CBR for CBR: A Case-Based Template Recommender System for Building Case-Based Systems

Juan A. Recio-García[1], Derek Bridge[2],
Belén Díaz-Agudo[1], and Pedro A. González-Calero[1]

[1] Department of Software Engineering and Artificial Intelligence,
Universidad Complutense de Madrid, Spain
jareciog@fdi.ucm.es, {belend,pedro}@sip.ucm.es
[2] Department of of Computer Science, University College Cork, Ireland
d.bridge@cs.ucc.ie

Abstract. Our goal is to support system developers in rapid prototyping of Case-Based Reasoning (CBR) systems through component reuse. In this paper, we propose the idea of *templates* that can be readily adapted when building a CBR system. We define a case base of templates for case-based recommender systems. We devise a novel case-based template recommender, based on recommender systems research, but using a new idea that we call *Retrieval-by-Trying*. Our experiments with the system show that similarity based on semantic features is more effective than similarity based on behavioural features, which is in turn more effective than similarity based on structural features.

1 Introduction

It is an aspiration of the software industry that software development proceeds, at least in part, by a process of reuse of components. The anticipated benefits are improvements in programmer productivity and in software quality.

Compositional software reuse consists of processes such as: identifying reusable components; describing the components; retrieving reusable components; adapting retrieved components to specific needs; and integrating components into the software being developed [1]. These are difficult processes, made more difficult by the high volume of reusable components with which a software developer must ideally be acquainted.

Over the last twenty years, researchers have been looking at ways of providing software support to programmers engaged in software reuse. A lot of this research has drawn ideas from Case-Based Reasoning (CBR). The CBR cycle [2], retrieve-reuse-revise-retain, has obvious parallels with the processes involved in software reuse [3]. For example, an ambitious CBR system for software reuse is proposed in [4]. Its design combines text retrieval on component documentation with similarity-based retrieval on a case base of software components represented in LOOM. In [5], information about a repository of Java class definitions is extracted using Java's reflection facilities, and this information is used (along with

human annotations) to index the repository for similarity-based retrieval. In [6] retrieval is from a case base of Java 'examplets' (that is, snippets that demonstrate Java usage), using a mixture of text retrieval and spreading activation over a graph-based representation of the examplet token stream.

The most sustained research effort is that of Gomes [7]. In his ReBuilder system, cases represent designs and design patterns expressed as class diagrams in the Unified Modeling Language (UML). The work is unusual in providing some support for automated adaptation of the user's retrieved cases.

There has been surprisingly little work in which CBR applications have themselves been the target of case-based reuse: CBR for CBR! Perhaps the only example is the work reported in [8, 9], where CBR is used at the corporate level to support organization learning in software development projects, including CBR projects.

On the other hand, there are now several *frameworks* for building CBR systems, including myCBR[3], IUCBRF[4], and, the most developed, jCOLIBRI[5].

jCOLIBRI, for example, is a Java framework for building CBR systems. Building a CBR system with jCOLIBRI is a process of configuration: the system developer selects *tasks* that the CBR system must fulfil and, for every primitive task, assigns one of a set of competing *methods* to achieve the task, where a method is an actual Java implementation. Non-primitive tasks decompose into subtasks, which themselves may be primitive (achieved by methods) or non-primitive (requiring further decomposition). Ideally, every task and method that a system designer needs will already be defined; more realistically, s/he may need to implement some new methods and, more rarely, to define some new tasks.

In jCOLIBRI 1, the emphasis was on supporting the novice system developer. A developer could build a CBR system using a visual builder, i.e. s/he used a graphical tool to select tasks and methods in point-and-click fashion. While easy to use, this offered low flexibility. For example, it was not easy to implement new methods; and to keep the visual builder simple, non-primitive tasks could be decomposed only into *sequences* of subtasks.

jCOLIBRI 2 is a more 'open' system: a white-box framework that make it easier for programmers to add new methods to its repository. Non-primitive task decomposition now supports conditionals and iteration, as well as sequence. This raises the question of how best to support system developers who wish to use jCOLIBRI 2 to build CBR systems. For novices building simple systems, a visual builder might again be appropriate, although as yet a visual builder for jCOLIBRI 2 has not been written. But for more complex systems, students, inexperienced designers and even experienced designers may benefit from greater support. In this paper, we explain how we have extended jCOLIBRI to have a case base of past CBR systems and system *templates*, and how we explain how we can give case-based support to these users of the jCOLIBRI framework: truly, CBR for CBR.

[3] http://mycbr-project.net/
[4] http://www.cs.indiana.edu/ sbogaert/CBR/index.html
[5] http://gaia.fdi.ucm.es/grupo/projects/jcolibri/

The contributions of this paper are as follows. We define the idea of templates, as abstractions over past systems (Section 2). We show how a case base of systems can be defined, where each can be described in terms of the templates from which it was constructed and features drawn from an ontology of CBR systems, and we describe Retrieval-by-Trying, which we have implemented in a case-based recommender system for case-based recommender systems (Section 3). We define alternative similarity measures that can be used in Retrieval-by-Trying (Sections 4). We give an example of our system in operation (Section 5). And we use ablation experiments to evaluate the different similarity measures (Section 6).

