

Ontology based CBR with jCOLIBRI *

Juan A. Recio-García, Belén Díaz-Agudo,
Pedro González-Calero, Antonio Sánchez-Ruiz-Granados
Dep. Sistemas Informáticos y Programación
Universidad Complutense de Madrid
Madrid, Spain

email: {jareciog,antonio.sanchez}@fdi.ucm.es, {belend, pedro}@sip.ucm.es

Abstract

jCOLIBRI¹ is a Java framework that helps designing Case Based Reasoning systems. This paper presents the incorporation of Description Logics reasoning capabilities to the new release of the framework. With this extension jCOLIBRI facilitates the development of Knowledge Intensive CBR applications. Ontologies are useful regarding different aspects: as the vocabulary to describe cases and/or queries, as a knowledge structure where the cases are located, and as the knowledge source to achieve semantic reasoning methods for similarity assessment and case adaptation that are reusable across different domains.

1 Introduction

jCOLIBRI is an object-oriented framework in Java for building Case Based Reasoning (CBR) systems. This framework promotes software reuse integrating the application of well proven Software Engineering techniques with a knowledge level description that separates the problem solving method (PSMs), that defines the reasoning process, from the domain model, that describes the domain knowledge.

This paper presents the incorporation of Description Logics (DLs) reasoning capabilities to the framework. With this extension jCOLIBRI can acquire the domain knowledge from ontologies defined in DLs, allowing the development of Knowledge Intensive CBR (KI-CBR) applications [5],[10]. This approach solves the traditional problem of knowledge acquisition in CBR systems and improves the framework functionally with new PSMs for retrieving and adapting cases.

The next section presents an overview of the framework, whereas Section 3 presents ontology based reasoning methods in a conceptual level. Section 4 shows how these methods have been implemented and included in jCOLIBRI, explaining real examples of KI-CBR applications. Ontologies used by these

*Supported by the Spanish Committee of Science & Technology (TIN05-09382-C02-01)

¹<http://sourceforge.net/projects/jcolibri-cbr/>

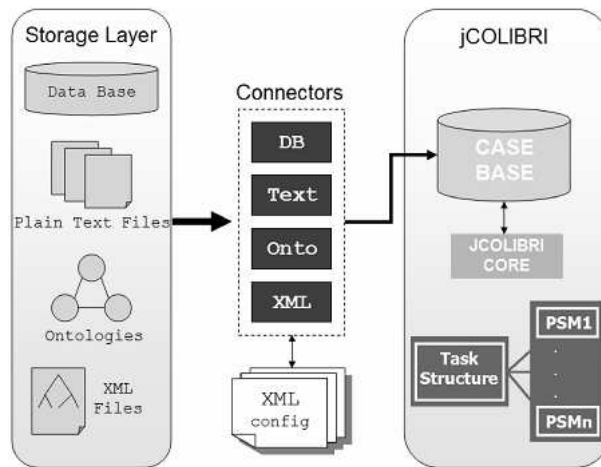


Figure 1: Connectors as abstraction of the storage medium

applications are formalized in the OWL language and are available through our web page ² together with the framework and its documentation.

2 The jCOLIBRI Architecture

In this section we present an overview on the main features of jCOLIBRI architecture. A more complete description about these topics can be found in the web page ³.

jCOLIBRI is distributed as a core module, which provides the basic functionality for building CBR applications, and several optional modules, which supply specific features useful only in some kind of systems (for example, the extension for Textual CBR [14]).

jCOLIBRI is built using three layers: interface, reasoning logic and persistence. The framework logic is composed by Java classes and interfaces, and some XML configuration files. Developing a new CBR system is made by writing some Java classes that extend classes of the framework, and configuring some XML files. To make easier this process the interface layer provides several graphical tools that help users in the configuration of a new CBR system.

Case Base management is split in two separate although related concerns: persistence and in-memory organization. This two-layer organization of the Case Base is a powerful approach that allows a number of different strategies for accessing the cases.

Persistence is built around connectors, objects that know how to access and retrieve cases from the medium and return them to the CBR system in a uniform way, giving flexibility against the physical storage (Figure 1). Using a

²<http://gaia.fdi.ucm.es/ontologies/index.html>

³<http://gaia.fdi.ucm.es/projects/jcolibri>

similar idea, the Case Base component implements a common interface for the CBR application to access the cases. This way the organization and indexation chosen for the Case Base will not affect the implementation of the reasoning methods.

