Lessons Learnt in the Development of a CBR Framework ⋆

Juan A. Recio-García, Belén Díaz-Agudo,
Antonio A. Sanchez-Ruiz, and Pedro A. González-Calero

Dep. Ingeniería del Software e Inteligencia Artificial
Universidad Complutense de Madrid, Spain
email: {jareciog,antonio.sanchez}@fdi.ucm.es {belend,pedro}@sip.ucm.es

Abstract. Developing a CBR application from scratch is a hard task. However many components of the system could be reused from previous developments. Based on this idea, our research group has developed jCOLIBRI, an object-oriented framework in Java for building CBR systems that greatly benefits from the reuse of previously developed CBR systems. In this paper we describe our experience during the development of jCOLIBRI, its features, drawbacks and lessons learnt during the evolution of the framework.

Keywords: Case-Based Reasoning, Framework Development, jCOLIBRI

1 Introduction

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 is a mature and established technology two necessities have became critical: the availability of tools to build CBR systems, and the accumulated practical experience of applying CBR techniques to real-world problems.

Many researchers in the field agree about the increasing necessity to formalize this kind of reasoning, define application analysis methodologies, and provide design and implementation assistance with software engineering tools [8, 3, 6]. The best way for achieving this goal is the development of a framework. Frameworks are a well-known technology related with software reuse that leverages the prior efforts of developers in order to avoid recreating and revalidating common solutions.

During the last few years our research group has developed jCOLIBRI, a framework for building CBR systems that greatly benefits from the reuse of previously

⋆ Supported by: Spanish Ministry of Science and Education under grant TIN2006-15140-C03-02; and Main Directorate of Universities and Investigation of the Council of Education of the Community of Madrid and Complutense University of Madrid under grant 910494 for consolidated Research Groups.

developed CBR systems. In this paper we describe our experience during the development of jCOLIBRI, its features, drawbacks and lessons learnt during the evolution of the framework. To address the issues identified we propose a new architecture that takes advantage of the newest Java technologies.

Following section examines reasons for the necessity of a CBR framework. In section 3 we report an analysis of CBR applications in order to summarize the common components that must be included in a framework. Section 4 describes the current state of our system whereas Section 5 describes its main drawbacks. Finally, Section 6 proposes an improved framework architecture.

2 Why a CBR Framework?

Developing a CBR application from scratch is a hard task. However many components of the system could be reused from previous developments. Adding the increasing necessity of the CBR community for developing prototypes quickly, reusing and comparing their improvements with other researchers, the requirement of a CBR framework becomes obvious. A framework is a partial design and implementation for an application in a given problem domain. The use of a framework for software development has many advantages [5]: modularity (improves software quality by localizing changes), reusability (lowers the required effort for developing an application from scratch) and extensibility (provides explicit hook methods that allow applications to extend their stable interfaces). But developing a good framework is not easy. It requires good domain knowledge and previous experience of developing the applications in the domain.

Trying to improve the process of development CBR systems our research group has created the jCOLIBRI framework. It has the mentioned advantages of frameworks but also presents several drawbacks that must be corrected for future releases. From the experience of two years of framework development, the following sections revise current features, problems and proposed improvements to obtain a reference framework in the CBR community.

There are some other efforts within the CBR community for developing this kind of systems. CBR*Tools [8] identifies 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. Orenge [12] has been designed as a component based platform similar to the paradigm used in jCOLIBRI. Finally, CAT-CBR [1] uses UPML for specifying CBR components.

For classifying a framework we can distinguish several types [10]. 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 has an associated builder that will generate the application's code. The visual builder allows the software designer to connect the framework objects and activate them. The current version of jCOLIBRI is

closer to a black-box framework with visual builder and lacks of a clear white-box structure. The new design presented in this paper attempts to remodel the architecture into a clear white-box system oriented to programmers and a separated white-box with visual builder layer that is oriented to designers.

3 Analysis of CBR Applications

Designing a CBR framework implies the analysis of different types of CBR systems. Case base reasoning is being used for a very wide spectrum of applications, so obtaining the common factors of all these applications is a difficult task. In order to create a CBR framework that can cover different approaches we reviewed the most important types of CBR systems and extracted some axes of variability in their architecture [13, 2]. We can organize these axes into two groups:

Case Base structure:

- Cases persistence: Databases, plain text files or ontologies.
- Case representation: There are many approaches: flat (attribute,value), structured, based on frames or object-oriented schemes, plain texts, semantic nets, conceptual graphs, annotated predicate calculus,...
- Case components: This usually depends on the type of application. Commonly they are: description, solution, justification of the solution and result of applying the case.
- In-memory case organization or indexing: Once cases are loaded they can be organized in several ways trying to improve the access to the case base: linear lists, trees, case retrieval nets, etc.