2 Template-Based Design

2.1 Templates

A *template* is a predefined composition of tasks. Each template is an abstraction over one or more successful systems (in our case, CBR systems). A template may contain primitive tasks, for which there will exist one or more methods that implement that task. A template may also contain non-primitive tasks, for which there may be one or more decompositions, each defined by a further, lower-level template. As already mentioned, in jCOLIBRI 2 tasks can be composed in sequence, conditionals and iteration. Templates are defined by experts who have experience of building relevant systems and are capable of abstracting over the systems they have built.

A system developer can rapidly build a new prototype CBR system by retrieving and adapting relevant templates, a process we will refer to as *template-based design*. The designer will select among retrieved templates, among decompositions of non-primitive tasks, and among methods that implement primitive tasks. There may be occasions too when the designer must modify templates, e.g. inserting or deleting tasks or altering control flow; and there may be times when s/he must implement new methods. The degree to which these more radical adaptations are needed will depend on the extent to which the template library covers the design space. It also depends on the extent to which CBR is suitable for CBR, i.e. the extent to which problems recur, and similar problems have similar solutions.

There is a knowledge acquisition bottleneck here: experienced CBR designers must define the templates. Furthermore, templates are not concrete instances. They are abstractions over successful designs. Nevertheless, we believe that this is feasible for CBR systems. They have a strong, recurring process model, whose essence the expert can capture in a relatively small number of templates.

As proof-of-concept, we have built a library of templates for case-based recommender systems. The second author of this paper acted as expert, while the other authors acted as knowledge engineers. Within a few hours, we obtained a library of templates with, we believe, good coverage, which we have refined, but not substantially altered, over the last twelve months. We will describe some of these templates in the next section. More of them are described in [10].

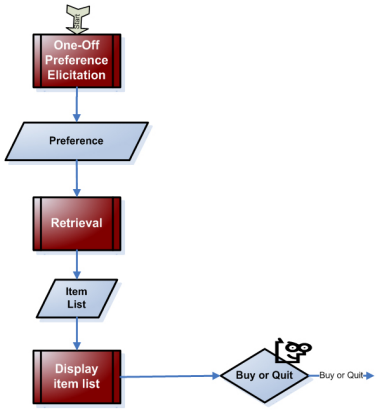

Fig. 1. Single Shot Systems

2.2 Templates for case-based recommender systems

We have defined twelve templates for case-based recommender systems, based in part on the conceptual framework described in the review paper by Bridge et al. [11]. We take the systems' interaction behaviour as the fundamental distinction from which we construct templates:

- *Single-Shot Systems* make a suggestion and finish. Figure 1 shows the template for this kind of system, where *One-Off Preference Elicitation* (for soliciting the user's 'query') and *Retrieval* (for finding items to recommend) are complex tasks that are solved by decomposition methods having other associated templates.
- After retrieving items, *Conversational Systems* (Figure 2) may invite or allow the user to refine his/her current preferences, typically based on the recommended items. *Iterated Preference Elicitation* might be done in navigation-by-proposing fashion [12] by allowing the user to select and critique a recommended item thereby producing a modified query, which requires that one or more retrieved items be displayed (Figure 2 left). Alternatively, it might be done in navigation-by-asking fashion [12] by asking the user a further question or questions thereby refining the query, in which case the retrieved items might be displayed every time (Figure 2 left) or might be displayed only when some criterion is satisfied (e.g. when the size of the set is 'small enough') (Figure 2 right). Note that both templates share the *One-Off Preference Elicitation* and *Retrieval* tasks with single-shot systems.

In the diagrams, non-primitive tasks are shown as red/dark grey rectangles. These tasks are associated with one or more further, lower-level templates. For the purposes of this paper, we will describe the decompositions of the *Retrieval* task, because it is common to all case-based recommender systems and because it will be familiar to everyone in a CBR audience. For information about the decompositions of the other non-primitive tasks in Figures 1 and 2, see [10].

