CASE-BASED REASONING TOOLS FROM SHELLS TO OBJECT-ORIENTED FRAMEWORKS

Essam Abdrabou, AbdEl-Badeeh Salem

*Abstract: A Case-Based Reasoning (CBR) tool is software that can be used to develop several applications that require cased-based reasoning methodology. CBR shells are kind of application generators with graphical user interface. They can be used by non-programmer users but the extension or integration of new components in these tools is not possible. In this paper we analyzed three CBR object-oriented framework development environments CBR*Tools, CAT-CBR, and JColibri. These frameworks work as open software development environment and facilitate the reuse of their design as well as implementations.*

Keywords: Case-Based Reasoning, Case-Based Reasoning Shells, CBR, CBR Shells

ACM Classification Keywords: I.2.5 Expert system tools and techniques - Conference proceedings

Conference: The paper is selected from XIVth International Conference "Knowledge-Dialogue-Solution" KDS 2008, Varna, Bulgaria, June-July 2008

Introduction

CBR is based on psychological theories of human cognition [Watson, 1997]. It rests on the intuition that human expertise does not depend on rules or other formalized structures but on experiences.

Since late 70s many CBR applications have been developed by research institutes or industrial companies in order for solving specific domain problems. In addition, there are several tools or shells have been built to facilitate the building of a CBR application by non-programmer users. Most of these tools aim to provide Application Programming Interfaces (APIs) which provide a set of functions that deal with CBR algorithms and methodologies. They intended to help programmers to embed these APIs in their application development. In order to access more complex problems the research goes to provide an open development environment that lead users to more uniform tool at the level of design [Jaczynski & Trousse, 1998].

The concept of object-oriented frameworks has been introduced in the late 80's and has been defined as "*a set of classes that embodies an abstract design for solutions to a family of related problems, and supports reuses at a larger granularity than classes* [Johnson & Foote, 1988]".

Frameworks allow the reuse of both code and design for a class of problems, giving the ability to non-expert to write complex applications efficiently. A framework can be considered as a semi-complete application than can be specialized to produce custom applications [Bello-Tomas et al. 2004]. A framework can be applied in a wide range of domain, and can be enhanced by the adding of new components.

This paper gives a survey on some of CBR shells, shows the need for the development of CBR tools based on open framework environment and discusses three object-oriented based CBR frameworks CBR*Tools, CAT-CBR, and JColibri. The paper shows the importance for developers of CBR applications to move from shells to object-oriented frameworks that facilitate the reuse of their design as well as the code to implement the intended CBR application. The paper is divided into five sections. Section one is this introduction. Section 2 gives a theoretical background on case-based reasoning. Section 3 discusses some of the CBR shells and developing environments. Section 4 discusses the importance of moving to object-oriented frameworks and studies the three proposed frameworks. Finally, section 5 concludes for the work.

Theoretical Background

In case-based reasoning (CBR) systems expertise is embodied in a library of past cases, rather than being encoded in classical rules. Each case typically contains a description of the problem, plus a solution and/or the outcome. The knowledge and reasoning process used by an expert to solve the problem is not recorded, but is implicit in the solution. To solve a current problem: the problem is matched against the cases in the case base, and similar cases are retrieved. The retrieved cases are used to suggest a solution that is reused and tested for success. If necessary, the solution is then revised. Finally the current problem and the final solution are retained as part of a new case.

CBR is a five-step problem solving process. Figure 1 shows the CBR cycle.

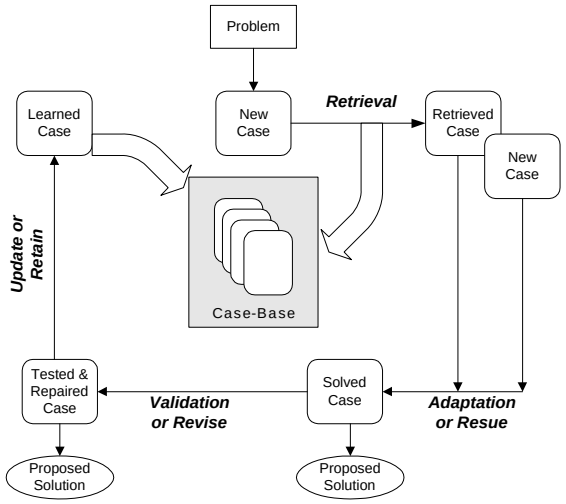

Figure 1: The CBR Cycle

Representation: Given a new situation, generate appropriate semantic indices that will allow its classification and categorization. This usually implies a standard indexing vocabulary that the CBR system uses to store historical information and problems. The vocabulary must be rich enough to be expressive, but limited enough to allow efficient recall [Kolodner, 1993].

