Knowledge Containers

Michael M. Richter
TU Kaiserslautern
FB Informatik, P.O.Box 3049
67653 Kaiserslautern, Germany
richter@informatik.uni-kl.de

Abstract
In this article we discuss some general and basic questions of knowledge representation with special emphasis on Case-Based Reasoning systems. We will discuss the four major elements of such a system, the knowledge base, the similarity measure, the solution transformation, and the vocabulary used. The major point is how to use them for representing knowledge in a systematic way and how to improve the system over time. For this purpose we take first a more general view.

1. Preliminaries
Knowledge bases have some structure, like programs they are organized in modules. Each module solves a specific task or subtask and can be called and used by other modules. The purpose of knowledge based systems is to store knowledge explicitly such that it can be used for solving problems. Part of the knowledge is represented directly and another part is indirectly stored; in that it can be derived using methods of inference. Hence, the totality of the represented knowledge is the deductive hull of the directly represented knowledge under the inference methods. A knowledge based system can certainly be and often is organized in modules. On the other hand, the system’s knowledge may be organized onto different modules as well. Representation formalisms usually have different description elements, which constitute the representation language. Each such language element, or more generally each element of the formalism (which in addition may be e.g. rules of inference or certain algorithms) contributes to the description of knowledge units or knowledge processing methods. So to speak, the totality of these description elements is able to represent the intended knowledge. Each element alone will only represent a certain part or aspect of the knowledge and will in general not be able to solve completely a subtask. These description elements are usually of different and sometimes of heterogeneous nature. As an example we mention firstly the language elements facts and rules in logic programming. To constitute a logic program one has to define the facts and the rules in such a way that all the intended knowledge is represented. This can of course be done in different ways. We will call such description elements knowledge containers.

The way the knowledge is expressed in a knowledge based system is given by the type of knowledge representation. It consists of certain data structures and additional inference operations, which allow manipulating these data structures. Hence data structures are essential for representing knowledge. On the other hand, we do not identify them with knowledge containers. The main difference is that data structures are of a more elementary character than knowledge containers but they are needed to constitute the containers. The same data structures can be used for different knowledge containers and a knowledge container may require several data structures.

In the center of our interest are the knowledge containers for Case-Based Reasoning (CBR) although we will also briefly discuss other representation formalisms. In almost all other knowledge representation systems all knowledge that is represented has to be understood and represented properly. For the moment we will call this a compilation process. In addition, it has mostly to be complete in the way a procedural program has to be complete: If something is missing then the system will not work at all or present solutions with major mistakes. A CBR – system will work even if the knowledge is incomplete, not totally exact and not very efficiently to use. In this situation it will not provide totally correct but more or less useful solutions. Hence one can start with an initial CBR – system, relying heavily on the knowledge in the case base and improve it by later compilation steps, which shift the knowledge between containers. The interplay between the containers is the major reason that the concept of knowledge containers is especially interesting for CBR.

Our investigation of knowledge containers will concentrate on the following topics:
• What kind of knowledge is in the containers and how to fill them?
• At which time of the system development will the containers be filled?
• How to use the containers for problem solving and improving the system?
• How to shift knowledge between containers?

The paper is partially based on an invited talk of the author at the ICCBR95 at Sesimbra, Portugal 1995, see also (Richter 1998).

2. Knowledge Containers in Case-Based Reasoning

In CBR we identify four major knowledge containers as represented in the following diagram.

![Diagram of four major knowledge containers]

There is an interaction between the containers:

The available knowledge is distributed over the containers, as indicated by the arrows. In a more detailed view the containers may be split up into different subcontainers. Because no container is able to solve completely a task the containers depend on each other. Therefore an inadequately filled container may provide a burden to other containers. The term inadequately filled refers to deficiencies in a knowledge container that affect quality aspects of a CBR system such as: Correctness of solutions, competence of the system, and efficiency of problem solving or maintenance aspects. Here the competence is roughly the ratio between the number of solvable problems and the number of possible problems, see e.g. (Smyth, & McKenna 1998).

