CASE BASED REASONING – A SHORT OVERVIEW

Z. Budimac, V. Kurbalija
Institute of Mathematics and Computer Science, Fac. of Science, Univ. of Novi Sad
Trg D. Obradovića 4, 21000 Novi Sad, Yugoslavia
zjb@im.ns.ac.yu, kurba@im.ns.ac.yu

Abstract: *Case based reasoning is a relatively new approach in intelligent search of (large) databases. Every new search is based on previous (similar) cases, thus including an experience in the search-engine. This approach gained much attention in e-business applications and search over the Internet. The paper presents basics of case based reasoning systems, while the talk concentrates on realistic examples.*

Keywords: artificial intelligence, case-based reasoning systems

1. Introduction

Generally speaking, case-based reasoning is applied for solving new problems by adapting solutions that worked for similar problems in the past.

In this section, some formal or informal definitions of the basic concepts will be given. Definitions are taken from (Lenz, 1998).

1.1. Knowledge

Knowledge can be understood as an informal notion describing something that a human, a formal system or a machine can possibly use in order to perform a certain task (to solve a problem). In order to use knowledge some entities need to have an access to it and to know how to apply it in solving problems.

The way the knowledge is expressed is the type of knowledge representation, which consists of certain data structures and some additional operators that allow changes on the data structure. The most common data structure in case-based reasoning is the *attribute-value representation*. Every attribute is given by:

- a name A
- a usually finite set $DOM(A)$ called the domain of the attribute A
- a variable x_A

For a finite set A_i, $1 <= i <= n$, of attributes an *attribute-value vector* is an n-tuple $(a_1, ... ,a_n)$ such that $a_i \in DOM(A_i)$. However, if one wants to deal with incomplete knowledge (which is occurring frequently in case-based reasoning), he must allow that some variables exist without values (value unknown).

1.2. Basic Concepts

Case-based reasoning is a problem solving technology. The basic scenario for case-based reasoning, from the simplified point of view, looks as follows: In order to find a solution of an actual problem one looks for a similar problem in an experience base, takes the solution from the past and uses it as a starting point to find a solution to the actual problem.

A general desire in every knowledge-based system is to make use of the past experience. An experience may be concerned with what was true or false, correct or incorrect, more or less useful. It can be represented by a rule, constraint, some general law or advice or simply by saving a past event. From all of this the main idea of the case can be obtained. The *case* is some recorded situation where the problem was totally or partially solved. In its simplest form, the case is represented as an ordered pair

(problem, solution)

The existence of the case means that the corresponding episode happened in the past. This episode contains some decisions that a decision maker found useful. However, somebody else may not be happy with such a case and neglect it. From this follows that cases must be selected carefully, so different categories of cases can exist: good, typical, important, misleading or unnecessary.

A case base is a set of cases, which is usually equipped with some additional structure. A structured case base is usually called a *case memory*.

In many practical applications, one deals with problems with incomplete information. Both, the problem and the solution part in the case may be incompletely described. In these situations we talk about *incomplete cases*. The case completion is another task that can be solved by using case-based reasoning technology.

The next very important concept in the case-based reasoning is *similarity*. While in classical databases information can be retrieved by using only exact matches, in the case-based reasoning cases can be retrieved by using even inexact matches. The notion of similarity is equivalent to a dual mathematical concept - *distance*.

In the functional way similarity can be defined as a function $sim : U \times CB \rightarrow [0 , 1]$ where U refers to the universe of all objects, while CB refers to the case base (just those objects which were examined in the past and saved in the case memory). The higher value of the similarity function means that these objects are more similar. The boundary case is $sim(x, x) = 1$, which means that each object is the most similar to itself.

Retrieval is a basic operation in databases and therefore in the case base too. A query to a database retrieves some information by an exact match by using a key, while a query to a case-based reasoning system presents a problem and returns a solution by using inexact matches with the problems from the cases in the case base. As in databases, trees play a major role in efficient retrieval. Some examples of retrieval structures are: kd-trees (k-dimensional trees), case retrieval nets, discrimination nets etc.

The simplest way to use retrieved case is simply to take the unchanged solution of that case as the solution to the actual problem. However, in many applications even small differences between the actual and the case problem may require significant modifications to the solution. Making the appropriate changes to the case solution is called a *case adaptation*. A general demand is that the solution of a similar problem should be easily adapted to a solution of an actual problem.

