ColibriCook: A CBR System for Ontology-Based Recipe Retrieval and Adaptation[⋆]

Juan DeMiguel, Laura Plaza, and Belén Díaz-Agudo

Department of Software Engineering and Artificial Intelligence,
Universidad Complutense de Madrid, Spain.
{ltkerensky@gmail.com, lplazamorales@gmail.com, belend@sip.ucm.es}

Abstract. In this paper we present ColibriCook: a CBR system for ontology-based cooking recipe retrieval and adaptation. The system's purpose is to participate in the 1st Computer Cooking Contest, organized by the European Conference on Case-Based Reasoning (ECCBR'08), at the University of Trier, Germany. CBR is based on a best-adaptation likeness paradigm between ingredient sets, with a domain ontology providing one-on-one fuzzy ingredient similarity. A number of other machine learning techniques are used to calculate, propagate, compare and adapt other recipe properties.

1 Introduction

In this paper we present ColibriCook[1], a CBR system for the cooking domain. This system was developed as the final evaluation assignment for a graduate course in Machine Learning at the Universidad Complutense de Madrid. The aim is to create a system capable of competing in the 1st Computer Cooking Contest, held at the ECCBR'08. The rules for the contest are quite simple: given a query asking for a recipe with a set of requirements, the system should retrieve and adapt an already existing recipe to meet those requirements. For further information, refer to the Cooking Contest website[2].

The implementation is based on the jCOLIBRI2 [8] case-based reasoning framework[3]. Basically, ColibriCook is a knowledge-intensive CBR system that deals with the four classical steps in the CBR cycle, and also supports a number of other machine learning techniques.

The paper is organized as follows: first, we take a look at the ColibriCook general system architecture (section 2). We then perform a running example to show how the system actually works (section 3). Next, we evaluate the final result (section 4). Finally, we draw some conclusions and describe some lines of future work (section 5).

[⋆] Supported by the MID-CBR project (TIN2006-15140-C03-02)
[1] http://gaia.fdi.ucm.es/projects/cookingContest/cookingContest.html#jcolibricook
[2] http://www.wi2.uni-trier.de/eccbr08/index.php?task=ccc&act=2
[3] http://gaia.fdi.ucm.es/projects/jcolibri/jcolibri2/index.html

2 System Architecture

In this section we describe the ColibriCook system architecture. We have used the jCOLIBRI2 java-based CBR framework [8] as the basis for the architecture. More specifically our system is based on the jCOLIBRI ontology extension[6] where we have include different variations. ColibriCook includes mechanisms to retrieve, reuse, revise and retain cases and it is designed to be easily extended with new components.

2.1 High Level Design

The Computer Cooking Contest rules and requirements are the direct inspiration for the high level decisions that guided us in the system's design phase. They hinted to a knowledge-rich approach to CBR. Based on this fact, the main features of the system design might be described as follows:

- **Ontology**: most of the system domain specific knowledge resides in a custom built ontology. Complete recipes are not stored in the ontology.
- **Case Base**: the cases (recipes) as a whole are stored in a custom built XML database. The original recipes are processed to expose relevant data (ingredients and types).
- **Ingredient Similarity**: the degree to which an ingredient might be substituted by another is computed as a real, continuous value.
- **Best adaptation likeness paradigm**: our similarity function compares the query with the best possible adaptation of the inspected case.
- **Other ML techniques**: The cuisine type of a recipe is not available in the database provided by the contest organization. However, this information is necessary to properly solve some queries. For this reason, we have used the Weka Data Mining Software[4] to construct a supervised classifier for predicting this type (see section 4.1).

2.2 Ontology Design and Use

The system uses a domain ontology built with Protégé[5]. We reviewed some already built domain ontologies, but we found them impractical for our purposes. The ontology is used to store the following concepts:

- **Ingredients**: all the knowledge about the ingredients is stored here. Ingredients might be classes, and thus denote "generic" ingredients, or instances, which are specific ingredients.
- **Formal Type**: the general type of a meal. Again, it is a hierarchy of classes and instances. For example, "cake" is a kind of "dessert".
- **Cuisine Type**: the ethnic origin or style of the recipe. Currently has a flat structure, but it can be easily extended to contain concepts such as that an Italian recipe is a Mediterranean recipe.

