Science of Computer Programming 79 (2014) 126–145

Contents lists available at ScienceDirect

Science of Computer Programming

journal homepage: www.elsevier.com/locate/scico

jCOLIBRI2: A framework for building Case-based reasoning systems☆

Juan A. Recio-García *, Pedro A. González-Calero, Belén Díaz-Agudo

Facultad de Informática, Universidad Complutense de Madrid, Spain

ARTICLE INFO

Article history:
Received 24 January 2011
Received in revised form 9 April 2012
Accepted 10 April 2012
Available online 24 April 2012

Keywords:
Case-based reasoning
Framework
jCOLIBRI

ABSTRACT

This paper describes the jCOLIBRI2 framework for building Case-based reasoning (CBR) systems. CBR is a mature subfield of artificial intelligence based on the reuse of previous problem solutions – cases – to solve new ones. However, up until now, it lacked a reference toolkit for developing such systems. jCOLIBRI2 aims to become that toolkit and to foster the collaboration among research groups. This software is the result of the experience collected over several years of framework development and evolution. This experience is explained in the paper, together with a description of the specialized CBR tools that can be implemented with jCOLIBRI: CBR with textual cases, recommenders, knowledge/data intensive applications or distributed architectures.

1. Introduction

Case-based reasoning is a subfield of Artificial Intelligence rooted in the works of Roger Schank in the early 80s, on dynamic memory and the central role that the recall of earlier episodes (cases) and scripts (situation patterns) has in problem solving and learning [1]. In spite of its maturity as a field of research, in 2003 when we started to develop jCOLIBRI [2], there was no open source tool that would serve as a reference implementation for the common techniques and algorithms developed and refined in the CBR community over the years. Our goal was to fill this gap by developing an open source framework that could fulfil several goals: to provide a readily useable implementation of common techniques in CBR useful for academic environments; to foster collaboration among research groups by providing an extensible framework which their prototypes can use and extend; to provide a toolset for assembling a CBR system without even writing a line of code. This last goal consists of facilitating the experimentation with different configurations of a CBR system, and enabling the use of the framework for those with a good understanding of the CBR processes but reluctant to program in Java.

As reported by Díaz-Agudo et al. [3], in 2007, with jCOLIBRI1 we had been quite successful in fulfilling two of the three initial goals. The framework was becoming popular as a programming resource in graduate courses in Case-based reasoning, and high-level tools resulted in a significant improvement in the process of configuring CBR systems for research purposes. Nevertheless, the framework was barely attracting any contribution from other research groups. Some colleagues reported that the framework architecture was too biased towards high-level configuration tools, which imposed some artificial constraints on the design of the framework. For example, in order to facilitate the definition of the case structure through a GUI, jCOLIBRI1 included its own type system in parallel with the basic types included in Java. Using terminology from the software reuse community, we had built a successful black-box framework which failed to succeed as a white-box one. jCOLIBRI2 was conceived to solve this problem.

jCOLIBRI2 was designed from the bottom up. First we concentrated on developing a white-box framework that facilitated source code reuse and promotes the integration of a third-party code. Furthermore, by adopting standard Java technologies

☆ Supported by Spanish Ministry of Science & Education (TIN2009-13692-C03-03) and Madrid Education Council and UCM (Group 910494).
* Corresponding author. Tel.: +34 913947599; fax: +34 913947547.
E-mail addresses: jareciog@fdi.ucm.es (J.A. Recio-García), pedro@sip.ucm.es (P.A. González-Calero), belend@sip.ucm.es (B. Díaz-Agudo).

0167-6423/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.scico.2012.04.002

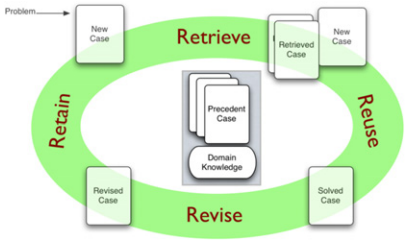

Fig. 1. The CBR Cycle (adapted from Aamodt and Plaza [6]).

for enterprise applications, we extended the scope of the framework which is now useful both for academic and real world applications.

This paper describes the architecture, components and features of jCOLIBRI2. Our goal is to outline the problems found in jCOLIBRI1 to help framework developers avoid them; consequently, to explain how that experience drove us to the new architecture of its successor. Once the architecture of the framework is described, we provide an overview of the components supplied for building CBR systems. Finally, we also explain the capabilities of the framework for building specialized CBR applications.

The rest of the paper runs as follows. The next section briefly introduces the main ideas of Case-based reasoning. Section 3 details lessons learnt from developing jCOLIBRI1 and describes the new architecture proposed for jCOLIBRI2 along with the main components in the framework. Section 4 describes further complements that extend the basic functionality provided by the framework. Section 5 explains how to obtain, install and use the jCOLIBRI2 framework. Section 6 describes some use cases of the framework in third-party large applications. Related work is presented in Section 7 and, finally, Section 8 concludes the paper and provides some hints about future work.

2. Case-based reasoning

Case-based reasoning is a paradigm for combining problem-solving and learning that has become one of the most successful applied subfields of artificial intelligence in recent years [4]. CBR is based on the intuition that problems tend to recur, so that new problems are often similar to previously encountered problems and, therefore, past solutions may be of use in the current situation [5]. When applied to classification problems, CBR, and more specifically instance-based learning, it is considered a lazy learning approach where instead of generating some kind of abstract representation of the set of training examples, it uses those training examples in the neighbourhood of the problem example (k Nearest Neighbours [5]) to determine its class.

Fig. 1 shows a high-level description of a generic CBR system as described by Aamodt and Plaza [6]. In its most general form a CBR system is composed of 4 consecutive processes that, given a collection of precedent cases, some kind of domain knowledge, and a problem to solve or an example to classify:

1. retrieve the most similar cases from the precedent ones;
2. reuse the knowledge in the cases retrieved to propose a solution or classification for the new one;
3. revise the solution proposed by some means external to the system; and
4. learn from the problem-solving or classification episode by retaining the new problem, along with its solution and revision.

Within this general schema we can identify different families of CBR systems, such as:

- Textual CBR, where the cases are given as text in natural language;
- Knowledge-intensive CBR, where a rich domain model is available and thus the system requires a smaller case base;
- Data-intensive CBR, where precedent cases are the main source of information with no domain knowledge available; or
- Distributed CBR, where different agents collaborate to reach conclusions based on their particular case bases.

jCOLIBRI2 includes source code components for supporting such a range of CBR approaches.

CBR, both as a problem-solving paradigm and machine learning approach, is best suited for complex domains where knowledge is scarce. As a problem-solving paradigm, CBR can be used in those domains where a deep domain model is not available but we have a collection of past problem-solving episodes. As a learning technique, CBR can be used in those domains where the number of available training examples, given the number of attributes used to describe the examples, is too small to obtain a useful generalization of the training examples by applying some non-lazy learning technique.

Its ability to cope with complex domains explains why CBR has been successfully applied to a number of real world problems such as diagnosis applications [7], planning [8], medical applications [9], law [10], e-learning [11], knowledge management [12], image processing [13], or recommender systems [14].

For a detailed description of these and other CBR-related aspects, we direct the interested reader to a number of CBR books [15], [5], the special issue of the Knowledge Engineering Review (vol. 20:3) dedicated to reviewing the state of the

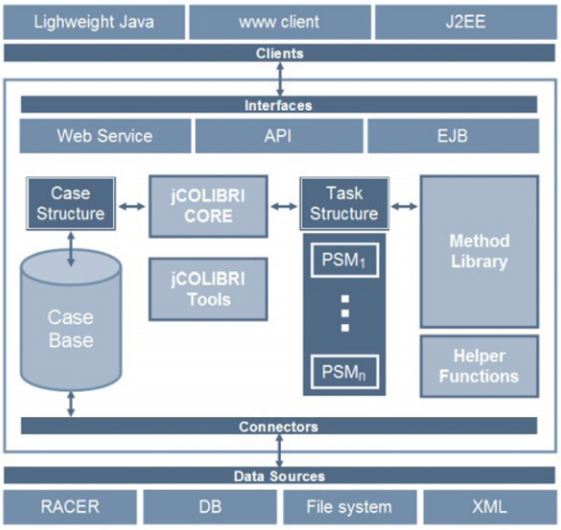

Fig. 2. jcolibri1 global architecture.

art in Case-based reasoning in 2006, and to the International Conference on Case-Based Reasoning,[1] as the most up-to-date source of information about the field.

3. The jCOLIBRI2 framework

The main goal of jcolibri2 is to provide a reference platform for developing CBR applications. With the term "platform" we mean that it defines a clear architecture to designing CBR systems plus a reference implementation of the components required to build them. This architecture has been designed to be extensible and reusable across different domains and CBR families. Although the main target of the framework is the research community, a relevant feature of jcolibri2 is its support for large-scale commercial applications. It means that our software can be used not only to do rapid prototyping of CBR systems but also to develop applications that will be deployed in real scenarios. This feature is exemplified by its straightforward integration into web environments. A listing of large-scale applications using jcolibri can be found on the web site.[2] Some examples are: an application to solve water stress problems in European countries [16], a decision support system for energy efficiency optimization in manufacturing companies in Europe [17], a CBR model oriented towards penal justice in Brazil [18], a software process model to encourage the software industry in Mexico [19], or an intelligent Web-based catalogue for the electronic presentation of cultural-historical heritage of Bulgaria [20].

jcolibri2 takes advantage of several years of experience from the first version released in 2005, called jcolibri1. Thus, the current architecture solves many problems found in its predecessor but reuses the design choices that proved to be good ideas. A complete description of jcolibri1 can be found in the first edition of these Special Issues on Experimental Software and Toolkits [3]. However, to understand the current architecture we will briefly outline its features and analyse the problems found after the exhaustive use of the framework by real developers.

3.1. Architectural features of jCOLIBRI1

From our point of view, the first version of the framework was really significant because it provided very valuable knowledge about the design of a CBR framework. This section describes its architecture to let us understand the problems found and later justify the current design of its successor jcolibri2. Recio-García et al. [21] thoroughly analyses this design experience.

jcolibri1 is aimed mainly at CBR system designers. A CBR application can be built by instantiating the framework, or through GUI-based configuration tools, which allow us to build the application without writing a line of code. Nevertheless, if we want to build a very complex CBR system or we need components – called methods – that are not available in the framework, then we can program new methods and incorporate them into the framework, thus contributing them for other CBR system designers to use.

The jcolibri1 system architecture, as depicted in Fig. 2, is organized around the following elements:

- *Run-time structuring of CBR applications*. jcolibri splits the behaviour of CBR systems into three stages. First, the precycle loads the resources required and pre-processes the knowledge used by the application. This step is executed only once.

1 http://www.iccbr.org/.

2 http://gaia.fdi.ucm.es/research/colibri/people-users-apps.