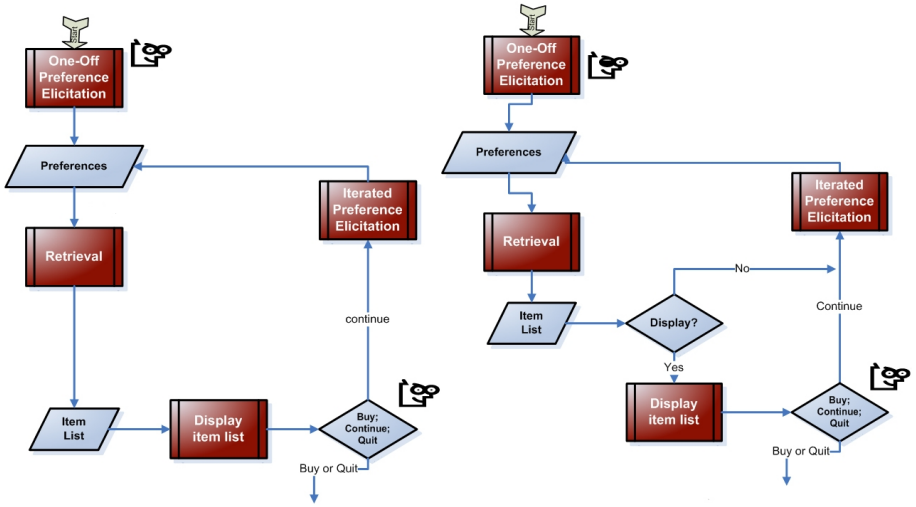

Fig. 2. Conversational Systems A and B

Retrieval is a complex task, with many alternative decompositions. Although Figure 3 shows only three decompositions, each of these three contains non-primitive tasks which themselves have more than one decomposition Commonly, for example, *Retrieval* comprises a scoring process followed by a selection process (Figure 3 top). For example, in similarity-based retrieval (k-NN), items are scored by their similarity to the user's preferences and then the k highest-scoring items are selected for display. Most forms of diversity-enhanced similarity-based retrieval follow this pattern too: items are scored by similarity, and then a diverse set is selected from the highest-scoring items [13–15].

But there are other types of recommender system in which *Retrieval* decomposes into more than two steps (Figure 3 bottom). For example, in some forms of navigation-by-proposing, first a set of items that satisfy the user's critique is obtained by filter-based retrieval, then these are scored for similarity to the user's selected item, and finally a subset is chosen for display to the user.

For completeness we mention also that some systems use filter-based retrieval (Figure 3 middle), where the user's preferences are treated as hard constraints. Although this is not commonly used in CBR in general, it can be found in some recommender systems. Despite its problems [16] it is still used in many commercial web-based systems. Also, systems that use navigation-by-asking often use filter-based retrieval: questions are selected using, e.g. information gain, and cases are retrieved in filter-based fashion [17].

A final observation about Figure 3 is that it shows optional tasks for updating a 'tabu list'. The tabu list can be used to prevent certain items from being recommended. A common use, for example, is to prevent the system from recommending an item that it has recommended previously.

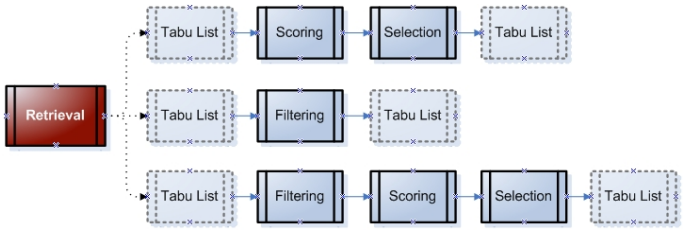

Fig. 3. Retrieval decomposition

2.3 Template recommendation

We envisage that a system developer will build a CBR system by adapting relevant templates. This implies a way of retrieving relevant templates.

We had originally thought that we would devise a description language for templates. With this language we would be able to describe each template in the library from the point-of-view of the functional and non-functional requirements that it might satisfy. The same description language could then be used by the CBR system developer to construct a query description of the functional and non-functional requirements of the system s/he is building. A retrieval engine would then find the best-matching templates from the template library.

We soon realized that this raised two formidable problems. There is the difficulty for system developers of expressing their requirements as a query. But more fundamentally, we realized that templates often do not lend themselves to a useful level of description. It is easier to say useful things about *systems*, rather than *templates*.

This insight led us to define the case-based template recommender system that we describe in the next section.

3 Case-Based Template Recommendation

3.1 Cases

In line with the insight of the previous section, each case in our case-based template recommender represents a successful CBR system (in our case, each is a case-based recommender system). But the templates that characterize the system are stored in the case as well. One can think of the system itself and its templates as the 'solution' part of the case.

The description part of the case is a set of feature-value pairs. The feature set includes the tasks of the system, the methods of the system, and semantic features from an ontology defined by the domain expert. We postpone a detailed explanation of the features to Section 4.

In some situations, the systems in the case base may be original systems, collected over the years. In other situations, this may not be possible. It was not the way in which we built our case base of case-based recommender systems, for example. The original systems, such as Entree [18] and ExpertClerk [12], are

not available. How then did we build a case base of systems? Very simply, we re-implemented versions of these systems and included these, along with their templates, in the case base. It is testimony to jCOLIBRI's repository of templates, tasks and methods that it did not take long to re-implement simplified versions of each of the main case-based recommender systems from the research and development literature. The case base we use in the rest of this paper contains fourteen such systems, although others could easily be implemented.

