Available online at www.sciencedirect.com
ScienceDirect
Science of Computer Programming 69 (2007) 68–75
Science of Computer Programming
www.elsevier.com/locate/scico

Building CBR systems with jCOLIBRI☆

Belén Díaz-Agudo*, Pedro A. González-Calero, Juan A. Recio-García, Antonio A. Sánchez-Ruiz-Granados

Facultad de Informatica, Universidad Complutense de Madrid, Madrid, Spain

Received 1 November 2005; received in revised form 28 July 2006; accepted 12 February 2007
Available online 16 October 2007

Abstract

Case-based reasoning (CBR) is a paradigm for combining problem solving and learning that has become one of the most successful applied subfields of AI in recent years. Now that CBR has become a mature and established technology two necessities have become critical: the availability of tools to build CBR systems, and the accumulated practical experience of applying CBR techniques to real-world problems. In this paper we are presenting jCOLIBRI, an object-oriented framework in Java for building CBR systems, that greatly benefits from the reuse of previously developed CBR systems.

Keywords: Case-based reasoning; Problem-solving methods; Ontologies

1. Introduction

Case-based reasoning (CBR) has become a mature and established subfield of artificial intelligence (AI), both as a means for addressing AI problems and as a basis for fielded AI technology.

Now that CBR fundamental principles have been established and numerous applications have demonstrated that CBR is an useful technology, many researchers agree on the increasing necessity to formalize this kind of reasoning, define application analysis methodologies, and provide a design and implementation assistance with software engineering tools [5,12,8,10]. While the underlying ideas of CBR can be applied consistently across application domains, the specific implementation of the CBR methods – in particular retrieval and similarity functions – is highly customised to the application at hand. Two factors have become critical: the availability of tools to build CBR systems, and the accumulated practical experience of applying CBR techniques to real-world problems.

Our work goes along all these increasing necessities. We intend to design a tool to help application designers to develop and quickly prototype CBR systems. Besides we want to provide a software tool useful for students who have little experience in the development of different types of CBR systems. jCOLIBRI has been designed as a wide spectrum framework able to support several types of CBR systems from the simple nearest-neighbor approaches based on flat or simple structures to more complex Knowledge Intensive ones. It also contains textual and conversational extensions [17,11].

☆ Supported by the Spanish Ministry of Education and Science (TIN2006-15140-C03-02).
* Corresponding author.
E-mail addresses: belend@sip.ucm.es (B. Díaz-Agudo), pedro@sip.ucm.es (P.A. González-Calero), jareciog@fdi.ucm.es (J.A. Recio-García), antonio.sanchez@fdi.ucm.es (A.A. Sánchez-Ruiz-Granados).

0167-6423/$ - see front matter
doi:10.1016/j.scico.2007.02.004

jCOLIBRI offers a simplified development process that is based on the reuse of past designs and implementations. It formalizes the CBR knowledge using a domain-independent CBR ontology (CBROnto) [9] which is mapped into the classes of the framework, a knowledge level description of the CBR tasks and a library of reusable Problem-Solving Methods (PSMs) [4]. Although there are other CBR tools our work goes beyond them in terms of reuse, flexibility, scope and usability. jCOLIBRI is an object-oriented framework in Java for building CBR systems. jCOLIBRI promotes software reuse integrating the application of well-proven Software Engineering techniques with a knowledge level description that separates the PSMs, which define the reasoning processes, from the domain model, that describes the domain knowledge. Framework instantiation is supported by a graphical interface that guides the configuration of a particular CBR system. This paper provides a general overview of the main characteristics of the framework that can be downloaded by visiting sourceforge: http://sourceforge.net/projects/jcolibri-cbr/.

The goal of this tool description paper is for the reader to obtain a general view of the main components of jCOLIBRI and learn how to use them to easily create CBR applications. Documentation provided with jCOLIBRI includes detailed usage examples (see http://gaia.fdi.ucm.es/grupo/projects/jcolibri/). Among others, we show the use of jCOLIBRI to create a travel recommendation system from the travel domain case base, a well-known example frequently used in the CBR community.

