Towards Semi-Automatic Composition of CBR Systems in jCOLIBRI ⋆ ⋆⋆

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

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

Abstract. jCOLIBRI [14, 13, 3] is a framework to build CBR systems. To define the behavior of a new CBR system, the user has to compound reusable software components (Problem Solving Methods or PSMs [6]). Nowadays, this process must be done by hand although jCOLIBRI helps the user by showing only the PSMs that are applicable at each step. In this paper we describe some ideas to apply planning techniques to compound PSMs semi-automatically. This way, the planner will ask the user for the required information as needed, and the user will only have to answer to get a fully functional CBR system.

Keywords: Case-Based Reasoning, Planning, Reusable Software Components, Framework

1 Introduction

jCOLIBRI [1] is an object-oriented framework in Java for building Case Based Reasoning (CBR) systems. This framework tries to be a reference in the fast development of CBR systems, and therefore, it has been designed using a flexible and scalable architecture that promotes the reuse of components.

jCOLIBRI is a wide spectrum framework able to support several types of CBR systems: from simple applications based on retrieving using the nearest-neighbor approach, to knowledge-intensive ones with complex reuse and retain tasks. It is also possible to develop textual CBR systems using a specific and optional framework extension [12].

Framework instantiation is supported by a graphical interface that guides the configuration of a particular CBR system. This intuitive interface alleviates the steep learning curve typical for these type of systems, and allows CBR applications to be built even without deep programming knowledge.

Different tasks must be done in order to build a new CBR system (define the case structure, the similarity functions, ...) but the most important one is to define the system's behavior. To do this, the user must select and compound different software components (Problem Solving Methods or PSMs) from a library of reusable software components. Nowadays, this is made manually in jCOLIBRI and therefore, the user must know about all the PSMs to choose the most suitable for every occasion. Since everyone can collaborate providing their own PSMs to improve the library, the number of available PSMs could become very large soon, and will be very hard to know all of them.

Another problem related to the composition of PSMs are the implicit dependences among them. Different information can be interchanged among them using a common blackboard in which they can read and write data. In this way, there are dependences if one PSM needs some data that must have been generated by a previous PSM.

In this paper we propose the use of planning techniques to help the user in the composition process. The idea is to describe the PSMs and define the goal CBR system in a formal language, and

⋆ Supported by the Spanish Committee of Education & Science (TIN2006-15140-C03-02)
⋆⋆ Parcialmente financiado por la Dirección General de Universidades e Investigación de la Consejería de Educación de la Comunidad de Madrid y por la Universidad Complutense de Madrid (Grupo de investigación consolidado 910494)
[1] Project web page http://gaia.fdi.ucm.es/grupo/projects/jcolibri/ (LGPL licenced)

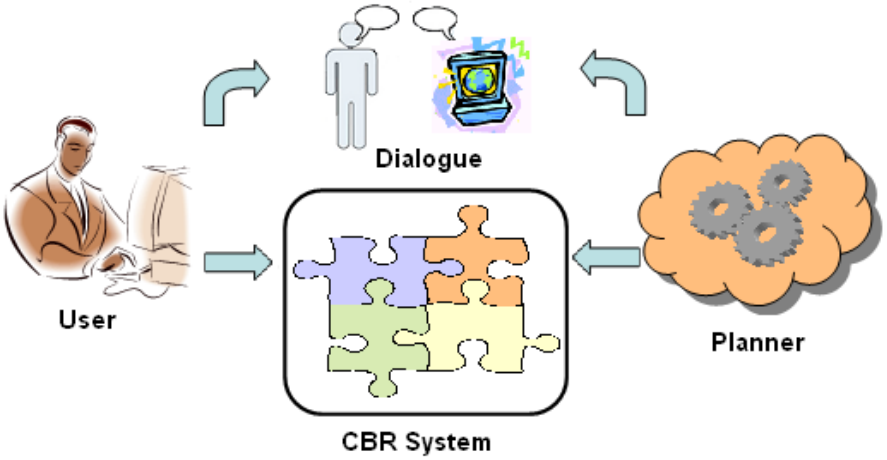

Fig. 1. Building a CBR system using an interactive approach

use a planner to manage the implicit dependences and to assist the user in the composition process. We propose to use a conversational approach in which the planner ask the user the information required to build a valid plan, and therefore, the final CBR system.