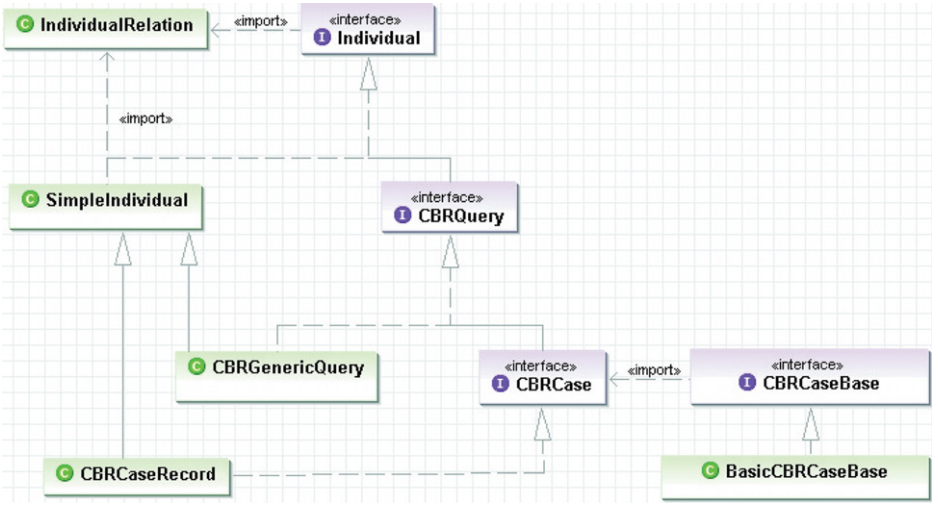

Fig. 3. Case representation in jCOLIBRI1.

Then the cycle is ready to be executed every time a query is received. It computes the CBR processes. Finally, the postcycle releases the loaded resources to shut down the system.

This design ended up being one of the most successful architectural decisions as was proved when developing future extensions like the textual CBR components. This kind of CBR systems requires an initial step to pre-process textual cases that is executed in the precycle stage.

- *Case base, data sources and connectors*. Cases – or experiences – can be stored using different media. Common media are data bases or plain text. However, there may be any other potential source of cases that can be exploited by the framework, such as ontologies, which are accessed through Description Logic reasoners like RACER [22]. Therefore, jCOLIBRI1 defines a family of components called *connectors* to load the cases from different media into the in-memory organization: the *case base*. There are several implementations of the in-memory case base using different data structures: linear lists, trees, hash maps, etc. This division into two layers enables an efficient management of cases, an issue that becomes more relevant as the size of the case base grows. This design based on connectors and in-memory storage of the case base is shown at the bottom of Fig. 2.
- *Case structure and similarity measures*. jCOLIBRI1 represents cases using a composite pattern of attributes containing other attributes as shown in Fig. 3. A case is just an individual that can have any number of relationships with other individuals (the attributes of the case). This representation, inspired by the Composite design pattern, allows for a uniform treatment of arbitrarily complex case structures as long as they conform to the *Individual* related through *IndividualRelation* to other *Individual*. Additionally, every individual includes its weight in the global case and a similarity function to compare itself with the same individual of another case.
- *Tasks structure and problem-solving methods (PSMs)*. To define the workflow of the application we used a decomposition schema rooted in the work of Problem-solving Methods [23]. Therefore, a CBR application is defined as a sequence of tasks that must be solved by a resolution or decomposition method. Examples of tasks are pre-process cases, obtain query, retrieve, reuse, revise, retain, and compute similarity.
- *Dataflow*. A critical part of the framework is the communication between methods. The dataflow is carried out by means of a blackboard organization where any method can read and write information. A blackboard structure – named *context* – is used by the methods to obtain the case base, interchange data and other configuration values or parameters. It is based on a hash table where every component can store any data labelled with an identifier. This way, if another component wants some data it needs the corresponding identifier to read the information. This organization, which is a critical part of the framework, has the main advantage of providing an open and simple communication mechanism, but also presents some drawbacks that will be discussed in the following sections.
- *Method library*. The method library includes the different components that are used to solve each task defined at the knowledge level (tasks structure). A CBR system designer will need to write Java code when the library does not include the required resolution method.
- *Core and tools*. The core of jCOLIBRI1 mixes up several elements: the definition of the architecture for CBR systems, the core components required to build and deploy the final CBR systems, and an application generator from the tasks & methods design. This application generator includes the tools to let developers build a new application by defining: the case structure, the connector configuration and the task structure. Therefore, there is a specific tool to graphically define each element and afterwards generate the source code of the application according to the settings configured by the user.
- *Interface*: The top layer of the system provides the façade to use a configured CBR application. As shown at the top of Fig. 2, jCOLIBRI1 supports standalone Java clients through its API. It also provides web compatibility through web

technologies like web services or Enterprise Java Beans (EJBs) used to wrap the cases and transfer them from the model to the presentation layer of the application.

3.1.1. Problems found in jCOLIBRI1

The architecture of jCOLIBRI1 presents several problems that were corrected in the following version of the platform and are described next.

In order to classify a framework we can distinguish several types [24]. A white-box framework is reused mostly by subclassing and a black-box framework is reused through parametrization. The usual development of a framework begins with a design as a white-box architecture that evolves into a black-box one. The resulting black-box framework (commonly) has a builder associated that will generate the application's code. A visual builder allows the software designer to visually compose the framework objects and activate them. jCOLIBRI1 is closer to a black-box framework with a visual builder and lacks a clear white-box structure. This is, in general, the origin of several drawbacks found in this version. The experience gained during the analysis of these problems was really useful in designing the architecture of jCOLIBRI2. As we will explain in Section 3.2 this new architecture is split into a clear white-box system oriented to programmers and a black-box with a builder layer that is oriented to designers.

The main drawback in jCOLIBRI1 is the difficulty in developing new applications by directly using the code without the user interface. Still, there were some other problems encountered that were useful for designing the architecture of jCOLIBRI2:

- *Cases.* The case structure mixes data and metadata (similarity, weight, etc). The reason is that the most common algorithm used in the retrieval stage of CBR applications is the k Nearest Neighbours (kNN). This algorithm is configured with similarity functions used to compare each attribute and the weight of each attribute. To support and facilitate the implementation of the visual composition of applications in jCOLIBRI1, we decided to include this metadata (similarity and weights) in the case representation. This additional data was very useful when using the kNN algorithm but was unnecessary when using other methods that were included in the framework afterwards.

  Another problem is that the Composite pattern (shown in Fig. 3) complicates the direct programming and debugging of the cases.
- *Metadata.* Besides the problem of the metadata contained in the case structure, there is also information that is stored in the configuration files or additional classes but which could be inferred from the Java code or runtime objects. For example, information about the variables read and stored by the methods is required by the high-level composition tools and is stored as static attributes in each method class.
- *Blackboard organization.* Blackboard organization is a simple and extensible way of communicating and developing new components. But, in general, blackboard-based applications are difficult to check for correctness because interaction between components is not precisely localized [25]. That was the case in jCOLIBRI1 when incorporating new methods (usually from third party developers) into the framework. The problem was rooted in the identification of the variables stored in the blackboard. For example, when a method $m2$ required a variable x stored in the blackboard by a previous method $m1$, $m2$ had to use the identifier of that variable found in the documentation of $m1$. This is a problem in an open environment where any developer can contribute with new methods and the number and names of the variables is completely open to these developers. A third-party developer could create a replacement method $m1'$ for $m1$ with equivalent behaviour but using different variable names (for example y instead of x). In this scenario we will find problems when replacing $m1$ with $m1'$ because of the name of the variables.
- *Inversion of control and method signature.* This problem is derived from the blackboard organization. Frameworks usually manage the control flow of the application and invoke the components developed by the user. This concept is also referred to as the "Hollywood Principle": "don't call us, we'll call you" [26]. This control flow is contrary to the design of the *component libraries* that offer several methods that are invoked by the developer.

  In a general way, jCOLIBRI1 is a framework with inversion of control because it invokes the methods following the control flow configured by the user in the precycle, cycle and postcycle through graphical tools. These tools generate the control flow and check that the variables required by every method were stored in the blackboard by a previous one. It means that the data flow between methods is also checked by these graphical tools. To do so, the signatures of the methods were designed to be integrated with those tools instead of providing a clear interface. As a consequence the signature of each method was just the *context* (blackboard) parameter. For example, the signature of the Nearest-Neighbour scoring method (which ranks cases according to their similarity to the query) is in jCOLIBRI1:

  ```
  public CBRContext execute(CBRContext context)
  ```

  Users had to find in the documentation that ranked cases are stored in the `_caseEvalList` variable of the `context`. However, in jCOLIBRI2 that method is defined as follows:

  ```
  Collection<RetrievalResult> evaluateSimilarity(
                                Collection<CBRCase> cases,
                                CBRQuery query,
                                NNConfig simConfig )
  ```

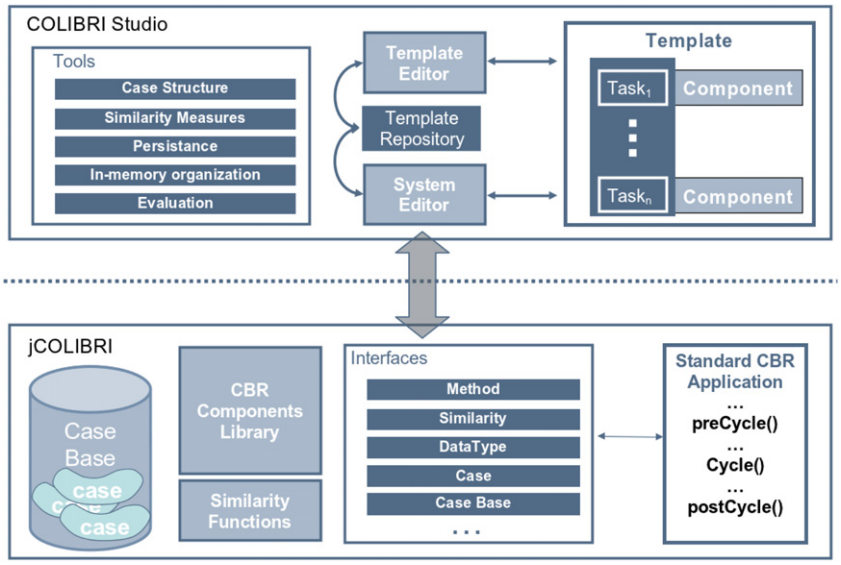

Fig. 4. The two layers architecture of jCOLIBRI2.

As we can see, it is much simpler to invoke the later signature from an external code. But the problem of the "clear" signature is that it complicates the generation of the application's code from the graphical tools as they have to "understand" the signatures to define the data flow.

This *mandatory* inversion of control limits the possibility of using the framework as a component library and represents an important drawback of this version.

- *Concurrency*. Blackboard can be also a problem when the application includes concurrent threads [27]. Normally a CBR application is not going to have several threads but some applications require concurrency (i.e. web applications).
- *Sequential execution of methods.* The jCOLIBRI1 design process based on task decompositions only allows developers to generate applications whose methods are executed in a sequential way. However, many times, applications require the inclusion of other constructions like conditions or loops.

The drawbacks found in the architecture of jCOLIBRI1 were very useful for the general evolution of the platform. Although these drawbacks could initially be seen as something negative, because they were problems with the framework, the conclusions obtained from their analysis directed the design of the architecture of jCOLIBRI2 as described next. This new architecture was born from the experience gained during the development and maintenance of jCOLIBRI1, and therefore is a much better design.

