Components for Case-Based Reasoning systems

Chema Abásolo, Enric Plaza and Josep-Lluís Arcos

IIIA - Artificial Intelligence Research Institute
CSIC - Spanish Council for Scientific Research
Campus UAB, 08193 Bellaterra, Catalonia, Spain.
{abasolo,enric,arcos}@iiia.csic.es
http://www.iiia.csic.es

Abstract In this paper we present CAT-CBR a component-based platform for developing CBR systems. CAT-CBR uses UPML (Universal Problem-solving Methods Language) for specifying CBR components. A collection of CBR components for retrieval of propositional cases is presented in detail. The CAT-CBR platform guides the engineer using a case-based recommendations system to develop a configuration of components that satisfies the requirements of a CBR system application. We also present how to develop a runtime CBR application from the configuration resultant of the configuring process.

Keywords: CBR, Knowledge modeling.

1 Introduction

Developing a CBR system is a very complex problem, many decisions have to be taken during the development. These decisions concern to the techniques we want to use and how they can be integrated in the CBR system; also a representation model of the cases must be chosen, according to the domain where the CBR system will be applied.

Knowledge modelling aims on solving this kind of problems, developing complex software systems. The approach we take is to model the different techniques or components as Problem-Solving Methods. These Problem-Solving Methods represent reasoning processes or software components that achieve a specific task.

UPML is used to specify these components. UPML is a language that allows us to describe Problem-Solving Methods, Tasks, Domain Models and the Ontologies used by them. UPML also provides, as a software architecture, connectors (Bridges) for these components.

In this paper we present the CAT-CBR platform; a platform for developing CBR systems. CAT-CBR provides to the engineer a library of CBR components that he can combine in an application (a CBR system). The platform provides to the engineer recommendations of which components will fit better to the requirements; these recommendations are given using a case-based recommendation system based on previous applications developed using CAT-CBR. Finally CAT-CBR gives the possibility to the engineer of generating a runtime application from the configuration he has made; this process of generating a runtime application has two parts: one automatic and the other has to be done manually by the engineer.

The paper is divided in the following sections. First, in section 2, the general aspects of the platform are presented. Section 3 presents a brief description of UPML. In section 4 we present the UPML library of CBR components. After this section 5 describes the process of configuring a CBR system using the platform. Section 6 describes how the platform operationalizes a CBR system. Finally section 7 presents some conclusions and future work.

2 The CAT-CBR platform

In this section, we present the general concepts of the CAT-CBR platform. This platform uses a library of CBR components to guide the engineer in the development of CBR systems. These components describe

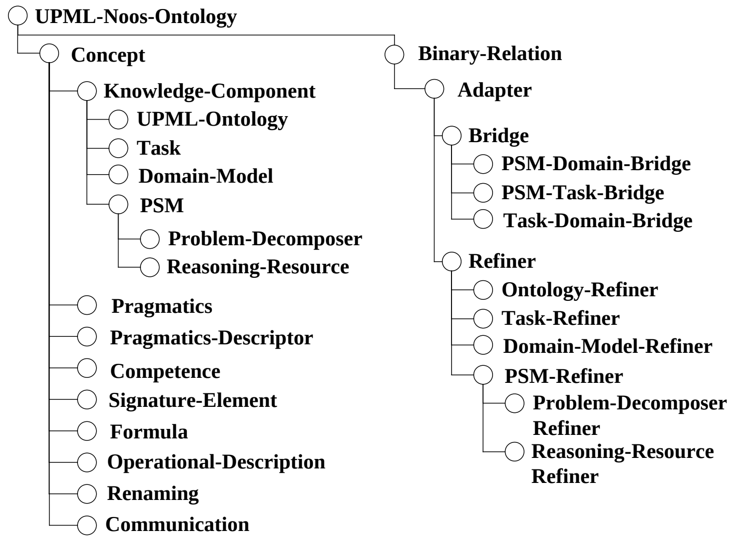


Fig. 1. The UPML hierarchy of elements.

the different tasks that can appear in a CBR system and also the problem-solving methods that can be applied to these tasks. That drives us to work with these components in a problem-solving methods framework. The CAT-CBR platform has been developed on Noos platform [2]. Noos uses *feature terms* as representation language. Noos is a platform used for developing CBR systems.

To describe CBR components, inside this framework, we have used UPML (Universal Problem-solving Methods Language) [5]. UPML is a language for specifying components of knowledge systems. Two levels can be differentiated in a component description: a specification level —where we use UPML– and an operational level; as operational language we use Noos, defining functions that implement the components. Using UPML and feature terms a library of CBR components has been specified.

The goal of CAT-CBR is, given the requirements of an engineer, to develop a CBR application; to achieve this objective two processes has been specified: Configuration Process and Operationalization Process. The configuration process consists in selecting different components and connect them in order to specify an application. The CAT-CBR has an interactive tool where the engineer chooses the components to be included in the application. This tool is built over a CBR system that guides and gives support to the engineer during this configuration process. Section 5 explains in more detail this configuration process.

Finally, the operationalization process takes an application specification (in the form of a configuration of components) and generates an executable application. The approach we have taken to operationalize a configuration is that the platform generates a file that links the Noos methods following the structure of the configuration of components. This operationalization process is described in more detail in section 6.