Section 2 introduces CBR as a problem-solving paradigm, describes the core tasks that all CBR methods have to deal with and the range of different methods that a CBR system designer could choose to solve these tasks. Before going into the details of jCOLIBRI in Section 4, Section 3 briefly describes related CBR tools. Section 5 concludes and presents our lines of 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 AI in recent years. CBR is based on the intuition that problems tend to recur. It means that new problems are often similar to previously encountered problems and, therefore, that past solutions may be of use in the current situation [7]. CBR is rooted in the works of Roger Schank on dynamic memory and the central role that a reminding of earlier episodes (cases) and scripts (situation patterns) has in problem solving and learning [18].

CBR is particularly applicable to problems where earlier cases are available, even when the domain is not understood well enough for a deep domain model. Helpdesks, diagnosis or classification systems have been the most successful areas of application. For example: to determine a fault or diagnose an illness from observed attributes, or to determine whether or not a certain treatment or repair is necessary given a set of past solved cases [20].

Central tasks that all CBR methods have to deal with are [1]: "*to identify the current problem situation, find a past case similar to the new one, use that case to suggest a solution to the current problem, evaluate the proposed solution, and update the system by learning from this experience. How this is done, what part of the process that is focused, what type of problems that drives the methods, etc. varies considerably, however*".

The underlying ideas of CBR can be applied consistently across application domains although a CBR system designer has to decide from among a range of different methods for organizing, retrieving and reusing the knowledge retained in past cases[1]: Cases may be kept as concrete experiences, or a set of similar cases may form a generalized case [15]. Cases may be stored as separate knowledge units, or distributed within the knowledge structure [6]. Cases may be indexed by a prefixed or open vocabulary, and within a flat or hierarchical index structure. The solution from a previous case may be directly applied to the present problem, or adapted according to differences between the two cases. The matching of cases, adaptation of solutions, and learning from an experience may be supported by a deep model of general domain knowledge [8], by more shallow and compiled knowledge, or be based on an apparent, syntactic similarity only. Some CBR methods assume a rather large amount of widely distributed cases in its case base, while others are based on a more limited set of typical ones. CBR methods may be purely self-contained and automatic, or they may be user guided.

For a detailed description of these and other CBR related aspects, we direct the interested reader to detailed surveys about the CBR field [1,14,7] and to the last European and International conferences in the field (ECCBR and ICCBR).

Our work within jCOLIBRI goes along the line of many researchers who agree about the increasing necessity to formalize case-based reasoning, define application analysis methodologies, and provide a design and implementation assistance with software engineering tools.

Before going into the details of jCOLIBRI, the next section briefly describes previously existing CBR tools.

3. Related tools

In the Machine Learning (ML) community we can mention a number of efforts directed to building reusable libraries of ML methods. MLC++ [13], developed at Stanford University by Ronny Kohavi, provides a library of C++ classes to aid in the development of new ML algorithms, especially hybrid algorithms and multistrategy algorithms, providing implementations for ID3, nearest neighbor, naive Bayes, lazy decision trees, etc. David Mount, from the University of Maryland has developed ANN [3], a library written in C++, which supports data structures and algorithms for both exact and approximate nearest-neighbor searching in arbitrarily high dimensions. The machine learning group of the University of Waikato, New Zealand, has developed Weka [21] a public domain class library in Java that incorporates several standard ML techniques. Weka contains implementations of a number of algorithms for classification and numeric prediction. Weka also includes a number of data filtering techniques, and several clustering algorithms. In order to exploit these efforts from the ML community we plan to incorporate into jCOLIBRI interfaces some of the aforementioned resources.

In the CBR community, the work by Michel Jaczynski [12] is closely related to the one presented here. CBR*Tools is an object-oriented framework, implemented in Java, designed to facilitate the development of CBR applications. They identify the following axes of variability: the delegation of reasoning steps, the separation of case storage and case indexing, the design of indexes as reusable components, and the design of adaptation patterns. The framework concentrates mainly on indexing, providing a large number of indexing alternatives. The key difference of jCOLIBRI with respect to CBR*Tools is the explicit model of task/method decomposition that imposes a high level architecture on top of the framework, facilitating framework use and evolution. However, in a certain sense, in CBR*Tools the knowledge level is also *explicit* as a part of its UML-based model. Besides, jCOLIBRI incorporates a GUI for alleviating framework instantiation effort that is based on its task/method description (at the knowledge level).

The architecture of Orenge (the Open Retrieval engine from empolis.com) [19] is also related to jCOLIBRI. Orenge has been designed as a component-based platform. Building an application includes the selection and composition of required components. Its pipeline/pipelet structure is very similar to the PSMs paradigm. In the same way as jCOLIBRI, Orenge also provides access to databases and XML files. However, Orenge is a commercial system and it is not available freely.