3.2. The new architecture for jCOLIBRI2

In the design of the jCOLIBRI2 framework there are many ideas that were inherited from the previous version because they proved to be successful from our development experience. Some examples are the common interface for connectors that unifies the loading of cases from different media or the organization of CBR applications in three stages (precycle, cycle and postcycle). These design decisions facilitated the incorporation of further components. The main drawback of the previous version was the coupling between methods and graphical interface because there is not a clear white-box architecture. Therefore, the main aim in jCOLIBRI2 was to design a good white-box framework that could be complemented afterwards by composition tools. As jCOLIBRI2 is a white-box tool it is oriented to programmer users. However, the new version contemplates a second layer over the white-box framework that includes and defines the composition tools to support designer users.

This new organization in two layers is referred to as the COLIBRI platform and is illustrated in Fig. 4. jCOLIBRI2 represents the white-box framework at the bottom level of the platform, while the composition tools are detached into the top level. These tools – named *COLIBRI Studio* – are under development and will be similar to the ones provided by jCOLIBRI1. The most remarkable difference is that the new architecture with two layers solves the problems of coupling between source code (white-box) and composition tools (black-box) as described next. Because the purpose of this paper is to describe the white-box framework – that is, jCOLIBRI2 – the composition tools are set aside although described as future work in Section 8.

Fig. 5 shows a UML diagram with the main elements of the new jCOLIBRI2 framework:

- Organization into persistence, core and presentation layers. It follows the structure of the previous version. However, these layers are implemented in a different way through J2EE[3] technologies. Persistence is again managed by *connectors*

[3] *Java 2 Enterprise Edition* (J2EE) is the Java platform oriented to enterprise applications http://www.oracle.com/technetwork/java/javaee/.

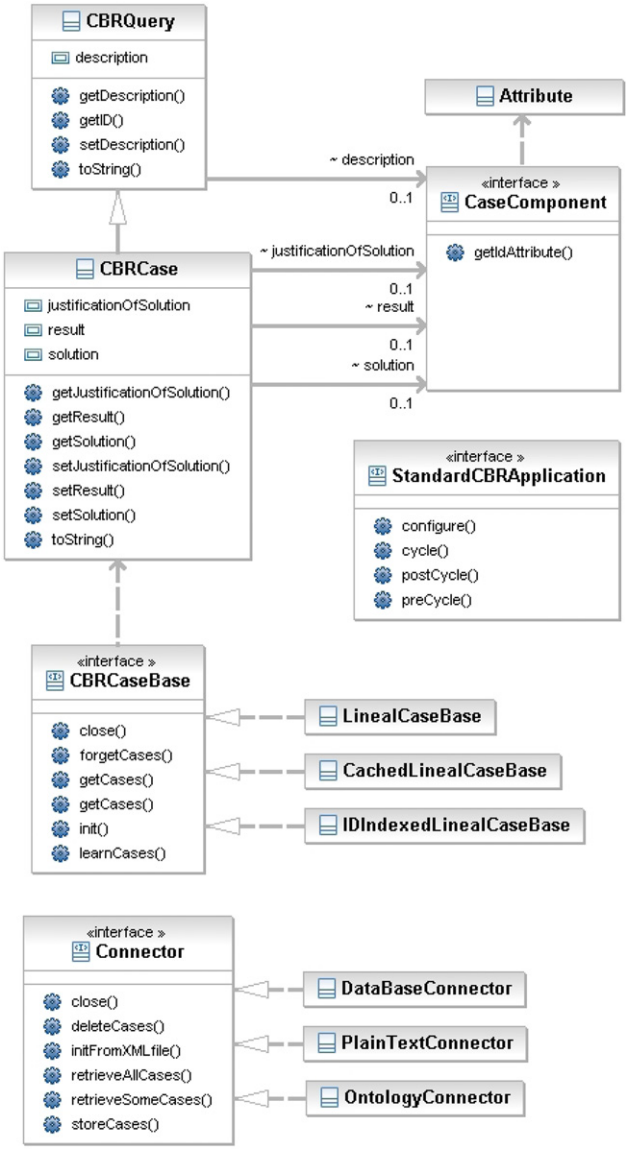

Fig. 5. UML diagram with main elements of jCOLIBRI2.

(`Connector` in Fig. 5) that access the persistence media and load the cases into different *in-memory organizations* (`CBRCaseBase` in Fig. 5). The core contains the same basic classes that were defined in the previous version, but there are no presentation methods to keep the core independent of the user interface.

- Organization of the applications into precycle, cycle and postcycle. This organization is defined by the `StandardCBRApplication` interface shown in Fig. 5. Moreover, new stages can be defined to be executed at different execution points. This feature enables the development of maintenance or evaluation procedures. For example, maintenance algorithms are usually executed each *n* cycles.
- Case structure. The structure of cases is organized in: *description*, *solution*, *result* and *justification of solution* after a revision of several works found in CBR literature [15,28]. This structure is defined by the `CBRQuery`, `CBRCase` classes shown in Fig. 5.

The new architecture of jCOLIBRI2 solves the problems found in the first version that were presented in Section 3.1.1. The following list summarizes how these problems are addressed in the second version of the framework:

- Problems managing cases. The most important change in the architecture of jCOLIBRI2 is the representation of cases. This new representation modifies the way of storing cases in the persistence media and the behaviour of the methods when accessing their attributes.

jCOLIBRI2 represents the cases using *Java Beans*. A Java Bean is any class that has a `get()` and a `set()` method for each public attribute.[4] Its modification and management can be performed automatically by using the introspection capabilities of the Java language. This way, both the framework and the developer can read and modify any attribute of the case. The complexity of introspection processes is hidden thanks to the `Attribute` class shown in Fig. 5, which offers a simple management of every attribute in the case.

Using Java Beans in jCOLIBRI2, developers can design their cases as normal Java classes, choosing the most natural design. These classes are referred to as *case components* because they must implement the `CaseComponent` interface shown in Fig. 5. This interface only imposes the definition of an attribute as the identifier of the component. It is used by the connectors and methods as a kind of primary key to identify them.

This representation simplifies programming and debugging CBR applications, and configuration files became simpler because most of the metadata of the cases can be extracted using introspection. Java Beans also offer automatically generated user interfaces that allow the modification of their attributes, and direct persistence into data bases and XML files. It is important to note that many Java web applications use Java Beans as a base technology, so the development of web interfaces is very straightforward. Moreover, Hibernate[5] – the library used to develop the data base connector in jCOLIBRI2 – uses Java Beans to store information in a data base. Java Beans and Hibernate are core technologies in the *J2EE* platform. By using these technologies in jCOLIBRI2 we guarantee the possibility of integrating CBR applications developed using this framework into large scale commercial systems. Recio-García et al. [29] provides an extended description of case representation in the framework.

- Metadata. The redundant metadata required by jCOLIBRI1 was necessary to support the graphical composition tools. In jCOLIBRI2 this problem is solved by leaving the blackboard organization and defining clear signatures for the methods where each input and output parameter is clearly defined. Therefore, the graphical tools (on the top layer) will be able to analyse these signatures through introspection techniques and check the correctness of the composition.

  This way, both layers are completely independent and methods do not comply any restriction to support the composition tools.
- Blackboard organization. This restriction disappears when developing new methods with an explicit signature. As we explained when exposing this problem in Section 3.1.1, methods in jCOLIBRI2 can receive any number of parameters and their signature is clear and easy to understand. Although this feature is counter-productive for the automatic composition of methods, it enormously simplifies their manual composition. However, as we explained before, new complexity for composition tools is solved by means of introspection mechanisms.
- Inversion of control. Now, methods do not receive the context (blackboard) but define the input and output parameters. Therefore they can be used as a *component library* where the control flow is defined elsewhere and invokes the methods. This external definition of the control flow must also define the data flow between the inputs and outputs of the methods. This is what developers do when writing code, but represents a complex task for composition tools.
- Concurrency. This problem disappears as the blackboard is no longer used.
- The problem with the sequential execution of methods is directly solved as there are no composition tools in this bottom layer. jCOLIBRI2 is a pure white-box framework and, therefore, developers can define their custom execution flows by means of the Java language (if, while,...).

As we have explained, most of the problems found in the first version are solved by means of the new architecture with two layers where the white-box framework defines clear signatures for the methods while setting aside the blackboard organization. However, this modification complicates the automatic composition of these methods by the high level tools in the top layer. We could say that we have moved the complexity of composing methods from the white-box to the black-box.

To address this problem, the composition tools will analyse through introspection the methods provided by the jCOLIBRI2 framework (the white-box) to "understand" their signature. They will create a "semantic representation" of the methods according to the standards used in the Semantic Web Services research (like OWL-S [30] or WSML [31]) that can be used to check the correctness of the systems generated. Moreover, as these standards include conditionals and loops in their composition languages, we can solve the problem of sequential composition of methods found in jCOLIBRI1. A discussion about the features of the top layer is presented in Section 8.

Now that we have discussed the major changes in the architecture, we can detail the functionality included in the white-box framework. The wide spectrum of methods provided by jCOLIBRI2 is presented in the following section. It summarizes the generic methods for standard CBR systems, whereas specialized methods are described in Section 4.

3.3. Components provided by jCOLIBRI2

It is important to clarify that with the term *component* we refer to any piece of encapsulated functionality. *Methods* are those components that implement algorithms that make up CBR reasoning. And, finally, we use the term *tool* for components involving graphical interfaces. The components of jCOLIBRI2 can be organized according to the distribution shown in Fig. 6.

[4] A Java Bean class must follow additional conventions like a nullable constructor, `is()` accessors for Boolean attributes and should be serializable.

[5] http://www.hibernate.org.

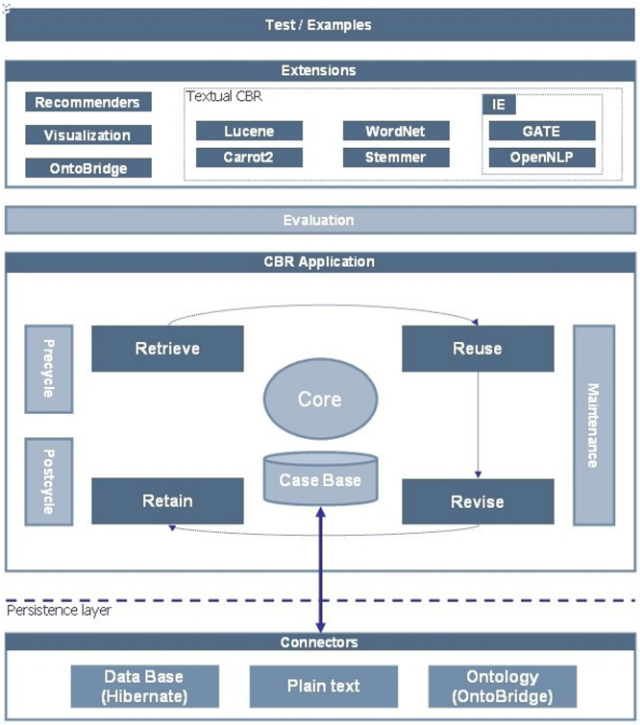

Fig. 6. New architecture of jcolibri2.