The *knowledge container* is the structural element, which contains some quantity of knowledge. The idea of the knowledge container is totally different from the traditional module concept in programming. While the module is responsible for a certain subtask, the knowledge container does not complete the subtask but contains some knowledge relevant to

many tasks. Even small tasks require the participation of each container. The concept of the knowledge container is similar to concepts of the nodes and propagation rules in neural networks.

In case-based reasoning we identify the following knowledge containers: a) the vocabulary used; b) the similarity measure; c) the case base; and d) the solution transformation. In principle, each container can carry almost all knowledge available. From a software engineering point of view there is another advantage of case-based reasoning - the content of the containers can be changed locally. This means that manipulations on one container have little consequences on the other ones. As a consequence, maintenance operations (Iglezakis 2001), (Reinartz 2000) are easier to be performed then on classical knowledge based systems.

The task of machine learning is to improve a certain performance using some experience or instructions. In inductive learning, problems and good solutions are presented to the system. The major desire is to improve a general solution method in every inductive step. Machine learning methods can be used in order to improve the knowledge containers of a case-based reasoning system (the case base, similarity measures and the solution transformation). However, one of the greatest advantages of the case-based reasoning system is that it can learn even through the work with users modifying some knowledge containers.

The case based reasoning system has not only to provide solutions to problems but also to take care of other tasks occurring when it is used in practice. The main phases of the case-based reasoning activities are described in the *CBR-cycle* of (Aamodt and Plaza, 1994) in Figure 1.

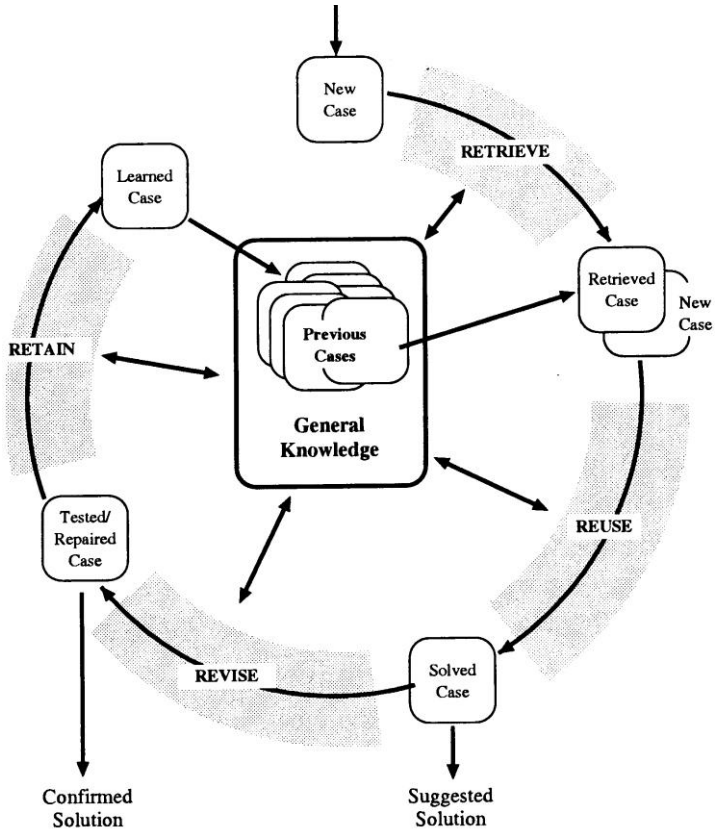


Figure 1. *CBR-cycle* of Aamodt and Plaza (1994)

In the *retrieve* phase the most similar case (or k most similar cases), to the problem case, is retrieved, while in the *reuse* phase some modifications to the retrieved case is done in order to provide better solution to the problem (case adaptation). As the case-based reasoning only suggests solutions, there may be a need for a correctness proof or an external validation. That is the task of the phase *revise*. In the *retain* phase the knowledge, learned from this problem, is integrated in the system by modifying some knowledge containers.

2. Extending some Foundations of Case-Based Reasoning

The case-based reasoning was developed in the context and in the neighborhood of problem solving methods, learning methods (Machine Learning, Statistics, Neural Networks) and retrieval methods (Data Bases, Information Retrieval). It has inherited the concepts of "problem" and "solution" and a notion of "similarity" based on the distance.