Also closely related to jCOLIBRI, Plaza and Arcos [16] proposed a generic architecture for adaptation that has evolved into CAT-CBR [2], a component-based platform for developing CBR systems. CAT-CBR uses UPML for specifying CBR components and provides: a library of CBR components (currently for retrieval and for classification; reuse components are being developed); a language and a graphical editor to specify new components and a syntax to implement their operational code; a broker service allowing to specify the requirements of a target CBR application and the available models in a domain, and to search for a configuration of components that satisfy the requirements. CAT-CBR generates the "glue code" that binds together the operational code of the configured components into a stand-alone application. The main difference between jCOLIBRI and CAT-CBR is a technological one. CAT-CBR is developed on top of the Noos framework, a monolithic Lisp system that cannot compete with a Java-based approach in terms of usability, extendability and user acceptance.

4. jCOLIBRI

jCOLIBRI is a system for building CBR applications that is an evolution of previous work on knowledge intensive CBR [8,9]. The first version of our software (COLIBRI[1]) was prototyped in LISP and was far from being usable outside of our own research group. However we learned good lessons from it, and we have designed jCOLIBRI as a technological evolution of COLIBRI that incorporates in a 3-tier architecture an object-oriented framework in Java and a number of GUI-based tools for assembling a CBR application from reusable components.

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

[1] Cases and Ontology Libraries Integration for Building Reasoning Infrastructures.

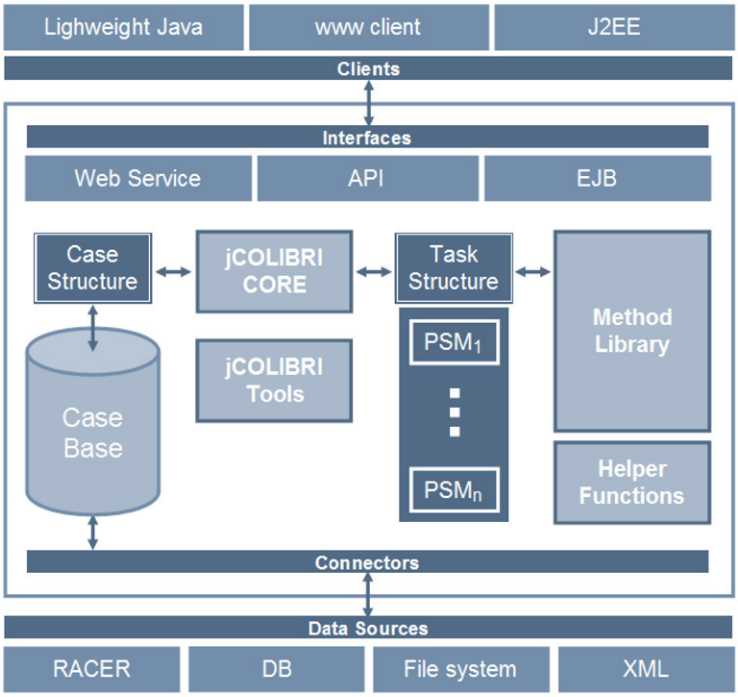

Fig. 1. jCOLIBRI framework structure.

The jCOLIBRI system architecture, as depicted in Fig. 1, is organized around the following elements:

- **Case base, data sources and connectors**. CBR systems must have access to the stored cases in an efficient way, a problem that becomes more relevant as the size of the Case Base grows. jCOLIBRI splits the problem of Case Base management in two separate although related concerns: persistency mechanism through connectors and in-memory organization. Different connectors and data structures for in-memory organization are provided.
- **Case structure and similarity measures**. Depending on the system, cases may be represented as simple plain attribute-value cases, textual cases, or complex hierarchical (object-oriented) structures where attributes are connected among them. Depending on the case structure different similarity functions can be used to compare the attributes of the cases.
- **Tasks structure and problem-solving methods (PSMs)**. A CBR application in jCOLIBRI is defined at the knowledge level as a sequence of tasks that must be solved by a resolution or decomposition method. Decomposition methods divide a task into several subtasks. Examples of tasks are PreProcess Cases, Obtain Query, Retrieve, Reuse, Revise, Retain, Compute Similarity, and many others. For each task, the CBR designer configures a method that solves it.