[4] The University of Waikato Weka Project: http://www.cs.waikato.ac.nz/ml/weka/
[5] Protégé: www.protege.stanford.edu

– **Dietary Type**: dietary restrictions that a recipe might satisfy. For example, a "vegetarian" recipe does not include meat.
– **Ingredient Type**: properties used to derive dietary types. For example, "vegetarian" recipes cannot contain ingredients of "animal" type, but might contain "milk" or "egg" ingredient types.

Each ingredient is described by the following properties:

– **Identifier**: obviously, a unique name. This name is intended, mostly, for internal use. An independent component is in charge of "translating" textual ingredients to ontology ones.
– **Father-Similarity**: this property tells us how well the parent class of this ingredient might be substituted by it. For example, "mutton" has a 1.0 father similarity with its parent class, "sheep", which means that mutton is basically equivalent to sheep meat. On the other hand, "sheep" has a 0.6 similarity with its parent "red meat" class, which means that, while certainly being a type of red meat, "sheep" it's not the most common substitution for "red meat".
– **Is-Ingredient-Type**: this property tells us to which ingredient class, if any, this ingredient belongs. For example, "mutton" is "animal meat".
– **Is-Made-Of**: for composite ingredients. For example, mayonnaise is made of "vegetable oil" and "eggs". This knowledge helps us to propagate ingredient properties.
– **Availability**: some ingredients can derail a similarity comparison between recipes. For example, whether a recipe contains water or not is usually irrelevant. To help the similarity function, ingredients are classified into availability groups.
– **Is-Substitutable**: an ingredient might have an arbitrary substitutability relationship with any other ingredient. Substitutability relationships are complex and do not necessarily follow any specific parent-child taxonomies.

One of the key aspects of the ontology usage is how substitutability values are computed. The substitutability value for a given ingredient pair is computed by following all possible paths between both ingredients. At each path step, substitutability values are combined in order to obtain a total path value. Finally, the shortest (i.e. whichever yields a higher substitutability value) path is chosen.

Currently, the ontology holds 570 ingredients, both generic and specific, and the recipe data base has 298 instances.

2.3 Similarity Computation

We have followed what we call the best-adaptation likeness paradigm. The idea is very simple: the similarity of a problem solution with a problem description is equal to the similarity between the best-possible adaptation of the problem solution to meet the query's requirements. The retrieve and reuse steps of the CBR cycle are thus combined.

The domain determines that we are basically going to compare ingredient sets. But defining a specific semantic for 'set-likeness' is elusive. Many different measures are available, but the problem is assigning them an intelligible semantic so the user understands what the system is trying to accomplish. We identified two very different but useful semantics:

– **"At least" semantics**: this means that we want at least all the desired ingredients in the target recipe, but we do not care if additional ingredients are needed. This will be used, for example, when trying to come up with ideas for a meal.
– **"Just" semantics**: here we do not only want all the desired ingredients in the target recipe, but any ingredient not explicitly specified is supposed to be unavailable. This will come in handy, for example, when matching the contents of a fridge to a viable recipe.

The details of the similarity function as a whole are somewhat complex, but we can coarsely describe it as:

– **Desired**: first we look into the target recipe for desired ingredients. Ingredients in the target recipe might be substituted to better match the query. Obviously, more and more like ingredients imply a higher similarity.
– **Forbidden**: next we check for forbidden ingredients. Ingredients might be forbidden for a variety of reasons: dietary practices, explicit input by the user or unavailability. Ingredients might get substituted accordingly, if needed and possible. Higher ratios of forbidden ingredients imply a lower similarity.
– **Types**: we check for similarity between the query's miscellaneous types and the target recipe's. Dietary practices are enforced by the previous step. Formal type is checked using the ontology of formal types and its "is-a" relationship. Cuisine type is inferred using a machine learned classifier.