The basic concepts of the case-based reasoning can be extended in the following way:

- we will use *acceptance* instead of *similarity*, because acceptance includes similarity but also other approaches related to "expected usefulness", "reminds on" etc.
- we will use the term *case completion* instead of *solution* (including solutions of problems but also a proposal of intermediate problem solving steps).
- we will consider cases as sets of *information entities* instead of *vectors* (whereby this approach includes vectors as well as textual documents).

2.1. Case completion

Case-based reasoning is considered as a problem solving method; given a problem we have to find the solution. This leads to a view where cases are split into the problem and the solution part. Given a new problem we search for related problems in the case memory and adapt their solution for the new problem. However, problem solving usually does not start with a complete problem description, which makes the identification of a final solution more difficult.

Nevertheless, the new case is often talked of as a given entity when only the first impression of the underlying task is given. Instead of this, we want to keep attention to the fact that the whole process of completing the task up to the final solution. The consequence is that the resulting case usually depends on a number of decisions. These decisions are initially open; they depend on future human decisions. Depending on different possible decisions, we can end up with different cases.

Most practical tasks are performed as processes with a lot of intermediate steps. Each of these steps could be considered as a single new problem-solving step. The question arises, as to whether we need different sets of cases to support each of these steps. This would mean splitting the whole story of the process into different cases for a later usage by the case-based reasoning. Case completion is an attempt to avoid such approach. A case should be a description of the whole performance of the task with all steps, and it should be useful for later tasks at intermediate situations, too. The main consequence of the case completion is that we do not actually need any distinction between the problem and the solution part in the case.

2.2. Information Entities

Information entities are the atomic constituents of cases and queries. We consider a case as the result of the case completion process. Each step of that process adds some information entities. The current situation during the elaboration of a task is described by the information entities known at the time point. The final case, as it later may appear in the case memory, is a completed set of information entities.

The collected information entities result from the real world (an outcome of the test, a decision in an intermediate design step, etc). They are not a direct result of the case-based reasoning process - case-based reasoning is used to propose the next step (some test, the next design decision etc).

The number of information entities in a case may be variable. It is up to a human decision at which time point the task is finished.

The information entities, which are later used for retrieval, (which appear in the case memory) may be only a subset of the information entities collected during the case completion. These information entities (in the case memory) serve as the indexes for retrieval. The case memory consists of cases, which are sets of such information entities. These cases may then point to related complete descriptions in a collection of "full cases".

The information entity is an atomic part of a case or query. E denotes the set of all information entities in a given domain.

- A case is a set of information entities: $c \subseteq E$.
- The set of cases (in the case memory) is denoted by C, $C \subseteq P(E)$.
- A query is a set of information entities: $q \subseteq E$.

In many applications, the information entities are simply attribute-value pairs. Some examples of information entities are: *<price, 1000>, <price, 324>, <color, blue>, <mass, 54 kg>*. We say that the first two information entities are comparable (because they have the same attribute) while the other information entities are not comparable. This causes a structuring of the set E into disjoint sets E_A, where E_A contains all attribute-value pairs from E for a certain attribute A. If cases and queries are considered as attribute-value vectors over a finite set of attributes $A_1,\ldots,A_n$, then each case or query may contain at most one information entity from each E_{Ai}.

2.3. Acceptance

We want to use the association of information entities for reminding cases with the expectation that these cases are useful for a given query. Usefulness of a case in the case completion process depends on real world circumstances that are not completely known at the retrieval time. This means that usefulness is only an a posteriori criterion. The retrieval from the case memory will be based on matching of certain information entities. Usefulness of former cases is not restricted to those cases that are similar to a given query for all information entities. Cases may contain information entities that have no counterpart in the query. It is also possible that some information entities of the query are not present in the useful case.

Some special desirable properties of acceptance are following:
P1: A case might be acceptable for a query even if there exist some information entities that are not comparable.
P2: A case might be unacceptable for a query if there exist an unacceptable information entity (a fix budget may forbid expensive offers).
P3: The same information entity may have different importance for different cases (information entity <sex, male> has different importance in pregnancy testing and in testing for influenza).
P4: The same information entity may have different importance for different queries according to the user's intentions (material have different priorities in design queries).
P5: Information entities may not be independent of each other.