3 The UPML Language

Problem-Solving Methods are reusable components for implementing the reasoning part of knowledge-based systems. The UPML language aims at describing and implementing such architectures and components to facilitate their semiautomatic reuse and adaptation.

UPML provides both a framework and a language to describe libraries of knowledge components and their relationships to form knowledge systems. UPML is a software architecture that defines a set of components and connectors between these components. Figure 1 shows the hierarchy of the different components and connectors defined in UPML.

There are three main entities in the UPML: Tasks, Problem-Solving Methods and Domain Models.

A *Tasks* specifies the goals to be achieved by the Problem Solving Methods of the library. A Task is characterized by its input roles and output roles together with preconditions and postconditions. Input and output roles describe the types of elements used as input and the type of the result of the task. Preconditions describe some properties needed by the task to be achieved; and postconditions describe the properties that we get when a task is achieved.

Problem-Solving Methods (PSM) describe which reasoning steps and which types of knowledge are needed to perform a task. A PSM specifies a particular way to solve a task. The main attributes of a PSM are the input/output roles, plus the preconditions and postconditions to be fulfilled by the input and output roles. These attributes determine when a PSM can solve a task. There are two subclasses of PSM: *Problem Decomposers* and *Reasoning Resources*. Problem Decomposers specify decomposition of tasks into subtasks and the data flow between the different subtasks (*operational description*). Reasoning Resources are the elementary PSMs; they specify how to solve a task using external knowledge described as Domain Models.

A *Domain Model* characterizes domain knowledge. A Domain Model consists of three elements: properties, meta-knowledge, and domain knowledge itself. The meta-knowledge captures the implicit and explicit assumptions made while building a Domain Model of the real world. The domain knowledge is the explicit knowledge of a Domain Model. The domain knowledge is build under the assumption that the meta-knowledge is true. Properties are statements of the domain knowledge and can be directly used in the configuration.

The UPML framework has the notion of *Ontology*. An Ontology defines a vocabulary used to describe the properties of the previous components. In our case we have an ontology for describing CBR concepts (section 4.1).

Finally UPML as a software architecture, defines different connectors between previous entities; these connectors are called bridges, and allow the interaction between two different entities. There are three kind of bridges, depending os which components it connects: PSM-Task Bridges, PSM-Domain Bridges and Task-Domain Bridges.

The different characteristics of the entities described above are defined using an Object Language. UPML leaves open the decision of which Object Language is used to describe the components. This Object Language will be used by the inference process in the configuration of an application. In our case, as the platform has been built over Noos platform, the Object Language is Feature Terms and the inference process is that of term subsumption [2].

The CAT-CBR platform supports in part the UPML software architecture. In particular, some assumptions have been made to focus on the characteristics that we deem more useful for CBR system engineering. Specifically, CAT-CBR does not support automatic use of bridges. In practice this amounts to two effects: first, the PSM-Domain and Task-Domain bridges need be defined manually by the engineer; second, PSM-Task bridges are not need because CAT-CBR uses only one CBR-ontology to describe Tasks and PSMs.

4 CBR components library

The main idea of a CBR system is to use past situations (cases) to solve a new problem. A case is composed by a past problem and the solution to this problem. A CBR system entails four phases: Retrieve, Reuse, Revise and Retain [1], as shown in Figure 2. In CAT-CBR we are developing components for three stages: Retrieve, Reuse and Retain. The Revise phase is not incorporated in the library of components, because normally this process is done by an expert externally.

Currently, Retrieve PSMs for propositional cases and relational cases have been analyzed and implemented. In Reuse we have started adding to the library of components for classification, and some components for adaptation, as Constructive Adaptation [12]. We plan to incorporate components for the Retain phase in the future.

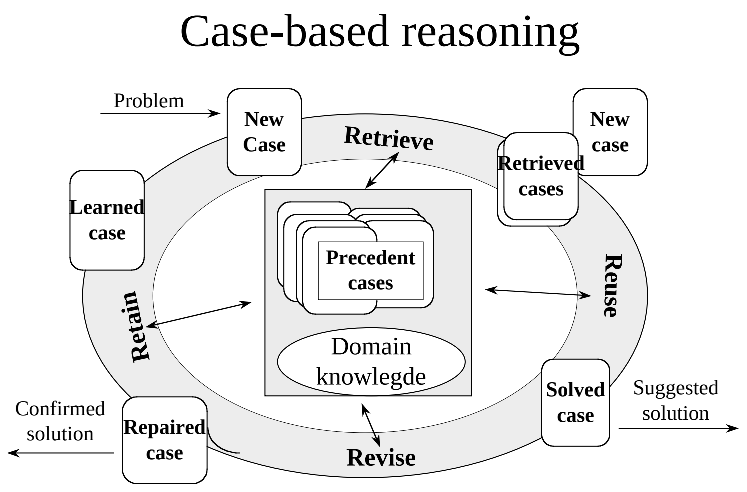


Fig. 2. The CBR cycle [1].