The steps for designing a new CBR system in jCOLIBRI are:

1. Define the case structure. A case is composed by three components: description (describes the problem), solution (represents a possible solution approach) and result (reveals if the proposed solution is able to solve the problem). Description and solution are collections of simple or compound attributes, letting us to build a hierarchical case structure.
2. Choose similarity functions. These functions computes the similarity between the query and a case, and are used to choose the most similar case to the query. There are two types of similarity functions: local (compute the similarity between simple attributes) and global (compute some kind of average over the local similarities).
3. Case Base management. The designer should choose a connector for persistence and the in-memory organization of the case base.
4. Task/Method decomposition. The developer selects the tasks the system must fulfill and for every task assigns the PSM that will do the job. There are two kind of methods: decomposition (solve a task by decomposing it in simpler subtask) and resolution (solve the task directly). As PSMs are domain-independent they must be configured with the domain information.

3 Ontology based CBR

For the last few years our research group has been working in Knowledge Intensive CBR using ontologies [8, 4, 5, 6]. We claim that ontologies have an important role in the context of KI-CBR systems. We state that ontologies are useful for designing knowledge intensive CBR applications because they allow the knowledge engineer to use knowledge already acquired, conceptualized and implemented in a formal language, like Description Logics, reducing considerably the knowledge acquisition bottleneck. Moreover, the reuse of ontologies from a library also benefits from their reliability and consistency.

Our approach proposes the use of ontologies to build models of general domain knowledge. Although in a CBR system the main source of knowledge is the set of previous experiences, our approach to CBR is towards integrated applications that combine case specific knowledge with models of general domain knowledge. The more knowledge is embedded into the system, the more effective is expected to be. Semantic CBR processes can take advantage of this domain knowledge and obtain more accurate results.

Several investigations have suggested the use of DLs to organize, represent and retrieve cases in CBR systems like MRL [11], CATO [2], RESYN/CBR [13], and a diagnosis system for the French telephone network [15]. The common

ground is to take advantage of the DLs reasoning mechanisms for solving some of the CBR tasks.

We state that the formalization of ontologies is useful for the CBR community regarding different purposes, namely:

1. Persistence of cases and/or indexes using individuals or concepts that are embedded in the ontology itself.
2. As the vocabulary to define the case structure, either if the cases are embedded as individuals in the ontology itself, or if the cases are stored in a different persistence media as a data base.
3. As the terminology to define the query vocabulary. The user can express better his requirements if he can use a richer vocabulary to define the query. During the similarity computation the ontology allows to bridge the gap between the query terminology and the case base terminology.
4. Retrieval and similarity [7, 15, 13], adaptation [8] and learning [1].

The main usage of ontologies in the CBR community has been centered on similarity assessment. However, we think the usage of ontologies is specially interesting for case adaptation. Ontologies facilitate the definition of reusable, rich and semantic adaptation methods. In our modular architecture the reasoning methods are reusable because they rely on domain specific knowledge models from ontologies –that are interchangeable. This approach contrasts with most CBR systems that have traditionally relied on an enormous amount of built-in adaptation knowledge in the form of adaptation rules.

Among the contributions of this paper we include the implementation of all these features in the modular and reusable architecture of jCOLIBRI. This modular approach solves one of the main usability inconveniences of the previously cited approaches. Namely, the use of DLs as an ad-hoc persistence media that is not well suited if we have to deal with previously existing case bases. We propose the explicit separation between the case persistency media and the reasoning mechanisms to be used over a case base.

We abstract the case persistency details so the same CBR method can be used over different types of case bases. This approach allows much more flexible uses of ontologies and DLs reasoning. For example, section 5 describes an example where the case base is stored in a SQL database, the retrieval and similarity computation methods are configured as NN based on numeric and standard similarity functions, while adaptation is defined as a substitution method that relies on DLs to find suitable substitutes on the domain model.

Section 4 explains in a detailed way how jCOLIBRI manages separately the persistency and reasoning layers and the mechanisms we use to connect both layers. In this section we abstract from the case persistency layer and describe the reasoning methods in a conceptual level.

3.1 Ontologies as the CBR system vocabulary

Regarding the *case vocabulary*, there is a direct approach consisting on the use of a domain ontology in an object-oriented way: concepts are types, or classes,

individuals are allowed values, or objects, and relations are the attributes describing the objects. There are also simple types like string or numbers that are considered in the traditional way. The case structure is defined using types from the ontology even if the cases are not stored as individuals in the ontology. For example, the *Destination* concept can be used as a type where every one of its instances are the type values: *Lanzarote*, *Fuerteventura*, *Gran Canaria*,...