On the other hand we face the fact that some knowledge can be more easily represented in some containers and is much more difficult to represent in others. This leads to a central aspect of CBR: One can start with a working system of lower quality and improve the system by reorganizing the containers over time.

Next we will discuss the different containers.

2.1 The Vocabulary

One of the first questions to be handled in a knowledge representation system is which data structures and which elements of the structures are used to represent primitive notions. These may be e.g. predicates, attributes, functions or related constructs. The structure most common in CBR is the attribute-value representation. Other structures, e.g. taxonomic ones can be built up from attributes. In addition, predicates are often used. For our purposes it is, however, sufficient to consider attributes.

For an attribute-value representation it is relevant to find out which attributes with which semantics are chosen. The completeness of the set of attributes can by judged by two criteria:

• Principal completeness: All relevant properties can be formulated. If the attribute set is incomplete in this sense then certain aspects or properties of interest cannot be represented in the system. This is known as the
phenomenon of missing parameters. Example: Missing attribute “age” of a patient when designing a therapy.

- Efficiency: For a complete attribute set it may happen that a certain relation between attribute values is important for a decision, e.g. the quotient of the values of attribute A and attribute B (which may be expressed by a new attribute C) is relevant for a therapy.

A related problem occurs when the original attributes are not operational and the value of C is computable in principle but the computation of the value is involved. This is related to the ideas used in explanation based learning. Here the missing knowledge has the consequence that the system is not directly aware of the relevance of C. We call such additional attributes virtual attributes. Their addition can improve the efficiency of the system significantly and may even lead to the deletion of other attributes.

In the vocabulary container one can identify various subcontainers; examples are:

- Retrieval attributes: Useful for computing similarities
- Input attributes: for textmining and completion rules
- Output attributes: For information of the user (e.g. in help-desk applications)

The vocabulary (e.g. the chosen attributes) is basic for all other containers. It can be used for various types of descriptions, ranging from logical constructs to free text.

2.2 The Similarity Measure

The similarity measure sim maps pairs of problem descriptions (i.e. elements of \( P \times P \)) to the real unit interval \([0, 1]\). We assume that sim satisfies the local – global principle, i.e. sim is constructed from local measures \( \text{sim}_i \) on the domains of the attributes \( A_i \) using an amalgamation function \( f \):

\[
\text{sim}(q, p) = f(\text{sim}_i(q_i, p_i) \mid i \in I).
\]

For efficiency reasons measures with linear \( f \) are preferred; then sim is a generalized weighted Hamming measure:

\[
\text{sim}(q, p) = \sum (g_i \times \text{sim}_i(q_i, p_i) \mid 1 \leq i \leq n).
\]

where \( g \) is a weight vector \( g = (g_1, ..., g_n) \) of non-negative real coefficients (often from the real unit interval \([0,1]\) and normalized to \( \sum g_i = 1 \)).

The measure has two subcontainers, the local measures and the amalgamation function, e.g. the weight vector. The local measures contain mainly domain knowledge while the amalgamation function is task oriented and contains utility knowledge (relevances for the task), see below.

An important property of a measure connected with the linearity is formulated in the

**Monotonicity axiom:**

If \( \text{sim}(q, p) > \text{sim}(q, r) \) then there is at least one \( i \in I \) such that \( \text{sim}_i(q_i, p_i) > \text{sim}_i(q_i, r_i) \).

(Burkhard & Richter 2001).

Weighted Hamming measures satisfy the monotonicity axiom. The question arises whether a given measure can be represented as a weighted Hamming measure. This axiom is related to the vocabulary container. Consider e.g. the XOR – problem: There are two binary attributes and four possible cases \((i, j), i, j = 0, 1\). Assume that three cases are already in the case base and the last one has to be classified correctly as XOR requires. Then no weighted hamming measure can do this because the monotonicity axiom fails. The way out is to introduce a third virtual attribute (similar to the introduction of a hidden neuron in a neural net). The introduction of such an attribute shifts the non-linearity from the computation of the measure to the computation of an attribute value.