3.2 Retrieval-by-Trying

In Section 2.3, we noted two problems with a simple approach to template recommendation. We have overcome one of these problems (that the most useful descriptions apply to systems rather than to templates) by basing our case base on systems (see above). The other problem was that system developers would find it difficult to express their requirements as a query.

This problem is not unique to case-based template recommendation. In other recommender systems domains, it is not uncommon for users to find it difficult to articulate their preferences. But recommender systems research offers a solution.

In some of the most influential recommender systems, users make their preferences known through one or other of the many forms of navigation-by-proposing [18, 19, 12, 20]. In navigation-by-proposing, the user is shown a small set of products from which s/he selects the one that comes closest to meeting his/her requirements. The next set of product s/he is shown will be ones that are similar to the chosen product, taking into account any other feedback the user supplies.

This is the idea we adopt in our case-based template recommender. We show the user (the system developer) two systems. In the style of McGinty & Smyth's *comparison-based recommendation*, s/he may choose one, and the system will try to retrieve more systems that are like the chosen one. This overcomes the problem that system developers may not be able to articulate their requirements.

But it raises another problem. On any iteration, how will the system developer know which system to choose? If all we do is show the names of the systems or abstruse descriptions of the systems, s/he is unlikely to be able to make a decision. But, what the user is choosing between here are implemented systems. Therefore, we allow the user to *run* these system. We call this *Retrieval-by-Trying*: the user can actually *try* the recommended items (in this case, the recommended recommender systems) to inform his/her choice.

Retrieval-by-Trying is a natural approach for systems that are relatively simple and interactive, like case-based recommender systems. The approach may not extend to other kinds of CBR system that are more complicated (e.g. case-based planners) or non-interactive (e.g. case-based process controllers).

In the next three subsections, we explain the following: how our implementation of Retrieval-by-Trying selects the initial pair of systems that it shows to the user; how it selects the pair of systems that it shows to the user on subsequent iterations; and how the user's feedback on every iteration is handled.

3.3 Entry points

Initially, our system selects two systems to show to the user. One is as close to the 'median' of the case base as possible; the other is as different to the median as possible. A similar idea is used in ExpertClerk system [12], except that it selects three cases, one near the median and two dissimilar ones. Whether it is better to show three systems rather than two is something we can evaluate in future work. We decided to use two in our initial prototype because it simplifies the comparisons the user must make.

The first system that we retrieve is the most similar case to the median of the case base. The median of the case base is an artificial case where the value for every numerical attribute is obtained as the median of the values of that attribute in the case base, and the value for non-numerical attributes is set to the most frequent value for that attribute in the case base.

The second system initially retrieved is chosen to be as different to the median of the case base as possible. We compute for every case in the case base the number of 'sufficiently dissimilar' attributes between that case and the median case, and select the one with the largest number of dissimilar attributes. Two values of a numerical attribute are sufficiently dissimilar when their distance is larger than a predefined threshold. Two values of a non-numerical attribute are sufficiently dissimilar simply when they are different.

Although the process of selecting the initial two cases may be computationally expensive, it does not need to be repeated until new cases are added to the case base.

3.4 Diversity-enhanced retrieval for comparison-based recommendation

The user is shown a pair of systems, which s/he may try. In the style of preference-based feedback, s/he may then select one, and we will retrieve a new pair of systems that are similar to the one s/he chooses.

We need to ensure that the two systems that we retrieve are similar to the user's choice but are different from each other. If they are too similar to each other, the chances that at least one of the systems will satisfy the user are reduced. In recommender systems terminology, we want to enhance the *diversity* of the retrieved set [13]. There are several ways to achieve this. We use the well known Bounded Greedy Selection algorithm which enhances diversity while remaining reasonably efficient [13].

3.5 Preference-based feedback

The case that the user chooses gives us information about his/her preferences. But the rejected case also gives important feedback. Therefore we have implemented several of the preference feedback elicitation techniques described in [19].

The *More Like This* (MLT) strategy just uses the selected case as the new query for the following iteration. A more sophisticated version named *Partial More Like This* (pMLT) only copies attributes from the selected case to the

query if the rejected case does not contain the same value for that attribute. Another option is the *Less Like This* (LLT) strategy that takes into account the values of the rejected case that are different from the values of the selected one. In the subsequent iteration, cases with these 'rejected values' will be filtered before retrieving the cases. Finally, the *More and Less Like This* strategy combines both MLT and LLT behaviors. Given that it is difficult for users to express their requirements as a query (as we have explained above), it is an advantage that none of these approaches requires the user him/herself to deal explicitly with features and their values.

Our tool to retrieve templates can be configured to work with any of these strategies. We compare them empirically in Section 6.

4 Similarity in Case-Based Template Recommendation

The description part of each case is a set of feature-value pairs. The feature set includes the tasks of the system, the methods of the system, and semantic features from an ontology defined by the domain expert. Thus we can compute similarity based on *what* the systems do (by comparing system task structure); we can compute their similarity based on *how* they do what they do (by comparing their methods); and we can compute their similarity using semantic features defined by an expert to describe structural and behavioural characteristics of the systems. Or, of course, we can use combinations of these. We will describe each of them in more detail in the next three subsections.