First we will present a vocabulary (ontology) used to describe CBR components (section 4.1). Then we will introduce a description of some components for the Retrieve and Reuse stages (section 4.2).

4.1 CBR components ontology

As it is presented in section 3, the UPML entities (Tasks and PSMs) are described using an ontology. In this section we present briefly the ontology defined for describing CBR components. This ontology allow us to describe the preconditions and postconditions, the assumptions and the type of the input and output roles of tasks and PSMs. Moreover, these concepts can be used to describe the type of knowledge and its properties used in a Domain Model.

Let us start with the concepts used to characterize preconditions, postconditions and assumptions of the components.

The CBR ontology is organized by different groups of concepts. A first group of concepts characterize performance issues of the components. In this group we have, as example, concepts that describe: the noise tolerance of a component (High Noise Tolerant, Noise Tolerant and Low Noise Tolerant), the accuracy of component (High Accuracy, Medium Accuracy and Low Accuracy), the space and time consumption (Low and High Space Consumption, and Low and High Time Consumption) and tolerance with missing values in the data.

Furthermore we need concepts to characterize properties of the inputs and output roles of the components. A retrieval PSM can return only a set of cases with no order, a set of cases partially ordered or a set of cases ranked by similarity. To characterize these properties the ontology has concepts such as: retrieve-similar-cases, retrieve-similar-cases-with-similarity.

Moreover, in this vocabulary there are concepts for describing the types of inputs and outputs used by a task or PSM, and the knowledge of a Domain Model. In this way we have, for example, *Dtree-Model* (that represents a Decision Tree), *Case-Base-Model* and *Case-Language-Model* (that represents the language of attributes used to describe propositional cases). Other models in CAT-CBR include: *Weight-Model* and *Order-Model* (used for aggregation PSMs); and *k-Model* (used in the *k-NN-Retrieval*).

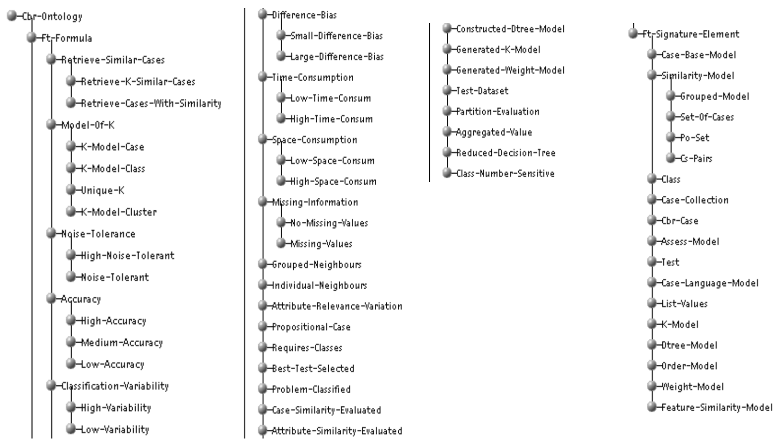


Fig. 3. Hierarchy of concepts defined in the CBR Ontology

4.2 CBR Components

In this section we will explain in detail part of the library, specifically we will focus on tasks and PSMs for retrieval with propositional cases[1]. Also we describe other Tasks and PSMs for classification (Retrieve and Reuse).

The process to construct this library involves analyzing existing techniques from a task-method decomposition point of view. Once this analysis is made, the different components are specified in UPML. The conclusion is that different components can be used in a same problem decomposition, giving us the notion of families of PSMs for solving a task.

The family of PSMs related with k-Nearest Neighbour can be described as a problem decomposer with two subtasks: *Assess-Similarity* and *Select-k*. These PSMs are used with propositional cases and achieves the retrieve stage. Figure 4 shows this decomposition and the different PSMs that can be applied for its subtasks, which are described below.

The *Asses-Similarity* task is achieved for propositional cases by a problem decomposer, called *Feature-Similarity-and-Aggregation*. This component has to subtasks: *Feature-Similarity*, that evaluates the similarity of the attributes one by one, and *Aggregation*, that aggregate the similarities of the attributes in an unique value. The *Feature-Similarity* task can be solved by two reasoning resources, depending if the cases have missing values or not; these two components are: *Feature-Similarity-without-Missing-Values* and *Feature-Similarity-with-Missing-Values*. For the *Aggregation* task the following PSMs can be applied: *City-Block-Aggregation*, *Euclidean-Aggregation*, *Txebitxev-Aggregation*, *Weighted Mean*, *OWA* and *WOWA*. These three first PSMs do not need any additional domain model to aggregate the *Feature Similarity*; while the others need a *Weight Model* of the attributes to achieve *Aggregation* (in *Weighted-Mean* and *WOWA*), and a *Order Model* between the attributes in *OWA* and *WOWA*. For other type of cases, more complex and more expressive, new PSMs will be added to the platform for achieving *Assess-Similarity* task.

For the *Select-k* task different PSMs can be applied depending on the available domain model. These PSMs are reasoning resources that use a model of k. These are the PSMs included in the library:

k-selection: It is a reasoning resource that uses a unique k as model to select the cases. It returns the k nearest neighbors of the problem. The value of k can be learned using techniques like leave-one-out or can be given by the engineer.