Another approach consists on using specific types (for example, value enumerations) to define the case structure. This approach is specially useful if we want to deal with existing databases where we can not expect to have exactly the same set of values, for example, the same set of destinations in the cases than in the ontology.

Regarding the *query vocabulary* we have two options to define the queries:

- Using exactly the same vocabulary used in the cases, i.e, the same types used in the case structure definition.
- Using the ontology as the query vocabulary, what allows richer queries. The user can express better his requirements if he can use a richer vocabulary to define the query. During the similarity computation the ontology allows to bridge the gap between the query terminology and the case base terminology.

Example: In the travel domain, lets suppose we have an existing case base where it is defined an enumerated type for the *Destination* attribute where the allowed values are countries: Spain, France, Italy and others. Suppose that *case_i* is a case whose destination is Spain. We do not want to restrict the query vocabulary to the same type but allow broader queries, for example:

- Query1: "I want to go to **Lanzarote**"
- Query2: "My favorite destination is **Europe**"
- Query3: "I would like to travel to **Spain**"

In the three queries and using the ontology of Figure 3 we could find *case_i* as a suitable candidate. Similarity assessment described in next section measures if it is the best candidate.

3.2 Case Retrieval using Ontologies

Every case retrieval method includes a similarity assessment method to measure the adequacy between the query and cases that are candidate to be retrieved. When we deal with ontologies it is clear that the concept hierarchy influences similarity assessment. Intuitively, it is obvious that the class hierarchy contains knowledge about the similarity of the objects. There are different approaches:

- *Classification based retrieval* using DLs classification capabilities. There are two different approaches:

$$f_{deep_basic}(i_1, i_2) = \frac{\max(\text{prof}(LCS(i_1, i_2)))}{\max(\text{prof}(C_i), C_i \in CN)} \quad f_{deep}(i_1, i_2) = \frac{\max(\text{prof}(LCS(i_1, i_2)))}{\max(\text{prof}(i_1), \text{prof}(i_2))}$$

$$\text{cosine}(i_1, i_2) = \text{sim}(t(i_1), t(i_2)) = \frac{\left| \left(\bigcup_{d \in t(i_1)} \text{super}(d, CN) \right) \cap \left(\bigcup_{d \in t(i_2)} \text{super}(d, CN) \right) \right|}{\sqrt{\left| \bigcup_{d \in t(i_1)} \text{super}(d, CN) \right|} \cdot \sqrt{\left| \bigcup_{d \in t(i_2)} \text{super}(d, CN) \right|}}$$

$$\text{detail}(i_1, i_2) = \text{detail}(t(i_1), t(i_2)) = 1 - \frac{1}{2 \cdot \left| \left(\bigcup_{d \in t(i_1)} \text{super}(d, CN) \right) \cap \left(\bigcup_{d \in t(i_2)} \text{super}(d, CN) \right) \right|}$$

Where

$\text{super}(c, C)$ is the subset of concepts in C which are superconcepts of c
 CN is the set of all the concepts in the current knowledge base
 $LCS(i_1, i_2)$ is the set of the least common subsumer concepts of the two given individuals
 $\text{Prof}(C_i)$: depth of the concept C_i
 $\text{Prof}(i)$: depth of the individual i

Figure 2: Concept based similarity functions in jCOLIBRI

1. Retrieval based on *concept classification*, where a concept description c_q is built using the restrictions specified in the query. This concept is then classified, and finally all its instances are retrieved.
 2. Retrieval based on *instance recognition*, where an individual is built and a number of assertions are made about it based on the features specified in the query. Instance recognition is applied to retrieve the most specific concepts of which this individual is an instance, and then all the instances of these concepts are retrieved.
- *Computational based retrieval* where numerical similarity functions are used to assess and order the cases regarding the query. The use of structured representations of cases requires approaches for similarity assessment that allow to compare two differently structured objects, in particular, objects belonging to different object classes. Similarity measures for structure case representations are often defined by the following general scheme [3]: The goal is to determine the similarity between two objects, i.e., one object representing the case (or a part of it) and one object representing the query (or a part of it). We call this similarity object similarity (or global similarity). The object similarity is determined recursively in a bottom up fashion, i.e., for each simple attribute, a local similarity measure determines the similarity between the two attribute values, and for each relational slot an object similarity measure recursively compares the two related sub-objects. Then the similarity values from the local similarity measures and the object similarity measures, respectively, are aggregated (e.g., by a weighted sum) to the object similarity between the objects being compared.