This figure is the entrance point to the framework documentation on the web page[6] and allows us to explore the different components provided:

- Persistence layer. Includes the connectors to store the cases in the persistence layer.
- Core. This package contains the basic classes and interfaces of the framework: `Connector`, `CBRCaseBase`, `CBRQuery`, `CBRCase`, ...
- Case base. Defines several in-memory organizations of the case base.
- Retrieval methods. The most important retrieval method is Nearest Neighbour scoring. It uses global similarity functions to compare compound attributes and local similarity functions in order to compare simple attributes. Although this method is very popular, there are other methods that are also included in the framework. For example, we implement Expert Clerk Median scoring from Shimazu [32] and a filtering method that selects cases according to Boolean conditions on the attributes. Both methods belong to the recommender's field and will be explained in Section 4.2. In the textual CBR field we also find specialized methods using several Information Retrieval or Information Extraction libraries (Apache Lucene, GATE, ...) that will be detailed in Section 4.1. Regarding knowledge intensive CBR we enable retrieval from ontologies (Section 4.3). Data intensive retrieval is addressed by means of clustering algorithms and a connector for the Weka ARFF format (Section 4.4). And, finally, jcolibri2 provides the infrastructure required to retrieve cases in a distributed architecture of agents (Section 4.5).

  Once cases are retrieved, the best ones are selected. The framework includes methods to select the best scored cases, but also the best cases that are also diverse. Most of these methods belong to the recommender domain, therefore they are detailed in Section 4.2.
- Reuse and revision methods. These two stages are coupled to the specific domain of the application, so jcolibri2 only includes simple methods to copy the solution from the case to the query, to copy only some attributes, or to compute direct proportions between the description and solution attributes. There are also specialized methods to adapt cases using ontologies that will be explained in Section 4.3.
- Retain. Includes methods in charge of adding new cases to the case base. There are strategies to reflect the changes to the persistence media or just modify the in-memory storage. This kind of behaviours are required to use the framework's evaluation capabilities.
- Evaluation tools measure the performance of a CBR application. jcolibri2 includes the following strategies: *Hold Out*, *Leave One Out* and *N-Fold*. These tools are shown in Fig. 7. A detailed explanation of these evaluation tools is presented by Recio-García et al. [29]. They automatically generate a chart showing the performance of the application. A detailed

6 http://gaia.fdi.ucm.es/research/colibri/jcolibri/documentation.

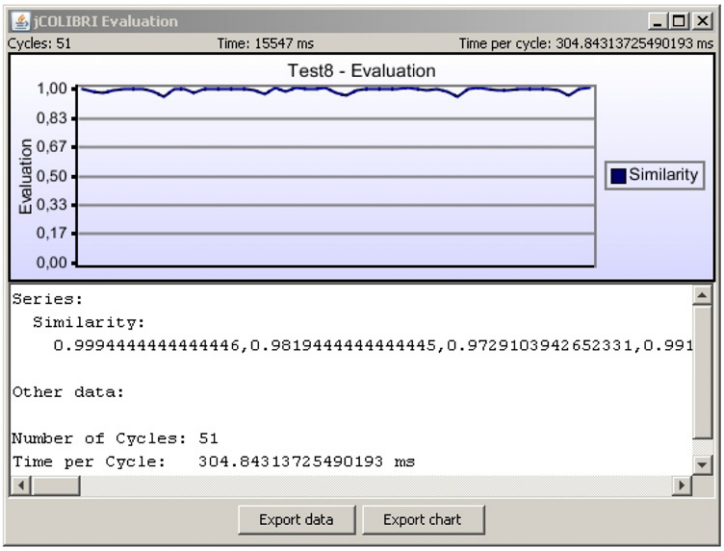

Fig. 7. Screenshot of the evaluation tool [29].

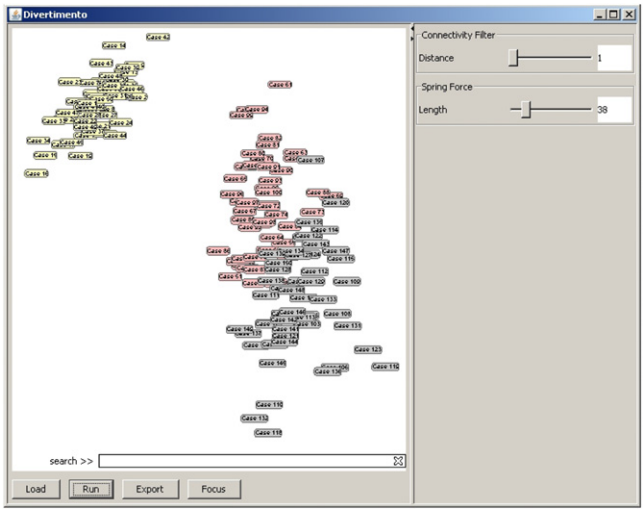

Fig. 8. Screenshot of the visualization tool [29].

log is displayed below, including information about the number of cycles executed during the evaluation, time per cycle, etc. Both chart and log can be exported to jpg and csv formats respectively.

- Maintenance. These methods allow developers to reduce the size of the case base.[7] Components provided are: BBNR (Blame-based noise reduction) and CRR (Conservative Redundancy Removal) [33], RENN (Repeated Edited Nearest Neighbour) [34], RC (Relative Cover) [35], or ICF (Iterative Case Filtering) [36].
- Visualization. This tool lays out the distance between cases.[8] This tool serves to debug the fitness of the similarity measure and is shown in Fig. 8. It assigns a colour to each type of solution and lays out the cases according to their similarity.
- Examples. The examples included in jCOLIBRI2 are useful in testing the framework and are part of its documentation.

Summarizing, jCOLIBRI2 offers 5 different retrieval strategies with 7 selection methods and provides more than 30 similarity metrics. It provides around 20 adaptation and maintenance methods plus several extra tools like system evaluation or the visualization of the case base.

Now that we have described the architecture and functionality provided by the framework, the following section details how to build specialized CBR systems.

7 Maintenance components were contributed to the library by Lisa Cummins and Derek Bridge from the University College of Cork, Ireland.

8 The visualization facility was contributed by Josep Lluis Arcos (Artificial Intelligence Research Institute, Spain).

4. Building specialized CBR applications

The previous section has described the building blocks of the framework and has been focused in the structure and organization of the applications. Up to now, we have mostly explained how to structure the applications, represent the cases and load them into memory. However, the services – functionality – provided by jCOLIBRI2 has just been outlined. These services are represented in the Retrieve, Reuse, Revise and Retain boxes of Fig. 6. jCOLIBRI2 includes generic methods that provide the basic services required for building simple CBR systems. Some examples are the methods to load cases, run the nearest-neighbour scoring or select the most similar cases. However, developers cannot create advanced applications using only these methods. To support the development of such advanced applications, jCOLIBRI2 includes several components that extend the core services of the framework.

These components, delivered as packages named *extensions*, offer additional services and behaviour beyond basic CBR processes: they serve to implement specialized CBR applications. The most relevant extensions are included in the main distribution but others can be obtained separately at the web site.[9] This web site also provides *contributions*, i.e. extensions developed by third-party research teams. Extensions include several examples to let developers understand the services provided. Then, the API documentation details how to use and integrate each component into an existing development. Next, we will describe the most relevant extensions provided by jCOLIBRI2 to offer an overview of the framework's capabilities.

4.1. Textual CBR applications

Textual Case-based reasoning (TCBR) is a subfield of CBR concerned with research and implementation on Case-based reasoners where some or all of the knowledge sources are available in textual format. It aims at using these textual knowledge sources in an automated or semi-automated way to support problem-solving through case comparison [37].

Although it is difficult to establish a common functionality for TCBR systems, several researchers have attempted to define the different requirements for TCBR [38,37]: how to assess similarity between textually represented cases, how to map from texts to structured case representations, how to adapt textual cases, and how to automatically generate representations for TCBR.

The textual CBR extension of jCOLIBRI2 provides methods to address some of these requirements using different strategies. These sets of methods are organized according to:

1. A library of CBR methods solving the tasks defined by the Lenz layered model [39].[10] The goal of these layers is to extract the information from the text into a structured representation that can be managed by a typical CBR application. This way, the CBR application can perform a retrieval algorithm based on the similarity of the extracted features and use them to adapt the text to the query.

   These layers are typical processes in Information Extraction (IE) systems: stemming, stop words removal, Part-of-Speech tagging, text extraction, etc. jCOLIBRI2 includes several implementations of these theoretical layers. A first group of methods uses the Maximum Entropy algorithms provided in the OpenNLP package.[11] The second implementation uses the popular GATE library for text processing.[12]

   See Recio-García et al. [40] and Recio-García et al. [41] for examples in the use of textual methods in different domains.
2. A natural language interaction module that facilitates querying CBR systems by using written natural language. Information Extraction techniques and Description Logics (DLs) reasoning [42] play a fundamental role in this interaction module, where it analyses a textual query from the user to generate a structured representation using the relevant information from the text. The details and an evaluation of this module are presented by Díaz-Agudo et al. [43].
3. There is another group of textual CBR methods that can be used when cases are texts without a fixed structure [44]. These methods are based on Information Retrieval (IR)+clustering techniques where reuse and retrieval are performed in an interleaved way. This third approach is clearly different from approaches 1 and 2, which are based on IE techniques to "capture the meaning", i.e, to engineer structured case representations.

   This set of textual methods belongs to the statistical approach that has given such good results in the IR field. jCOLIBRI2 methods are based on the Apache Lucene search engine [45]. Lucene uses a combination of the Vector Space Model (VSM) of IR and the Boolean model to determine how relevant a given document is to a user's query.

   The main advantages of this search method are its positive results and its applicability to non-structured texts. The big drawback is the lack of knowledge regarding the semantics of the texts. The details about this set of methods are presented by Recio-García et al. [44]

For interested readers, Recio-García et al. [29] present an extended tutorial on textual CBR with jCOLIBRI.

9 http://gaia.fdi.ucm.es/research/colibri/jcolibri/contributions.

10 This extension was developed in collaboration with Nirmalie Wiratunga from The Robert Gordon University, U.K.

11 http://opennlp.sourceforge.net.

12 http://gate.ac.uk.

