Similarity Measures for Object-Oriented Case Representations

Ralph Bergmann and Armin Stahl

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

Abstract. Object-oriented case representations require approaches for similarity assessment that allow to compare two differently structured objects, in particular, objects belonging to different object classes. Currently, such similarity measures are developed more or less in an ad-hoc fashion. It is mostly unclear, how the structure of an object-oriented case model, e.g., the class hierarchy, influences similarity assessment. Intuitively, it is obvious that the class hierarchy contains knowledge about the similarity of the objects. However, how this knowledge relates to the knowledge that could be represented in similarity measures is not obvious at all. This paper analyzes several situations in which class hierarchies are used in different ways for case modeling and proposes a systematic way of specifying similarity measures for comparing arbitrary objects from the hierarchy. The proposed similarity measures have a clear semantics and are computationally inexpensive to compute at run-time.

1. Introduction

Several recent CBR systems apply object-oriented techniques for representing cases (Manago, et al., 1994; Wess 95). Such representations are particularly suitable for complex domains in which cases with different structures occur. Cases are represented as collections of *objects*, each of which is described by a set of attribute-value pairs. The structure of an object is described by an *object class* that defines the set of attributes (also called slots) together with a *type* (set of possible values or sub-objects) for each attribute. Object classes are arranged in a *class hierarchy*, that is, usually a n-ary tree in which sub-classes inherit attributes as well as their definition from the parent class (predecessor). Moreover, we distinguish between *simple attributes*, which have a simple type like Integer or Symbol, and so-called *relational attributes*. Relational attributes hold complete objects of some (arbitrary) class from the class hierarchy. They represent a directed binary relation, e.g., a part-of relation, between the object that defines the relational attribute and the object to which it refers.

Relational attributes are used to represent complex case structures. The ability to relate an object to another object of an arbitrary class (or an arbitrary sub-class from a specified parent class) enables the representation of cases with different structures in an appropriate way.

Similarity measures for such object-oriented representations are often defined by the following general scheme (Wess, 1995): The goal is to determine the similarity between two objects, i.e., one object representing the case (or a part of it) and one object representing the query (or a part of it). We call this similarity *object similarity* (or *global similarity)*. The object similarity is determined recursively in a bottom up fashion, i.e., for each simple attribute, a *local similarity measure* determines the similarity between the two attribute values, and for each relational slot an object similarity measure recursively compares the two related sub-objects. Then the similarity values from the local similarity measures and the object similarity measures, respectively, are aggregated (e.g., by a weighted sum) to the object similarity between the objects being compared.

Unfortunately, such object similarity measures are currently developed more or less in an ad-hoc fashion. It is mostly unclear, how the structure of the object-oriented case model, e.g., the class hierarchy, influences similarity assessment. Intuitively, it is obvious that the class hierarchy contains knowledge about the similarity of the objects. Objects that are closer in the hierarchy should normally be more similar to each other than objects which are more distant in the hierarchy. However, how this knowledge relates to knowledge that could be represented in similarity measures that also consider the local similarity of the attributes is not obvious at all. Consequently, there is no clear view about how the similarity between two objects belonging to two different object classes should be determined. Therefore, many existing CBR systems and applications restrict object similarity to the comparing objects of the same object class only, not taking advantage of the high flexibility that object-oriented representations provide.

This paper provides a framework for object similarities that allow to compare objects of different classes while considering the knowledge contained in the class hierarchy itself. We will show that knowledge about similarity contained in class hierarchies is quite similar to the knowledge contained in taxonomies of symbols, which has been analyzed in a previous paper (Bergmann, 1998c). The next section presents four related examples of how class hierarchies can be used and what kind of object similarities are appropriate. Based on these examples a new framework for determining object similarities is developed.

2. Example Use of Class Hierarchies and Object Similarities

We now describe possible uses of class hierarchies in different related application examples in which personal computers are represented as part of the case. The class hierarchy (see Fig. 1) contains a class for representing a PC as well as different classes for representing components.