In this paper we are dealing with numerical similarity functions based on ontologies. In general, the similarity computation between two structured cases

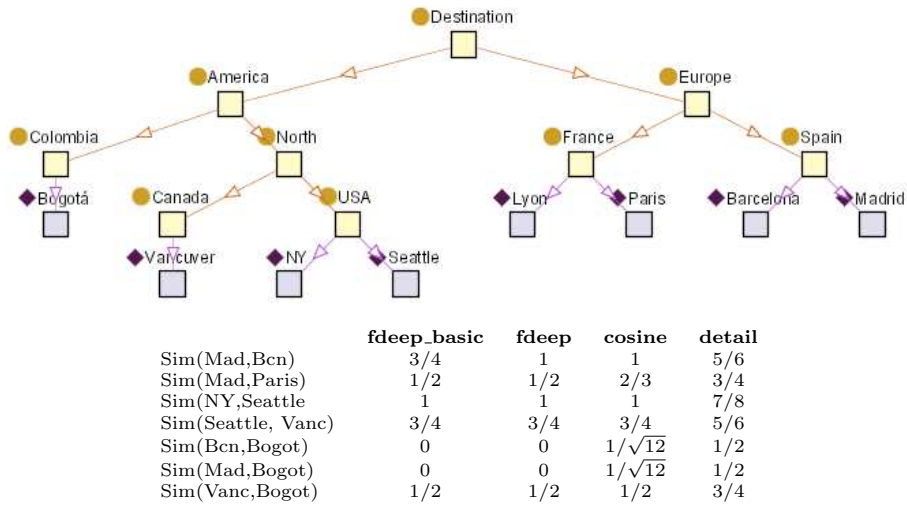


Figure 3: Example of application of the similarity functions

can be divided into two components that are aggregated [8]: the computation of a *concept based similarity* that depends on the location of the cases in the ontology (or intra-class similarity [3]) and the computation of a *slot-based similarity* (or inter-class similarity [3]) that depends on the fillers of the common attributes between the compared objects. This scheme is recursively applied until we reach individuals without slots, whose similarity is computed only by the concept-based similarity term (local similarity function).

There are different approaches for local similarity measures based on ontologies that are provided in jCOLIBRI. It initially offers four functions to compute the *concept based similarity* that depends on the location of the cases in the ontology. Of course, other similarity functions can easily be included. These similarity functions are shown in Figure 2 whereas Figure 3 explains how they work with a small ontology taken as example.

3.3 Case Adaptation based on Ontologies

Many authors agree about the fundamental role of case adaptation in the problem solving capabilities of CBR systems. However, it is in many ways the Achilles' heel of CBR. Case adaptation is a knowledge-intensive task and most CBR systems have traditionally relied on an enormous amount of built-in adaptation knowledge in the form of adaptation rules. In order to determine which rules must be included in the system, a deep analysis of the domain is required. Unfortunately, CBR is often applied to domains poorly understood or difficult to codify in the form of rules. So the leaders in the field have sometimes argued for postponing or avoiding the automatic adaptation.

In this section we propose an ontology based model and an adaptation scheme based on substitutions based on the method we described in [9]. Section

4 extends this example and describes implementation details of this adaptation method that is included in jCOLIBRI. We illustrate the method using a CBR system to recommend training exercise sessions to users according to their characteristics (this example is extended in Section 5).

1. The list L of items in the solution that need to be adapted is obtained. We find these items following a *relation path + a concept*. For example, if we want to substitute *training exercises* I would give the relation path that joins the case individual with the individuals representing the *training exercises*.
2. Every item in L is substituted by a proper new item. The search of proper substitutes is accomplished as a kind of *specialized search* which takes advantage of the DL knowledge base organization and applies similarity functions to find a substitute that is similar to the substituted one.
3. Substitute items that depend on other items of the solution that have already been adapted. For example, if we substitute the “biceps training exercise” we should change the warm-up exercises that are related to biceps.

Specialized search, as described in [12], is a way of finding candidates for substitutions in a case solution, where instructions are given about how to find the needed item. In our model memory instructions are given using the language that is explained in Section 5.