In order to provide better understanding of acceptance we will define the *preference relation* ($\geq_q$), over the set of all potential cases, in the following way:

$c' \geq_q c''$ *if case c' is preferable to case c'' in regard to the query q.*

The definitions of acceptance functions will be given for cases and queries represented as feature vectors (and not as a set of information entities) because it is a more convenient form of representation. Both the queries and the cases are considered as feature vectors ($a_1, \ldots, a_n$), where a_i specifies the value for the i-th attribute A_i. Formally, there is no difference between case vectors $c = (c_1, \ldots, c_n)$ and query vectors $q = (q_1, \ldots, q_n)$.

Definition 1. (Global Acceptance Function): *Let* $U := dom(A_1) \times \ldots \times dom(A_n)$ *denote the set of all queries and cases. The acceptance of a case for a query is expressed by a global acceptance function*

$$acc : U \times U \rightarrow R \qquad (1)$$

such that a higher value acc(q,c) denotes a higher acceptance of the case c for the query q. The preference relation $\geq_q \subseteq U \times U$ *induced by a query* $q \in U$ *is defined by*

$$c' \geq_q c'' \text{ iff } acc(q,c') \geq acc(q,c''). \qquad (2)$$

Definition 2. (Local Acceptance Functions for Attributes): *A local acceptance function* σ_i *for the attribute* A_i *is defined over the domain* $dom(A_i)$*:*

$$\sigma_i : dom(A_i) \times dom(A_i) \rightarrow R \qquad (3)$$

such that higher value $\sigma_i(q_i,c_i)$ *denotes a higher acceptance of the value* c_i *(of a case c) for the value* q_i *(of a query q).*

Global acceptance function can be obtained from local acceptance functions by a related composition function as follows:

Definition 3. (Composite Acceptance Function): *A global acceptance function acc is called composite if it is composed by a composite function* $\Phi : R \times \ldots \times R \rightarrow R$ *from related local acceptance functions* σ_i*:*

$$acc((q_1,\ldots,q_n),(c_1,\ldots,c_n)) = \Phi(\sigma_1(q_1,c_1),\ldots,\sigma_n(q_n,c_n)). \qquad (4)$$

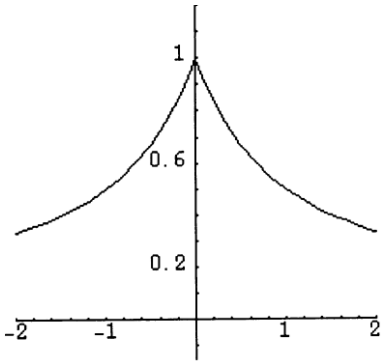
The natural demand is that composition functions must be monotonously increasing.

An example for the composition of local acceptance values is given by a weighted sum with only positive weights g_i (because of monotonously increasing composition function):

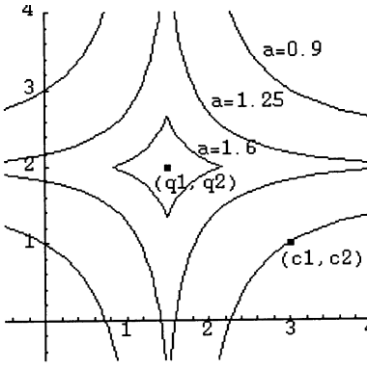
$$acc((q_1,\ldots,q_n),(c_1,\ldots,c_n)) = \sum g_i \cdot \sigma_i(q_i,c_i) \qquad (5)$$

Addition is widely used for the combination of local values. It has an intuitive interpretation concerning acceptance in the sense of "collecting arguments" in favor of something. Positive arguments have positive values, while negative arguments are expressed by negative values. The value 0 does not change the result, so unimportant or unknown attributes can be treated with value 0. Therefore, properties P1 and P2 are satisfied. However, a more general combination is necessary to satisfy properties P3, P4 and P5.

For illustration, we consider two simple local acceptance functions over the real numbers, which are composed by addition (weighted sum whose weights are all equal to 1). In Figure 2. we consider the local acceptance function $\sigma_i(q_i,c_i) = 1 / (1+|q_i - c_i|)$, while in Figure 3. we consider the function $\sigma_i(q_i,c_i) = 1-|q_i - c_i|$.