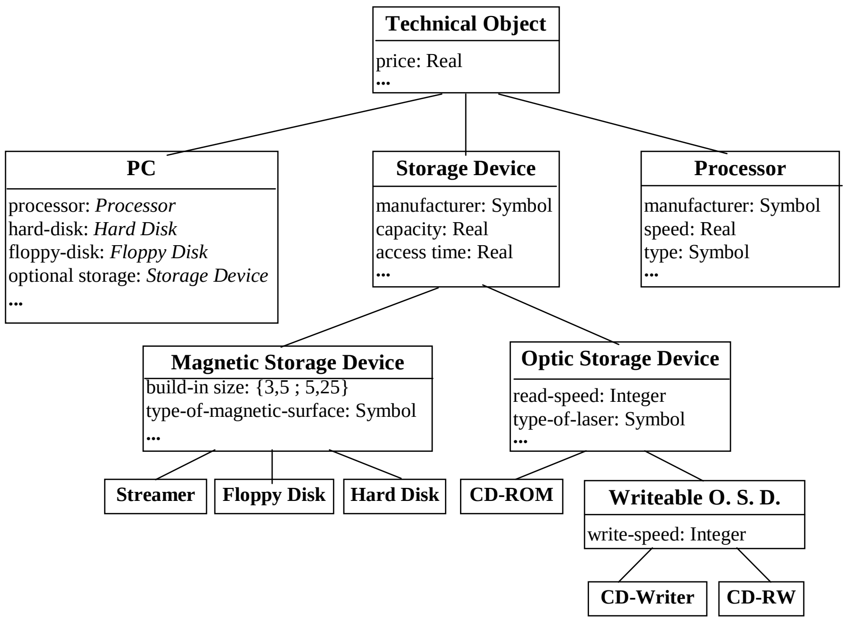

Fig. 1. A part class hierarchy in an example domain

The PC class contains attributes like „processor", „hard-disk", and „price" (inherited from "Technical Object") which are used to describe the properties of a PC in detail. Because a PC consists of a set of components (part-of-relation) which have properties themselves most attributes are relational (printed in italic font). Like for simple attributes, it is necessary to assign a relational attribute a class, for example to express that the relational attribute „hard-disk" can only have an instance of the object class *Hard Disk*. In the example the class *Hard Disk* has no sub-classes and consequently, every object that this attribute refers to has the same structure, i.e., the same set of attributes. In contrast, the relational slot „optional storage" does not have a unique class, because the object-class *Storage Device* has several direct and indirect sub-classes. Hence, the attribute can relate to objects of different structures, but they still have a common super-class (e.g., *Storage Device*) and therefore share at least some common attributes. In our example, one PC can have a second hard-disk as optional storage device, while another PC can have a CD-ROM described by a few different attributes (e.g., type of laser) than a hard-disk.

Consider again the „optional storage" attribute. Now we describe four examples in which this attribute is used differently. We first would like to focus on the knowledge contained in the class hierarchy and therefore don't take different values for simple attributes into account.

Example 1a. Consider a CBR system for the sales support of Personal Computers. A case represents an available PC from the stock. Consider a case c_1 with a second hard-disk as optional storage device and a case c_2 with a CD-Writer. If we assume that a customer enters a query to such a CBR system in which she/he specifies that she/he wants a CD-ROM, then c_2 is certainly closer than c_1, because a CD-ROM and a CD-Writer have obviously more in common than a hard-disk and a CD-ROM. In general, we could use a similarity measure that assesses similarity based on the distance between the class of the case object (of the respective relational attribute) and the class of the query object in the class hierarchy.

Example 1b. Imagine the customer states in the query a request for an optic storage device, i.e., in the query, the relational attribute refers to an instance of the class "Optic Storage Device". Then any of the devices in the *Optic Storage Device* sub-tree are perfectly suited. Hence, we expect the similarity value for the relational slot between this query and case c_2 (from Example 1a) to be equal to 1[1]. From this consideration we can conclude that whenever the class of the query object is located above the class of the case object, the similarity should be 1.