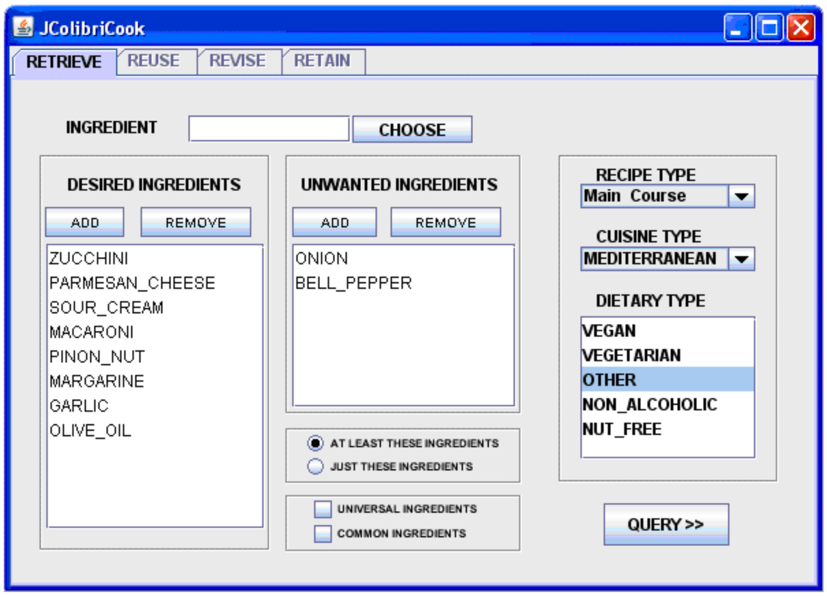
Each step generates a similarity value between 0 and 1. Finally, we combine these three values to arrive to a final similarity value. This value and the corresponding adaptation are then returned as output.

3 Running Example

In this section we run an example query through the system to help to understand how it works. The application is built around the four classical steps in a CBR cycle: retrieve, reuse, revise and retain. Each step has an associated action: query, select, done and store.

Retrieve. In this step the user enters a query. The application's interface is shown in figure 1. The query is made of:

– **Desired Ingredients**: the user might enter a list of desired ingredients for the requested recipe.



INGREDIENT

DESIRED INGREDIENTS
ZUCCHINI
PARMESAN_CHEESE
SOUR_CREAM
MACARONI
PINON_NUT
MARGARINE
GARLIC
OLIVE_OIL

UNWANTED INGREDIENTS
ONION
BELL_PEPPER

[● AT LEAST THESE INGREDIENTS] ○ JUST THESE INGREDIENTS

☐ UNIVERSAL INGREDIENTS
☐ COMMON INGREDIENTS

RECIPE TYPE: Main Course

CUISINE TYPE: MEDITERRANEAN

DIETARY TYPE
VEGAN
VEGETARIAN
[✓ OTHER]
NON_ALCOHOLIC
NUT_FREE

Fig. 1. JColibriCook Retrieve Step

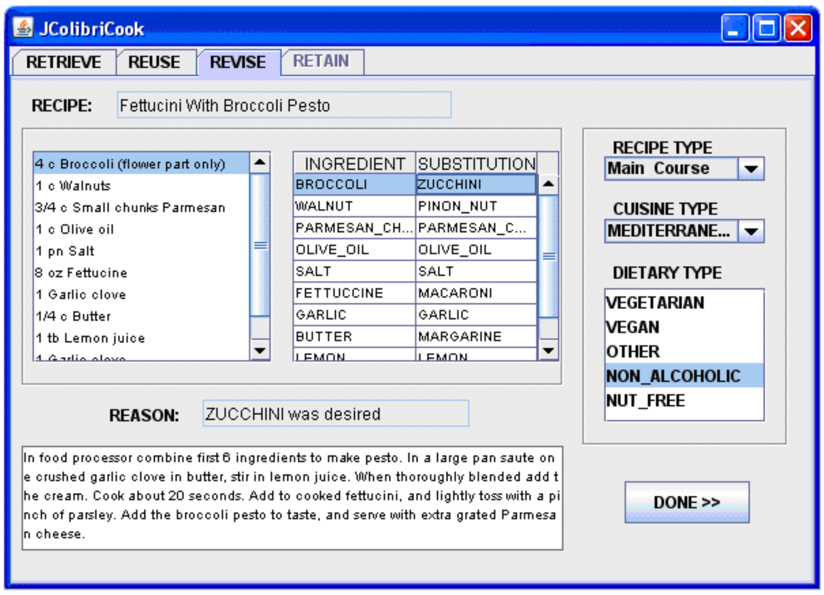
– **Forbidden Ingredients**: the user might enter a list of ingredients that she *does not* want in the requested recipe, too.
– **Comparison Semantics**: the preferred semantic for the ingredient set comparison. Under the "Just" semantics the two small checkboxes allow the user to specify that universal and / or common ingredients might be assumed as available.
– **Types**: each recipe belongs to a formal or general type ("main course", "dessert", "soup", etc.), a cuisine type ("Mediterranean", "Mexican", "Oriental", etc.) and any number of dietary types ("nut free", "non alcoholic", "vegetarian", etc.). The user might left them unspecified.

For the example we chose zucchini, parmesan cheese, sour cream, macaroni, pine nuts, margarine, garlic and olive oil as desired ingredients. We do not want onion or bell pepper. Finally, we want a Mediterranean first course, but we don't care about dietary practices.