Retrieval: Given a new, indexed problem, retrieve the best past cases from memory. This requires answering three questions: What constitute an appropriate case? What are the criteria of closeness or similarity between cases? How should cases be indexed? Indexing a case is essential in establishing similarity, because the indices help define the important elements of a problem, those that should be considered when studying the problem. Thus, part of the index must be a description of the problem that the case solved, at some level of abstraction. Part of the case is also the knowledge gained from solving the problem represented by the case [Kolodner, 1993].

Retrieving a case starts with a (possibly partial) problem description and ends when best matching cases found. The subtasks involve identifying a set of relevant problem descriptors, matching the case and returning a set of sufficiently similar cases; and selecting the best case from the set of cases returned.

Adaptation: Modify the old solutions to confirm to the new situation, resulting in a proposed solution. With the exception of trivial situations, the solution recalled will not immediately apply to the new problem, usually because the old and the new problem are slightly different [Kolodner, 1993].

Reusing the retrieved case solution in the context of the new case focuses on: identifying the differences between the retrieved and the current case; and identifying the part of a retrieved case that can be transferred to the new case. Generally the solution of the retrieved case is transferred to the new case directly as its solution case.

Validation: Determine whether the proposed solution is successful. Checking a solution can take many forms, depending on the domain. Whatever the means, after the system checks a solution, it must evaluate the results of

this check. If the solution is acceptable, based on domain criteria, the CBR system is done with reasoning. Otherwise, the case must be modified again, and this time the modifications will be guided by the results of the solution's evaluation [Kolodner, 1993]. Revising the case solution generated by the reuse process is necessary when the solution proves incorrect. This provides an opportunity to learn from failure.

Update: If the solution fails, explain the failure and learn it, to avoid repeating it. If the solution succeeds and warrants retention, incorporate it into the case memory as a successful solution and stop. The CBR system must decide if a successful new solution is sufficiently different from already-known solutions to warrant storage. If it does warrant storage, the system must decide how the new case will be indexed, on which level of abstraction it will be saved, and where it will be put inside the case-base organization [Kolodner, 1993].

Case-Based Reasoning Shells and Development Environments

CBR shells are kind of application generators with graphical user interface. They can be used by non-programmer users but the extension or integration of new components in these tools is not possible. There is a clear difference between a CBR application and a CBR shell. A CBR application is a direct implementation of CBR methodology to a specific domain problem in order to solve this problem. On the other hand, a CBR shell is an application that enables developers to develop a domain specific CBR application.

In the late 1980s, the U.S. DARPA program funded a series of workshops on CBR and the development of a CBR tools [DARPA, 1991]. This tool became Cognitive System's ReMind and marked the transition of CBR from purely academic research in cognitive science and artificial intelligence into the commercial area.

Many CBR shells have been developed to make the theory practically feasible [Watson, 1997]. Table 1 shows a summary of the key features of the major CBR shells. In addition to the previous tools there are three major CBR development environments CASPIAN, CASUEL, and CBR-Works.

Table 1: A Summary of the Major CBR Sheels [Watson, 1997]

Product	Platform	Representation	Retrieval	Interface
ART Enterprise	PC, Workstation	flat attribute:value pairs supporting a full range of variable types	Nearest-neighbor	Fully featured GUI builder
CaseAdvisor	PC Windows	Flat records supporting text and weighted questions	Nearest-neighbor and knowledge-guided	Use Netscape
CBR3	PC Windows	Flat records supporting text and weighted questions	Nearest-neighbor and knowledge-guided	CasePoint available as a DLL or API and CGI scripts
Eclipse	Any ANSI C environment	Flat attribute	Nearest-neighbor	No interface, only supply as a C library
ESTEEM	PC Windows	Case can be nested	Nearest-neighbor with inductive weight generation	Simple form-based GUI builder
KATE	PC Windows and UNIX	Hierachical cases	Nearest-neighbor and induction	ToolBook interface can be customized