  Our approach to the specification of PSM competence (post-condition) and requirements (pre-condition) makes use of ontologies and provides two main advantages. First, it allows formal specifications that add a precise meaning and enables reasoning support. Second, it provides us with important benefits regarding reuse because task and method ontologies can be shared by different systems.
- **Method library and helper functions**. The method library includes the operational level (i.e. the implementation) of the resolution problem-solving methods at the knowledge level. A CBR system designer will need to write Java code when the library does not include a required resolution method. The helper functions provide support for the development of new methods and similarity functions.
- **Core and tools**. Notice that this is both the architecture for jCOLIBRI, an application generator, and the architecture for a particular CBR application built with it. jCOLIBRI tools allow one to build a new application by defining: the case structure, the connector configuration and the task structure. Configuration data is saved in XML files that serve as input to the framework core to execute the CBR application. The framework core also plays a role in the design process since a partially configured system can also be interactively tested.

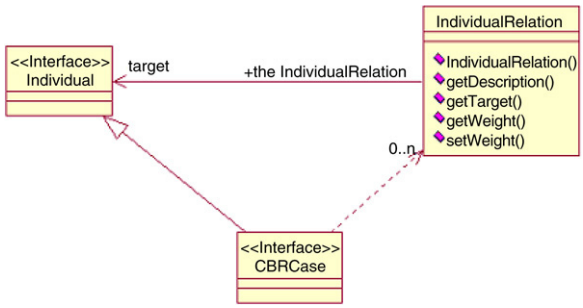

Fig. 2. Case representation.

- **Interface**: The top layer of the system provides the facade to use a configured CBR application. Currently, jCOLIBRI supports stand-alone Java clients and web interfaces through a Tomcat server.

The following sections describe the main steps for developing a CBR application in jCOLIBRI along with the supporting tools to do that. A more detailed description of system usage can be found in the documentation which we recommend to carry out with the step-by-step desing scenarios found in the tutorial videos.[2]

4.1. Case structures and case bases

jCOLIBRI represents a case in a very general way. As shown in Fig. 2, a case is just an individual that can have any number of relationships with other individuals (the attributes of the case). The framework is populated with a number of predefined primitive data types that support the definition of any case structure, and a GUI-based tool is provided to build it. This abstract settlement, inspired on 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*.

Cases are often derived from legacy databases, thereby converting existing organizational resources into exploitable knowledge. To take advantage of these previously existing resources, facilitate intelligent access to existing information, and incorporate it as the seed knowledge in the CBR system (the case base) jCOLIBRI offers a set of connectors to manage persistence of cases.

Connectors are objects that *know* how to access and retrieve cases from the storage media and return those cases to the CBR system in a uniform way. Therefore connectors provide an abstraction mechanism that allows users to load cases from different storage sources in a transparent way. jCOLIBRI includes connectors that work with plain text files, XML files, relational data bases and Description Logics (DL) systems. Other connectors can be included depending on the specific application requirements by means of the jCOLIBRI.cbrcase.Connector java interface. Besides jCOLIBRI offers a graphical tool that is used to easily configure connectors to load existing case bases in different formats. Fig. 3 shows the tool for configuring a connector to a database by assigning attributes of the case structure to columns in a given table.

4.2. Task/method configuration

jCOLIBRI offers a simplified development process that is based on the reuse of past designs and implementations. jCOLIBRI formalizes the CBR knowledge using a CBR ontology (CBROnto), a knowledge level description of the CBR tasks and a library of reusable PSMs. The CBR system designer creates a CBR application following an iterative process:

(1) Select one of the not configured tasks of the system.
(2) Choose from the library of reusable methods one that is suitable to solve – directly or by decomposition – the selected task. Note that task/method constraints are being tracked during the configuration process so that only applicable methods in the given *context* are offered to the system designer.

[2] http://gaia.fdi.ucm.es/projects/jcolibri/jcolibri1/tutorials.html.