The basic demand on the measure is that the nearest neighbor technique provides in fact the best available solution. To be more precise, if for some query problem \( q \) the case \((p, s)\) has the property that \( p \) is the nearest neighbor to \( q \) then \( s \) is indeed the most useful solution for \( p \) that is contained in the case base. This leads to the utility concept, which has two versions, a relational and a functional one. Suppose \( p \) is a problem and \( s_1 \) and \( s_2 \) are two elements of the solution space are given.

**Def.:**

(i) A preference relation is of the form \( \text{pref}(p, s_1, s_2) \), read as “\( s_1 \) is preferred to \( s_2 \) for \( p \)” (or is “more useful than”).

(ii) A utility function \( u \) is a real valued function of the form \( u(p, s_i) \) verbalized as “\( x = u(p, s_i) \) is the utility of \( s_i \) for \( p \)”.

A common semantics for a similarity measure is that it reflects or at least approximates the utility of the user. This means that for a problem \( p \) and a case \((p, s)\) the equation \( \text{sim}(q, p) = u(q, s) \) is at least approximately true. In this view the knowledge contained in a measure is concerned with the knowledge about the underlying utility
function, it contains utility knowledge. In particular, the nearest neighbor to a query problem should give the most useful solution. (See e.g. (Bergmann et al. 2001).

In addition, a measure can contain also efficiency knowledge. It may happen that the utility can be sufficiently well approximated by two measures, which need different computational effort, e.g. a weighted Hamming measure and a non-linear one. In such situation the measures contain different efficiency knowledge.

2.3 The Case Base

The case base CB contains the experiences. These experiences have to be available or can be constructed by variations of existing cases.

The first requirement is that the case base should only contain cases \((p, s)\) where the utility of \(s\) is maximal or at least very good for the problem \(p\). This is knowledge contained in the individual cases. There are two other but conflicting demands on CB:

1) There should be as many cases as possible in CB because each additional case can possibly enlarge the competence of the system (concerned with competence knowledge).
2) The case base should be as small as possible because each new case can extend the search time (concerned with efficiency knowledge).

2.4 The Solution Transformation

The solution transformation has to take care of fact that the solutions obtained from the case base using the nearest neighbor principle may still be insufficient (either by a not very well defined similarity measure or simply by the fact that the case base does not contain a better solution). In this situation the solution is adapted.

The adaptation is usually performed by rules; in this case the adaptation container is filled with such rules.

The set of problems covered by the system then is given as the deductive closure \(\text{CB}\) of the case base \(\text{CB}\) under the adaptation rules. Hence the nearest neighbor search takes place in \(\text{CB}\), i.e. in a set, which is only partially given explicitly. The relation of the rules to the similarity measure is complicated by the fact that a necessary rule application may not increase the similarity to the query.

3. Further Relations between the Containers; Compile and Run Time

A simple observation is that any of the four containers can in principle contain essentially all relevant solution knowledge. In practice, however, this is usually impossible and in addition, such containers would contain little efficiency knowledge.

The vocabulary: We need for each problem description \(p\) simply an additional (ideal) virtual attribute sol which has as domain all solutions and where \(s = \text{sol}(p)\) gives the correct solution.

The similarity measure: For an ideal measure \(\text{sim}_{\text{ideal}}\) we could have \(\text{sim}_{\text{ideal}}(q, p) = 1\) if the case \((p,s)\) provides some best solution \(s\) and \(\text{sim}_{\text{ideal}}(q, p) = 0\) otherwise.

The case base: An ideal case base in this sense would simply contain all possible cases.

The solution transformation: One can simply ignore the cases and construct the solution from scratch using the adaptation rules.

Such „ideal“ objects are of course neither available nor always desirable.

The answer to the question which knowledge is filled in which containers in which way depends to a large degree on the way the knowledge is presented. To discuss the filling procedure it is useful to distinguish between compile time and run time, borrowed from traditional programming terminology. By compile time we mean the time when a system is built, in particular before any actual problem is presented; here we assume that system development and problem solving is not interleaved. By run time we mean the time when an actual problem is solved.