4.1 Task structure similarity

We take a simple approach to task similarity for the moment, which relies to an extent on the fact that our 'top-level' templates (Figures 1 and 2) contain very similar non-primitive tasks. A more complex approach might be needed for case bases whose templates share less top-level structure. Let G be the set of non-primitive tasks $\{C_1, C_2, C_3, \ldots, C_n\}$, (such as *Retrieval*) and Q the set of possible decompositions of tasks in G into primitive tasks $Q = \{Q_1, Q_2, Q_3, \ldots, Q_n\}$. Each sequence Q_i is composed of a set of primitive tasks $S = \{S_1, S_2, S_3, \ldots, S_n\}$ (e.g. see *Retrieval* decompositions in Figure 3).

We define one attribute for each non-primitive task in G. The allowed values of these attributes are the specific sequences of primitive tasks Q used in this case. Comparing two cases by their structure means comparing the attributes of their templates using the equals function that returns 1 if they have the same value and 0 otherwise.

4.2 Methods similarity

Computing system similarity based on *how* the systems do what they do means comparing their methods. To be able to do that we include in the case representation structure different attributes, one for each primitive task. The allowed values for each one of these attributes are the set of methods that implement the

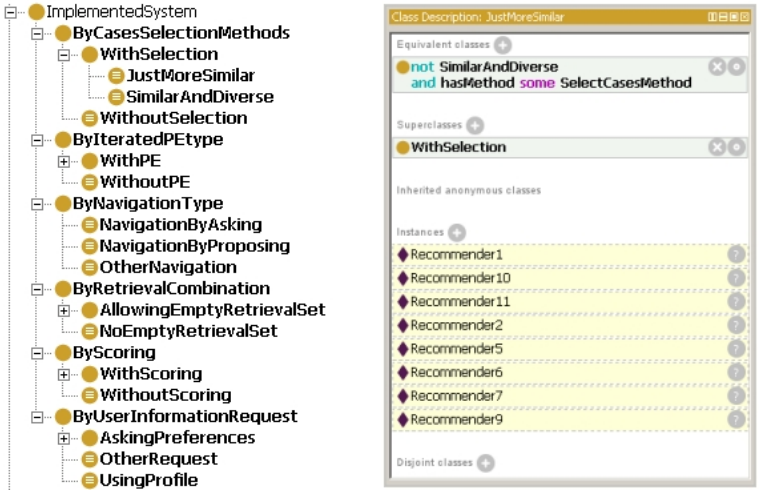

Fig. 4. Semantic features classification

primitive task. If a primitive task occurs more than once, then it is represented by different attributes.

To be able to compare methods we have created a concept in CBROnto for each method, and we have organized them into subconcepts of the *Method* concept. These method concepts are hierarchically organized according to their behaviour. Then we apply the ontological similarity measures implemented in jCOLIBRI2 to compare the methods. This family of ontological measures use the structure of the ontology to compute the similarity between instances. The CBROnto ontology and the similarity measures are described in [21].

4.3 Semantic feature similarity

As well as comparing systems by their tasks and methods, we let the expert define semantic features to describe structural and behavioural characteristics of the systems. In the recommenders domain, for example, we can classify systems depending on their preference elicitation approach: navigation-by-asking (asking questions to the user) or navigation-by-proposing (showing items to the user). We can also classify them according to their retrieval process: filtering, scoring or both. These features (navigation type and retrieval) define two different ways of classifying recommenders, and by extension the templates associated with those systems. There are other axes to classify systems, like the type of interaction with the user and the type of user information that it collects. The left-hand side of Figure 4 illustrates some of these semantic features.

Each case (system) in the case base is represented by an individual and is assigned properties describing the tasks and methods that define its behaviour. Using the classification capabilities of a Description Logic reasoner, each individual (system) is classified into the concepts that this individual fulfils, based