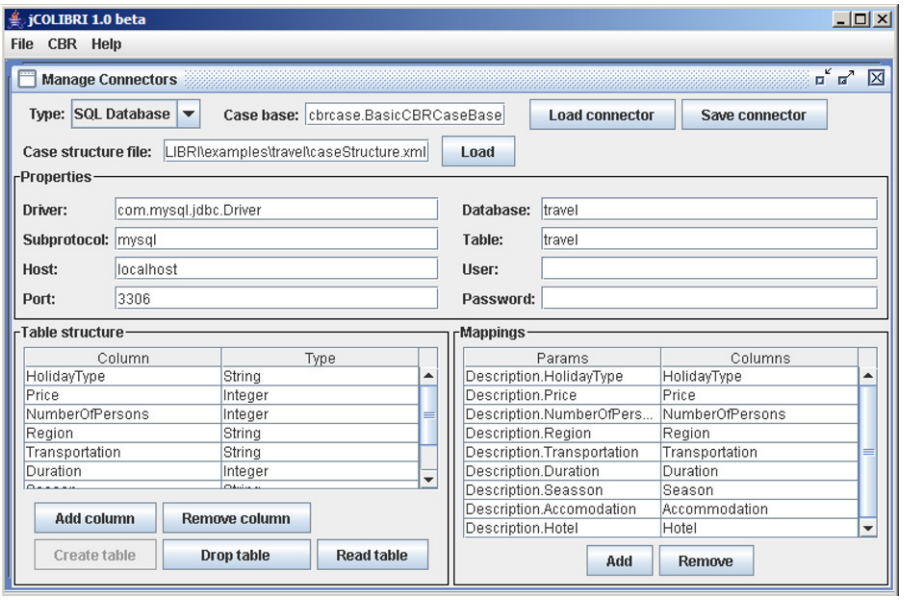

Fig. 3. jCOLIBRI connector configuration.

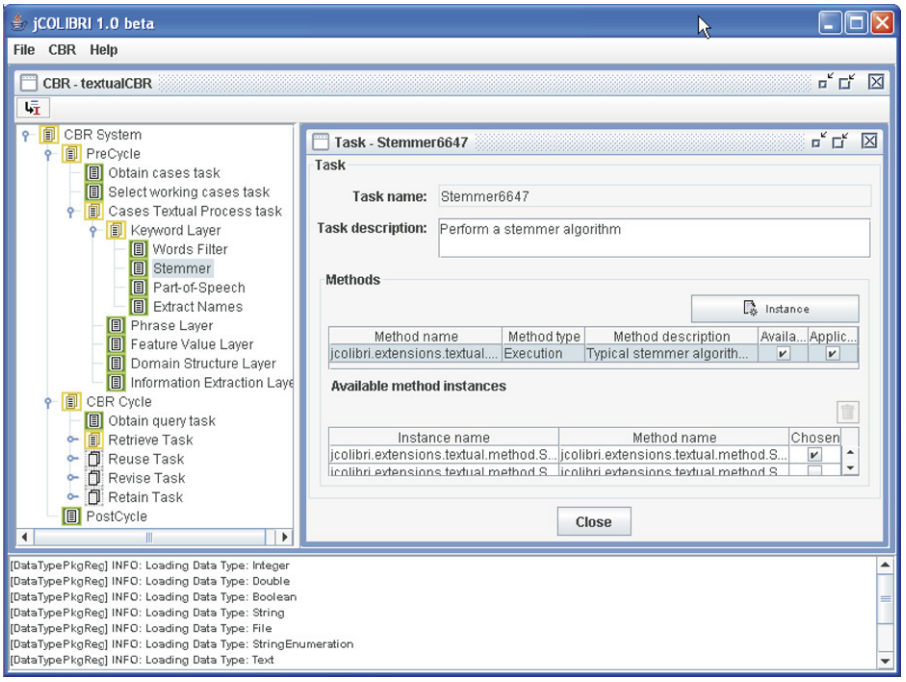

Fig. 4. Task/Method configuration example.

Task structure in a CBR system distinguishes between:

- Pre-cycle or tasks that are solved once before the main cycle, like computing the index structure or processing texts in textual CBR (see Fig. 4).
- Main cycle of four tasks (4 Rs) [1]: *Retrieve* the most similar case/s, *Reuse* it/their knowledge to solve the problem, *Revise* the proposed solution and *Retain* the experience.
- Post-cycle or maintenance tasks, like forgetting not useful cases or recomputing the index structure.