Of course, all containers are in some sense filled at compile time except for the case base which may be filled incrementally. An important point is, however, that not all knowledge has to be understood at compile time, in contrast to the situation in procedural programs.

The primary example is the case base. The cases can be stored in the case base without understanding them at all. The cases can e.g. simply be copied from existing files. They have to be understood only at run time when they are used to solve problems. For all other containers one has to understand, in some sense, the knowledge:

- One has to collect all necessary and useful attributes.
- One has to define a suitable similarity measure.
One has to define useful and correct adaptation rules. Of course, to store knowledge is easier than to compile it. Hence the container “case base” is the easiest to fill (if cases are available). On the other hand, a purpose of compiling knowledge is to improve efficiency.

There are two ways to improve the knowledge in the containers:

- Improving the knowledge for the individual containers separately
- Shifting knowledge between containers.

These two aspects are important for:

- Development of a CBR system (see Bergmann et al. 1999).
- Maintenance of a CBR system, in particular as a reaction to changing contexts. (see e.g. Roth-Berghofer 2002).

There are basically two ways of improvement:

- Improvements by humans
- Improvements by machine learning techniques.

First we will discuss the improvement of individual containers.

(a) The vocabulary: There are the three principal methods of removing, adding or modifying terms; the latter can be considered as a macro and will be neglected. Removing terms is triggered by redundancy and functional dependencies of attributes while adding attributes is triggered by the detection of useful virtual attributes. The need for virtual attributes is shown primarily by failures of the monotonicity axiom. Generating the appropriate attributes is an induction procedure and is challenging for symbolic learning methods; little progress has been made on this problem.

(b) The similarity measure: There are no results on obtaining the general structure of a similarity measure in a systematic way. The research has concentrated on learning feature weights of weighted Hamming measures; see (Wettscherek & Aha 1995), (Patterson et al. 2000).

(c) The case base: The first developed methods deal with removing cases. This means, to eliminate cases, which do not contribute at all, or contribute little to the problem solving quality of the system. Early but useful methods are provided by the IBLi, i = 1, 2, 3, algorithms as described in (Aha et al. 91). The idea is to list all cases in a linear order, to investigate this list step by step and to eliminate those entries seem to be unnecessary at this point of time. Although this is quite useful in principle there are some pitfalls, which are due to the fact that something superfluous in a certain situation may be useful in a later situation.

In order to improve a case base one needs a concept of the quality of a case base. Such concepts one finds in (Racine 1997) or (Smyth & McKenna 1998). These concepts are closely related to the concept of competence and contain criteria like consistency, redundancy or reachability.

A different quality concept for the case base was given in (Leake & Wilson 1999). It was concerned with the property that problems which really encountered in practice could be solved which means that utility orientation played the major role.

(d) The solution transformation: There is little known about a systematic improvement of adaptation rules. One refers here mainly to standard techniques of rule based systems. Examples of learning adaptation rules from a case base are e.g. presented in (Hanney, & Keane 1996), (Hanney & Keane 1997), (Wilke et al. 1997).

Learning adaptation rules was explored in (Leake & Kinley 1995). This describes a shift of knowledge between the containers case base and solution transformation, see below.

4. Shifts between Knowledge Containers, Learning

The improvement of containers is a major aspect from the viewpoint of software engineering. Besides improving an individual container the shift from one container to another one is important. In fact, the techniques mentioned in the last section for improving one container have an influence on the other containers too. One has to observe that shifting knowledge does not always imply that the knowledge from the source has to be deleted.

The way to built a system can in principle be of the following form:

- start with a given or an obvious vocabulary
- take an existing set of cases
- take a very simple and obvious similarity measure
- start with no adaptation rules.

Of course, from such an initial system one cannot expect wonderful solutions. On the other hand, it may still be useful and support e.g. a human operator at a help desk.

Next we come to the problem of improving such an initial system.