Example 2a. Consider now a trouble-shooting CBR system for PCs in which cases encode diagnostic situations and faults that have occurred previously. The domain expert describes a fault that can occur with any optic storage device. Therefore, the respective case contains an instance of the class *Optic Storage Device* in the relational attribute „optional storage device". Now, assuming a PC user has a problem and she/he states that there is a CD-RW device in the PC, then the similarity for the respective relational slot should be equal to 1 because the case matches exactly w.r.t. this attribute. From this consideration we can conclude that whenever the class of the case object is located above the class of the query object the similarity should be 1.

Example 2b. For the same trouble-shooting example, imagine now a different query in which the user does not exactly know what kind of storage device is installed in the PC, but she/he knows that it is a writeable optic storage device. Therefore, she/he enters an instance of the class *Writeable Optic Storage Device* as attribute value in the query. Again, the case about the *Optic Storage Device* mentioned in Example 2a matches exactly because, whatever storage device the user has, we known it is an *Optic Storage.* Hence, the situation described in the case applies. However, if we consider a different case that describes a problem with a CD-RW device, then this case does not match exactly. Since we don't know what writeable optic storage device the user has (it can be a CD-Writer but it can also be a CD-RW) we expect a similarity value less than 1 to represent this kind of uncertainty. Therefore, we cannot conclude that whenever the class of the query object is located above the class of the case object the similarity should be 1.

Although these four examples are based on the same class hierarchy, it is obvious that they have to be treated differently for the similarity computation. In the query and in the cases from example 1a, only instances of classes without subclasses are used. The examples 1b to 2b make use of abstract classes (classes with subclasses) of the

[1] We assume that similarity measures compute values between 0 and 1.

hierarchy, but in each example the semantics of the use these abstract classes is different, which must lead to different similarity measures.

3. Computing Object Similarities

3.1 Basic Notions

We briefly introduce a few notions (see Fig. 2) that will be further used in the paper. Let *K* be an inner node of the class hierarchy, then L_K denotes the set of all leaf nodes (classes) from the sub-tree starting at *K*. Further, $K_1 < K_2$ denotes that K_1 is a successor node (sub-class) of K_2. Moreover, $<K_3,K_4>$ stands for the most specific common object class of K_3 and K_4, i.e., $<K_3,K_4> \geq K_3$ and $<K_3,K_4> \geq K_4$ and it does not exist a node $K' < <K_3,K_4>$ such that $K' \geq K_3$ and $K' \geq K_4$ holds.

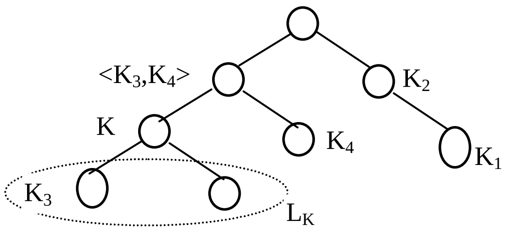

Fig. 2. Illustration of basic notions

3.2 Basic Considerations about Object Similarities

In general, the similarity computation between two objects can be divided into two steps: the computation of an *intra-class similarity* SIM_{intra} and the computation of an *inter-class similarity* SIM_{inter}.

Intra-Class Similarity. The common properties of the two objects can be used to the *intra-class similarity*. For this it is necessary to take the most specific common class of the two objects and to compute the similarity based on the attributes of this class only. By considering only the attributes of the most specific common class, the object similarity computation can be done as usual, since the objects being compared are from the same class. That is, local similarities or object similarities are computed for all attributes and the resulting values are aggregated to the intra-class similarity, e.g., by a weighted sum. Formally written:

$$SIM_{\text{intra}}(q,c) = \Phi(sim_{A_1}(q.A_1, c.A_1), \ldots, sim_{A_n}(q.A_n, c.A_n)),$$

where Φ is the aggregation function, $q.A_i$ and $c.A_i$ denote the value of the attribute A_i in the query and case object, respectively, and sim_{A_i} is the local or object similarity of the attribute A_i.