4.2. Recommender systems

CBR has played a key role in the development of several classes of recommender systems [46,47]. The jCOLIBRI2 extension for building recommender systems is based in part on the conceptual framework described in the review paper by Bridge et al. [46].[13] The framework distinguishes between collaborative and Case-based, reactive and proactive, single-shot and conversational, and asking and proposing. Within this framework, the authors review a selection of papers from the Case-based recommender systems literature, covering the development of these systems over the last ten years (we could cite Wilke et al. [48], Bergmann [49], Burke [50], Shimazu [32] and McSherry [51] as illustrative examples). Based on this revision jCOLIBRI2 includes the following set of methods:

- *Methods for retrieval.* Different approaches to obtain and rank items to be presented to the user: typical similarity-based retrieval [48], filter-based retrieval based on Boolean conditions, ExpertClerk's median method [32] and collaborative retrieval method based on users' ratings to predict candidate items [52,53].
- *Methods for case selection.* Approaches to select one or more items from the set of items returned by the retrieval method: (1) Select all or k cases from the retrieval set. (2) The Compromise-driven selection [54] method chooses cases according to a number of attributes compatible with the user's preferences. (3) Finally, the Greedy Selection [55] method considers both similarity and diversity (inverse of similarity).
- *Methods for navigation by asking.* Navigation by asking is a conversational strategy where the user is repeatedly asked about attributes until the query is good enough to retrieve relevant items. There are different methods based on heuristics to select the next best attribute to ask about. For example, Information Gain, which returns the attribute with the greatest information gain in a set of items [56,49], and Similarity Influence Measure [49], which selects the attribute that has the highest influence on Nearest Neighbour similarity.
- *Methods for navigation by proposing.* The navigation by proposing strategy asks the user to select and critique one of the items recommended. The selected item is modified according to the critique and produces a new query. There are different strategies to modify the query as enumerated by Smyth and McGinty [57]. The More Like This (MLT) strategy replaces the current query with the description of the selected case. Partial More Like This (pMLT) strategy partially replaces the current query with the description of the selected case. It only transfers a feature value from the selected case if none of the rejected cases have the same feature value. Another option is to use MLT but weighting the attributes (Weighted More Like This, wMLT). Less Like This (LLT) is a simple one: if all the rejected cases have the same feature-value combination, which is different from the preferred case, then this combination can be added as a negative condition. Finally, More + Less Like This (M+LLT) combines both More Like This and Less Like This.

There are other methods to display item lists, make critiques, display attribute questions, manage user profiles, etc. Details are provided by Recio-García et al. [58] and the jCOLIBRI2 code examples.

4.3. Knowledge intensive CBR

Our research group[14] has been working for more than a decade on knowledge intensive CBR (KI-CBR) using ontologies [59–62]. Commonly, KI-CBR is appropriate when developers do not have enough experiences available but there is a considerable amount of knowledge of the domain. Recio-García et al. [29] provide further details about the implementation of KI-CBR applications in jCOLIBRI.

We state that the formalization of ontologies is useful for the CBR community regarding different purposes, and therefore, jCOLIBRI2 supports the following services:

1. Persistence: jCOLIBRI2 provides a connector that loads cases represented as concepts or individuals in an ontology (see Fig. 6). This connector delegates to our library for managing ontologies named *OntoBridge*.
2. Definition of the case structure through elements in ontologies. This extension provides a specialized data type used to represent attributes of the case structure that point to elements in an ontology. For example, an attribute *city* used in the representation of a case can be linked to the concept *City* in an ontology. This way, cases following that structure will store the values *Madrid, London, N.Y., Tokyo* in the attribute *city* because they are the individuals belonging to the concept *City* in the ontology.

   This approach that links case attributes to elements from an ontology can be used either if the cases are embedded as individuals in the ontology itself, or if the cases are stored in a different persistence medium, such as a data base, but some attributes contain values from the ontology.
3. Retrieval and similarity. There are different strategies to compute the local similarity based on ontologies [63–65]. Following the previous example, a more elaborate ontology will classify cities according to continents. Therefore the concept *city* will be specialized by the subconcepts *EuropeanCity*, *AmericanCity*, *AsianCity*, ... the individuals being organized consequently. This way we can use this hierarchy to compute similarity according to the distance between

[13] The recommenders extension has been developed in collaboration with Derek Bridge from University College Cork, Ireland.

[14] http://gaia.fdi.ucm.es/.

individuals. In this case, the similarity between *Madrid* and *London* will be higher than *Madrid* and *Tokyo* because *Madrid* and *London* belong to the same subconcept. This approach to compute similarity based on the distances – named *concept-based similarity* – can be performed in different ways [63], and jCOLIBRI2 provides the implementation of these similarity metrics to developers.

4. Adaptation. The usage of ontologies is especially interesting for case adaptation [59], as they facilitate the definition of domain-independent, but knowledge-rich adaptation methods. For example, imagine that we need to modify the *city* attribute of a retrieved case because the current value *Madrid* is not compatible with the restrictions of the query. According to the ontology, the best substitute is *London* because it is also a *EuropeanCity*. Here again we use the structure of the ontology to adapt the case. Because this schema is based on distances within the ontology, jCOLIBRI2 offers several domain-independent methods to perform this kind of adaptation. These methods only need to be configured with the ontology (which contains the domain knowledge).

   See González-Calero et al. [59], Recio-García et al. [66], Díaz-Agudo et al. [67] for a detailed description of different adaptation proposals.
5. Learning [68]. As ontologies are used as a persistence media, ontologies can be reused to store the experiences learnt. This is performed by means of the connector able to manage ontologies.

Now that we have explained the main methods in jCOLIBRI2 for KI-CBR we can move on to an opposite family of CBR systems where many cases are used instead of using few but rich ones.

4.4. Data intensive CBR

One of the problems to solve when dealing with real world problems is the efficient retrieval of cases when the case base is huge and/or it contains uncertainty and partial knowledge. There are many examples of domains and applications where a huge amount of data arises; for example, image processing, personal records, recommender systems, textual sources, and many others. Many authors have focused on proposing case memory organizations to improve retrieval performance. For example, there are different proposals to manage huge case memories organized in clusters such as the ones by Vernet and Golobardes [69] and Fornells et al. [70]. However none of the existing tools has incorporated capabilities to efficiently manage large case bases. jCOLIBRI2 provides an extension called *Thunder* to address this issue. Thunder[15] allows CBR experts to manage case memories organized in clusters and incorporates a case memory organization model based on Self-Organizing Maps (SOM) [71] as the clustering technique.

Clustering is implemented by grouping cases according to their similarities and representing each one of these groups by prototypes. Thus, the retrieve phase carries out a selective retrieval focused on using only the subset of cases potentially similar to the new case to solve. The new case retrieval procedure consists of (1) selecting the most suitable group of cases by comparing the input case with the prototypes and, (2) comparing the new input case with the cases from the selected clusters. The benefits of such an approach are both the reduction of computational time and improved robustness with uncertain data. Nevertheless, some open issues remain such as to what extent the accuracy rate is degraded due to the cluster-based retrieval, and furthermore how many clusters and cases should be used according to given requirements of computational time and accuracy degradation.

To support a uniform way for loading these large case bases, the Thunder extension provides a connector compatible with the ARFF format. This format is a standard defined by the popular Weka[16] toolkit for data mining and machine learning [72]. jCOLIBRI2 includes a graphical interface to test the components provided as shown in Fig. 9. We point interested readers to the works by Fornells et al. [73] for further details.

Next we describe the last extension of jCOLIBRI2: an infrastructure for developing distributed CBR systems.

4.5. Distributed CBR

Research efforts in the area of *distributed CBR* concentrate on the distribution of resources within CBR architectures and study how it is beneficial in a variety of application contexts. In contrast to single-agent CBR systems, multi-agent systems distribute the case base itself and/or some aspects of the reasoning among several agents. Plaza and Mcginty [74] categorized the research efforts in the area of distributed CBR using two criteria: (1) how knowledge is organized/managed within the system (i.e. single vs. multiple case bases), and (2) how knowledge is processed by the system (i.e. single vs. multiple processing agents).

Much of the work in distributed CBR assumes multi-case base architectures involving multiple processing agents that differ in their problem solving experiences [75]. The "ensemble effect" [76] shows that a collection of agents with uncorrelated case bases improves the accuracy of any individual. Multiple sources of experience exist when several CBR agents need to coordinate, collaborate, and communicate. Within this purpose jCOLIBRI2 provides two extensions to design

15 Thunder was developed in collaboration with Albert Fornells, from Universitat Ramon Llull, Spain. Available at: http://gaia.fdi.ucm.es/research/colibri/jcolibri/contributions.

16 http://www.cs.waikato.ac.nz/ml/weka/.

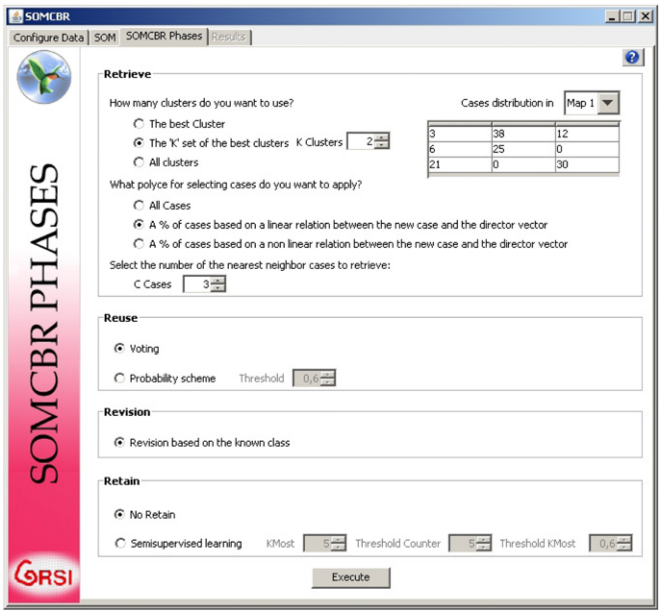

Fig. 9. jcolibri2 test GUI for the data intensive extension.

deliberative and distributed multi-agent CBR systems where the case base itself and/or some aspects of the reasoning process are distributed among several agents. A deliberative system can predict the future state that will result from the application of intentional actions. These predicted future states can be used to choose between different possible courses of actions in an attempt to achieve system goals [77]. Our work focuses on distributed *retrieval* processes working in a network of collaborating CBR systems.

The basic extension to support distributed CBR applications in jcolibri2 is called *ALADIN* (Abstract Layer for Distributed Infrastructures). This layer defines the main components of every distributed CBR system: agents, directory, messages, etc. and could be implemented by using different alternatives: JADE,[17] sockets, shared memory, ... It was defined after reviewing the existing literature on distributed CBR and is mostly compatible with IEEE FIPA[18] standards for multi-agent systems. Because ALADIN is only composed of interfaces that define the behaviour of the system, we have developed an implementation of this abstract layer using standard network sockets. This extension is called *SALADIN* (Sockets implementation of ALADIN[19]) and provides a fully functional multi-agent environment for building distributed CBR systems that can be particularized in many ways. For further details and a description of a case study we refer the interested reader to the paper by Recio-García et al. [78].

5. Using jCOLIBRI2

This section explains how to obtain, install and use the jcolibri2 framework. It can be downloaded from the web page: http://www.jcolibri.net. It can be used in any platform with the Java runtime environment.[20] jcolibri2 is distributed under the terms of the GNU Lesser General Public License (LGPL).[21]

The Windows installer provides a step-by-step wizard that will guide users during the installation process. Once the framework is installed, users can launch an application (explained next) including several code examples through the start menu. These shortcuts link to the files located in the installation directory. Linux and MacOS users just have to unzip the jcolibri2 file into any folder. Several shell scripts are provided to launch the same applications as under Windows.

There are three sources of documentation for jcolibri2. Firstly, there is a complete tutorial available at the web site[22] that describes the installation of the framework and explains how to create a complete CBR application. The code of this application is also included in the installation. Secondly, there are several code examples that exemplify how to use the different features of the framework. These 30 test applications are run by means of an "examples launcher" application,

17 JADE is a framework for building multi-agent systems following the FIPA specifications. It is available at: http://jade.tilab.com/.