Tasks and methods of the CBR cycle:

- Retrieval algorithms: There are many methods for retrieving cases according to an specific query. The approach depends on the persistence media, case representation or in-memory organization. The simplest one is the Nearest Neighbor using local and global similarity functions. There are many other algorithms like Inductive Retrieval or Ontology Classification.
- Adaptation Methods: This is also a very domain dependent component of a CBR system. Their implementation differs in each system and there is not a common way for acquiring or representing the required knowledge. In general we can divide the adaptation algorithms into three categories: No adaptation, Structural adaptation (parameters adjustment, reinstantiation,...) and Derivational adaptation (replay, model guided replay, ...).
- Adaptation Knowledge: none, low or intensive.
- Learning: Commonly, CBR systems retain the adapted case in the case base but it is not always required.
- User interaction: In some systems the user only interacts when defining the query. Others need the feedback from the user in the adaptation step. Finally, there are some special types of CBR applications like conversational CBR where users must collaborate with the system to find the proper query.
- User Interface: CBR systems can be final applications that are executed stand-alone, in web environments or integrated into a bigger software system.

4 jCOLIBRI 1.0

jCOLIBRI [9] is an evolution of the COLIBRI architecture [3], consisting of a library of Problem Solving Methods (PSMs) for solving the tasks of a knowledge intensive CBR system along with an ontology, CBROnto[4]. The tasks and methods library guides the framework design, determines possible extensions and supports the framework instantiation process. Tasks and methods are described in terms of domain-independent CBR terminology which is mapped into the classes of the framework. This terminology is defined in CBROnto, separating the domain knowledge and the tasks and methods terminology.

jCOLIBRI splits the problem of Case Base management into two separate although related concerns: persistence mechanism and in-memory organization.

Persistence is built around connectors. Connectors are objects that know how to access and retrieve cases from the medium and return those cases to the CBR system in a uniform way. Currently, jCOLIBRI implements different connectors to load a cases from data base, plain text file, XML file or Description Logics ontology.

The second layer of Case Base management is the data structure used to organize the cases once loaded into memory. The organization of the case base (linear, k-d trees, etc.) may have a big influence on the CBR processes, so the framework leaves open the choice of the data structure that is going to be used.

Cases are represented in a very general way. 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 simple case. This abstract view allows for a uniform treatment of arbitrarily complex case structures. Individual objects contain information about each of a case's attributes: the value of the attribute, and meta-information about the attribute such as the similarity function applied to the attribute, the weight of the attribute among the case, etc.

Framework instantiation is aided by several tools that allow task and method definitions, PSM composition, case description and case base connector configuration. These tools have been described in [7] and provide many graphical interfaces to configure CBR systems. The configuration data is stored in different XML configuration files. When the application is executed, the framework core reads these files to know how to configure the CBR system. You can write or modify configuration files by hand, but this can be a very tedious task and graphical tools are provided to do it.

To orchestrate all the components of the framework, jCOLIBRI uses a blackboard structure named *Context*. The context is used by the methods to obtain the case base, interchange data and other configuration values or parameters. The organization, access and representation of data into the Context are critical parts of the framework. It is based on a hash table where every component can store any data labeled with an identifier. This way, if another component requires some data it needs the corresponding identifier to read the information. This organization has advantages because it offers an open and simple communication

between present and future components, but also presents many drawbacks that will be discussed in the following sections.

jCOLIBRI is aimed at CBR system designers. They can design a CBR application using the graphical interface without writing any code. Even so, expert programmers can use Java to instantiate the framework or add new components. During the evolution of the framework its components have been developed keeping in mind the designers' point of view. That implies an easier use of the framework but on the other hand it generates a dependency of the components on the user interface. This dependency makes the code harder to reuse for developers.