When the user is satisfied with her selection, she can press the *query* button in order to advance to the next step.

Reuse. When the query is processed, the user is prompted to choose a recipe from the list of the five closest matches to her query. When the user has chosen a recipe, she can press the *select* button to proceed.

Revise. The next screen shows the selected recipe and the system's best adaptation effort. In the example we can see how the system has (sensibly) substituted



RECIPE: Fettucini With Broccoli Pesto

4 c Broccoli (flower part only)
1 c Walnuts
3/4 c Small chunks Parmesan
1 c Olive oil
1 pn Salt
8 oz Fettucine
1 Garlic clove
1/4 c Butter
1 tb Lemon juice
1 Garlic clove

INGREDIENT	SUBSTITUTION
BROCCOLI	ZUCCHINI
WALNUT	PINON_NUT
PARMESAN_CH...	PARMESAN_C...
OLIVE_OIL	OLIVE_OIL
SALT	SALT
FETTUCCINE	MACARONI
GARLIC	GARLIC
BUTTER	MARGARINE
LEMON	LEMON

RECIPE TYPE: Main Course

CUISINE TYPE: MEDITERRANE...

DIETARY TYPE: VEGETARIAN, VEGAN, OTHER, [✓ NON_ALCOHOLIC], NUT_FREE

REASON: ZUCCHINI was desired

In food processor combine first 6 ingredients to make pesto. In a large pan saute one crushed garlic clove in butter, stir in lemon juice. When thoroughly blended add the cream. Cook about 20 seconds. Add to cooked fettucini, and lightly toss with a pinch of parsley. Add the broccoli pesto to taste, and serve with extra grated Parmesan cheese.

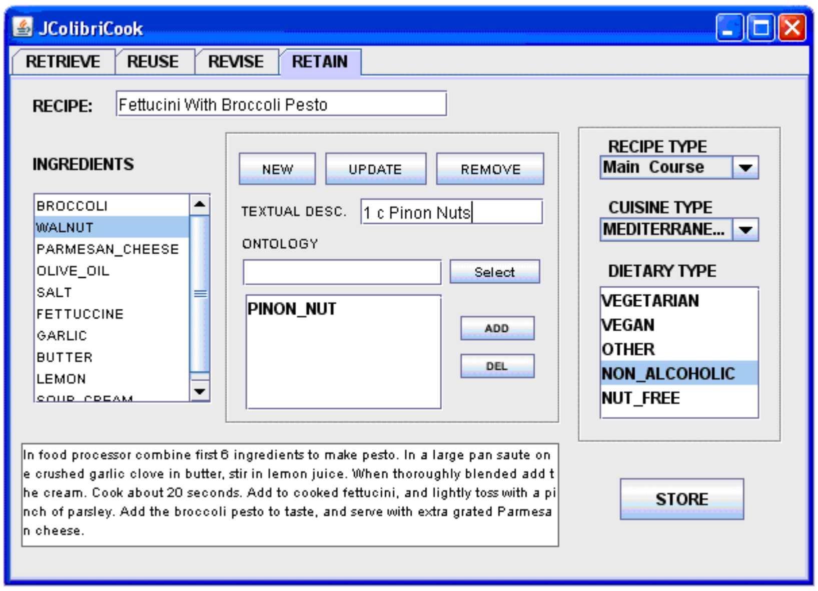
Fig. 2. JColibriCook Revise Step

broccoli by zucchini, walnuts by pine nuts and fettuccine by macaroni. Both formal and cuisine type match, and, additionally, we're told that the recipe is alcohol free. The user might now go back to a previous step to refine her query or, if she is satisfied with the result and wants to retain it, press the *done* button to advance to the next and final step.

Retain. In this step the user might retain the result for use in future queries. A new recipe must have, at least, a unique name. Additionally, the user is expected to modify the textual description of the recipe to match the new one. The application's interface is shown in 3. In the example, we have retyped the "1c Walnuts" as "1c Pine Nuts". The instructions should be update to reflect this fact, too. When the user is done, she might press the *store* button to update the case base with the new recipe.

4 Evaluation

The present section is organized into two subsections: in the first one we evaluate different classifiers at the task of learning the cuisine type of a recipe, while the second one is devoted to analyze the results achieved when running the contest query examples on ColibriCook.