18 Foundation for Intelligent Physical Agents. http://www.fipa.org/.

19 ALADIN and SALADIN extensions are available at: http://gaia.fdi.ucm.es/research/colibri/jcolibri/contributions.

20 Windows, Linux and MacOS platforms have been tested.

21 http://www.gnu.org/licenses/lgpl.html.

22 http://gaia.fdi.ucm.es/research/colibri/jcolibri/documentation.

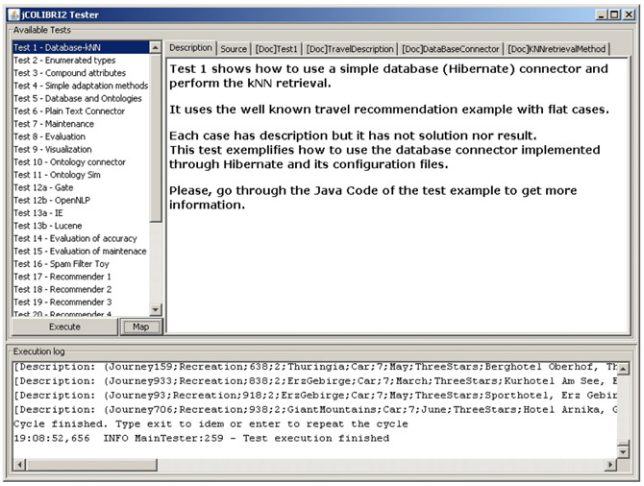

Fig. 10. jCOLIBRI2 examples launcher.

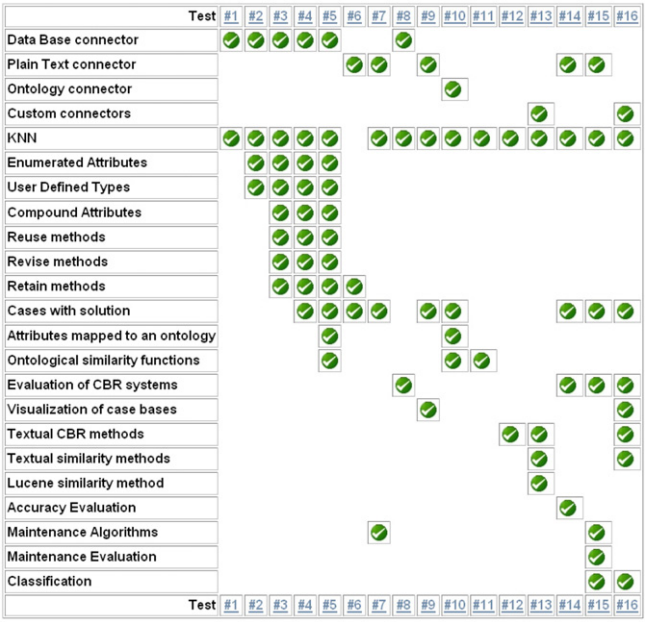

Fig. 11. jCOLIBRI2 examples table.

shown in Fig. 10, that provides access to the source code and the API documentation of the most significant components in the example. This API is the third source of documentation included in jCOLIBRI2.

Moreover, documentation is summarized in the *Code Examples* section of the jCOLIBRI2 web page[23] This section provides a table that summarizes which features are explained by each test (see Fig. 11). This table is also shown through the "examples launcher" application.

6. The framework in use

To illustrate the usefulness of the framework we have summarized some of the applications, both in the industry and the academia that take advantage of the framework and its extensions. The complete list of applications and their references to jCOLIBRI is available on the project web page [24]

We have described the use of 20 applications from different domains, countries and scopes. Some of them are outlined here to illustrate the capabilities of the framework. Aquastress is a tool to mitigate water stress problems in Europe

23 http://gaia.fdi.ucm.es/research/colibri/jcolibri/examples.

24 http://gaia.fdi.ucm.es/research/colibri/people-users-apps.

developed by a consortium of companies and universities under a European Project [16]. They point out the out-of-the-box support of jCOLIBRI for building CBR systems and its compatibility with web applications. Another application developed under a European project that integrates jCOLIBRI into a web environment is Ami-Moses. This is a web system for energy efficiency optimization in manufacturing companies [17]. Another relevant system using jCOLIBRI for developing standard CBR systems is DEEP, from the US Air Force Research Laboratory, which utilizes past experiences to suggest courses of action for new situations [79].

Regarding applications using the extensions provided by jCOLIBRI we find an application for assisting criminal justice [18] that uses the textual capabilities of the framework (see Section 4.1). There is also a recommender system (see Section 4.2) developed by the (European) ONE Project that supports users during negotiation processes with recommendations on how best to conduct a given negotiation [80]. Many of the applications use the knowledge intensive capabilities described in Section 4.3 and point out the facilities to incorporate ontologies and semantic resources in their CBR applications. Some examples are a digital library management system [81], a Web-based catalogue for the electronic presentation of Bulgaria's cultural-historical heritage [20], or a medical application for classifying breast cancer [82].

After collecting the list of applications we asked developers to report their comments and opinions on the framework.[25] According to our users jCOLIBRI enhances other CBR frameworks in several aspects: availability (open source framework), implementation (the Java implementation in J2EE environment) and the GUI (the graphical tools provided facilitate system design).

6.1. Lessons learnt

Setting aside the point of view of the users, this section summarizes some "lessons learnt" from our experience in the development of the framework that could be useful for the design of similar tools.

- Who are the users? Tool designers must carefully identify the potential users of the tool. Different users have different requirements, and therefore, the design of the tool is going to be influenced by the activities of each role. For example, in jCOLIBRI1 we did not clearly identify the two main target roles: developers and designers. These two roles require different tools (i.e. a black- or white-box framework) and it was a problem in jCOLIBRI1 that had to be solved in jCOLIBRI2 with the two-layer architecture.
- Who defines the control flow of the application? The tool can define the control flow of the application (inversion of control) or it can be left up to the developer. For example, in jCOLIBRI the control flow is defined by the framework (as there is a precycle, cycle and postcycle). In this case, the tool imposes the control flow on the developer. This requirement is generated by means of a template pattern where the developer must fill/implement the hooks defined by the tool designer. In the opposite case, it is the developer who defines the control flow of the application and invokes the framework's methods/components. This kind of tool can be identified as libraries that offer a collection of components that can be used by the guest application.
- How to extend the tool? When designing a framework, designers are supposed to anticipate any kind of application that could potentially be implemented and create software artefacts to support its development. Obviously this is an almost impossible task, and for that reason, framework designers must always leave an open door to extend the published components. This is specially significant when defining the interfaces of the classes/components in the framework. These signatures are imposed on future components so they must be extensible or at least provide a mechanism to do so. Of course, developers can use inheritance to extend the functionality of a component, but the existing components of the framework will use the parent signature as it was originally designed. For example, `CaseComponents` in jCOLIBRI2 are not serializable[26] and that has been an inconvenience for some users. The simplest solution is to add the `Serializable` interface to this component, but this modification is a problem when there are many classes already using the original signature of `CaseComponent`. We cannot simply add a new interface to that class because we would be breaking our API. As it is a building block of the framework it cannot be modified without a major impact on the remaining components. In general, this is the problem with adding new features to an existing class that has already been released. This problem is still present in jCOLIBRI2 as we did not foresee it when designing its components. We tried to define simple and generic interfaces to support extension through inheritance. It worked for most of the components in the framework; however, as the previous example shows, this mechanism is not always enough. To address it we will include the Extension Object pattern for future releases and we recommend using it in similar tools. The description of that design pattern is beyond the scope of this paper, although we point readers to the explanation by Martin [83].
- How is the framework growing? A framework is supposed to be extended by third party users, and it is very probable that they will contribute by including new functionality. The inclusion of third-party contributions to the tool involves several issues regarding authorship, licencing, publishing, and distribution. It should be clear what the mechanisms and conditions are to create contributions for the tool. In the case of jCOLIBRI we did not plan it from the beginning and we had to publish a "contributions guideline" after the release of the framework to open the framework to third-party

25 http://gaia.fdi.ucm.es/research/colibri/people-users-apps/users-opinion.

26 Serialization is a method of persisting objects.

contributors. This guideline[27] defines how to submit the contribution, how the authorship and licence is going to be considered, etc.

- When is a major change really needed? As the framework evolved, we learnt from the development experience and drawbacks found by the users, causing us to think of better designs, architectures, techniques, and many other aspects. That led us to re-design jCOLIBRI1 and create jCOLIBRI2. Although it was a necessary step, it generated a complete incompatibility between the existing applications that were using the framework. From that experience, our advice is not to make major changes unless it is completely necessary. Sometimes a (relatively) bad design could be better than breaking the code of existing applications.

7. Related work

jCOLIBRI is nowadays the most popular CBR framework due to its broad scope, number of applications, users and contributors. There are other related CBR frameworks in the literature, with myCBR [84] being one of the most closely related to jCOLIBRI2. myCBR is also an open-source tool for developing CBR systems, although there are some important differences in its scope and architecture. myCBR is intended primarily for prototyping applications that focus on the similarity-based retrieval step while jCOLIBRI2 includes components for supporting the whole CBR cycle, including retrieval, reuse, revision and adaptation. To a certain extent myCBR and jCOLIBRI can be used in collaboration, using myCBR to define the case structure and similarity measures through their Protégé-based interface, and having jCOLIBRI import those definitions from the XML files generated by myCBR, through a number of wrapper methods that were developed in collaboration by both project teams.[28]

The Indiana University Case-Based Reasoning Framework (IUCBRF[29]) was conceived with a similar goal of providing an open-source solution for academic CBR projects but never achieved a mature enough development state and has not been actively maintained in the last years [85].

We have not included other discontinued frameworks in this comparison, like CAT-CBR, CBR*Tools or Orenge, as we have no knowledge of recent work on them. Díaz-Agudo et al. [3] present a comparison of these frameworks with the jCOLIBRI1 version of our framework.

Although in a different domain jCOLIBRI2 is also related to Weka, a collection of machine-learning algorithms for data-mining tasks. Weka originally served as an inspiration for jCOLIBRI which was intended to play a similar role in the CBR community, as an open-source reference tool in academia, like the one played by Weka in the data mining community. jCOLIBRI2 is also influenced by Weka in the way it is designed to facilitate the construction of different configurations of a CBR system which can be compared along different dimensions. The two frameworks also share the idea of including both a collection of reusable methods organized in a white-box framework, plus a software composition tool that allows to assemble a running system without writing a line of code. They even share some common methods, since Weka also includes an implementation of the Nearest Neighbours algorithm for instance-based classification. Nevertheless, and in addition to the obvious differences coming from covering different application domains, a key difference between the frameworks is that jCOLIBRI2 is designed for supporting not only classification but also problem-solving tasks, and can deal with complex object-based data and not only simple attribute-value pairs like Weka.