Inter-Class Similarity. The intra-class similarity alone would not be an adequate object similarity for the two objects. For example, in the domain shown in Fig. 1 two instances of *Hard Disk* and *CD-ROM* can have an intra-class similarity of 1, provided that they have the same values in the attributes which they inherit from their common superclass „Storage Device“. But it is obvious that there is a significant difference between a hard disk and a CD-ROM. Hence, the similarity should definitely be less than 1. It is important to note that the difference between two objects is not represented by their shared attributes but by the structure of the class hierarchy. Therefore, it is necessary to use this structure to compute an *inter-class similarity* for the two objects. This inter-class similarity represents the highest possible similarity of two objects, independent of their attribute values, but dependent on the positions of their object classes in the hierarchy. Formally, the inter-class similarity $SIM_{\text{inter}}(Q,C)$ is defined over the **classes of the objects** from the query and case being compared.

The final object similarity *sim(q,c)* between a query object *q* and a case object *c* can then be computed, by the product of the inter- and the intra-class similarity, i.e.,

$$sim(q,c) = SIM_{\text{intra}}(q,c) \cdot SIM_{\text{inter}}(class(q), class(c)),$$

where class(q) and class(c) denote the object class of the object q and c, respectively.

Next, we analyze how the inter-class similarity should be determined, which is quite similar to the similarity computation between two symbols arranged in a taxonomy (Bergmann, 1998c).

3.3 Different Semantics of Nodes

In a class hierarchy as well as in a taxonomy of symbols, we must distinguish between leaf nodes and inner nodes. In a taxonomy leaf nodes represent *concrete objects* of the real world. Inner nodes, however, describe *classes* of real world objects. An inner node *K* represents a class with certain properties that all of the concrete objects from the leaf nodes L_K have in common. Unlike classes that occur in the object-oriented paradigm, the classes that are represented by the inner nodes of a taxonomy are not described intentionally by a set of properties, but extensionally through the set of concrete objects L_K that belong to the class. Therefore, an inner node K stands for the set L_K of real world objects.

If we look at the class hierarchy shown in Fig. 1, we can notice a difference in the semantics of its nodes compared to the semantics of taxonomy nodes. While a leaf node of a taxonomy represents a concrete object of the real world, a leaf node of a class hierarchy is naturally a class and therefore represents a set of objects. As shown above, inner nodes of a taxonomy describe classes of real world objects, but if we look at the inner nodes of class-hierarchies, we can see that these nodes represent abstract classes. Because of this, such a node does not represent a set of real world objects, but a set of *abstract objects*. The instances which belong exclusively to the class „Storage Device“ or „Optic Storage Device“ for example are obviously not objects of the real world. However, abstract objects are sets of real world objects. An instance of „Optic Storage Device“, for example, can be used as abbreviation for the set of all instances of the classes „CD-ROM“, „CD-Writer“, and „CD-RW“ that have the same attribute-values in the common attributes as the respective „Optic Storage Device“ instance, e.g., the same manufacturer, the same capacity, the same access time, and the same speed.

There is also a difference in the use of the two different structures. A taxonomy tree consists of the symbols that are directly used as values for the attributes. On the other hand, the classes of a class hierarchy are not used as values for the relational slots themselves, but the instances of the classes. If we take this fact into account, we will see that now there is no difference in the semantics of the corresponding values, because the taxonomy symbols must be compared with the instances and not with the classes of the class hierarchy. An instance of a class without subclasses (a leaf node of the hierarchy) represents a concrete object of the real world, and as we have seen before an instance of an abstract class (inner node) can be treated as a set of real world objects. This semantics is equivalent to the semantics of the taxonomy nodes. Therefore, it is possible to apply the similarity measures used to compute similarities between taxonomy symbols for computing of the inter-class similarity between objects.

3.4 Inter-Class Similarity Between Concrete Objects

A class hierarchy encodes some knowledge about the inter-class similarity of the real-world objects, i.e., the instances of the leaf nodes. The deeper we descend in the class hierarchy, the more features the instances of the classes will have in common. We can therefore define the inter-class similarity as a measure of how many “features”[2] the compared objects have in common. The more “features” are shared, the higher is the inter-class similarity. For example, the inter-class similarity between a CD-Writer and a CD-RW is higher than the inter-class similarity between a CD-Writer and a CD-ROM.