Traditionally, we distinguish between vertical and horizontal compilation. A vertical compilation translates between different levels of abstraction. A classical compiler translates to a level closer to the hardware level; this is called vertical compilation. Horizontal compilation in knowledge based systems takes place between expressions on the same language level, e.g. going to a more compact or abstract formulation. The purpose of
Horizontal compilation is to improve e.g. efficiency, size of the knowledge base or understandability. Major examples are replacing terms by variables, omitting details and introducing new abstract concepts. This kind of abstraction takes also place in CBR – systems, e.g. by introducing generalized cases (cf. Bergmann, Vollrath, Wahlmann 1999).

The shifts between the knowledge containers are often not of explicit nature, they make implicit use of the relation between containers. The easiest container to handle is the case base; the modifications are simply deletions and additions of cases. The deletion of cases is more delicate because it can be an implicit consequence of improving another container. Examples mentioned above are the introduction of virtual predicates or the changes of feature weights in the measure. Another example is the learning of similarity measures from adaptation rules, see (Leake et al 1996). Here the adaptation rules are not deleted, only the measure is improved. It should be mentioned, however, that the explicit interplay between the containers is not yet fully understood.

5. Other Representation Systems

Knowledge containers do not only occur in CBR, they are relevant in many other areas. Implicitly they have been dealt with at many situations. For illustration, we will briefly mention containers of two other knowledge representation systems.

Rule based systems:

Here we have facts and rules of the form

\[ A_1, A_2, \ldots, A_n \rightarrow B. \]

We distinguish the forward application of the rule and backward chaining which reduces as e.g. in Prolog the satisfaction of B to that of the premises. A major difference to the CBR-approach is that only exact matches are admissible in order to achieve the conclusion B.

Knowledge can be distributed over facts and rules in different ways. Initially it is simpler to express knowledge in terms of facts; they often are stored in a data base.

The replacement of many facts by few rules is a horizontal compilation step, which compactifies the program. Example:

Set of facts = \{p(a), q(a), p(b), q(b), \ldots, p(u), q(u)\}

After shifting: \{p(a), p(b), \ldots, p(u), p(x) \rightarrow q(x)\}

The rule container can be split into the container ground rules and generic rules (i.e. rules with variables). A shift between these containers is again a compactification. Example:

\{p(ai) \rightarrow q(ai), i = 1, \ldots, n\} is shifted to \{p(x) \rightarrow q(x)\}.

The learning of rules is extensively treated in inductive logic programming.

Another type of shift takes place between rules, which are used in a forward mode, and rules, which are used, in a backward mode.

Fuzzy-Logic:

This has the following containers:

1. Linguistic rules
2. Fuzzy membership functions
3. t-norms and co-t-norms
4. Defuzzification methods
5. Adaptation rules

These containers have very different tasks to perform and it is presently not very well understood how knowledge can be transformed between them in a systematic way.

6. Summary

Containers provide the basis for efficiently building and maintaining systems, in particular CBR systems. This is due to the possibility of changing the content of a container, a process that we called compilation.

Because the compilation is intended to improve the represented knowledge in some way there is an optimization involved, which is initially not fully understood. In such situations the compilation process itself is a priori unknown. The establishment of the compilation is then usually the result of a learning or data mining process. It may also be that the content of a container is not compiled at all but interpreted at run time. This allows applying CBR systems even in some initial stage of understanding. In this view containers are an important concept from the view of software engineering.

As we have pointed out, the concept of knowledge containers is connected with many interesting aspects. Besides special CBR problems machine learning techniques, software engineering questions and general aspects of knowledge representation enter the scenario. This opens the view for many interesting and promising research topics.
Acknowledgements

The author first thanks David Leake and Thomas Roth-Berghofer for reading the paper and providing very valuable improvements. Thanks also to Ian Watson and David Aha for encouraging me to write this paper finally. Also thanks go to Ralph Bergmann, Sascha Schmitt and Armin Stahl for many discussions in the past about this topic.

References