		Median Case	Recommender 6	Recommender 8
Generic Template		Conversational A	ConversationalA	ConversationalB
Task Structure	One-Off P.E.	Form_Filling	Form_Filling	Ask_Question
	Retrieval	Filtering_Scoring_Selection	Filtering_Scoring_Selection	Filtering
	Iterated P.E.	Create_Complex_Query	Create_Complex_Query	ExpertClerck_Iterated_P.E.
	Display	Display	Display	ExpertClerck_Display
Methods	FormFilling	FormFillingWithInitialValues	FormFillingWithoutInitialValues	
	SelectQuestion	InformationGain		InformationGain
	AskQuestion	ObtainQueryWithAttributeQuestion		ObtainQueryWithAttributeQuestion
	ReadProfile	ObtainQueryFromProfile		
	Scoring	ExpertClerkMedianScoring		
	SelectCases	SelectTopK		
	DisplayCases	DisplayCasesTableWithCritiques		
	Filtering	FilterBasedRetrievalMethod	FilterBasedRetrievalMethod	FilterBasedRetrievalMethod
	Scoring	NNScoringMethod	NNScoringMethod	NNScoringMethod
	Selection	SelectTopK	SelectTopK	BoundedGreedySelection
	RemoveTabu	null		
	Display	DisplayCasesTableWithCritiques	DisplayCasesTableWithCritiques	DisplayCasesTableWithCritiques
	UpdateTabu			
	FormFilling	FormFillingWithInitialValues		
	SelectQuestion	InformationGain		InformationGain
	AskQuestion	ObtainQueryWithAttributeQuestion		ObtainQueryWithAttributeQuestion
	CreateComplexQuery	MoreLikeThis	MoreLikeThis	MoreLikeThis
Global Features	CasesSelectionMethods		JustMoreSimilar	SimilarAndDiverse
	IteratedPEtype		ModifyingQueryWithUserSelection	ModifyingQueryWithUserSelection
	NavigationType		NbP	NbP and NbA
	RetrievalCombination		NotEmptyRetrievalSet	NotEmptyRetrievalSet
	Scoring		BasicScoring	BasicScoring
	UserInformationRequest		AskingUserForAllPreferences	AskingUserForSomePreferences

Fig. 5. Case values during retrieval

on the properties of the individual. The concepts into which each individual is classified define different relevant features of the recommender. For example, in the right-hand side of Figure 4 we show the definition of the feature "JustMoreSimilar". It is a defined concept described as follows:

$$\textit{JustMoreSimilar} \equiv \textbf{not}\ \textit{SimilarAndDiverse}\ \textbf{and}\ \textit{hasmethod}\ \textbf{some}\ \textit{SelectCasesMethod}$$

This definition applies to systems whose retrieval methods do not use any mechanism to enhance diversity but which do contain some method for selecting cases. The right-hand side of Figure 4 shows the systems in our case base that have been automatically classified as instances of this defined concept: eight of the fourteen recommenders are classified according to this feature.

The ontology allows us to compare two systems by classifying them into the hierarchy and comparing the concepts under which they are classsified. We use one of the ontological similarity metrics included in jCOLIBRI: the cosine function [21]. Similarity in the different semantic features can be computed separately as each feature represents a subtree in the hierarchy. Then the similarity results for each feature are aggregated.

5 Example

Let's illustrate the first step of our Retrieval-by-Trying template recommender. When the system is launched, it finds the most similar case to the median of the case base and the case that has most different attributes with respect to this median case. Figure 5 shows the content of the two retrieved cases and

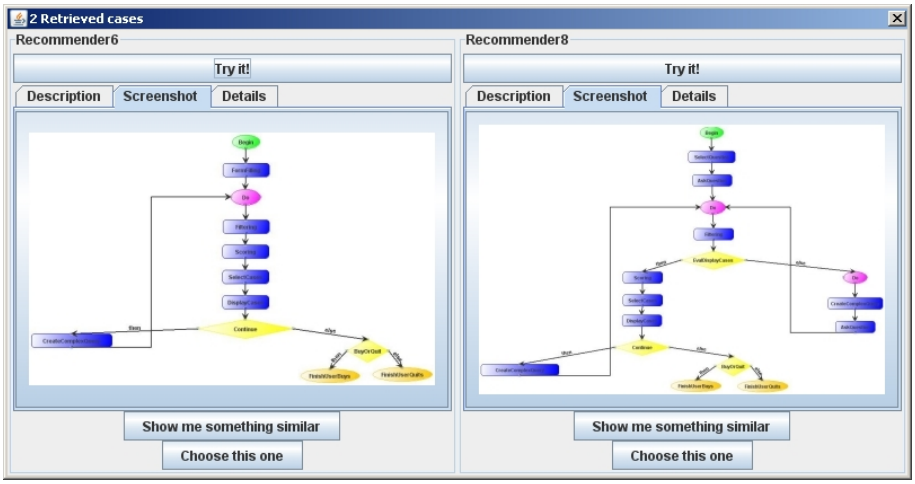

Fig. 6. Templates recommender screenshot

the median case computed by our method. The table contains the value of the attributes for each component of the case description: tasks, methods and semantic features. The first group of attributes describes the task decomposition of the templates associated with each case. Our templates have five non-primitive tasks: *One-Off Preference Elicitation*, *Retrieval*, *Display* and *Iterated Preference Elicitation*. Each one of these tasks can be decomposed into several sequences of primitive tasks as shown in Figure 3. This way, the values of this set of attributes reflect the decomposition into primitive tasks. The method attributes describe which methods were assigned to solve each task of the template to obtain the recommender. Finally, the semantic features refer to the roots of each classification hierarchy of our ontology (shown in Figure 4). The values of these features are the leaves of the hierarchy where each recommender is classified by the Description Logic reasoner.