CASPIAN [Pegler & Price, 1996] is CBR tool in the public domain developed at the University of Aberstwyth in Wales. It was used as the CBR component of the Wayland system. It has a simple command line interface, but can be integrated with a GUI front end if required. CASPIAN is written in C and can run on MS-DOS, MAC or UNIX but without the GUI. CASPICAN performs nearest-neighbor matching and used rules for case adaptation. It stores a case-base, including adaptation rules, in ASCII file. An individual case comprises a series of attributes and a solution. CASPIAN has an internal engine sophisticated enough to allow its use in industrial applications.

CASUEL [Manago et al., 1994], the Common Case Representation language developed by the European INRECA project (Integrated Reasoning from Cases), is the interface language between the INRECA component systems. It is also intended to serve as the interface language between the INRECA integrated system and the external world, and as a standard for exchanging information between classification and diagnostic systems that

use cases. CASUEL is a flexible, object-oriented, frame-like language for storing and exchanging descriptive models and case libraries as ASCII files. It is designed to model naturally the complexities of real cases. CASUEL represents domain objects in a class hierarchy using inheritance, slots being used to describe the objects, with typing constraints on slot values, as well as different kinds of relationships between objects.

CASUEL also supports rule formalism for exchanging case completion rules and case adaptation rules, as well as a mechanism for defining similarity measures. CASUEL is more concise than flat feature values vectors for representation of objects with a large number of potentially relevant attributes of different types, only a few of which are applicable to any given case. Its use reduces the number of information-gain calculations needed for induction systems or similarity computations required for case-based reasoning.

CASUEL does not require applications to use all of them. CASUEL is a keyword-driven language that allows different system components to ignore irrelevant definitions. CASUEL is also open in the sense that new features can be defined, if necessary for a particular kind of application of component [Watson, 1997].

CBR-Works can be seen as a CBR-Shell providing all necessary tools to model, maintain, and consult a case base [Schulz, 1999]. CBR-Works comes from the German company TECINNO, running on MS Windows, Mac, OS/2, and various UNIX platforms. Written in SMALLTALK, it supports an object-oriented model and flexible retrieval methods. It also supports the definition of concept and type hierarchies to help define similarity of symbolic concepts. CBR-Works includes an attribute editor, a rule editor, similarity criteria editor, distributed processing support and is easily integrated to existent applications. CBR-Works can import case-bases from Microsoft Excel and in the CASUEL case format.

Table 2 shows a summary of the three discussed CBR development environments.

Table 2: A Summary of the Major CBR Development Environments

Product	Platform	Representation	Retrieval	Interface
CASPIAN	DOS, MAC, or UNIX	Attribute-Value for feature representation	Nearest-neighbor	Can be integrated with a GUI
CASUEL	Portable	Frame-like language for storing and exchanging descriptive models as ASCII files	Not Applicable	Not Applicable
CBR-Works	MS Windows, MAC, or UNIX	Flat records supporting text and weighted questions	Nearest-neighbor with support of feature weights	Fully featured GUI

Case-Based Reasoning Object-Oriented Frameworks

Most of the CBR tools presented in scientific papers aim to provide Application Programming Interfaces (APIs) which provide a set of functions that deal with CBR algorithms and methodologies. They intended to help programmers to embed these APIs in their application development [Jaczynski & Trousse, 1998]. Usually these APIs can be extended by the programmer to modify the provided algorithms. However, none of these tools are designed to provide an open development environment that lead users to more uniform tool at the level of design [Jaczynski & Trousse, 1998].

The concept of object-oriented frameworks has been introduced in the late 80's and has been defined as "*a set of classes that embodies an abstract design for solutions to a family of related problems, and supports reuses at a larger granularity than classes*" [Johnson & Foote, 1988].

The goal of a framework is to capture a set of concepts related to a domain and the way they interact. In addition a framework is in control of a part of the program activity and calls specific application code by dynamic method binding. A framework can be viewed as an incomplete application where the user only has to specify some classes to build the complete application [Jaczynski & Trousse, 1998].