8. Conclusions and future work

In this article, we have described the jCOLIBRI2 framework for building Case-based reasoning systems. The mail goal of our research is to cover the need for open software engineering tools for CBR. We can conclude that jCOLIBRI2 is fulfilling the goal of becoming a reference tool for academia, having, as of this writing, hit the 10,000 downloads mark with users in 100 different countries. The jCOLIBRI2 framework greatly facilitates the development of new CBR applications by reusing components that can be customized to meet CBR application requirements. Additionally, re-designing the framework in order to facilitate the integration of third party code has also proved correct since jCOLIBRI2 is attracting an increasing number of contributions from other research groups.[30]

In this paper we have presented the basic functionality together with several complements that extend this basic functionality, namely textual CBR (see Section 4.1), recommendation (Section 4.2), knowledge-intensive CBR (Section 4.3), data-intensive CBR (Section 4.4) or distributed CBR (Section 4.5).

As future work, we intend to finish the implementation of high level configuration tools where we are trying new ideas on software configuration through a collection of system templates that can be reused in different applications. These templates are composed of tasks that dictate the behaviour of the CBR system. It is an evolution of the task decomposition schema implemented in jCOLIBRI1, where more complex control flows can be defined (loops, conditionals, etc.). It follows a

27 The "contributions guideline" of jCOLIBRI2 is available at http://gaia.fdi.ucm.es/research/colibri/jcolibri/contributions.

28 This extension is also available at http://gaia.fdi.ucm.es/research/colibri/jcolibri/contributions.

29 http://www.cs.indiana.edu/~sbogaert/CBR/.

30 http://gaia.fdi.ucm.es/research/colibri/jcolibri/contributions.

parallelism with the evolution of Problem-solving Methods methodologies to Semantic Web Services standards. What we propose is to use these standards (OWL-S, WSML or similar) to represent the control flow of the templates. The platform provides a catalogue of templates and lets users select the most suitable template and adapt it to the concrete requirements of the target application. This development process, named Template-Based Design, consists mainly of retrieving a template and selecting a suitable component that implements the behaviour dictated by each task in that template. This *component selection* activity is the connection with the jCOLIBRI2 framework. To address this issue we can use an approach based on *annotations*. Annotations represent meta-information in the source code and allow us to document classes, methods or even variables. Then there are several ways to process and extract this information. The most popular example of annotations is the JavaDoc tool, which generates HTML documentation of the source code from simple tags. jCOLIBRI2 uses this mechanism to annotate every component with information about its behaviour. In this way, composition tools can automatically analyse the source code of jCOLIBRI2 and obtain the methods able to solve a concrete task. Later, introspection techniques are used to obtain the signature of these methods and aid the user in defining the data flow between them. An early prototype of the composition tools, named COLIBRI Studio, is available in the jCOLIBRI web site, and a description of the Template-Based Design is presented by Recio-García et al. [86].

Appendix. Supplementary data

Supplementary data to this article can be found online at http://dx.doi.org/10.1016/j.scico.2012.04.002.

References

[1] R. C. Schank, R. P. Abelson, Scripts, Plans, Goals and Understanding: an Inquiry into Human Knowledge Structures, L. Erlbaum, Hillsdale, NJ, 1977.
[2] J. Bello-Tomás, P. González-Calero, B. Díaz-Agudo, JColibri: an object-oriented framework for building CBR systems, in: [87], pp. 32–46.
[3] B. Díaz-Agudo, P. A. González-Calero, J. A. Recio-García, A. Sánchez, Building CBR systems with jCOLIBRI, in: Experimental Software and Toolkits, Journal Science of Computer Programming 69 (2007) 68–75. Special Issue.
[4] Case-Based Reasoning (CBR) - Using Similar Situations from the Past to Solve a Present Problem, Association for the Advancement of Artificial Intelligence (AAAI), 2011. Online: http://aaai.org/AITopics/CaseBasedReasoning (accessed 06.02.12).
[5] D. B. Leake, Case-Based Reasoning: Experiences, Lessons and Future Directions, MIT Press, Cambridge, MA, USA, 1996.
[6] A. Aamodt, E. Plaza, Case-based reasoning: Foundational issues, methodological variations, and system approaches, AI Communications 7 (1994).
[7] M. H. Göker, R. J. Howlett, J. E. Price, Case-based reasoning for diagnosis applications, Knowledge Engineering Review 20 (2005) 277–281.
[8] M. T. Cox, H. Muñoz-Avila, R. Bergmann, Case-based planning, Knowledge Engineering Review 20 (2005) 283–287.
[9] A. Holt, I. Bichindaritz, R. Schmidt, P. Perner, Medical applications in case-based reasoning, Knowledge Engineering Review 20 (2005) 289–292.
[10] E. L. Rissland, K. D. Ashley, K. Branting, Case-based reasoning and law, Knowledge Engineering Review 20 (2005) 293–298.
[11] J. L. Kolodner, M. T. Cox, P. A. González-Calero, Case-based reasoning-inspired approaches to education, Knowledge Engineering Review 20 (2005) 299–303.
[12] K.-D. Althoff, R. O. Weber, Knowledge management in case-based reasoning, Knowledge Engineering Review 20 (2005) 305–310.
[13] P. Perner, A. Holt, M. M. Richter, Image processing in case-based reasoning, Knowledge Engineering Review 20 (2005) 311–314.
[14] D. G. Bridge, M. H. Göker, L. McGinty, B. Smyth, Case-based recommender systems, Knowledge Engineering Review 20 (2005) 315–320.
[15] J. Kolodner, Case-based Reasoning, Morgan Kaufmann, 1993.
[16] N. Kukuric, F. Robijn, J. Griffioen, The i3S Document Series: Using Case based reasoning for the solution of water stress problems, Aquastress, 2008. Online: http://i3s.aquastress.net/tools/CBR/Aquastress I3S-Case Based Reasoning Users Guide.pdf (accessed 06.02.12).
[17] Ambient-Intelligent Interactive Monitoring System for Energy Use Optimisation in Manufacturing SMEs, Aml-Moses project, 2008. Online: http://www.ami-moses.eu/fileadmin/templates/amimoses/files/AmI-MoSES_D7.6_EP_Platform_Public_v1.0.pdf (accessed 06.01.12).
[18] E. Lopes, U. Schiel, Integrating context into a criminal case-based reasoning model, in: Information, Process, and Knowledge Management, 2010. eKNOW'10. Second International Conference on, pp. 37 –42.
[19] E. Vargas, H. Oktaba, S. Guardati, A. Laureano, Agents, case-based reasoning and their relation to the mexican software process model (moprosoft), in: Computer Software and Applications Conference, 2007. COMPSAC 2007. 31st Annual International, vol. 2, pp. 326 –334.
[20] N. Govedarova, S. Stoyanov, I. Popchev, An ontology based CBR architecture for knowledge management in BULCHINO catalogue, in: B. Rachev, A. Smrikarov (Eds.), Proceedings of the 9th International Conference on Computer Systems and Technologies and Workshop for Ph.D. Students in Computing, CompSysTech, ACM, 2008, p. 67.
[21] J. A. Recio-García, B. Díaz-Agudo, A. Sánchez, P. A. González-Calero, Lessons learnt in the development of a cbr framework, in: M. Petridis (Ed.), Proccedings of the 11th UK Workshop on Case Based Reasoning, CMS Press, University of Greenwich, 2006, pp. 60–71.
[22] V. Haarslev, R. Möller, Description of the racer system and its applications, in: Working Notes of the 2001 International Description Logics Workshop (DL-2001), Stanford, CA, USA, August 1-3, 2001.
[23] V. R. Benjamins, D. Fensel, Editorial: problem-solving methods, International Journal of Human-Computer Studies 49 (1998) 305–313.
[24] D. Roberts, R. E. Johnson, Patterns for evolving frameworks, in: Pattern Languages of Program Design 3, Addison Wesley, 1997, pp. 471–486.
[25] O. Nierstrasz, D. Tsichritzis (Eds.), Object-Oriented Software Composition, Prentice-Hall, 1995, Online: http://scg.unibe.ch/archive/oosc/.
[26] S. Sparks, K. Benner, C. Faris, Managing object-oriented framework reuse, Computer 29 (1996) 52–61.
[27] J. McManus, W. Bynum, Design and analysis techniques for concurrent blackboard systems, Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on 26 (1996) 669–680.
[28] K.-D. Althoff, E. Auriol, R. Barletta, M. Manago, A Review of Industrial Case-Based Reasoning Tools, AI Intelligence, Oxford, 1995.
[29] J. A. Recio-García, B. Díaz-Agudo, P. A. González-Calero, jCOLIBRI2 Tutorial, IT/2007/02, Department of Software Engineering and Artificial Intelligence. University Complutense of Madrid, 2007.
[30] The OWL Services Coalition, OWL-S: Semantic Markup for Web Services., http://www.daml.org/services/owl-s/1.1/overview/, 2004.
[31] J. de Bruijn, H. Lausen, A. Polleres, D. Fensel, The web service modeling language wsml: An overview, in: Y. Sure, J. Domingue (Eds.), ESWC, in: Lecture Notes in Computer Science, vol. 4011, Springer, 2006, pp. 590–604.
[32] H. Shimazu, ExpertClerk: a conversational case-based reasoning tool for developing salesclerk agents in e-commerce webshops, Artificial Intelligence Review 18 (2002) 223–244.
[33] S. J. Delany, P. Cunningham, An analysis of case-base editing in a spam filtering system, in: [87], pp. 128–141.
[34] I. Tomek, An experiment with the edited nearest-neighor rule, IEEE Transactions on Systems, Man, and Cybernetics 6 (6) (1976) 448–452.
[35] E. McKenna, B. Smyth, Competence-guided case-base editing techniques, in: [88], pp. 186–197.
[36] H. Brighton, C. Mellish, Advances in instance selection for instance-based learning algorithms, Data Mining and Knowledge Discovery 6 (2002) 153–172.