[1] PSMs for relational cases include: LAUD, LID and Perspectives, but they are not presented here for lack of space.

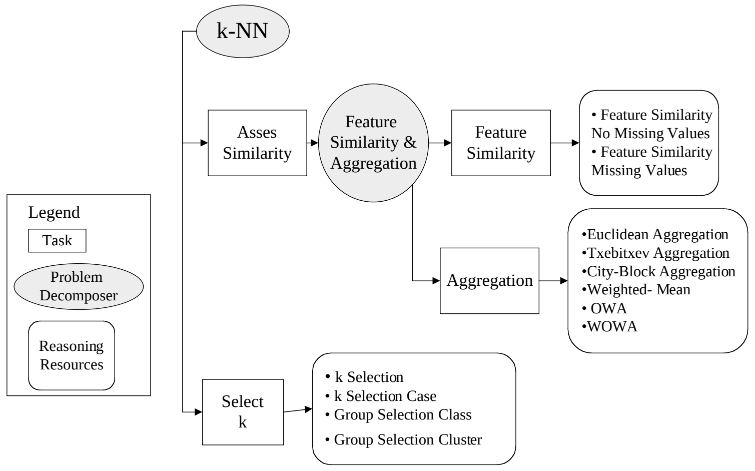


Fig. 4. K Nearest Neighbour decomposition and the different PSMs that can be applied.

k-selection-case: It is a reasoning resource that adaptively selects a value of k for a problem given a *k-case model* (a model characterizing for each case in the Case Base the k values that are better classifying that case).

Group-selection-class: It is a reasoning resource that uses a *k-class model* (this model has a k value for each class). This reasoning resource returns the neighbors grouped by classes (*Grouping Model*).

Group-selection-cluster: It is a reasoning resource that uses k-cluster model (this model has a k value for each cluster). This reasoning resource returns the neighbors grouped by clusters (*Grouping Model*).

The CBR library also includes PSMs for learning this different models of k given a Case Base.

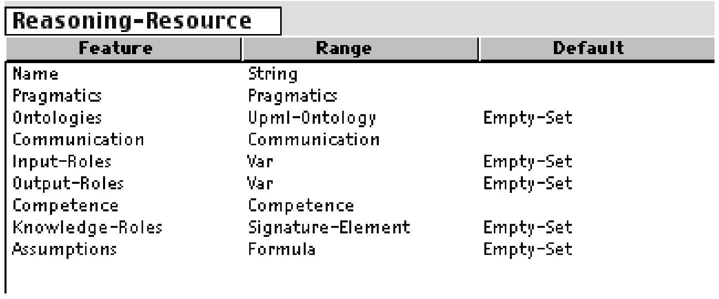
Another family of PSMs for the Retrieve phase use Decision Trees to retrieve similar cases. Retrieval using *Decision-Trees* is a PSM that uses a Decision Tree model indexing cases in a Case Base. This Decision Tree model has cases in its leaves nodes; the cases in a leaf node are those which satisfy all the test nodes in the path from the root to the leaf. There are four reasoning resources that solve Decision Tree Retrieval task:

DT-Retrieval: It is a reasoning resource that assumes no unknown values on the current problem. *DT-Retrieval-Missing-Values*: It is a reasoning resource that retrieves a set of cases from a decision tree allowing missing values in the current problem. When a missing value is found the PSM returns all the cases found on the leaves under the node where the missing value has been found. *DT-Retrieval-MostFV*: in this reasoning resource when a missing value is found, the PSM evaluates which is the most frequent value and continues the retrieve through this branch. *DT-Retrieval-AllBranch*: in this reasoning resource when a missing value is found, this PSM continues the retrieve process through all the branches (notice that this PSM is different from *DT-Retrieval-Missing-Values* in that it continues the retrieve process while *DT-Retrieval-Missing-Values* stops the retrieve process in the node where a missing value is found).

The library has two PSMs for inducing Decision Trees:

DT-Construction: It is a problem decomposer that has three subtasks: *Stop-Criteria*, *Select-Test* and *Branching*. The PSM evaluates whether a collection of cases satisfies the *Stop-Criteria*. If they do not satisfy it, then *Select-Test* determines the best criteria to split the collection into subcollections. After this, *Branching* generates all the branches for each subcollection. This PSM is applied recursively until all the final branches satisfy the *Stop-Criteria*. *DT-Construction-Pruning*: It is a problem decomposer that works in the same way than the previous one, but after the decision tree is made, a *Pruning* task eliminates branches in the Decision Tree in such a way that the accuracy improves.

The two main subtasks in these two PSMs are *Select-Test* and *Pruning*. For *Select-Test* task the library has several heuristic reasoning resources: *Gain* [13], *Chi-Square* ([6],[11]), *G-Statistic* [11], *GINI-Index* [4],



Reasoning-Resource

Feature	Range	Default
Name	String	
Pragmatics	Pragmatics	
Ontologies	Upml-Ontology	Empty-Set
Communication	Communication	
Input-Roles	Var	Empty-Set
Output-Roles	Var	Empty-Set
Competence	Competence	
Knowledge-Roles	Signature-Element	Empty-Set
Assumptions	Formula	Empty-Set

Fig. 5. Features of a *reasoning resource* and default values.