4 KI Case Representation and Retrieval in jCOLIBRI

In Section 2 we have introduced the architecture of jCOLIBRI based on 3 layers: persistence, reasoning logic and presentation. We also explained that one of the most important features of jCOLIBRI is that it allows to persist the cases independently of the applications logic. When a developer creates a new CBR system, he must define the structure of the cases using a graphical tool (shown in Figure 4) that generates a XML file with the structure information. In jCOLIBRI cases follow a composite pattern where an attribute (represented as a Java class) can be composed by other attributes, obtaining an easily extensible tree structure. Each attribute has a data type so jCOLIBRI can determine its persistence features, allowed similarity functions, etc.

When the developer has defined the case structure he configures a connector that uses that information for mapping the cases to the chosen persistence media. This mapping is also saved to a XML file. This approach allows us to create framework components that can be parameterized with these XML files containing the case structure and the connector configuration. This way, the CBR cycle is going to be independent of the persistence media.

With this architecture as the starting point our goals for integrating ontologies capabilities into the framework were: the use of ontologies as persistence media on one side, and to take advantage of DLs reasoning process to perform the retrieval, similarity and adaptation process on the other side. In a first approach we implemented the DLs extension following the idea that any case structure could be mapped into the persistence layer. But we noticed that this simple idea has side effects when using ontologies. For example, it has no sense to store in an ontology attributes like numbers (floats, integers, ...) because you should have an instance per each number, have order capabilities in the ontology... For solving this problem our final design divides case structures into two groups:

- Standard cases are composed by several attributes with different data types and can be stored into all the persistence medias excepting ontologies. If an attribute represents an instance of an ontology it has the *Concept* data type. This data type can be stored in any persistence media and connected with its instance into the ontology when loading the case.
- Pure ontological cases are completely composed by Concept-typed attributes. These cases don't need the connector information because are directly embedded into the ontology.

The Concept data type allows developers to indicate that the slot-filler of an attribute is going to be an instance of an ontology. This data type can also be stored in any kind of persistence media as a string of characters. It uses JENA⁴ to link the attribute with the instance loaded by a DLs reasoner.

This approach allows us to offer persistence, similarity and adaptation capabilities for the standard cases composed for attributes with different data types. Similarity and adaptation methods could connect a Concept typed attribute to the reasoner and then compute the algorithms exposed in previous sections.

The pure ontological case structure allows to store cases embedded into the ontology structure and perform several algorithms (like the classification based retrieval method described in Section 3.2) that cannot be used in other persistence medias. There is another reason for having this special case structure: we are representing the case structure twice. jCOLIBRI represents cases using Java classes to represent the attributes (slots), and then, the connector configuration relates these slots with the slot-fillers of the persistence media. However, using ontologies as persistence media means that the slots of the case structure are defined by the concepts and properties of the ontology and the slots-filler are the instances of these concepts. This way, it has no sense to do a jCOLIBRI representation of the case structure and then map it (using the connector) with the same case structure that is in the ontology. To solve this problem our pure ontological case structure represents directly the concepts and properties of the ontology using the Java classes that jCOLIBRI uses for reasoning. With this approach the connector for DLs does not need any configuration file and can load the cases from the ontology using only the case structure information. As

⁴JENA is a Java framework for building Semantic Web applications. It provides a programmatic environment for RDF, RDFS and OWL. <http://jena.sourceforge.net>

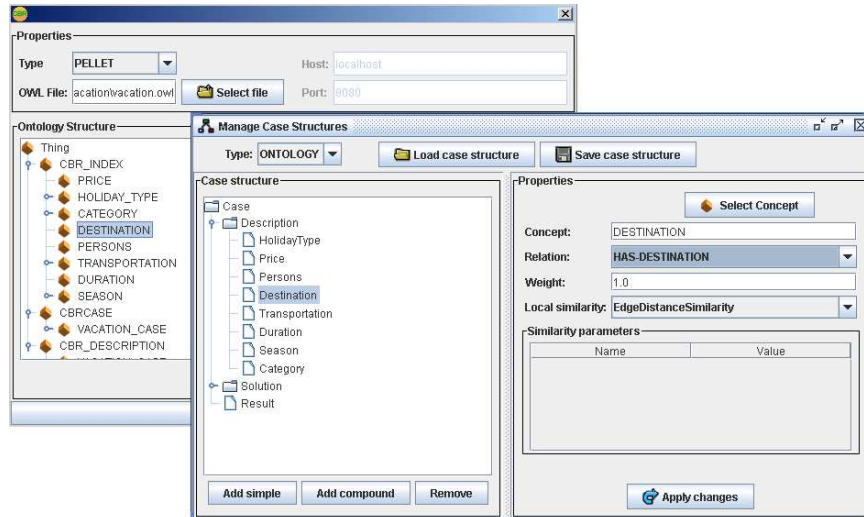


Figure 4: Case Attributes mapping with Ontologies

cases are directly mapped to the ontology, jCOLIBRI can execute methods that don't use numerical approaches to compute similarities, retrievals, etc.