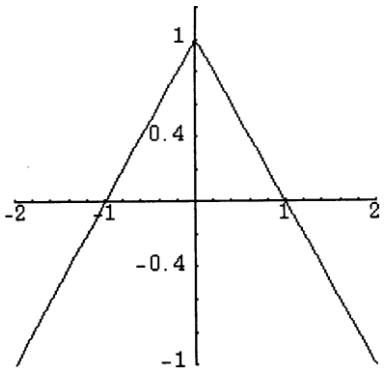


$f(x) = 1/(1+|x|)$

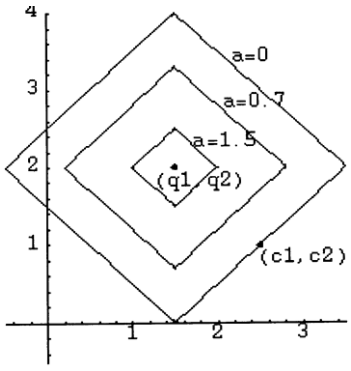


$1/(1+|q_1-c_1|) + 1/(1+|q_2-c_2|) = a$

Figure 2. The local acceptance function $\sigma_i(q_i,c_i) = 1 / (1+|q_i - c_i|)$



$f(x) = 1 - |x|$



$(1-|q_1-c_1|) + (1-|q_2-c_2|) = a$

Figure 3. The local acceptance function $\sigma_i(q_i,c_i) = 1-|q_i - c_i|$.

The left sides of these pictures present the graphic of the function $f(x) = \sigma_i(0,x)$, which shows the values of the local acceptance function for the fixed query ($q = 0$). The right parts of these pictures show some characteristics of the resulting global acceptance functions in the two-dimensional universe. All points $c = (c_1,c_2)$ on a characteristic have the same acceptance a for a fixed query $q = (q_1,q_2)$.

In Figure 2, we have closed characteristics for large values a, but the characteristics are opened for lower values a (these curves actually never meet each other). The reason is that a

high local acceptance value of a single attribute is sufficient to reach desired global acceptance value independently of other attributes (the local acceptance values are always positive, so the global acceptance is never reduced).

The situation changes if unlimited negative local acceptance values are allowed as in the Figure 3. On this picture all curves are closed. This situation is applied when we have to accumulate even "rejection" and not just "acceptance".

2.4. The general case

In this section we will generalize the previous concepts in order to satisfy the properties P1, … ,P5.

Queries have been defined as sets of information entities. A *weighted query* is the generalization of this concept.

Definition 4. (Weighted query): *The weighted query assigns an importance value to each information entity by a function:*

$$\alpha_q : E \to R, \qquad (6)$$

where $\alpha_q(e)$ denotes the importance of the information entity e for the query q.

High values indicate a high importance; negative values indicate the rejection of related cases. The value 0 is used as a neutral element ($\alpha_q(e) = 0$, means that information entity e is unimportant to the query q). Of course, values for $\alpha_q(e)$ can be taken from the set {0,1}, meaning that "e is (un)important for the q".

By using σ we can compute the acceptance of the information entity *e'* from the case for a single information entity *e* of a query. However, a query may contain several information entities *e* such that σ(e,e') is defined for the single information entity *e'*. The question is: how these values can be combined to a single value for *e'* which express the resulting acceptance value of *e'* for that query.

Definition 5. (Local Accumulation Function): *Let $E_e = \{e_1, \ldots, e_n\}$ denote the set of all information entities to which the information entity e is comparable concerning acceptance ($E_e = \{e' \mid \sigma(e',e)$ is defined$\}$). The local accumulation function π_e for e is a function:*

$$\pi_e : \underbrace{R \times \ldots \times R}_{n \text{ times}} \to R \qquad (7)$$

such that $\pi_e(a_1, \ldots, a_n)$ denotes the accumulated acceptance in e. The values a_i denote the contributions of the information entities $e_i \in E_e$ according to their occurrence in the query q and their local acceptance computed by $\sigma(e_i,e)$.
The contributions are computed by a function:

$$f : R \times R \to R \qquad (8)$$

such that $a_i = f(\alpha_q(e_i),\sigma(e_i,e))$.

We consider the retrieval of the cases as a process of reminding. Reminding may be of different strength; cases are in competition for retrieval according to the query. The cases receiving more reminders of more strength are the winners. The strength (importance, relevance) of reminding for an information entity $e \in c$ is given by a relevance function:

Definition 6. (Relevance Function): *The relevance between information entities and cases is described by a relevance function:*

$$\rho : E \times C \to R. \qquad (9)$$

The relevance $\rho(e,c)$ is considered as a measure for the relevance of information entity e for the retrieval of a case c. $\rho(e,c)$ is defined if and only if $e \in c$.