Gain-Ratio [8] and *RLM-Distance* [10]. For *Pruning* task the library has the following reasoning resources: *Error-Complexity* [4], *Critical-Value* [7], *Minimum-Error* [14], *Reduce-Error* [9] and *Pessimistic-Error* [8].

Classification Classification in CBR can be described as a problem decomposer with two subtasks; a *Retrieve* task and a *Reuse* task. For the retrieve task, the PSMs that can be applied are the ones described above. Reuse is a task that receives a *Similarity Model* (from the retrieve task) holding the cases retrieved that are more similar to current problem. The goal of the *Reuse* task is to determine a class for the current problem based on the information of the similarity model. For the reuse task the following three PSMs are included in the library:

Majority-Classification: This is a reasoning resource that selects the majority class from the set of cases in the similarity model. *Probabilistic-Classification*: This is a reasoning resource that selects a class probabilistically from the set of cases in the similarity model. *Grouping-Classification* is a reasoning resource that selects the most representative class, using a *Grouping-Model* (this is a model obtained from PSMs *Group-Selection-Class* and *Group-Selection-Cluster*). This similarity model has a grouping for each class. The reasoning resource determines the grouping that is best (according to an entropic criterion) and classifies in the class of that grouping.

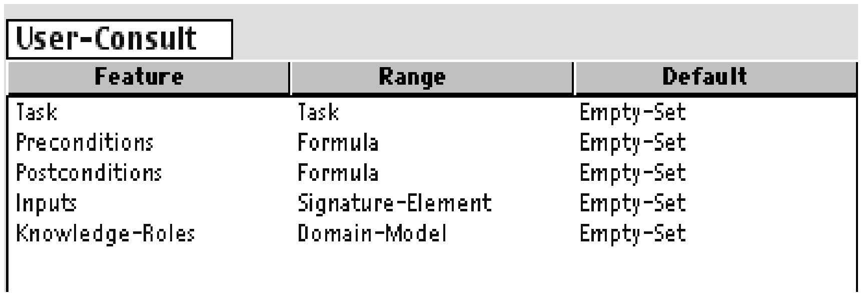
5 CBR system Configuration

Before explaining the configuration process itself, the representation of the elements used in the CAT-CBR will be introduced. As is presented before, this platform is developed on the Noos platform, and the representation language are feature terms.

CAT-CBR uses feature terms to represent both UPML and the Object Language. That is to say, every element of UPML is represented as a feature term and every type of UPML element is represented as a sort; see, for instance, figure 5 where a sort for *Reasoning Resource* is shown.

The configuration process starts with an engineer input, that is what we call a *Consult*. A Consult contains the requirements for the target application, specifying its inputs, preconditions and postconditions, plus the Domain Models available for configuring an application. As preconditions the engineer describes all the properties that he can assure that are true, while the postconditions represents the properties that he wants to be satisfied by the application. Figure 6 shows the Consult sort.

A target application will be a configuration of components of the library that satisfies the consult. A configuration specifies a PSM for each Task. A PSM can be decomposed in new subtasks (problem decomposer) or be elementary (reasoning resource). A configuration also specifies which Domain Models available will be used by each reasoning resource. Figure 7 shows an example of configuration.



User-Consult

Feature	Range	Default
Task	Task	Empty-Set
Preconditions	Formula	Empty-Set
Postconditions	Formula	Empty-Set
Inputs	Signature-Element	Empty-Set
Knowledge-Roles	Domain-Model	Empty-Set

Fig. 6. Features and default values of the *Consult* sort.

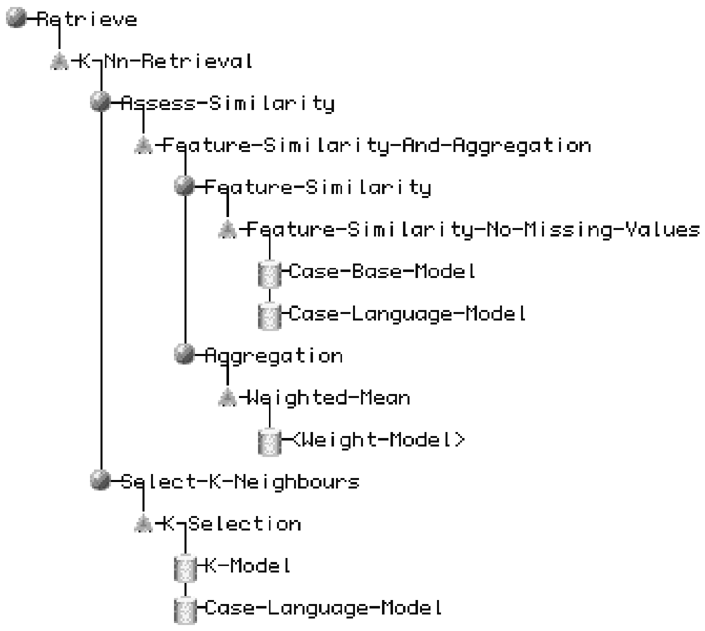
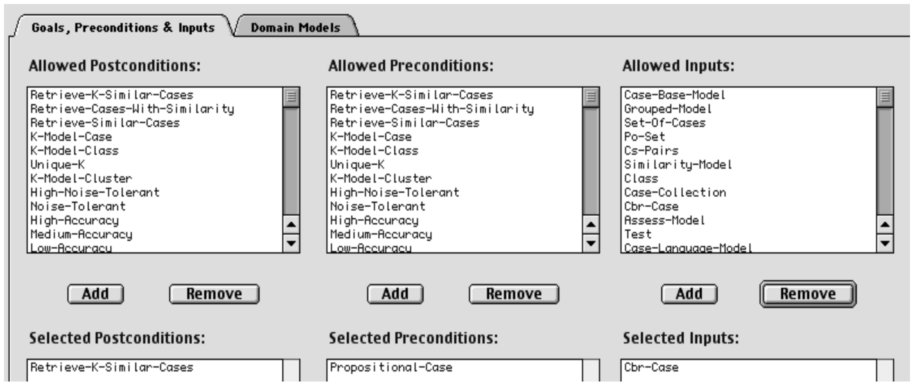