As result of the development process jCOLIBRI 1.1 includes the required components to develop a KI-CBR application using Description Logics. The main component of this extension is a component that allows developers to connect with a DLs reasoner. This component, called *OntoBridge*, uses the JENA library to implement most of the required methods for accessing an ontology loaded in a reasoner and perform inferences. The *OntoBridge* is used by the DLs Connector for loading the cases, and by the retrieval, similarity and adaptation methods that connect with the reasoner to link the Concept attributes with their corresponding instances and then perform their task.

When a developer creates a case structure he needs to choose between a standard or ontological structure. If a standard structure is selected any attribute can be typed as Concept and then choose its corresponding ontology mapping as shown in Figure 4 (left). Then, the developer can select a similarity function for computing the similarity of the instances. As explained before, the similarity functions use the information stored in the attribute to access the ontology using the *OntoBridge* component. If the developer needs a pure ontological structure, he must completely map between the jCOLIBRI case structure and the concepts of the ontology that defines the case. To perform this action he must select the concepts and the ontology relations of the ontology as shown in Figure 4 (right).

jCOLIBRI includes the ontology-based retrieval methods explained in section 3.2: *classification-based retrieval* and *computational based retrieval*. This last method applies the similarity functions explained in Figure 2 to compute the most similar cases of the given query.

```

IDONTO:= /(Concept/Relation)* /Concept
IDPROPERTY:= (Relation/Concept)*
IDCASE:= "CASE."(attribute[.])*
RULE:= IDONTO,@,CONDITION,@,ADAPTATION
CONDITION:= (IDPROPERTY (|=|!=) IDCASE) | (IDCASE (|=|!=) String)
| [not] (IDPROPERTY instanceOf Concept)
ADAPTATION:= SUBSTITUTION | MODIFY [FOLLOWDEPENDENCIES Relation]
SUBSTITUTION:= "SUBSTITUTE" [#CONDITION#]
MODIFY:= DIRECTMODIFICATION | ANYOTHERINSTANCMODIFICATION
DIRECTMODIFICATION:= "DIRECT":IDPROPERTY:instance
ANYOTHERINSTANCMODIFICATION:= "ANYOTHERINSTANCEOF":IDPROPERTY:concept
[#CONDITION#]

```

IDONTO identifies the path to the instance to adapt.
IDPROPERTY identifies a property of a concept.
IDCASE identifies an attribute of the case
SUBSTITUTION substitutes the instance by another one chosen randomly. Accepts can include a condition that the substitute instance must obey.
DIRECTMODIFICATION substitutes an attribute of the instance with the instance indicated by the developer.
ANYOTHERINSTANCMODIFICATION substitutes an attribute of the instance by another instance of a concept. Accepts a condition for the substitute instance.
FOLLOWDEPENDENCIES applies the rule recursively to the instances related with the specified relation.

Figure 5: Adaptation rules syntax

5 Ontology-based Adaptation in jCOLIBRI

This method was described in Section 3.3 and can perform several transformations in an instance that represents the solution of a case. To implement this functionality in an extensible way we have defined a small but powerful ontology adaptation language. This language allows developers to create rules for adapting solutions and store them in textual files that will be loaded and interpreted by our generic adaptation method.

These rules are composed by 3 parts. The first one identifies the instance (I) to adapt following a `<concept,relation>` chain. Then the rule has a condition that will be evaluated for deciding if the adaptation of I should be performed. The last part defines the adaptation process. By now the method supports the following adaptations:

- Substitution of I by another instance specified by the developer. It is also possible to indicate conditions that the substitute instance must obey.
- Direct substitution of an instance that is related with I using a property of the ontology. Note that this related instance should define a characteristic of I . This option allows to indicate directly the substitute instance.
- Substitution of an related instance indicating conditions for the substitute instance.

These rules can also follow dependencies (i.e. properties) that relates instances to perform the adaptation process recursively.