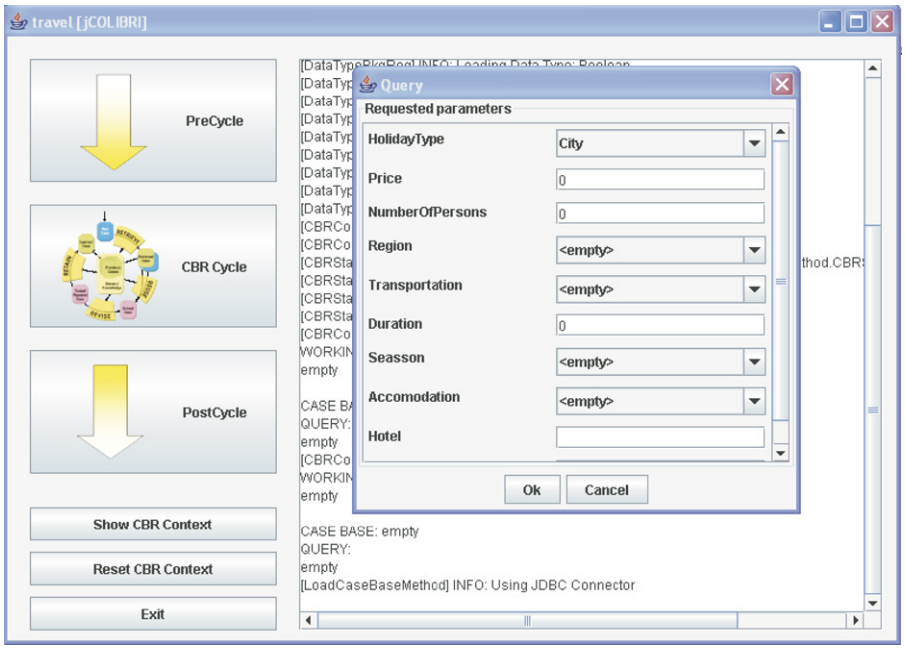

Fig. 5. CBR system interface built by jCOLIBRI.

4.3. Deploying an application

A designed CBR system can be executed from the task/method configuration interface (see Fig. 4). The configured system can also be loaded/stored from jCOLIBRI.

jCOLIBRI offers facilities to generate a stand-alone CBR application (.java) including user interfaces like the one shown in Fig. 5 and that correspond to the travel domain application. This is an interface mainly for testing purposes where the user may execute pre-cycle tasks, main cycle tasks and post-cycle tasks. The source code of this generated application allows developers to easily integrate the CBR system into another application.

4.4. Advanced features

jCOLIBRI allows the development of Knowledge Intensive CBR applications using Semantic Web technologies like OWL ontologies, Information Extraction methods or Description Logic reasoners. Ontologies are useful regarding different aspects: as the vocabulary to describe cases and/or queries, as a knowledge structure where the cases are located, and as the knowledge source to achieve semantic reasoning methods for similarity assessment and case adaptation that are reusable across different domains. The framework allows one to connect with DL reasoners that work with the ontlogies.

The Textual Extension of jCOLIBRI allows one to create structured cases from plain texts using Information Extraction methods. As Semantic Web applications do, this extracted information (cases) can be tagged and indexed using an ontology.

Other advanced features are the Evaluation module that allows one to measure the quality of the CBR applications, or the Web Extension that can be used to develop web interfaces.

5. Conclusions and future directions

We have presented an object-oriented framework in Java to build CBR systems. This framework is built around a task/method ontology that facilitates the understanding of an intrinsically sophisticated software artifact. A key aspect of this work is the availability of configuration tools that facilitate the framework instantiation process.

The framework implementation is evolving as new methods are being included. We are already porting previously developed CBR systems into the framework which is publicly available to the CBR community. Our (ambitious) goal

is to provide a reference framework for CBR development that would grow with contributions from the community. This reference would serve pedagogical purposes and would be a bottom line implementation for prototyping CBR systems and comparing different CBR approaches to a given problem.

The main problem we foresee for the adoption of jCOLIBRI as reference tool in the CBR community is that of interoperability. Although accepting the benefits of participating in an open source initiative, other CBR research groups may be reluctant to put away their own technology to adopt jCOLIBRI, even when our framework may already include the problem-solving methods that they have. The solution we are exploring is the use of web services as a component technology that allows one to interoperate without imposing further restrictions on the implementation. jCOLIBRI would then work on semantic descriptions of the web services and it could be applied to the composition of problem-solving methods implemented outside the jCOLIBRI framework. This way different levels of jCOLIBRI adoption are defined: low level, source code framework integration; component level interoperations; and just profiting from jCOLIBRI at the tool level.