Negative values ρ(e,c) may be used in the meaning "do not retrieve the case *c* if one asks for the information entity *e*".

The acceptance of a case *c* for the query *q* is accumulated from the contributions of the information entities $e \in c$ according to their relevancies $\rho(e,c)$. The contributions p_e of the information entities are computed by their accumulation functions π_e as described in the definition 5. The accumulation in the cases is evaluated by *Global accumulation function.*

Definition 7. (Global Accumulation Function): *The global accumulation function π_c has the form:*

$$\pi_c : \underbrace{R \times \ldots \times R}_{k \text{ times}} \to R, \qquad (10)$$

for $c = \{e_1, \ldots, e_k\}$. The accumulated acceptance of the case c regarding its constituting information entities is then computed by $\pi_c(p_1, \ldots, p_k)$, where p_i is the contribution of the information entity $e_i \in c$. This contribution p_i depends on $\rho(e_i,c)$ and another real value x_i assigned to e_i (x_i is the accumulated local acceptance value computed by $\pi_{ei}(a_1,\ldots,a_n)$ from definition 5.). The contributions p_i are computed by a function:

$$g : R \times R \to R, \qquad (11)$$

such that $p_i = g(x_i,\rho(e_i,c))$.

The global acceptance function, which satisfies properties P1,…,P5, is calculated in the following way:

Definition 8. (Extended Acceptance Function): *Acceptance between weighted queries and cases is expressed by an extended acceptance function:*

$$acc : R^E \times P(E) \to R. \qquad (12)$$

The acceptance $acc(\alpha_q,c)$ of a case *c* for a weighted query α_q can now be accumulated by using the introduced functions:

$$acc(\alpha_q,c) = \pi_c\,(g\,(\pi_{e'1}\,(f\,(\alpha_q\,(e_{1,1}),\ \sigma(e_{1,1},e'_1)),\ldots,f\,(\alpha_q\,(e_{1,n1}),\ \sigma(e_{1,n1},e'_1)),\ \rho(e_1',c)),$$

$$\ldots$$

$$g\,(\pi_{e'k}\,(f\,(\alpha_q\,(e_{k,1}),\ \sigma(e_{k,1},e'_k)),\ldots,f\,(\alpha_q\,(e_{k,nk}),\ \sigma(e_{k,nk},e'_k)),\ \rho(e_k',c))\ )$$

where $c=\{e'_1,\ldots,e'_k\}$ and $E_{e'i} = \{e_{i,1},\ldots e_{i,ni}\}$ for $i = 1,\ldots,k$. (13)

If, for example, we consider *f* and *g* as products and π_c and π_e as sums then we get:

$$acc(\alpha_q, c) = \sum_{e' \in c} \rho(e', c) \sum_{e \in E_{e'}} \sigma(e, e') \cdot \alpha_q(e) \qquad (14)$$

Here, the properties P1,…,P4 are satisfied, but for satisfaction of the property P5, the appropriate selection of the functions f, g, π_c and π_e is needed.

3. Conclusion

Case-based reasoning is a relatively new and promising area of artificial intelligence. Case-based reasoning is a reasoning method that facilitates knowledge management in which knowledge is a case base acquired by a learning process. Case-based reasoning can be used for solving problems in many practical domains such as: mechanical engineering, medicine, business administration etc. Furthermore, for each domain, various task types can be implemented. Some of them are: classification, diagnosis, configuration, planning, decision support etc.

4. References

1. Aamodt, A., Plaza, E., (1994), "Case-Based Reasoning: Foundational Issues, Methodological Variations and System Approaches", *AI Commutations*, pp. 39-58.
2. Iglezakis, I. (2001), "The Conflict Graph for Maintaining Case-Based Reasoning Systems", *4th International Conference on Case-Based Reasoning (ICCBR 2001),* pp. 263-276, Vancouver, Canada, July/August 2001.
3. Lenz, M., Brtsch-Sporl, B., Burkhard, H., Wess, S. (1998), *Case-Based Reasoning Technology: From Foundation to Applications*, Springer, 1998.
4. Reinartz, T., Iglezakis, I., Roth-Bergofer, T., (2000), "On Quality Measures for Case Base Maintenance", *5th European Workshop (EWCBR 2000),* pp. 247-260, Trento, Italy, September 2000.