Frameworks allow the reuse of both code and design for a class of problems, giving the ability to non-expert to write complex applications quickly. Frameworks also allow the development of prototypes which could be extended further on by specialization or composition. A framework once understood, it can be applied in a wide range of domain, and can be enhanced by the adding of new components [Jaczynski & Trousse, 1998].

Before exploring the CBR frameworks, there are some points inside the framework that need to be addressed [Jimenez-Diaz & Gomez-Albarran, 2004]:

- Users must know the type of application which the framework can be used. Users should understand whether or not the application could be developed based on their choice of the framework.
- The mapping between application domain concepts and framework classes should be well studied to avoid the normal indirect mapping between domain entities and framework class.
- The framework users need to know behavior of elements within the framework in order to identify the hierarchy of classes that will be involved in the design of the application.
- Users need to study carefully the communication between the classes of the framework in order to avoid the integrity problem of the framework.
- Some problems like the duplication of functionality and extension of some parts of the framework can be avoided by the knowledge of framework architecture.

The following is a discussion of three object-oriented CBR frameworks CBR*Tools, CAT-CBR, and JColibri. We discuss their architecture and how CBR methodologies are applied in them.

CBR*Tools [Jaczynski, 1998] is an object-oriented framework for CBR which is specified with the Unified Modeling Language (UML) notation [Booch, 1994] and written in Java. It offers a set of abstract classes to model the main concepts necessary to develop applications integrating case-based reasoning techniques: case, case base, index, measurements of similarity, reasoning control. It also offers a set of concrete classes which implements many traditional methods (closest neighbors indexing, Kd-tree indexing, neuronal approach based indexing, standards similarities measurements). CBR*Tools contains more than 220 classes divided in two main categories: the core package for basic functionality and the time package for the specific management of the behavioral situations. The programming of a new application is done by specialization of existing classes, objects aggregation or by using the parameters of the existing classes.

CBR*Tools delegates each CBR step retrieve, reuse, revise or retain to a different object. Each class defines an abstract interface to a step of the reasoning while the Reasoner class defines how to control the reasoning. The step classes must be specialized to implement a specific reasoning. The Reasoner class allows the implementation of different reasoning control methods. In order to ensure that the reasoning step implementations and the reasoning object are consistent, the ReasonerFactory class is provided. Figure 2 shows the class diagram of CBR*Tools object model.

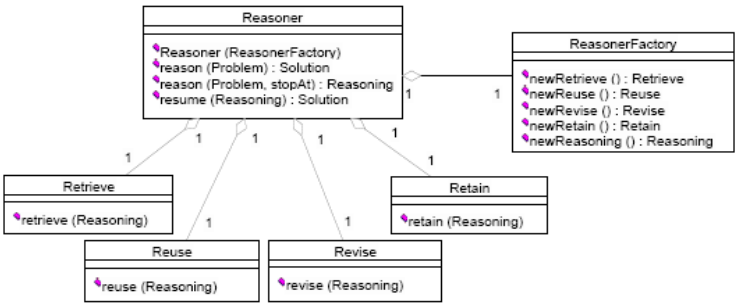

Figure 2: CBR*Tools Object Model

CAT-CBR platform uses a library of CBR components to guide the user in the development of a CBR application [Abasolo et al., 2002]. These components describe the different tasks that can appear in a CBR system and also the problem solving methods that can be applied to these tasks. The CAT-CBR platform has been developed on Noos platform [Arcos, 1997]. Noos uses feature terms as representation language.

Universal Problem-solving Methods Language (UPML) has been used to describe the CBR components used inside the framework [Abasolo et al., 2002]. Two levels can be differentiated in a component description: a specification level in which UPML is used and an operational level in which the Noos is used.

CAT-CBR uses two processes to enable users to develop a CBR application the configuration process and the operationalization process. The configuration process focuses on selecting different components and connecting them in order to specify an application. CAT-CBR has an interactive tool where users choose the components that need to be included in an application. This tool is built over a CBR system that guides and gives support to users during the configuration process. The operationalization process takes an application specification and generates an executable application. The platform generates a file that links with Noos methods following the structure of the configuration of components. Figure 3 shows the process of developing a CBR system. It is done in three steps: Configure, Enable, and Enact.