The median case is a Conversational A recommender where each attribute has the most repeated value among all cases. This median case has no semantic features because it does not correspond to a real system in the case base. Recommender 6 is the closest case to the median and it is also a Conversational A system. Finally, Recommender 8 is the most different case to the median and to Recommender 6. The first feature that makes it different is that it is a Conversational B recommender. Also, our application has retrieved this recommender because it is an implementation of the ExpertClerk system [12] and thus has several features that make it different from other recommenders. For example, it acts both as a navigation-by-asking and a navigation-by-proposing system.

The result displayed to the user is shown in Figure 6. The user can read descriptions of the two recommender systems and choose to execute one or both. Once the user has selected the closest recommender to his/her preferences, s/he can ask the system for something similar. The system uses the Bounded Greedy algorithm to select the next pair of recommenders.

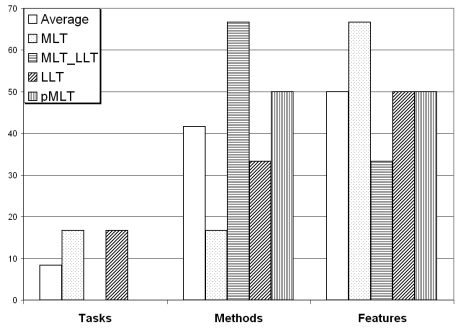

Fig. 7. Similarity approaches comparison

6 Evaluation

Our experimental evaluation is an ablation study. It is a leave-one-in study, where a chosen case from the case base is taken to be the user's target system. We simulate user preferences by deleting some of the case's attributes and take the resulting partial description to be a representation of the user's preferences.

Six representative recommenders were selected to act as target systems/queries (two Single-Shot systems, two Conversational A systems, and two Conversational B systems). We used random deletion, and hence we repeated each cycle twenty times to allow for deletion of different sets of attributes.

Our experiments measured the number of steps required to retrieve the same recommender using our tool. Obviously, the number of steps (or depth) when using 100% of the attributes is always 0 but depth will increase when using only 75%, 50% and 25% of the attributes.

During the first stage of our experiment we used only one of the three similarity approaches: either tasks, methods or semantic features. We also tried each preference feedback elicitation strategy: MLT, LLT, pMLT, MLT_LLT (see Section 3.5). Figure 7 shows, for every similarity approach and every preference feedback elicitation strategy, the percentage of queries where that particular combination results in the minimum number of retrieval steps. Averaging those results we find that task-based similarity provides the best results in 10% of the queries, method-based in 40% and feature-based in 50%. As might be expected, the semantic feature similarity is most often the best because it is a knowledge-intensive measure.

Next we tested our hypothesis that the best similarity measure would be a weighted combination of the three similarity approaches using as weights the percentages discovered in the previous experiment. This hypothesis was actually confirmed in the experiments as shown in Figure 8 (left) where the proposed weight combination is shown to outperform other weight combinations (70%-15%-15%, 15%-70%-15%, and 15%-15%-70%), and Figure 8 (right) where it outperforms pure task, method and semantic feature approaches.

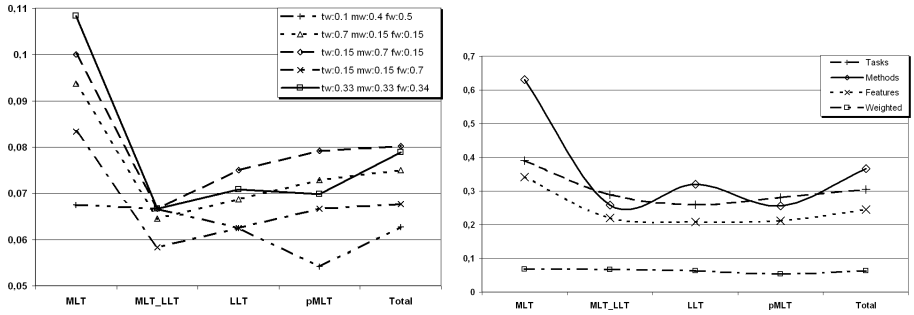

Fig. 8. Optimum weighted combination of similarity approaches

We can also propose a set of weights to use in the case where semantic features are not available. It is important to consider this scenario because it may not always be possible for an expert to define an ontology. In this case, our experiments show that the best weights are: Tasks = 34% and Methods = 66%. These values demonstrate that the behaviour of the system (methods) is more important than its structure (tasks) when computing the similarity.

7 Conclusions

In this paper, we have extended jCOLIBRI 2 with facilities to support reuse during the construction of CBR systems. In particular, we propose the use of templates, which a system developer can adapt to his/her new purposes. Our case-based template recommender draws ideas from case-based recommender systems research, and uses a new approach that we call Retrieval-by-Trying. We have illustrated the ideas by building a case-based recommender system for case-based recommender systems. We have defined and empirically evaluated different approaches to the measurement of similarity. We found, as might be expected, that knowledge-intensive semantic features are more important than behavioural features, which are in turn more important than structural features.