5 Drawbacks of the Current Architecture

Current architecture presents several problems that we want to correct for future versions. We think that the main problem is the difficulty of developing new applications using the code directly without the user interface. We have detected several drawbacks that must be corrected:

- **Cases.** The case structure mixes data and metadata. The data is the value of an attribute and the metadata is the information about similarity, weight, etc. For simple CBR applications it is a good solution, but for more advanced applications it presents several problems. For example, in some scenarios we would like to select the similarity measures when introducing the query. And in the retrieval from ontologies it is completely unnecessary.
- **Development of applications.** It is not easy to develop applications "by hand" using directly the classes of the framework. jCOLIBRI is a good black-box framework but not such a good white-box one. The origin of this problem is the coupling between logical classes and user interface.
- **Metadata.** Besides the problem of the metadata contained in the case structure, there is also information that is stored in the XML configuration files but could be inferred from the Java code or runtime objects. Although actually this is not a real inconvenience, it could become a problem of redundancy.
- **Blackboard architecture.** The blackboard organization is a simple and extensible way for communicating and developing new components. But, in general, blackboard-based applications are often discouraged because they present more drawbacks than advantages when the software systems grow. With many components reading and writing information in a hash table, it is difficult to manage the identifiers of the interchanged data and it is impossible to guarantee the order of execution of the methods. This is also a problem when the application includes concurrent threads. The simplest example is the web interface for CBR applications, where several users can execute the system concurrently.
- **Query expressiveness.** There is no standard way for expressing queries in a CBR system. jCOLIBRI includes a basic interpretation of queries that uses an AND boolean operator applied to the attributes. Users can express queries like: "I want a car with diesel motor, 3 doors, red color,..." weighting

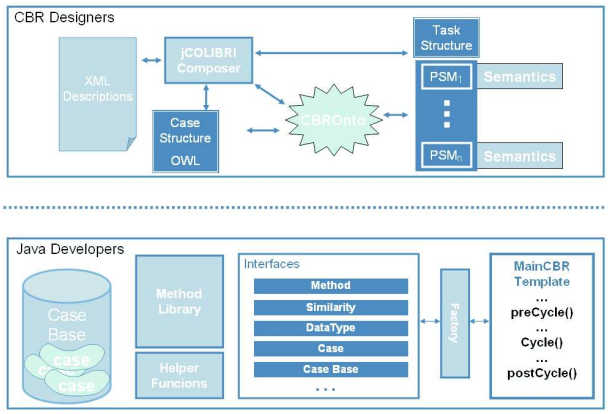

Fig. 1. Two layer architecture for jCOLIBRI 2.0

each component. But this is not enough in real applications. Imagine a real car sales recommender where the user can say: "I want a car with diesel or gas motor, I don't care about the number of doors but I don't like the blue or green color." This query implies more complex expressions like NOT, OR, ANY, NONE. The definition of the query can be even more complex if we also allow numerical comparison (like "less than 50000 km.").

6 New Architecture for jCOLIBRI 2.0

The key idea in the design of the new framework consists in separating core classes and user interface. That separation will give us the two layer architecture shown in Figure 1. The bottom layer contains the basic components of the framework with well defined and clear interfaces. This layer does not contain any kind of graphical tool for developing CBR applications; it is simply a white-box object-oriented framework that must be used by programmers. The top layer contains semantic descriptions of the components and several tools that aid users in the development of CBR applications (black-box with visual builder framework). It is important to note that the new framework design is going to be highly compatible backwards, so the transition to the new version will not cause any problems for current users.

The programmers' layer has new features that solve most of the problems described in Section 5. It takes advantage of the new possibilities offered by the newest versions of the Java language. The most important change is the representation of cases as *Java beans*[1]. A Java bean is any class that has a get() and set() method for each public attribute. With this change, developers can design their cases as normal Java classes, choosing the most natural design. This

[1] http://java.sun.com/products/javabeans/

simplifies programming and debugging the CBR applications, and the configuration files became simpler because most of the metadata of the cases can be extracted using a Java technology called Introspection. The similarity and weight information is now stored in separate configuration files. Java beans also offer automatically generated user interfaces that allow the modification of their attributes and automatic persistence into data bases and XML files. It is important to note that every Java web application uses Java beans as a base technology, so the development of web interfaces should be very straightforward.

The persistence of cases is going to be managed by the Hibernate[2] package. Hibernate can automatically store Java beans in a relational data base, using one or more tables. It also supports XML files as storage media and has many tools that aid the persistence mapping. Hibernate offers a standard and well documented query language that could be used to express the CBR queries.

Java Beans and Hibernate are core technologies in the Java 2 Enterprise Edition platform[3] that is oriented to business applications. Using these technologies in jCOLIBRI 2.0 we guarantee the future development of commercial CBR applications using this framework.