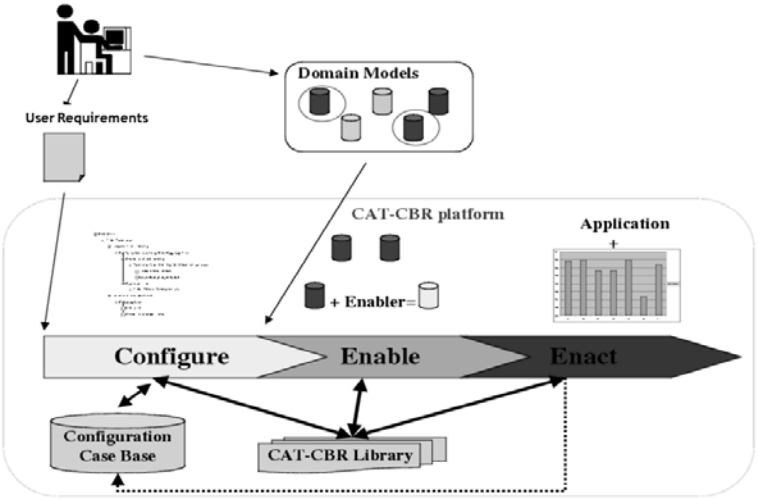

Figure 3: CAT-CBR Process of Developing a CBR System [Abasolo et al., 2002]

The goal of the Configure step is to decide which technique will be used in the CBR system. Only general information about the desired CBR system is required; this information is about general objectives (i.e. classify), or performance characteristics (i.e. noise tolerance). As result of the configure step, users get a configured CBR system; this configured CBR system is a task-method decomposition of components from the CAT-CBR library. This configured system specifies also which models will be used by each method.

The goal of the Enable step is to link the configured system with the concrete domain. In this step user have two options, first, they can assign the concrete models that the configuration needs to be carried out; second, they can use methods to acquire these models that the configuration needs and they are not currently available.

Enact: Finally in the Enacting step, the configuration and models will be translated into an executable code. As the platform is developed over Noos framework the resultant code will be Lisp functions. Once the configuration is operationalize, the application can run to solve new problems.

JColibri: The application framework of JColibri [Bello-Tomas et al., 2004] comprises a hierarchy of JAVA classes plus a number of XML files. The framework is organized around four main elements: tasks and methods, case-base, cases, and problem solving methods.

Tasks and Methods: XML files explain tasks supported by the framework and the methods to solve these tasks. Tasks are the key elements that represent the method goal and can identify it by name and description in an XML file. Users can add task to the framework at anytime.

Case Base: jColibri has a memory organization interface that assumes that whole case-base can be read into memory for the CBR to work with it. It is not feasible for big size. JColibri implemented a new interface who allows retrieving cases enough to satisfy a SQL query. A second layer of case base is a data structure which will organize cases after they loads into memory. The two layer approach is efficient enough to allow different strategies for retrieving cases.

Cases: jColibri represent cases in a very simple way. A case is individual which has number of relationships with other individuals. Framework is supported by different data types which define any simple case.

Problem Solving Methods: JColibri deals with the CBR methodology as follows:

- Retrieval: Main focus of methods in this category is to find similarity between cases. Similarity function can be parameterized through system configuration.
- Reuse: a complete design where case-based and slot-based adaptation can be hooked is provided.
- Revise: It is not supported by JColibri framework.
- Retain: Process of updating the case base is totally based on implementation of the case-base.

Conclusion

In this paper three object-oriented CBR frameworks have been studied CBR*Tools, CAT-CBR and JColibri. CBR*Tools is an object oriented framework implemented in JAVA. The framework identifies 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. CAT-CBR uses UPML for specifying CBR components and provides: a library of CBR components, a language and a graphical editor to specify new components and syntax to implement their operational code, and a broker service that allows specifying the requirements of a target CBR application and the available models in a domain. JColibri is an object-oriented framework implemented in Java. It uses XML to configure its data which make it interchangeable between computers. The development framework of JColibri supports the simple to most complex knowledge support.