Fig. 7. A Configuration of a CBR application, where balls represent tasks, triangles represents PSMs and containers represents domain models.



Allowed Postconditions:

Retrieve-K-Similar-Cases
Retrieve-Cases-With-Similarity
Retrieve-Similar-Cases
K-Model-Case
K-Model-Class
Unique-K
K-Model-Cluster
High-Noise-Tolerant
Noise-Tolerant
High-Accuracy
Medium-Accuracy
Low-Accuracy

Allowed Preconditions:

Retrieve-K-Similar-Cases
Retrieve-Cases-With-Similarity
Retrieve-Similar-Cases
K-Model-Case
K-Model-Class
Unique-K
K-Model-Cluster
High-Noise-Tolerant
Noise-Tolerant
High-Accuracy
Medium-Accuracy
Low-Accuracy

Allowed Inputs:

Case-Base-Model
Grouped-Model
Set-Of-Cases
Po-Set
Cs-Pairs
Similarity-Model
Class
Case-Collection
Cbr-Case
Assess-Model
Test
Case-Language-Model

Selected Postconditions:

Retrieve-K-Similar-Cases

Selected Preconditions:

Propositional-Case

Selected Inputs:

Cbr-Case

Fig. 8. Interface that shows the options to the engineer to start the configuring process.

The configuration process has been developed as a case-based recommendation. CAT-CBR needs information about past configurations in order of giving the recommendations to the engineer. These past configurations are modeled as *Configuration Case*, and are stored in a Case Base.

Once we have seen the representation of the components and some extra elements, let us start describing the configuring process.

First of all, the engineer has to determine his requirements of the CBR system he wants to develop. For that purpose, CAT-CBR presents a first interface to the engineer (see fig.8) where he can select the goals, assumptions, inputs and domain models (knowledge that is available) for his CBR system and that constitutes the Consult. After this the platform searches, in the configuration-case case base, for a past configuration that satisfies these requirements; if there is one configuration, the CAT-CBR presents to the engineer the possibility of reusing this past configuration as a solution.

During the configuring process, the engineer selects a task as top level task of his CBR system. Then, CAT-CBR presents to the engineer an ordered list of PSMs that can be used to solve that task. The PSMs presented by the system are those which *match* with the task, in the sense of component matching explained presently.

Component matching has two parts, signature matching and specification matching. Thus, for two components, a task and a PSM, they match when they have signature matching and specification matching.

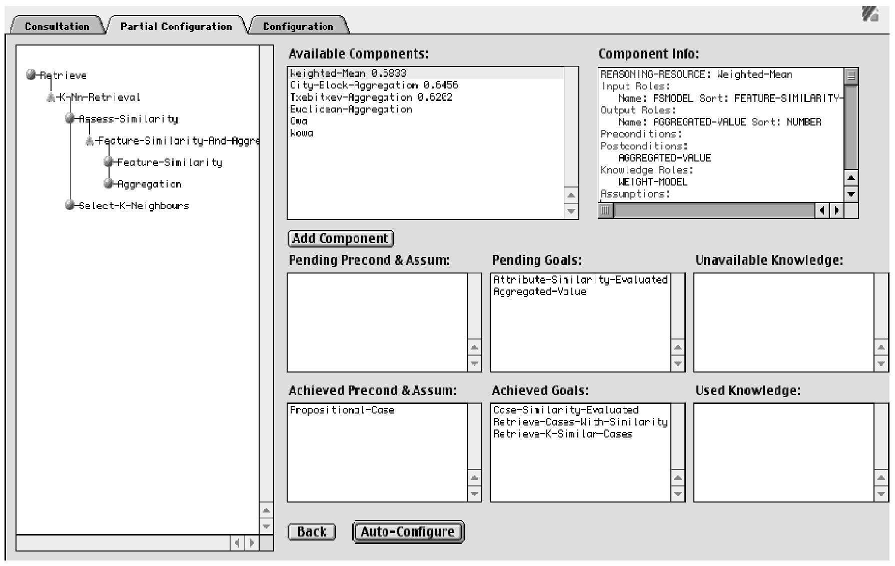
Signature matching requires that their input and output signatures match, and in our context this means that a problem-solving method has input and output signatures that are equal or that refine those of a task.