Another important change is the representation of tasks and methods. In the new architecture they will be coded as Java interfaces and classes that implement the interfaces. With this change, the application work flow is going to be controlled using Java code directly. As PSMs are going to be normal Java classes, they can interchange data through their parameters and be ordered using the Java code. This makes the blackboard unnecessary and gives maximum expressiveness, although it needs more effort in the designers' layer.

Regarding the top layer, it is oriented to designers and includes several tools with graphical interfaces. It is the black-box version of the framework. This layer helps users to develop complete CBR applications guiding the configuration process. To perform this task, the tools need semantic information about the Java components that compose the above layer. It is going to be done using Java Annotations, which is a new technology that allows to include meta-information in the source code. The annotation process is a technical subject that is not going to be discussed in this paper except to say that it is a very simple task. The annotations in the code are going to describe the behavior of a component with the corresponding information in CBROnto. Then, the GUI tools can read the components information and reason using CBROnto to aid users in the design of the CBR application.

An important feature of the designers layer is the representation of the case structure using OWL. Although cases are going to be Java beans, jCOLIBRI 2.0 is going to need semantic information about cases to aid in the composition of the CBR applications. Moreover, the CBR community is looking for a standard way for representing and interchanging cases so we are going to include the OWL based representation that we presented in [11].

[2] http://www.hibernate.org/
[3] http://java.sun.com/javaee/

7 Conclusions

In this paper we have argued for the necessity of a CBR framework and the main ideas behind the design of such a software system. We have made an analysis and classification of CBR applications trying to find the common components that must be included in a high quality framework. During the last few years our research group has developed the CBR framework jCOLIBRI and this paper presents the lessons learnt during this process. We point out the most important drawbacks of our approach proposing solutions based on the newest Java technologies. In our opinion, with these modifications jCOLIBRI could become a reference technology in the development of CBR applications both in the academic and industrial fields.

References

1. C. Abásolo, E. Plaza, and J.-L. Arcos. Components for case-based reasoning systems. *Lecture Notes in Computer Science*, 2504, 2002.
2. K.-D. Althoff, E. Auriol, R. Barletta, and M. Manago. *A Review of Industrial Case-Based Reasoning Tools*. AI Intelligence, Oxford, 1995.
3. B. Díaz and P. González. An architecture for knowledge intensive CBR systems. In *Advances in Case-Based Reasoning – (EWCBR'00)*. Springer-Verlag, 2000.
4. B. Díaz-Agudo and P. A. González-Calero. CBROnto: a task/method ontology for CBR. In S. Haller and G. Simmons, editors, *Procs. of the 15th International FLAIRS'02 Conference*. AAAI Press, 2002.
5. M. E. Fayad, D. C. Schmidt, and R. E. Johnson. *Building application frameworks: object-oriented foundations of framework design*. John Wiley & Sons, Inc., New York, NY, USA, 1999.
6. P. Funk and P. A. González-Calero, editors. *Advances in Case-Based Reasoning, 7th European Conference, ECCBR 2004, Madrid, Spain*, volume 3155 of *Lecture Notes in Computer Science*. Springer, 2004.
7. P. A. González-Calero, B. Díaz-Agudo, J. A. Recio-García, and J. J. Bello-Tomás. Authoring Tools in JColibri. In B. Lees, editor, *Proceedings of the 9th UK Workshop on Case-Based Reasoning*, pages 1–11, December 2004.
8. M. Jaczynski and B. Trousse. An Object-Oriented Framework for the Design and the Implementation of Case-Based Reasoners. In *Procs of the 6th German Workshop on CBR*, 1998.
9. J. A. Recio-García, A. A. Sánchez-Ruiz, B. Díaz-Agudo, and P. A. González-Calero. jCOLIBRI 1.0 in a nutshell. A software tool for designing CBR systems. In M. Petridis, editor, *Procs. of the 10th UK Workshop on CBR*. CMS Press, 2005.
10. D. Roberts and R. E. Johnson. Evolving frameworks: A pattern language for developing object-oriented frameworks. In *Pattern Languages of Program Design 3*. Addison Wesley, 1997.
11. A. A. Sánchez-Ruiz, J. A. Recio-García, B. Díaz-Agudo, and P. A. González-Calero. Case structures in jCOLIBRI. Best Poster Award. Twenty-fith SGAI Int. Conf. on Innovative Techniques and Applications of Artificial Intelligence, UK, 2005.
12. J. Schumacher. Empolis Orenge – an open platform for knowledge management applications. In *1st German Workshop on Experience Management*, 2002.
13. I. Watson. *Applying case-based reasoning: techniques for enterprise systems*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1998.