[37] R. O. Weber, K. D. Ashley, S. Brüninghaus, Textual case-based reasoning, The Knowledge Engineering Review 20 (2006) 255–260.
[38] R. Weber, D. W. Aha, N. Sandhu, H. Munoz-Avila, A textual case-based reasoning framework for knowledge management applications, in: Proceedings of the 9th German Workshop on Case-Based Reasoning. Shaker Verlag., pp. 244–253.
[39] M. Lenz, Defining knowledge layers for textual case-based reasoning., in: [89], pp. 298–309.
[40] J. A. Recio-García, M. A. Gómez-Martín, B. Díaz-Agudo, P. A. González-Calero, Improving annotation in the semantic web and case authoring in textual CBR, in: T. R. Roth-Berghofer, M. H. Göker, H. A. Güvenir (Eds.), Advances in Case-Based Reasoning, 8th European Conference, ECCBR'06, in: Lecture Notes in Artificial Intelligence, subseries of LNCS, vol. 4106, Springer, Fethiye, Turkey, 2006, pp. 226–240.
[41] J. A. Recio-García, B. Díaz-Agudo, M. A. Gómez-Martín, N. Wiratunga, Extending jCOLIBRI for textual CBR, in: H. Muoz-Avila, F. Ricci (Eds.), Proceedings of Case-Based Reasoning Research and Development, 6th International Conference on Case-Based Reasoning, ICCBR 2005, in: Lecture Notes in Artificial Intelligence, subseries of LNCS, vol. 3620, Springer, Chicago, IL, US, 2005, pp. 421–435.
[42] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, P. F. Patel-Schneider (Eds.), The Description Logic Handbook: Theory, Implementation, and Applications, Cambridge University Press, 2003.
[43] B. Díaz-Agudo, J. A. Recio-García, P. A. González-Calero, Natural language queries in CBR systems, in: 19th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2007), 2007, Patras, Greece, vol. 2, IEEE Computer Society, 2007, pp. 468–472.
[44] J. A. Recio-García, B. Díaz-Agudo, P. A. González-Calero, Textual CBR in jCOLIBRI: From Retrieval to Reuse, in: D. C. Wilson, D. Khemani (Eds.), Proceedings of the ICCBR 2007 Workshop on Textual Case-Based Reasoning: Beyond Retrieval., pp. 217–226.
[45] E. Hatcher, O. Gospodnetic, Lucene in Action (In Action series), Manning Publications Co., Greenwich, CT, USA, 2004.
[46] D. Bridge, M. H. Göker, L. McGinty, B. Smyth, Case-based recommender systems, Knowledge Engineering Review 20 (2006) 315–320.
[47] B. Smyth, Case-based recommendation, in: P. Brusilovsky, A. Kobsa, W. Nejdl (Eds.), The Adaptive Web, in: Lecture Notes in Computer Science, vol. 4321, Springer, 2007, pp. 342–376.
[48] W. Wilke, M. Lenz, S. Wess, Intelligent sales support with CBR, in: Case-Based Reasoning Technology, From Foundations to Applications, Springer-Verlag, London, UK, 1998, pp. 91–114.
[49] R. Bergmann, Experience Management: Foundations, Development Methodology, and Internet-Based Applications, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2002.
[50] R. Burke, Interactive critiquing forcatalog navigation in e-commerce, Knowledge Engineering Review 18 (2002) 245–267.
[51] D. McSherry, Diversity-conscious retrieval, in: S. Craw, A. D. Preece (Eds.), ECCBR '02: Proceedings of the 6th European Conference on Advances in Case-Based Reasoning, in: Lecture Notes in Computer Science, vol. 2416, Springer-Verlag, London, UK, 2002, pp. 219–233.
[52] J. Kelleher, D. Bridge, An accurate and scalable collaborative recommender, Artificial Intelligence Review 21 (2004) 193–213.
[53] J. L. Herlocker, J. A. Konstan, A. Borchers, J. Riedl, An algorithmic framework for performing collaborative filtering, in: SIGIR'99: Proceedings of the 22nd annual international ACM SIGIR Conference on Research and Development in Information Retrieval, ACM, New York, NY, USA, 1999, pp. 230–237.
[54] D. McSherry, Similarity and compromise, in: K. D. Ashley, D. G. Bridge (Eds.), Case-Based Reasoning Research and Development, 5th International Conference on Case-Based Reasoning, ICCBR, in: Lecture Notes in Computer Science, vol. 2689, Springer, 2003, pp. 291–305.
[55] B. Smyth, P. McClave, Similarity vs. diversity, in: [90], pp. 347–361.
[56] S. Schulz, CBR-works: a state-of-the-art shell for case-based application building, in: E. Melis (Ed.), Proceedings of the 7th German Workshop on Case-Based Reasoning, GWCBR'99, Würzburg, Germany, University of Würzburg, 1999, pp. 166–175.
[57] B. Smyth, L. McGinty, The power of suggestion, in: G. Gottlob, T. Walsh (Eds.), IJCAI, Morgan Kaufmann, 2003, pp. 127–132.
[58] J. A. Recio-García, B. Díaz-Agudo, P. A. González-Calero, Prototyping recommender systems in jCOLIBRI, in: RecSys'08: Proceedings of the 2008 ACM Conference on Recommender Systems, ACM, New York, NY, USA, 2008, pp. 243–250.
[59] P. A. González-Calero, M. Gómez-Albarrán, B. Díaz-Agudo, A Substitution-based Adaptation Model, in: Challenges for Case-Based Reasoning - Proc. of the ICCBR'99 Workshops, Univ. of Kaiserslautern, 1999, pp. 2–12.
[60] B. Díaz-Agudo, P. A. González-Calero, An architecture for knowledge intensive CBR systems, in: [88], pp. 37–48.
[61] B. Díaz-Agudo, P. A. González-Calero, Knowledge intensive CBR through ontologies, in: B. Lees (Ed.), Procs. of the 6th UK Workshop on Case-Based Reasoning, UKCBR 2001, CMS Press, University of Greenwich, 2001.
[62] B. Díaz-Agudo, P. A. González-Calero, An ontological approach to develop knowledge intensive cbr systems, in: R. Sharman, R. Kishore, R. Ramesh (Eds.), Ontologies, in: Integrated Series in Information Systems, vol. 14, Springer US, 2007, pp. 173–213. 10.1007/978-0-387-37022-4_7.
[63] P. A. González-Calero, M. Gómez-Albarrán, 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.
[64] S. Salotti, V. Ventos, Study and Formalization of a CBR System using a Description Logic, in: [89], pp. 286–301.
[65] A. Napoli, J. Lieber, R. Courien, Classification-Based Problem Solving in CBR, in: I. Smith, B. Faltings (Eds.), Proceedings of the Third European Workshop on Advances in Case-Based Reasoning, EWCBR'96, in: LNCS, vol. 1168, Springer-Verlag, 1996, pp. 295–308.
[66] J. A. Recio-García, B. Díaz-Agudo, P. A. González-Calero, A. Sánchez-Ruiz-Granados, Ontology based CBR with jCOLIBRI, in: R. Ellis, T. Allen, A. Tuson (Eds.), Applications and Innovations in Intelligent Systems XIV. Proceedings of AI-2006, the Twenty-sixth SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence, Springer, Cambridge, United Kingdom, 2006, pp. 149–162.
[67] B. Díaz-Agudo, E. Plaza, J. A. Recio-García, J. L. Arcos, Noticeably new: case reuse in originality-driven tasks, in: [91], pp. 165–179.
[68] A. Aamodt, Knowledge intensive case-based reasoning and sustained learning, in: Proceedings of the ninth European Conference on Artificial Intelligence –, ECAI-90, pp. 1–6.
[69] D. Vernet, E. Golobardes, An unsupervised learning approach for case-based classifier systems, Expert Update. The Specialist Group on Artificial Intelligence 6 (2003) 37–42.
[70] A. Fornells, E. Golobardes, D. Vernet, G. Corral, Unsupervised case memory organization: Analysing computational time and soft computing capabilities, in: ECCBR 2006, in: LNAI, vol. 4106, Springer-Verlag, 2006, pp. 241–255.
[71] T. Kohonen, Self-Organizing Maps, 3rd edition, Springer, 2000.
[72] I. Witten, E. Frank, Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations, Morgan Kaufmann, 2000.
[73] A. Fornells, J. A. Recio-García, B. Díaz-Agudo, E. Golobardes, E. Fornells, Integration of a methodology for cluster-based retrieval in jColibri, in: L. McGinty, D. C. Wilson (Eds.), ICCBR, in: Lecture Notes in Computer Science, vol. 5650, Springer, 2009, pp. 418–433.
[74] E. Plaza, L. Mcginty, Distributed case-based reasoning, The Knowledge Engineering Review 20 (2006) 261–265.
[75] L. McGinty, B. Smyth, Collaborative case-based reasoning: applications in personalised route planning, in: [90], pp. 362–376.
[76] S. Ontañón, E. Plaza, Arguments and counterexamples in case-based joint deliberation, in: Argumentation in Multi-Agent Systems, ArgMAS 2006, Selected and Invited Papers, in: LNCS, vol. 4766, Springer, 2006, pp. 36–53.
[77] J. P. Gunderson, L. F. Gunderson, L. F. Gunderson, J. P. Gunderson, Deliberative system, in: Robots, Reasoning, and Reification, Springer US, 2009, pp. 1–17.
[78] J.A. Recio-García, B. Díaz-Agudo, S. González-Sanz, L. Quijano-Sánchez, Distributed deliberative recommender systems, T. Computational Collective Intelligence 1 (2010) 121–142.
[79] J. A. Carozzoni, J. H. Lawton, C. DeStefano, A. J. Ford, J. W. Hudack, K. K. Lachevet, G. R. Staskevich, Distributed Episodic Exploratory Planning (DEEP), Technical Report, U.S Air Force Research Laboratory, 2008.
[80] A. Serra, P. Avesani, A. Malossini, Recommendation and Learning, ONE Project, 2008. Online: http://files.opaals.eu/ONE/Deliverables/D4.3_SoftwareComponents_RecommenderSystem.pdf (accessed 06.02.12).
[81] A. Martín, C. León, Expert knowledge management based on ontology in a digital library, in: J. Filipe, J. Cordeiro (Eds.), ICEIS (2), SciTePress, 2010, pp. 291–298.
[82] E. Lotfy Abdrabou, A. Salem, A breast cancer classifier based on a combination of case-based reasoning and ontology approach, in: Computer Science and Information Technology (IMCSIT), Proceedings of the 2010 International Multiconference on, pp. 3 –10.

[83] R. Martin, Extension object, in: Pattern Languages of Program Design 3, Addison Wesley, 1997, pp. 471–486.
[84] A. Stahl, T. Roth-Berghofer, Rapid prototyping of cbr applications with the open source tool mycbr, in: [91], pp. 615–629.
[85] S. Bogaerts, D. Leake, IUCBRF: A Framework For Rapid And Modular Case-Based Reasoning System Development, Technical Report 617, Indiana University, http://www.cs.indiana.edu/~sbogaert/CBR/IUCBRF.pdf (accessed 06.02.12), 2005.
[86] J. A. Recio-García, B. Díaz-Agudo, P. A. González-Calero, Template Based Design in COLIBRI Studio, in: M. Minor, S. Montani, J. A. Recio-García (Eds.), Proccedings of the ICCBR-2011 Workshop on Process-oriented Case-based Reasoning, pp. 101–110.
[87] P. Funk, P. A. González-Calero (Eds.), Advances in Case-Based Reasoning, 7th European Conference, ECCBR 2004, Madrid, Spain, August 30 - September 2, 2004, Proceedings, in: Lecture Notes in Computer Science, vol. 3155, Springer, 2004.
[88] E. Blanzieri, L. Portinale (Eds.), Advances in Case-Based Reasoning, 5th European Workshop, EWCBR 2000, Trento, Italy, September 6-9, 2000, Proceedings, in: Lecture Notes in Computer Science, vol. 1898, Springer, 2000.
[89] B. Smyth, P. Cunningham (Eds.), Advances in Case-Based Reasoning, 4th European Workshop, EWCBR-98, Dublin, Ireland, September 1998, Proceedings, in: Lecture Notes in Computer Science, vol. 1488, Springer, 1998.
[90] D. W. Aha, I. Watson (Eds.), Case-Based Reasoning Research and Development, 4th International Conference on Case-Based Reasoning, ICCBR 2001, Vancouver, BC, Canada, July 30 - August 2, 2001, Proceedings, in: Lecture Notes in Computer Science, vol. 2080, Springer, 2001.
[91] K.-D. Althoff, R. Bergmann, M. Minor, A. Hanft (Eds.), Advances in Case-Based Reasoning, 9th European Conference, ECCBR 2008, Trier, Germany, September 1-4, 2008. Proceedings, in: Lecture Notes in Computer Science, vol. 5239, Springer, 2008.