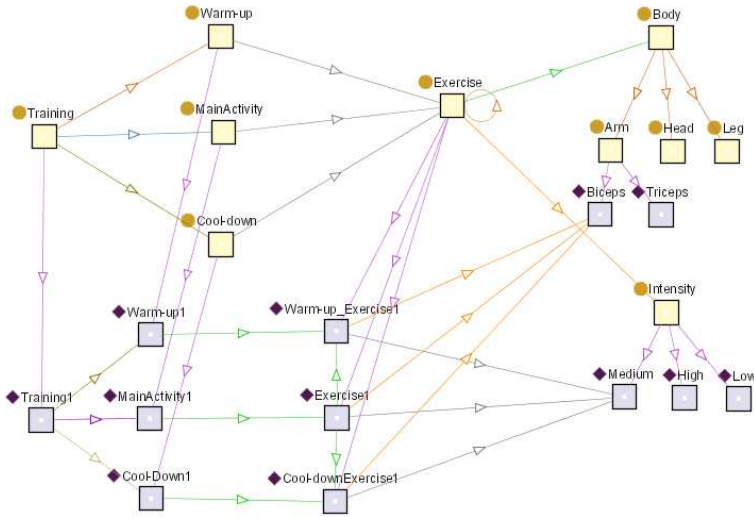


Figure 6: Sport Training Ontology

The generic adaptation method of jCOLIBRI explained in this paper uses several rules defined by the developer that indicates how to adapt the solution of the cases. Rules are composed by three parts: identification of the instance to adapt, condition to evaluate for performing the adaptation, and modification of the instance. They are stored in a text file and loaded and executed at runtime. Figure 5 shows their syntax. To illustrate how it works we are going to use a jCOLIBRI application named “Personal Sports Trainer” that recommends exercise sessions to the users. The query contains information about the user: age, sex, state of health, lesions, ... and the case base contains several cases with its solution stored in an ontology. The concept referenced by the solution is “Training” which is composed by: “Warm-up”, “Main Activity” and “Cool-down”. Each of these parts contains several exercises (“Exercise”) that have many properties: “hasIntensity”, “exercisedMuscles”... The exercises of the main activity are connected with the corresponding exercises in the warm-up and cool-down depending on the exercised muscles. Figure 6 shows a simplified view of the ontology. Now imagine the scenario where a user with a lesion in the arm sends a query and the system returns a training that contains exercises that use that body part. The following rule will adapt the retrieved solution changing the inappropriate exercises:

```

/Training/hasMainActivity/MainActivity/hasExercise/Exercise@
exercisedMuscles/Body == CASE.Description.lesion@
ANYOTHERINSTANCEOF:exercisedMuscles/Body:Body
#exercisedMuscles/Body not instanceof Arm#
FOLLOWDEPENDENCIES relatedExercise

```

Here “exercisedMuscles” is a property with domain “Exercise” and range “Body”. *Body* is a concept of the ontology that has subconcepts like *head*, *arm*,

leg,... In the condition we are comparing with the attribute of the case that stores the lesion. “relatedExercise” is the ontology property that relates the exercises in the main activity with the exercises of the warm-up and cool-down.

Next rule adjusts the intensity of the exercises depending on the state of health of the user. The state of health is included in the query (CASE-Description.HealthState). The adaptation changes directly the property “hasIntensity” of the exercises with the instance “Low” of the concept “Intensity”. As in the previous rule we propagate the transformations to the related concepts in the warm-up and cool-down.

```
/Training/hasMainActivity/MainActivity/hasExercise/Exercise@
CASE.Description.HealthState == low@
DIRECT:hasIntensity/Intensity: Low FOLLOWDEPENDENCIES relatedExercise
```

6 Conclusions

jCOLIBRI is a Java framework that helps during the development of Case Based Reasoning systems. The main advantage of using jCOLIBRI is mainly an easier development of CBR systems. To reach this goal we propose a design process that is based on reusing existing CBR knowledge (terminology, designs, tasks, methods, implementations), and on the integration of new components and the extension of existing components and their collaborations.

This paper describes the advantages that ontologies provide when they are used in CBR systems. The advantages are referred to different aspects: ontologies as the cases/queries definition languages independently of the persistence media we use for cases; ontologies as the knowledge structure where the cases are embedded, so that the persistence media for the case base uses the ontology formalization language; and ontologies as the knowledge source to get semantic reasoning processes, like retrieval, similarity and adaptation.

jCOLIBRI provides a well-defined and usable implementation of KI-CBR through ontologies. Though these ideas have repeatedly appeared in the literature this is to our knowledge the first general purpose standard-based implementation. This should promote the development of ontology-based CBR applications, but that is not only in our hands.