This consideration leads to the following general constraint for defining the inter-class similarity for the instances of the leaf nodes of a class hierarchy:

$$SIM_{\text{inter}}(K, K_1) \leq SIM_{\text{inter}}(K, K_2) \quad \text{IF} \quad \langle \mathrm{K}, \mathrm{K}_1 \rangle > \langle K, K_2 \rangle$$

Because the class hierarchy only represents the constraint shown above but does not define numeric values for the similarity between two leaf node objects (that are used for the computation of an object similarity), it is necessary to add additional knowledge to the hierarchy. For this purpose it is possible to annotate every inner node K_i with a similarity value $S_i \in [0..1]$, such that the following condition holds: if $K_1 > K_2$ then $S_1 \leq S_2$. The semantics of the similarity value is as follows:

The value S_i represents a lower bound for the inter-class similarity of two arbitrary

[2] Here, feature does not necessarily mean attribute in the case representations.

instances of the classes from the set L_{Ki}, or formally written:

$$\forall X, Y \in L_{Ki}\, SIM_{\text{inter}}(X,Y) \geq S_i$$

With regard to this semantics one may define the inter-class similarity between two objects as follows:

$$SIM_{\text{inter}}(K_1, K_2) = \begin{cases} 1 & \text{if } K_1 = K_2 \\ S_{\langle K_1, K_2 \rangle} & \text{otherwise} \end{cases}$$

3.5 Semantics and Inter-Class Similarity of Abstract Objects

If we now recall again the examples that we have presented in section 2, it is obvious that the "optional storage" attribute must be treated differently in the different examples, although they all use the same class hierarchy. From that it becomes clear that some additional knowledge which we have not yet discussed plays a role during similarity assessment. However, this knowledge is not contained in the class hierarchy itself.

The knowledge that we are looking for is the knowledge about the semantics of the instances of inner nodes, i.e., the semantics of the abstract objects, which can be treated as sets of real-world-objects (see section 3.3). In our example, the question is: what does it mean when the case or query contains the statement:

"optional storage: <an Optic Storage Device instance >"?

In fact, there are different interpretations of this statement that will be further discussed.

Any value in the query. The user specifies an abstract object *k* in the query. This means that she/he is looking for a case that contains a real-world-object that matches with the features of the specified abstract object, i.e., a case that contains an object that belongs to a class of L_K. This was the situation in example 1b.

Any value in case. The case contains an abstract object *k*, which describes a situation in which the case is valid for all objects that are a specialization of *k*. This leads to a kind of generalized case. This occurred in example 2a.

Uncertainty. This situation differs significantly from the previous ones. Here, the use of an abstract object *k* means that we do not know the concrete object for this relational slot, but we know that it is a specialization of *k*. This situation occurred in example 2b.

Depending on the appropriate semantics we can now define an inter-class similarity measure $SIM_{inter}(Q,C)$ which computes a value for the inter-class similarity between two objects Q and C where each can be either a leaf node (concrete object), an inner node (abstract object) with the "any value" interpretation or an inner node (abstract object) with the “uncertainty“ interpretation. This leads to nine possible combinations shown in Table 1. Seven of the nine combinations in the table are marked with a roman number that is further used to reference the formulas for computing the similarity. These are the ones that occur most likely. However, the following considerations can easily be extended also to the two missing combinations.

Table 1. Combinations of different semantics for objects in query and case

Query \ Case	**Leaf Node concrete object**	**Any Value abstract object**	**Uncertainty abstract object**
Leaf Node concrete object	I	II	V
Any Value abstract object	III	IV	
Uncertainty abstract object	VI		VII

I: Only the similarity between concrete objects must be computed as described in section 3.4.

II: The query contains a concrete object and the case contains an abstract object (inner node) representing a set of concrete objects each of which is a correct object for the case. Therefore, the use of this abstract object in the attribute is a shortcut for the use of several cases, one for each concrete object belonging to the abstract object. Since we are looking for the most similar case in the case base, the object similarity, and therefore also the inter-class similarity, between the query and our case containing the abstract object is equal to the highest similarity between the query and one of the concrete objects. Hence:

$$SIM_{\text{inter}}(Q,C) = \max\{SIM_{\text{inter}}(Q,C') \mid C' \in L_C\} = \begin{cases} 1 & \text{if } \; Q < C \\ S_{\langle Q,C \rangle} & \text{otherwise} \end{cases}$$

holds. This definition ensures that the similarity is the same as the similarity that arises when each of the concrete objects would have been stored in the case base. This measure is appropriate for example 2a.