Moreover, specification matching requires that a problem-solving method has weaker preconditions than a task and stronger postconditions than a task.

The PSMs are presented, to the engineer using a case-based recommendation criterion. First it computes the similarity of the new consult with the consults of the configuration-case case base. This similarity measure allows us to derive a ranking of the most similar configurations to the new problem. As similarity function we have used LAUD [3], that evaluates the similarity between two structures of feature terms. The PSMs are ordered using this ranking in such a way, a PSM is better than other one if the configuration where it appears is more similar than the other one.

Once the PSMs are presented, the engineer selects the one that will be used to solve the task. To make this selection easier to the engineer, the interface presents some extra information (see Fig.9). This extra information includes the goals and assumptions that are achieved and the ones that are not yet satisfied in the partial configuration. Also the interface shows the specification of the PSMs that match a particular task.

The process of selecting PSMs for each open task continues until the partial configuration is considered a final one. A configuration is final when: a) all the requirements of the engineer are achieved and b) it is



```
Retrieve
  K-Nn-Retrieval
    Assess-Similarity
      Feature-Similarity-And-Aggre
        Feature-Similarity
        Aggregation
    Select-K-Neighbours
```

Available Components:

```
Weighted-Mean 0.6833
City-Block-Aggregation 0.6456
Txebitxev-Aggregation 0.6202
Euclidean-Aggregation
Owa
Wowa
```

Component Info:

```
REASONING-RESOURCE: Weighted-Mean
Input Roles:
   Name: FSMODEL Sort: FEATURE-SIMILARITY-
Output Roles:
   Name: AGGREGATED-VALUE Sort: NUMBER
Preconditions:
Postconditions:
   AGGREGATED-VALUE
Knowledge Roles:
   WEIGHT-MODEL
Assumptions:
```

Pending Precond & Assum:

Pending Goals:

```
Attribute-Similarity-Evaluated
Aggregated-Value
```

Unavailable Knowledge:

Achieved Precond & Assum:

```
Propositional-Case
```

Achieved Goals:

```
Case-Similarity-Evaluated
Retrieve-Cases-With-Similarity
Retrieve-K-Similar-Cases
```

Used Knowledge:

Fig. 9. Interface that shows a partial configuration and all the extra information.

complete. A configuration is complete when all the tasks have a PSM to solve them and all the domain models needed by the reasoning resources of a configuration are available.

During the configuration process the engineer can take back his decisions; moreover he can use an auto-configure option, this starts an automatic configuring process starting in the last partial configuration. This automatic process is done using Constructive Adaptation [12]. This automatic process is a best-first search process in the space of legal configurations. Constructive Adaptation uses similarity with past configurations to direct the search process. The search is exhaustive, so cases are used only to order the sequence in which alternatives are considered. Once the process is ended the engineer can store the configuration in the Configuration case base.

6 Operationalization

The configuration process yield a configuration of UPML components (see fig.7). A configuration must be operationalized to get an executable application. As we have seen a configuration is a task-method decomposition. To operationalize a configuration, the different PSMs that solve the tasks and the Domain Models used by the reasoning resources must be linked. CAT-CBR provides support for operationalizing a CBR system. CAT-CBR provides the "glue code" that links the implementations of the different PSMs following the structure of the configuration. CAT-CBR does not generate automatically the connectors between reasoning resources and Domain Models; these connectors must be manually defined by the engineer.

The two kinds of PSMs (problem decomposer and reasoning resource) have different implementation. First, the *problem decomposer* has an operational description; this expresses the control and data flow between the different subtasks of a PSM. In our case the operational description is described in a functional way. Let us see an example, we present, for instance, the operational description of the *k-NN-Retrieval* Problem Decomposer:

```
(use-subtask
        (Select-K-Neighbours
             Problem
             (use-subtask
                (Assess-Similarity Problem))))
```

This Problem Decomposer (PD) has two subtasks (*Assess-Similarity* and *Select-k-Neighbours*) and one input (*Problem* of sort CBR-Problem). The operational description expresses (with the `use-subtask` construct) that the *Assess-Similarity* subtask has as input `Problem` (the input of *k-NN-Retrieval*) and that *Select-k-Neighbours* has as inputs the `Problem` and the result of the Assess-Similarity task.

On the other hand, a Reasoning Resource (RR) solves a Task using input roles and some domain models. For instance, the *K-Selection* reasoning resource will be used to solve the *Select-k-Neighbours* in our example. This reasoning resource has as inputs the problem to be solved (`Problem` of sort CBR-Problem) and the output of the *Assess-Similarity* Task (a Assess-Model). This RR also uses two domain models, a *Case-Language-Model* (that characterizes the kind of attributes used for describing a propositional case) and *k-Model* (model of k). The *resource identifier* in the *pragmatics* feature determines the name of the function (*k-Selection*), that implements the reasoning resource.