RECIPE: Fettucini With Broccoli Pesto

INGREDIENTS

- BROCCOLI
- WALNUT
- PARMESAN_CHEESE
- OLIVE_OIL
- SALT
- FETTUCCINE
- GARLIC
- BUTTER
- LEMON
- SOUR_CREAM

TEXTUAL DESC. 1 c Pinon Nuts

ONTOLOGY

PINON_NUT

RECIPE TYPE: Main Course

CUISINE TYPE: MEDITERRANE...

DIETARY TYPE

- VEGETARIAN
- VEGAN
- OTHER
- NON_ALCOHOLIC
- NUT_FREE

In food processor combine first 6 ingredients to make pesto. In a large pan saute on
e crushed garlic clove in butter, stir in lemon juice. When thoroughly blended add t
he cream. Cook about 20 seconds. Add to cooked fettucini, and lightly toss with a pi
nch of parsley. Add the broccoli pesto to taste, and serve with extra grated Parmesa
n cheese.

Fig. 3. JColibriCook Retain Step

4.1 Cuisine Type Classifier

As already mentioned (section 2), in order to build a classifier for inferring the cuisine type of a given recipe, we have conducted experiments using several of the machine learning algorithms implemented in Weka.

For this purpose, we first constructed the training set, in which each instance represents a recipe and each attribute indicates the presence or the absence of an ingredient in the recipe. The class attribute is the cuisine type of the recipe and can take any value defined in the ontology under the Cuisine Type concept.

Next, we applied dimensionality reduction to the dataset, using the Weka filter AttributeSelection and the algorithm CfsSubsetEval, and reducing the original attribute set from 282 to 26 attributes. Due the full detailed experiment results being too extensive to be fully depicted in this paper, we present just the most significant ones. Finally, both experiments were carried out using cross validation on the training set with a fold value of 10 (see table 1). The Bayes Net method seems to give the best results.

4.2 Cooking Contest Challenges

To finish the evaluation section, we now run the exercises queries provided for the Computer Cooking Contest on the ColibriCook system. These results can be seen in table 2 and are further discussed below.

	Original Set	**Reduced Set**
DecisionTrees		
J48	63.7255%	71.5686%
LMT	71.5686%	78.4314%
Rules		
JRIP	61.7647%	63.7255%
PART	62.7455%	66.6667%
NNGE	63.7255%	70.5882%
Functions		
SimpleLogistic	70.5882%	78.4314%
PerceptronMultiLayer	76.4706%	75.4902%
SMO	76.4706%	79.4118%
Lazy Methods		
IB1	62.7455%	69.6078%
IBK k=2	59.8039%	75.4902%
LWL	53.9216%	61.7647%
Bayesian Methods		
NaiveBayet	63.7255%	80.3922%
BayetNet	77.4510%	82.3529%

Table 1. Weka experimentation results

Exercise 1: Cook a main dish with meat and cauliflower. The retrieved recipe is a main course as requested, and there is no need modify it. No restrictions are imposed on the cuisine or dietary types.

Exercise 2: I would like to have a nut-free cake. The retrieved recipe is a nut-free cake, and so it observes the restrictions imposed by the query. Again, it's not necessary to change or remove any ingredient. No restriction is imposed on the cuisine type.

Exercise 3: Prepare a Chinese dessert with fruit. The retrieved recipe is a "sweet", which is classified in the ontology as a specific kind of "dessert". It also obeys the restrictions on the desired ingredients and cuisine type. There is neither need to perform any substitution nor to remove any ingredient. No restrictions are imposed on the dietary type.

Exercise 4: Cook a main dish with turkey, pistachio, and pasta. In order to have a recipe that contains the desired ingredients, the "chicken" is replaced by "turkey". The remaining ingredients, "pistachio" and "pasta", were present in the original recipe, so no further changes are needed.

Exercise 5: I would like to cook eggplant soup. The "zucchini" in the original recipe has been replaced by "eggplant". The rest of the ingredients remain unaltered. It observes the restrictions imposed by the query on the formal type.

Exercise	Sim	Name	Formal Type	Cuisine Type	Dietary Type
1	0.975	Scalloped Turkey and Cauliflower	Main Course	Unknown	Nut Free Non alcoholic
2	1.0	Chocolate Cake	Cake	Unknown	Nut Free Non alcoholic Vegetarian
3	1.0	Lychee Sherbet	Sweet	Oriental	Nut Free Non alcoholic Vegetarian
4	0.93	Chinese Oriental and Pistachio Chicken	Main Course	Oriental	Nut Free
5	0.87	Vegetable Soup	Soup	Mediterranean	Nut Free
6	1.0	Picnic Rice Salad	Salad	Unknown	Nut Free Non alcoholic Vegan

Table 2. Contest exercises results

Exercise 6: I want to have a salad with tomato but I hate garlic and cucumber. The retrieved recipe is a salad, as desired, with "tomato" and without "garlic" and "cucumber", so there is no need to modify it. No restrictions are imposed on the cuisine and dietary types.

