introduced PC domain) is in principle able to solve the type of problem that should be handled by the case-based system alone, i.e., without the use of cases. However, in practice such a pure generative problem solver is usually insufficient because of the computational complexity of the generative problem solving process or because of the insufficient quality of the solutions it produces. During generative adaptation, the generative problem solver is of course not used to solve the whole new problems from scratch, but only to generate those parts of the solution that are inadequate due to differences between the current problem and the retrieved case. Thus, drawbacks like lack of efficiency or quality don’t have a great impact as long as the retrieved case is sufficiently similar to the current problem.

As a consequence of the use of a generative problem solver, generative adaptation requires a different kind of knowledge than transformational adaptation. Instead of having knowledge that describes how differences in the problem description lead to differences in the solution, knowledge that allows to construct a solution from scratch is required. The particular knowledge needed is strictly dependent from the generative problem solver that is used. For example, in the introduced
PC domain, a configuration systems needs knowledge about the components like the function they fulfill and the constraints that exist between different components. For a particular graphics adapter, for instance, the following knowledge must be available: The Matrox Millennium graphics adapter is suited for high performance graphics and it has a PCI bus interface. The knowledge about the bus interface poses a constraint on the mainboard that can be used and therefore restricts the set of correct configurations.

Unlike transformational adaptation which transfers a previous solution to a new problem, generative adaptation transfers the derivation of a previous solution (the reasoning trace) to the new problem. Therefore, this kind of adaptation is also called derivational analogy [Carbó nell, 1986]. Reasoning traces record information about the derivational process that lead to a particular solution in the past, including decision information, justifications, and reasoning alternatives. During generative adaptation, these traces are essentially replayed in the context of the new target problem, i.e., the same or similar decisions are also tried to solve the current problem. If certain decisions from a previous reasoning trace cannot be transferred to the current problem, the generative problem solver will come up with a decisions by itself, without reusing information contained in the cases. In order to enable replay of reasoning traces, they must of course be stored in the cases, i.e., the representation of cases must be extended.

We can basically distinguish two different kinds of replay strategies: one shot replay and interleaved replay (see Figure 2) which we will discuss in the following sections.

![Fig. 3. Example for one shot and interleaved replay.](image-url)

**One Shot Replay** In one shot replay, one first identifies which portions of the solution trace can be reused in the context of the new situation. These decisions are then replayed by the problem solver. After the replay is finished, the problem solver will make the remaining decisions on its own.

The left side of Figure 3 shows an example in the PC domain. The new problem is to configure a PC...
basically used for music applications and not used for games. The retrieved case in the case-base, however, is configured for being used for games and not for music. The depicted solution trace for the case shows the configuration steps that had been taken to configure this particular PC. In one shot replay, the decisions 1, 2, and 5 from the case are replayed first because they are also valid decisions for the new problems. Then, the remaining decisions concerning a different graphic adapter and the sound card are taken. Caplan/Cbc [Muñoz and Weberskirch, 1996] is an example of a case-based planning system that uses one-shot replay.

**Interleaved Replay** In *interleaved replay*, we can switch several times between the replay of a previous solution trace and the generation of certain new decisions by the problem solver. The solution trace is followed up to a certain point where following it further would not be beneficial any more. Then, the generative problem solver takes over until a point is reached where the previous solution trace can be followed again. The right side of Figure 3 shows interleaved replay for the same problem used already for demonstrating one shot replay. First, decisions 1 and 2 are replayed. Then, the problem solver takes over and determines a different graphic adapter. Then, again, step 5 from the case is replayed, i.e., the CD-ROM is selected. Finally, the problem solver configures the sound card being used. Prodigy/Analogy [Veloso, 1994] is an example of a case-based planning system that uses interleaved replay.

We see that the basic difference between the two approaches is only the sequence in which replay occurs and not the steps that are replayed. In this examples it might seem easy to decide whether a step can be replayed for a new situation or not. However, in many situations this is not the case, particularly when there is a lot of interaction between the reasoning steps. In interleaved replay, it can also become very difficult to decide when to switch between replay and problem solving. Up to now, there is no general solution to this problem.

**Solution trace replay vs. complete decision replay** An other important difference between different techniques is the amount of information about the solution trace that is reused. In *solution trace replay*, only the selection of the solution elements (selection of PC components in Figure 3) is recorded and replayed. In *complete decision replay* all reasoning information in the problem solving process are recorded and possibly take into account. These reasoning information can be justifications, and reasoning alternatives, or failed attempts.

### 3.6 Compositional Adaptation

In addition to these adaptation models, recent research has demonstrated the power of delivering solutions through the retrieval, adaptation, and subsequent composition of multiple cases. This
leads to compositional adaptation [Redmond, 1990, Sycara and Navinchnadra, 1991], in which newly adapted solution components from multiple cases are combined to produce a new composite solution.

Newer approaches indicate that it makes sense to compose a solution from parts of several old cases. This is possible if the solution consists of different parts which can be adapted more or less independently and it is effective if there are few conflicts between these components so that a change in one component does not have several side-effects on other components. Compositional Adaptation is used in Cadsyn [Maher and Zang, 1993], Composer [Purvis and Pu, 1995], Déjà Vu [Smyth and Cunningham, 1992], Prodigy/Anology [Veloso, 1994] or Caplan/Cbc [Muñoz and Weberskirch, 1996].

3.7 Hierarchical Adaptation

Hierarchical adaptation is an other recent development that is used in combination with the shown adaptation models [Bergmann and Wilke, 1996]. Cases are stored at several levels of abstraction and the adaptation is performed in a top-down fashion. At first, the solution is adapted at the highest level of abstraction (omitting less relevant details). Then, the solution is refined in a stepwise manner and the required details are added. Hierarchical adaptation can either reuse a single case or it can reuse different cases for different levels of abstraction or for refining different details of the solution. Examples of systems using hierarchical adaptation are Déjà Vu [Smyth and Cunningham, 1992], Paris [Bergmann and Wilke, 1995], or the stratified CBR approach by [Branting and Aha, 1995].

4 Discussion

Adaptation is still a big search issue in case-based reasoning that is also getting increasingly important in practice as the complexity of applications increases. During the last years, there have been several workshops and published papers [Kolodner, 1993, Hanney et al., 1995, Voß, 1996] that try to systematically analyse the different approaches to adaptation that can be found in current CBR systems. However, all these approaches lack a systematic approach to design an appropriate adaptation method particularly suited for a given application and considering the knowledge available in this domain. We consider this one of the major challenges that need to be addressed in the future.

References