We will soon report on an empirical evaluation in which students template-based and other approaches to build recommender systems. In the future, we want to apply these ideas to CBR systems other than recommender systems. For example, we are looking at textual CBR systems. And we want to gain more practical experience of using the approach for rapid prototyping in educational, research and industrial environments.

References

1. Smolárová, M., Návrat, P.: Software reuse: Principles, patterns, prospects. Journal of Computing and Information Technology **5**(1) (1997) 33–49
2. Aamodt, A., Plaza, E.: Case-based reasoning: Foundational issues,methodological variants, and system approaches. Artificial Intelligence Communications **7**(1) (1994) 39–59

3. Tautz, C., Althoff, K.D.: Using case-based reasoning for reusing software knowledge. In Leake, D.B., Plaza, E., eds.: Procs. of the ICCBR'97. LNAI 1266, Springer (1997) 156–165
4. Fernández-Chamizo, C., González-Calero, P.A., Gómez, M., Hernández, L.: Supporting object reuse through case-based reasoning. In Smith, I., Faltings, B., eds.: Procs. of EWCBR96. LNAI 1168, Springer (1996) 135–149
5. Tessem, B., Whitehurst, A., Powell, C.L.: Retrieval of java classes for case-based reuse. In Smyth, B., Cunningham, P., eds.: Procs. of the EWCBR'98. LNAI 1488, Springer (1998) 148–159
6. Grabert, M., Bridge, D.: Case-based reuse of software examplets. Journal of Universal Computer Science **9**(7) (2003) 627–640
7. Gomes, P.: A Case-Based Approach to Software Design. PhD thesis, Departamento de Engenharia Informática, Faculdade de Ciêcias e Tecnologia, Univerisdade de Coimbra (2003)
8. Althoff, K.D., Birk, A., von Wangenheim, C.G., Tautz, C.: Cbr for experimental software engineering. In Lenz, M., Bartsch-Spörl, B., Burkhard, H.D., S.Wess, eds.: Case-Based Reasoning Technology: From Foundations to Applications. LNAI 1400. Springer (1998) 235–254
9. Jedlitschka, A., Althoff, K.D., Decker, B., Hartkopf, S., Nick, M.: Corporate information network: The Fraunhofer IESE Experience Factory. In Weber, R., von Wangenheim, C., eds.: Workshops ICCBR'01. (2001) 9–12
10. Recio-García, J.A., Bridge, D., Díaz-Agudo, B., González-Calero, P.A.: Semantic templates for designing recommender systems. In: Procs. of the 12th UK Workshop on Case-Based Reasoning, University of Greenwich (2007) 64–75
11. Bridge, D., Göker, M.H., McGinty, L., Smyth, B.: Case-based recommender systems. Knowledge Engineering Review **20**(3) (2006) 315–320
12. Shimazu, H.: ExpertClerk: A conversational case-based reasoning tool for developing salesclerk agents in e-commerce webshops. Artificial Intelligence Review **18**(3–4) (2002) 223–244
13. Smyth, B., McClave, P.: Similarity vs. diversity. In Aha, D.W., Watson, I., eds.: Procs. of the ICCBR'01, Vancouver, Springer (2001) 347–361
14. McSherry, D.: Diversity-conscious retrieval. In Craw, S., Preece, A., eds.: Procs. of the ECCBR'02, Springer (2002) 219–233
15. McSherry, D.: Similarity and compromise. In Ashley, K.D., Bridge, D.G., eds.: Procs. of the ICCBR'03, Springer (2003) 291–305
16. Wilke, W., Lenz, M., Wess, S.: Intelligent sales support with CBR. In Lenz, M., Bartsch-Spörl, B., Burkhard, H.D., Wess, S., eds.: Case-Based Reasoning Technology: From Foundations to Applications. Springer (1998) 91–113
17. Doyle, M., Cunningham, P.: A dynamic approach to reducing dialog in on-line decision guides. In Blanzieri, E., Portinale, L., eds.: Procs. of the EWCBR'00, Springer (2000) 49–60
18. Burke, R.D., Hammond, K.J., Young, B.C.: The FindMe approach to assisted browsing. IEEE Expert **12**(5) (1997) 32–40
19. McGinty, L., Smyth, B.: Comparison-based recommendation. In Craw, S., Preece, A., eds.: Procs. of the ECCBR'02, Aberdeen, Scotland, Springer (2002) 575–589
20. Smyth, B., McGinty, L.: The power of suggestion. In Gottlob, G., Walsh, T., eds.: Procs. of the IJCAI'03, Morgan-Kaufmann (2003) 127–132
21. Recio-García, J.A., Díaz-Agudo, B., González-Calero, P.A., Sánchez, A.: Ontology based CBR with jCOLIBRI. In: Procs. of the 26th SGAI Int. Conference AI-2006. Springer-Verlang (2006)