The next two sections explain the actual process of composition in jCOLIBRI and its limitations. Section 4 describes the main ideas about the HTN planning and the planner that we are going to use. Section 5 describes the necessary changes in jCOLIBRI for automatic composition of CBR systems and the limitations of this approach. Finally, in section 6 we try to solve these limitations using an interactive approach in which the user and the planner can collaborate to obtain the desired system.

2 Basic composition of CBR systems

In jCOLIBRI , the behavior of the new CBR system is defined combining software components or PSMs. Most approaches consider that a PSM consists of three related parts [6]. The *competence* is a declarative description of *what* can be achieved. The *operational specification* describes the reasoning process, i.e. *how* the method delivers the specified competence if the required knowledge is provided. And the *requirements* describe the knowledge needed by the PSM to achieve its competence.

The specification of a jCOLIBRI PSM is divided into several elements: *Name*, *Description*, *ContextInputPrecondition*, *Type* (decomposition or execution), *Parameters*, *Competences* (tasks solved by the PSM), *Subtasks* (generated by decomposition methods) and *ContextOutputPostcondition*. For our discussion, the most important elements are the *ContextInputPrecondition*, that describes the applicability requirements for the method, and the *ContextOutputPostcondition*, that represents the state after the method execution.

To create a new CBR system the user starts solving the general task *CBRSystemTask* that represents a generic CBR system. To solve it, a decomposition PSM can be used, and so, the initial task is decomposed into smaller sub-tasks. The process goes on, and these new tasks can be decomposed into smaller subtasks recursively until we reach trivial tasks that can be solved directly using resolution PSMs. The system is not completed until all the tasks have been solved by a method. An example of a possible decomposition task tree is shown in the Figure 2.

PSMs interchange data using a blackboard mechanism that we call *Context*. At each execution step the Context represents the actual state of the system, and contains all the shared data: query, cases, intermediate calculated results, etc.

We want to reuse the knowledge as much as possible among different kinds of CBR application, and therefore, we need to describe all that knowledge using a formal and domain independent language. In jCOLIBRI the PSMs and the *Context* are described in terms of CBROnto [4], a

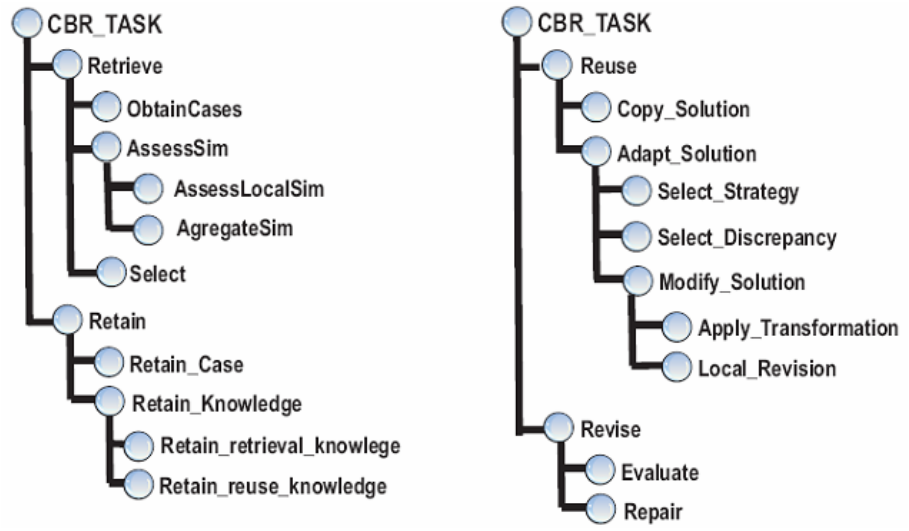

Fig. 2. A possible decomposition task tree of the CBR cycle.