III: Here, the specification of this abstract object can be viewed as a shortcut for posing several queries to the system, one for each of the concrete objects from the set that the abstract object represents. Since we are again interested in the most similar case, we should again select the most similar concrete object from the set. Hence:

$$SIM_{\text{inter}}(Q,C) = \max\{SIM_{\text{inter}}(Q',C) \mid Q' \in L_Q\} = \begin{cases} 1 & \text{if } \; C < Q \\ S_{\langle Q,C \rangle} & \text{otherwise} \end{cases}$$

holds. This measure is appropriate for example 1b.

IV: This is a combination of II and III. We are looking for the highest possible similarity between two concrete objects from the two sets represented by the abstract objects since both, the query and the case, represent alternatives that are suited equally well. Hence,

$$SIM_{\text{inter}}(Q,C) = \max\{SIM_{\text{inter}}(Q',C') \mid Q' \in L_Q, C' \in L_C\} = \begin{cases} 1 & \text{if } \; C < Q \text{ or } Q < C \\ S_{\langle Q,C \rangle} & \text{otherwise} \end{cases}$$

holds.

V: The case contains an abstract object which represents a set of concrete objects from which only one value is actually correct for the case, but we don't know which one. Therefore, our similarity measure has to reflect this lack of information. There are three possible approaches: we can assess the similarity in a pessimistic or optimistic fashion, or we can follow an averaging approach. We only demonstrate the pessimistic approach; see (Bergmann, 1998c) for more details on the other approaches.

Pessimistic approach: We assess the similarity between the known object (in the query) and the partially unknown object (in the case) by computing the lower bound for the similarity as follows: $SIM_{\text{inter}}(Q,C) = \min\{SIM_{\text{inter}}(Q,C') \mid C' \in L_C\} = S_{\langle Q,C \rangle}$.

VI: The uncertain information is contained in the query; the information in the case is certain. This case is quite similar to the previous case V. For the pessimistic approach holds: $SIM_{\text{inter}}(Q,C) = \min\{SIM_{\text{inter}}(Q',C) \mid Q' \in L_Q\} = S_{\langle Q,C \rangle}$

VII: The uncertain information is contained in the query and in the case. The similarity is computed as follows for the pessimistic approach:

$$SIM_{\text{inter}}(Q,C) = \min\{SIM_{\text{inter}}(Q',C') \mid C' \in L_c, Q' \in L_Q\} = S_{\langle Q,C \rangle}$$

In all of these cases, the inter-class similarity can be computed very easily by determining the position of the class of the query object and the case object in the class hierarchy and by looking up the similarity value associated with the most specific common super class.

4. Status of Implementation and Related Work

The described object similarities are realized as part of the recent version of the commercial CBR tool CBR-Works from TECINNO GmbH. The approach was applied for the HOMER application (**HO**tline **M**it **ER**fahrung; hotline support for troubleshooting Workstations and CAD software at Daimler Benz, see Göker et al., 1998 for details) developed as part of the INRECA-II project. The approach to the construction of similarity measures will also enter the INRECA-II methodology for building CBR applications (Bergmann & Althoff, 1998a; Bergmann, et al. 1998b).

Currently, there is no other work that proposes similarity measures for object-oriented case representations that make use of the class hierarchy, relational attributes, and flexible local similarity measures for simple attributes. However, similarity measures for different kinds of structured representations are discussed throughout the CBR and instance-based learning literature during recent years.

To some extend, object-oriented representations can be compared to representations in first-order logic where a case is a conjunction of atomic formulas. Each atomic formula $P(id,a_1,...,a_n)$ stands for a single object. The argument *id* of the formula denotes an object identification and the remaining arguments $a_1,...,a_n$ represent the attributes. Relational attributes can be represented by using the object identifications as attribute values. Emde & Wettschereck (1997) present a relational instance-based learning approach that is based on the computation of similarity between cases represented in first-order logic. However, there approach does not consider class hierarchies or flexible local similarity measures for simple attributes. However, they can handle cases in which one relational attribute points to several objects. In object-oriented representations like CASUEL, this representational feature is called a *multi-value slot*. Similarity measures for such attributes have been discussed in an earlier paper (Bergmann & Eisenecker, 1995), but have shown to be computationally too expensive in most applications. Therefore, we suggest to avoid these kinds of representations, if possible.