Besides the use of ontologies, jCOLIBRI 1.1 also includes new features for improving the development of CBR systems. The Web Interface extension includes several methods for launching a Tomcat server and communicate data between the PSMs of jCOLIBRI and the web applications running on the server. The Evaluation module test the efficiency of the CBR applications using the typical N-fold, Hold-Out or LeaveOneOut evaluation algorithms. This extension is designed for being extended by users that want to create their own evaluation algorithms, provides facilities for storing partial results and displays graphically the results.

References

- [1] A. Aamodt. Knowledge intensive case-based reasoning and sustained learning. In *Procs. 9th European Conference on Artificial Intelligence (ECAI-90)*, pages 1–6, 1990.
- [2] K. D. Ashley and V. Aleven. A Logical Representation for Relevance Criteria. In *EWCBR '93: Selected papers from the First European Workshop on Topics in CBR*, volume 837 of *LNCS*, pages 338–352, London, UK, 1994. Springer-Verlag.
- [3] R. Bergmann and A. Stahl. Similarity Measures for Object-Oriented Case Representations. In B. Smyth and P. Cunningham, editors, *Procs. of the 4th European Workshop on Advances in Case-Based Reasoning (EWCBR '98)*, volume 1488 of *LNCS*, pages 25–36. Springer-Verlag, 1998.
- [4] B. Díaz-Agudo and P. A. González-Calero. An Architecture for Knowledge Intensive CBR Systems. In E. Blanzieri and L. Portinale, editors, *Proceedings of the 5th European Workshop on Advances in Case-Based Reasoning (EWCBR '00)*, volume 1898 of *LNCS*, pages 37–48. Springer-Verlag, 2000.
- [5] B. Díaz-Agudo and P. A. González-Calero. Knowledge Intensive CBR through Ontologies. In B. Lees, editor, *Procs of the 6th UK CBR Workshop*. 2001.
- [6] B. Díaz-Agudo and P. A. González-Calero. *Ontologies in the Context of Information Systems*, chapter An ontological approach to develop Knowledge Intensive CBR systems, page 45. Springer-Verlag, 2006.
- [7] P. A. González-Calero, M. Gómez-Albarrán, and B. Díaz-Agudo. Applying DLs for Retrieval in Case-Based Reasoning. In *Procs. of the 1999 Description Logics Workshop (DL '99)*. Linkopings universitet, Sweden, 1999.
- [8] P. A. González-Calero, M. Gómez-Albarrán, and B. Díaz-Agudo. A Substitution-based Adaptation Model. In *Challenges for Case-Based Reasoning - Proc. of the ICCBR'99 Workshops*. Univ. of Kaiserslautern, 1999.
- [9] P. A. González-Calero, M. Gómez-Albarrán, and B. Díaz-Agudo. A Substitution-based Adaptation Model. In *Procs. of the ICCBR'99 Workshops*. University of Kaiserslautern, Germany, 1999.
- [10] G. Kamp. Using Description Logics for KI-CBR. In B. Faltings and I. Smith, editors, *Third European Workshop on Case-Based Reasoning (EWCBR'96)*, volume 1168 of *LNCS*, pages 204–218. Springer-Verlag, Switzerland, 1996.
- [11] J. Koehler. An Application of Terminological Logics to CBR. In J. Doyle, E. Sandewall, and P. Torasso, editors, *KR'94: Principles of Knowledge Representation and Reasoning*, pages 351–362. Morgan Kaufmann, 1994.
- [12] J. Kolodner. *Case-Based Reasoning*. Morgan Kaufmann, San Mateo, 1993.
- [13] A. Napoli, J. Lieber, and R. Courien. Classification-Based Problem Solving in CBR. In I. Smith and B. Faltings, editors, *Proceedings of the Third European Workshop on Advances in Case-Based Reasoning (EWCBR '96)*, volume 1168 of *LNCS*, pages 295–308. Springer-Verlag, 1996.
- [14] J. A. Recio, B. Díaz-Agudo, M. A. Gómez-Martín, and N. Wiratunga. Extending jCOLIBRI for textual CBR. In *Procs. of 6th International Conference on CBR, ICCBR 2005*, volume 3620 of *LNCS*, pages 421–435. Springer-Verlag, 2005.
- [15] S. Salotti and V. Ventos. Study and Formalization of a CBR System using a Description Logic. In *Advances in Case-Based Reasoning (EWCBR'98)*, volume 1488 of *LNCS*, pages 286–301. Springer-Verlag, 1998.