The operationalization process generates the "glue code" that calls, at the appropriate places, the operational descriptions of PDs and the code of the "resource identifier" of the RRs. Moreover, the domain models used by a RR, are referenced using an identifier. CAT-CBR provides a Project file where the engineer declares the files or URLs containing each domain model, and determines an identifier for each one.

Let us see an example: consider the PD *k-NN-Retrieval* and assume that for subtask *Assess-Similarity* the PSM *Feature-Similarity-And-Aggregation* is chosen, and for subtask *Select-k-Neighbours* task the PSM *k-Selection* is chosen. CAT-CBR generates the following Noos-method for this PD:

```
(define-method k-NN-Retrieval-Code
                  ((Problem CBR-Problem))
  (k-Selection
     Problem
     (Feature-Similarity-And-Aggregation-Code
                Problem))
     !Case-Language-Model
     !k-Model))
```

The name of the Noos-method is the name of the PD with the `-Code` ending and the input parameter is a `Problem` of sort CBR-Case. The body of this method corresponds with the operational description of the PD, substituting the `use-subtask` calls with the methods that implement the PSMs that solve the tasks. In our example `use-subtask` of *Select-k-Neighbours* is substituted by *k-Selection* with input parameters *Problem* and the result of the *Assess-Similarity* task; moreover, as *k-Selection*, being a RR, uses domain models they are added as parameters (*Case-Language-Model* and *k-Model*). The `use-subtask` of *Assess-Similarity* is substituted by `Feature-Similarity-And-Aggregation-Code` (that is the implementation of the PD selected) with input parameter `Problem`; notice that the result of this method (of sort *Assess-Model*) is the second input parameter of the `k-Selection` method.

All the Noos-methods generated and the functions of the resources identifiers, together with the domain models declared by the engineer, constitutes the runtime application that implements the CBR system configured by the engineer.

7 Conclusions

This paper presents the CAT-CBR platform for rapid and flexible development of CBR systems. The platform uses a library of CBR components described with UPML. UPML is a language for describing problem-solving methods, tasks, domain models and ontologies. In this paper we have described only the relevant aspects of UPML used in our work.

Using UPML we have analyzed in more detail the phases of the CBR cycle as tasks. We have presented, for propositional cases, the retrieval methods more frequently used and have modeled them in a family of components that use a common CBR ontology.

We have also presented how CAT-CBR guides the engineer in the development of a CBR system using a case-based recommendation system. In this configuration process, CAT-CBR only presents to the engineer those PSMs that match with a task and presents them in a ranking recommendation (based on a similarity between past configurations and the current requirements).

CAT-CBR provides the engineer a runtime CBR application for the components configuration. This runtime application is generated automatically, except for the connectors between PSMs and domain models that have to be manually defined by the engineer.

Currently Reuse phase includes components for classification; we plan to incorporate components for design and configuration tasks. Finally, components for the Retain phase of the CBR cycle also will be incorporated.

Acknowledgements

This research has been supported by the Esprit Long Term Research Project 27169: IBROW the TIC Project 2000-1094-C02 Tabasco and FPI TIC2000-1094-C02-02 grant.

References

1. Agnar Aamodt and Enric Plaza. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *Artificial Intelligence Communications*, 7(1):39–59, 1994.
2. Josep Lluís Arcos. *The Noos representation language.* PhD thesis, Universitat Politècnica de Catalunya, 1997.
3. Eva Armengol and Enric Plaza. Similarity assessment for relational cbr. In *Proceedings ICCBR 2001*, LNAI. Springer Verlag, 2001.
4. Breinman L. et al. Classification and regression trees. *Wadsworth International*, 1984.
5. D. Fensel, V. R. Benjamins, M. Gaspari S. Decker, R. Groenboom, W. Grosso, M. Musen, E. Motta, E. Plaza, G. Schreiber, R. Studer, and B. Wielinga. The component model of upml in a nutshell. In *Proceedings of the International Workshop on Knowledge Acquisition KAW'98*, 1998.
6. A. Hart. Experience in the use of an inductive system in knowledge engineering. In M. Bramer, editor, *Research and Developments in expert systems.* Cambrdge University Press, 1984.
7. Mingers J. Expert systems-rule induction with statistical data. *Journal of the Operational Research Society*, 1987.
8. Quinlan J.R. Inducing of decision trees. *Machine Learning, 1, 81-106*, 1986.
9. Quinlan J.R. Simplifying decision trees. *International Journal of Man-Machine Studies*, 1987.
10. Ramon López de Mántaras. A distance-based attribute selection measure for decision tree induction. *Machine Learning*, 6:81–92, 1991.
11. J. Mingers. Expert systems-rule induction with statistical data. *Journal of the Operational Research Society*, 1987.
12. E. Plaza and J.L. Arcos. Constructive adaptation: A search-based approach to reuse in cbr. In *Submitted*, 2002.
13. J.R. Quinaln. Learning efficient calssification procedures and their application to chess and games. In J. G. Carbonell R. S. Michalski and T. M. Mitchell, editors, *Machine Learning: An Artificial Intelligence approach.* Morgan-Kaufmann, 1983.
14. Niblett T. Construction decision trees in noisy domains. *Progress in machine Learning*, 1986.