5 Conclusions and Future Work

The first conclusion we have reached as a result of our work on the system is *adaptation likeness*, meaning that the best-adaptation likeness paradigm seems to work well. Its main disadvantage is, obviously, the run-time efficiency. But even in our blatantly non-optimized prototype, complex queries are resolved in a matter of seconds.

Fuzzy Similarity, the way in which the ingredient substitutability property values are computed, has shown itself to be simple yet powerful. It allows us to maintain an intuitive taxonomy while, at the same time, it lets us infer arbitrarily complex similarity relationships.

Another important conclusion is that the use of machine learned classifiers can help CBR systems when some case properties are hard to manually infer. In our example, the cuisine type is often not available with the recipe description but, still, is needed to properly process queries and case base updates.

Regarding the scalability of the system, we have found that it's quite easy to enter new recipes. Entering new ingredients is more problematic, as extending an ingredient that already was an individual requires some further modifications. From a efficiency point of view, we have little doubt that system, being still a non optimized prototype, can handle the contest's required data set.

We think that the ColibriCook system does a fairly good job at what it was designed to do. Even then, it is obvious that it merely scratches the surface of the problem domain. Cooking is a complex task, and there's a lot of further work to do.

The similarity function, the core of the CBR system, has been shown to be fairly competent. But, unsurprisingly, it fails to fully capture the concept of recipe similarity. An area that allows for a lot of work is the comparison semantics, specifically on what means to compare two sets of ingredients. "At least" and "Just" semantics are only a coarse grained approach to the problem.

The most glaring limitation of system's adaptation method is that it can only replace an ingredient with another, or remove it altogether. It can't add an ingredient to an already complete recipe. In order to add ingredients to an existing recipe, substitutability relationships are not enough.

We could, for example, build an ingredient co-occurrence model: which ingredients always appear with which ingredients. For example, "meat" is often accompanied with potatoes, so an existing recipe using meat could be adapted to have them, too. An added bonus is that it's plausible to automatically deduce this model from the case base using inductive techniques like Formal Concept Analysis [2].

The system does not currently support the computer cooking contest's menu challenge: the composition of a three-course menu based on the available recipes. Ideally, we would want to support this type of query in the final implementation.

Finally, it would be an interesting improvement to include some kind of creativity in the system. This would allow to produce recipes significantly different to those retrieved from the case database and with a touch of originality.

References

1. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F.: The Description Logic Handbook: Theory, Implementation, Applications. Cambridge University Press (2003)
2. Díaz-Agudo, B., Gervás, P., González-Calero, P.: Adaptation guided retrieval based on formal concept analysis. In Ashley, K., Bridge, D., eds.: Case-Based Reasoning Research and Development, Procs. of ICCBR 2002. LNAI 2689, Springer Verlag (2003) 131–145
3. Hammond, K.J.: Case-based planning: A framework for planning from experience. Cognitive Science **14** (1990) 385–443
4. Kolodner, J.L.: Case–Based Reasoning. Morgan Kaufmann Publishers (1993)
5. Lenz, M., Bartsch-Spörl, B., Burkhard, H.D., Wess, S.: Case-based reasoning technology, from foundations to applications. In: Case-Based Reasoning Technology. Volume 1400 of Lecture Notes in Computer Science., Springer (1998)
6. Recio-García, J., Díaz-Agudo, B., P.González-Calero, Sánchez-Ruiz-Granados, A.: Ontology based CBR with jCOLIBRI. In: Applications and Innovations in Intelligent Systems XIV. Proceedings of AI-2006, Springer (2006) 149–162
7. Witten, I.H., Frank, E.: Data Mining: Practical maching learning tools and techniques. Second edition edn. Morgan Kaufmann (2005)
8. Díaz-Agudo, B., González-Calero, P., Recio-García, J., Sánchez, A.: Building cbr systems with jcolibri. Special Issue on Experimental Software and Toolkits of the Journal Science of Computer Programming **69**(1-3) (2007) 68–75 Impact Factor 0.734 Journal Citation Reports 2005, published by Thomson Scientific.