Supplementary data

Supplementary data associated with this article can be found, in the online version, at doi:10.1016/j.scico.2007.02.004.

References

[1] A. Aamodt, E. Plaza, Case-based reasoning: Foundational issues, methodological variations, and system approaches, AI Communications 7 (i) (1994).
[2] C. Abásolo, E. Plaza, J.-L. Arcos, Components for case-based reasoning systems, in: Lecture Notes in Computer Science, vol. 2504, 2002.
[3] ANN. http://www.cs.umd.edu/~mount/ANN/.
[4] R. Benjamins, D. Fensel, Problem Solving Methods, International Journal of Human Computer Studies 49 (4) (1998) (special issue).
[5] R. Bergmann, S. Breen, M. Goker, M. Manago, J. Schumacher, A. Stahl, E. Tartarin, S. Wess, W. Wilke, The inreca-ii methodology for building and maintaining cbr applications, 1998.
[6] R. Bergmann, W. Wilke, I. Vollrath, S. Wess, Integrating general knowledge with object-oriented case representation and reasoning, in: H.-D. Burkhard, M. Lenz (Eds.), 4th German Workshop: Case-Based Reasoning — System Development and Evaluation, in: Informatik-Berichte Nr, vol. 55, Universität Berlin, 1996, pp. 120–127.
[7] D.B. Leake (Ed.), Case Based Reasoning: Experiences, Lessons and Future Directions, AAAI Press, MIT Press, USA, 1996.
[8] B. Díaz-Agudo, P.A. González-Calero, An architecture for knowledge intensive CBR systems, in: E. Blanzieri, L. Portinale (Eds.), Advances in Case-Based Reasoning, EWCBR'00, Springer-Verlag, Berlin, Heidelberg, New York, 2000.
[9] B. Díaz-Agudo, P.A. González-Calero, CBROnto: A task/method ontology for CBR, in: S. Haller, G. Simmons (Eds.), Procs. of the 15th International FLAIRS'02 Conference, AAAI Press, 2002.
[10] P. Funk, P.A. González-Calero (Eds.), Advances in Case-Based Reasoning, in: 7th European Conference, ECCBR 2004, Madrid, Spain, August 30–September 2, 2004, Proceedings, in: Lecture Notes in Computer Science, vol. 3155, Springer, 2004.
[11] H. Gomez-Gauchia, B. Díaz-Agudo, P.P. Gomez-Martin, P.A. González-Calero, Supporting conversation variability in cobber using causal loops, in: Case-Based Reasoning Research and Development—Proc. of the ICCBR'05, in: LNCS/LNAI, Springer Verlag, 2005.
[12] M. Jaczynski, B. Trousse, An object-oriented framework for the design and the implementation of case-based reasoners, in: Proceedings of the 6th German Workshop on Case-Based Reasoning, 1998.
[13] R. Kohavi, D. Sommerfield, J. Dougherty, Data mining using MLC++: A machine learning library in C++, in: Tools with Artificial Intelligence, IEEE Computer Society Press, 1996, pp. 234–245.
[14] J. Kolodner, Case-Based Reasoning, Morgan Kaufmann Publ., San Mateo, 1993.
[15] K. Maximini, R. Maximini, R. Bergmann, An investigation of generalized cases, in: Proc. of the ICCBR 2003, in: LNCS/LNAI, Springer Verlag, 2001, pp. 261–275.
[16] E. Plaza, J.L. Arcos, Towards a software architecture for case-based reasoning systems, in: Foundations of Intelligent Systems, 12th International Symposium, ISMIS 2000, in: LNCS, vol. 1932, Springer-Verlag, 2000, pp. 265–276.
[17] J.A. Recio, B. Díaz-Agudo, M.G. Martin, N. Wiratunga, Extending jcolibri for textual cbr, in: Procs. of the ICCBR 2005, Springer-Verlag, 2005.
[18] R.C. Schank, Dynamic Memory, Cambridge Univ. Press, 1983.
[19] J. Schumacher, Empolis orenge — an open platform for knowledge management applications, in: 1st German Workshop on Experience Management, 2002.
[20] I. Watson, Applying Case-Based Reasoning: Techniques for Enterprise Systems, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1998.
[21] I. Witten, E. Frank, Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations, Morgan Kaufmann, 2000.