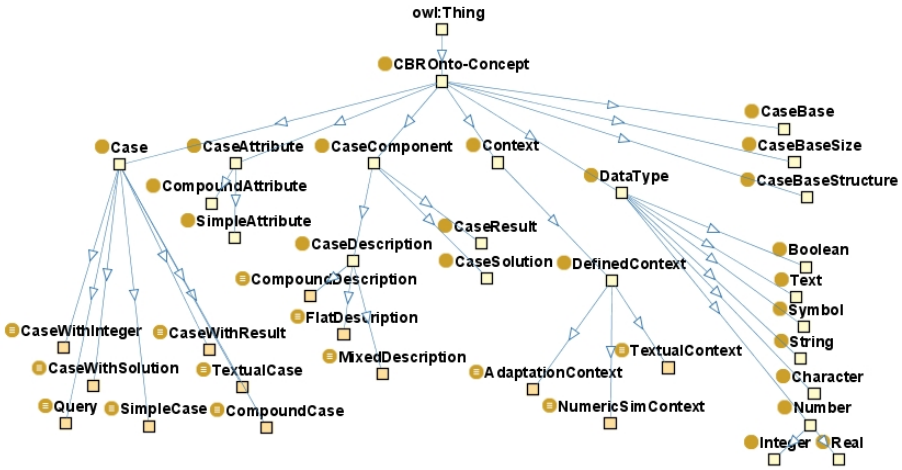

Fig. 3. Context definition subset of CBROnto

domain-independent ontology about CBR, using the OWL-DL standard formal language. Figure 3 describes the CBROnto subset for representing jCOLIBRI contexts (this figure only shows the *is-a* relations but the *context* object is related with the remaining concepts using several properties). The *ContextInputPrecondition*, *ContextOutputPostcondition* and *Competencies* are part of the PSM description and thus they are also represented as OWL-DL descriptions in CBROnto.

jCOLIBRI uses these elements to check if a PSM can be used to solve a task in a specific step of the composition process, taking care of the implicit dependencies because of the context. This is done in the following way: it uses a *context* instance C_i for representing the actual state of the CBR system. Each PSM contains a context precondition concept and returns a context postcondition instance. So, if we want to compose two PSMs PSM_i and PSM_{i+1} our framework must compute if the context instance C_i returned by PSM_i can be classified as an instance of the precondition concept of PSM_{i+1}.

The complete algorithm to guide the user in the composition process is described in [15] and follows these steps: (1) obtains the available methods; (2) looks for the current task into their *Competencies* list and selects the methods that can perform the current task; (3) computes the reasoning explained in the previous paragraph to offer only the applicable methods to the user; (4) and finally, the user chooses *manually* the method that will solve the current task.

3 Limitations of this approach

In the current framework, the choice of the method related to each task must be done by hand. The system shows the applicable methods and the user has to select the most suitable for every occasion. To do it, the user must know about all the methods. As jCOLIBRI has been designed as a collaborative framework where everyone can contribute providing their own PSMs to improve the library, the number of available PSMs could become very large, making hard to know all of them.

In addition, we have a more ambitious goal, we want to compound the PSMs to get the goal CBR system without the mediation of the user at each step of the process. Using planning techniques we want to be able to compound the PSMs starting from a description of the goal CBR system. As we will see, an automatic planning approach is not very useful to develop complex CBR systems, and it is better a semi-automatic approach in which the planner can ask the user the required information.

However, to apply planning techniques we need more detailed descriptions of the preconditions and postconditions of the methods. We could get these descriptions by adding concepts to CBROnto, but this would make the ontology too much complex. Instead of adding complexity to CBROnto we have chosen to describe the methods using custom-made first order logic formulas in the formal language of SHOP2, a hierarchical planner.

4 HTN planning and JSHOP2

The goal of HTN planning [5] is to find a sequence of actions that perform some task. To accomplish such task, the planner decomposes it into subtasks in a recursive process until achieving subtasks so simple that they can be solved directly. This planning technique has been used successfully in real and complex applications [19, 16, 9, 8].

SHOP2 [10] is a domain-independent HTN planner which won one of the top four awards in the 2002 International Planning Competition. JSHOP2 is an implementation of SHOP2 in Java. In order to work, JSHOP2 needs a description of the *domain model* and the *planning problem*. The *domain model* contains the tasks that the planner can perform, different ways to accomplish each one (*operators* and *methods*), and some axioms. The *planning problem* describes the initial state and the task or tasks that must be solved. The result of the planning process is a *plan*, that is, a sequence of actions that accomplish the goal tasks from the initial state.

There are two kinds of tasks: *primitive* and *non-primitive*. Primitive tasks can be performed directly using *operators*, and non-primitive tasks are decomposed into simpler subtasks using *methods*.