Plaza (1995) proposes feature terms for representing structured cases and proposes a similarity computation based on antiunification. This approach allows to consider some kinds of background knowledge during similarity computation but cannot deal with class hierarchies and local similarity measures for simple attributes.

The framework by Osborne & Bridge (1996) is based on lattices of values (somehow similar to a class hierarchy) and applies a logic-oriented approach for defining similarities. However, their work does not cover object-oriented representations, e.g., relational attributes and inheritance of attributes are not considered.

There is also some resemblance to graph representations as, for example, discussed by Bunke & Messmer (1994) and Sanders at al. (1997). However, object-oriented representations as discussed here are comparable to labeled graphs in which the matching problem, which is NP complete in the general case, does not occur.

Acknowledgment. Funding for this work has been provided by the Commission of the European Union (INRECA-II: Information and Knowledge Reengineering for Reasoning from Cases; Esprit contract no. 22196) to which the authors are greatly indebted. The partners of INRECA-II are: AcknoSoft (prime contractor, France), Daimler Benz (Germany), TECINNO (Germany), Irish Multimedia Systems (Ireland), and the University of Kaiserslautern (Germany). The authors would like to thank all members of the INRECA-II team for valuable comments on object similarities. Thanks also to the anonymous reviewers who significantly helped to improve the paper.

References

Bergmann, R. & Eisenecker, U. (1995). Fallbasiertes Schließen zur Unterstützung der Wiederverwendung objektorientierter Software: Eine Fallstudie. Proceedings der 3. Deutschen Expertensystemtagung, XPS-95, pp. 152-169, Infix-Verlag.

Bergmann, R. & Althoff, K.-D. (1998a). Methodology for building CBR applications. Chapter 12 of Lenz, Bartsch-Spörl, Burkhard, Wess (Eds*). Case-Based Reasoning Technology*. LNAI 1400, Springer.

Bergmann, R., Breen, S., Fayol, E., Göker, M., Manago, M., Schumacher, J., Schmitt, S., Stahl, A., Wess, S. & Wilke, W. (1998b). Collecting experience on the systematic development of CBR applications using the INRECA-II Methodology *This volume.*

Bergmann, R. (1998c). On the use of taxonomies for representing case features and local similarity measures. In Gierl & Lenz (Eds.) *6th German Workshop on CB*R.

Bunke, H. & Messmer, B. (1994). Similarity measures for structured representations. In Wess, Althoff & Richter (Eds.) *Topics in Case-Based Reasoning,* pp. 106-118, LNAI 837, Springer.

Göker, M., Roth-Berghofer, T. Bergmann, R., Pantleon, T., Traphöner, R., Wess, S., & Wilke, W. (1998). The development of HOMER: A case-based CAD/CAM help-desk support tool. *This volume.*

Manago, M. Bergmann, R. et al. (1994). CASUEL: A common case representation language. *Deliverable D1 of the INRECA Esprit Project*.

Osborne H. & Bridge, D. (1996). A case base similarity framework. In Smith & Faltings (Eds.) *Advances in Case-Based Reasoning, pp. 309-325, LNAI 1168,*. Springer.

Plaza, E. (1995). Cases as terms: A feature term approach to the structured representation of cases. In Veloso & Aamodt (Eds.) *Case-Based Reasoning Research and Development,* pp. 265-276, LNAI 1010, Springer.

Sanders, K., Kettler, B., & Hendler, J. (1997). The case for graph-structured representations. In Leake & Plaza (Eds.) *Case-Based Reasoning Research and Development,* LNAI 1266, Springer.

Wess, S. (1995). *Fallbasiertes Problemlösen in wissensbasierten Systemen zur Entscheidungs-unterstützung und Diagnostik: Grundlagen, Systeme und Anwendungen*. Dissertation, Universität Kaiserslautern, Infix-Verlag.