The move to object-oriented CBR frameworks has many advantages including modularity which helps improve software quality, reusability that leverages the domain knowledge and prior effort of experienced developers in order to avoid recreating and revalidating common solutions, and extensibility which enhanced by providing explicit methods that allow applications to extend its stable interfaces. So, it is recommended for researchers or for industrial applications that need to solve their domain problems to use the available CBR frameworks.

Bibliography

[Aamodt & Plaza, 1994] A.Aamodt, and E.Plaza. Case-Based Reasoning: Foundational Issues, Methodological Variation and System Approaches. AICOM, Vol.7, No.1, pp. 39-58.

[Abasolo et al., 2002] C.Abasolo, E.Plaza, J.L.Arcos. Components for Case-based Reasoning Systems, In Topics in Artificial Intelligence, Lecture Notes in Artificial Intelligence, Vol. 2504, p. 1-12.

[Althoff et al., 1995] K-D.Althoff, E.Auriol, R.Barletta, M.Manago. A Review of Industrial Case-Based Reasoning Tools. In Goodall, A. (Ed.), An AI Perspectives Report. Oxford: AI Intelligence.

[Arcos, 1997] J.L.Arcos. The Noos representation language. PhD thesis, Universitat Politecnica de Catalunya.

[Bello-Tomás et al., 2004] J.J.Bello-Tomás, P.A.González-Calero, B.Díaz-Agudo. JColibri: An Object-Oriented Framework for Building CBR Systems. In Advances in Case-Based Reasoning, Lecture Notes in Computer Science. Springer Berlin/ Heidelberg, Vol. 3155/2004, p. 32-46.

[Booch, 1994] G.Booch. Object-oriented analysis and design. Redwood City, CA: The Benjamin/ Cummings Publishing Company, Inc.

[DARPA, 1991] Proceedings from the Case-Based Reasoning Workshop, Washington D.C., May 8-10, 1991. Sponsored by DARPA. Morgan Kaufmann, 1989.

[Jaczynski & Trousse, 1998] 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, Berlin.

[Jimenez-Diaz & Gomez-Albarran, 2004] G.Jimenez-Diaz, M.Gomez-Albarran. A Case-Based Approach for Teaching Frameworks: Universidad Computense de Madrid, Juan del Rosal 8. 28040 Madrid Spain.

[Johnson & Foote, 1988] R.E.Johnson, B.Foote. Designing Reusable Classes. Journal of Object-Oriented Programming, 1(2), 22–35.

[Kolodner, 1993] J.L.Kolodner. Case-Based Reasoning, California: Morgan Kaufmann Publishers.

[Manago et al., 1994] M.Manago, R.Bergmann, N.Conruyt, R.Traph ner, J.Pasley, J.Le Renard, F.Maurer, S.Wes, K.D.Althoff, S.Dumont. CASUEL: a common case representation language. ESPRIT project 6322,Task 1.1, Deliverable D1.

[Pegler & Price, 1996] I.Pegler, C.J.Price. CASPIAN: A Freeware Case-Based Reasoning Shell. In proceedings of the Second U.K. Workshop on Case-Based Reasoning, edited by I. Watson. Salford, UK: Salford niversity.

[Plaza & Arcos, 2000] E.Plaza, J.L.Arcos. "Towards a software architecture for case-based reasoning systems", Foundations of Intelligent Systems, 12th International Symposium, ISMIS 2000. Ras, Z. W. and Ohsuga, S., (Eds.), Lecture Notes in Computer Science 1932.

[Schulz, 1999] S.Schulz. CBR-Works- A State-of-the-Art Shell for Case-Based Application Building. Proceedings of the CWCBR-1999, Wurzburg.

[Watson, 1994] I.Watson. The Case for Case-Based Reasoning, Proceedings of EPSRC/DRAL, November 1994, pp.55-64.

[Watson, 1997] I.Watson. Applying Case-Based Reasoning: Techniques for Enterprise Systems. California: Morgan Kaufmann Publishers.

Authors' Information

Essam Abdrabou – General-manager, Cairo Engineering Support Labs CESLabs, 36 Al-Imam Ali St., Heliopolis-11351, Cairo, Egypt; e-mail: gm@ceslabs.com

AbdEl-Badeeh Salem – Professor; Faculty of Computer and Information Sciences, Ain-Shams University, Abbassia-11566, Cairo, Egypt; e-mail: absalem@asunet.shams.edu.eg