Definition 1. *An operator has the form* $(:\ operator h P D A [c])$ *where:*

- h *is the primitive task that this operator perform.*
- P *is the logical precondition that must be satisfied to apply this operator.*
- D *is the list of things that will become false after the execution of the operator.*
- A *is the list of things that will become true after the execution of the operator.*
- c *(cost) optional cost.*

Definition 2. *A method has the form* $(:\ method h [name_1] L_1 T_1 \dots [name_n] L_n T_n)$ *where:*

- h *is the compound task that this method can preform.*
- $name_i$ *is an optional name for the succeeding* $(L_i T_i)$ *pair.*
- L_i *is a logical precondition that must be satisfied to apply this method.*
- T_i *is a task list.*

5 Automatic composition of CBR systems

As we have explained before, the tasks in jCOLIBRI can be performed using two kinds of methods: decomposition and resolution. Decomposition methods decompose the tasks into smaller subtasks and resolution methods solve the tasks directly. When a user has to define the behavior of the new

CBR system, he begins using a method to solve the *CBRSystemTask*, usually by decomposition into the subtasks *CBRPreCycleTask*, *CBRCycleTask* and *CBRPostCycleTask*. This process goes on until all the tasks have been performed.

If we represent the tasks of jCOLIBRI as tasks of JSHOP2 and the methods of jCOLIBRI as operators and methods of JSHOP2, the problem of building a new CBR system can be represented as a planning problem that has as goal to perform the *CBRSystemTask*. The resulting plan is the sequence of jCOLIBRI methods (JSHOP2 methods and operators) that conform the CBR system. Presuming that the library of methods is stable, different CBR systems can be built providing different initial states to the planner in the description of the planning problem.

We can resume the process of representing the tasks and methods of jCOLIBRI in JSHOP2 as follows:

- Each task of jCOLIBRI will be represented as a compound task of JSHOP2.
- Each decomposition method of jCOLIBRI will be represented as a method of JSHOP2.
- To represent each execution method of jCOLIBRI we have to do several things. Let E_c be an execution method that performs the task T_c of jCOLIBRI , let T_s be the compound task of JSHOP2 related to T_c. We will create a new JSHOP2 method M_s that decomposes T_s into a primitive task P_s, and an operator O_s that solves P_s. The operator O_s represents the jCOLIBRI method E_c, but we need an intermediate step because in jCOLIBRI the same task can be performed by both an execution and a decomposition method, and though in JSHOP2 each kind of task only can be accomplished in one way.

In addition, we have to define the necessary predicate symbols to describe the precondition and postconditions of the methods. A few examples of these predicates are:

- *conectorFile ?x*: x is a file that describes how to read and write cases from the persistence media.
- *activePackage ?x*: the tasks and methods of the package x are available.
- *oncontext ?x*: x is available in the context and the methods can access to it.

Example 1. In jCOLIBRI the task *CBRCycleTask* represents the CBR cycle and is performed using a decomposition method (of the core package) called *CBRMethod* that decomposes it into the tasks *ObtainQueryTask*, *RetrieveTask*, *ReuseTask*, *ReviseTask* and *RetainTask*. We can represent this in JSHOP2 using the following code:

```
(:method (CbrCycleTask)                              ; Non-primitive task to solve
  CBRMethod                                          ; Optional method's name
  ((activePackage core))                             ; Preconditions
  ((ObtainQueryTask) (RetrieveTask) (ReuseTask)      ; Subtasks
   (ReviseTask) (RetainTask))) ; Subtasks
```

Example 2. In jCOLIBRI the task *ObtainCasesTask* loads the case base in memory, and can be performed using the execution method *LoadCaseBaseMethod* (of the core package). This method has a file as input parameter, and such file describe how to read the cases from text files, databases, etc. We can represent this in JSHOP2 using the following code:

```
(:method (ObtainCasesTask)                           ; Non-primitive task to solve
  LoadCaseBaseMethod                                 ; Optional method's name
  ((activePackage core) (connectorFile ?cf))         ; Precondition
  ((!LoadCaseBaseMethodOp ?cf)))                     ; Subtasks

(:operator (!LoadCaseBaseMethodOp ?cf)               ; Primitive tasks to solve
  ((connectorFile ?cf))                              ; Precondition
  ()                                                 ; Delete list
  ((casebase casebase1) (oncontext casebase1)))      ; Add list
```

As we have explained before, the planner needs a domain model and a planning problem to work. The domain model is built using the description of all the tasks and methods. The planning

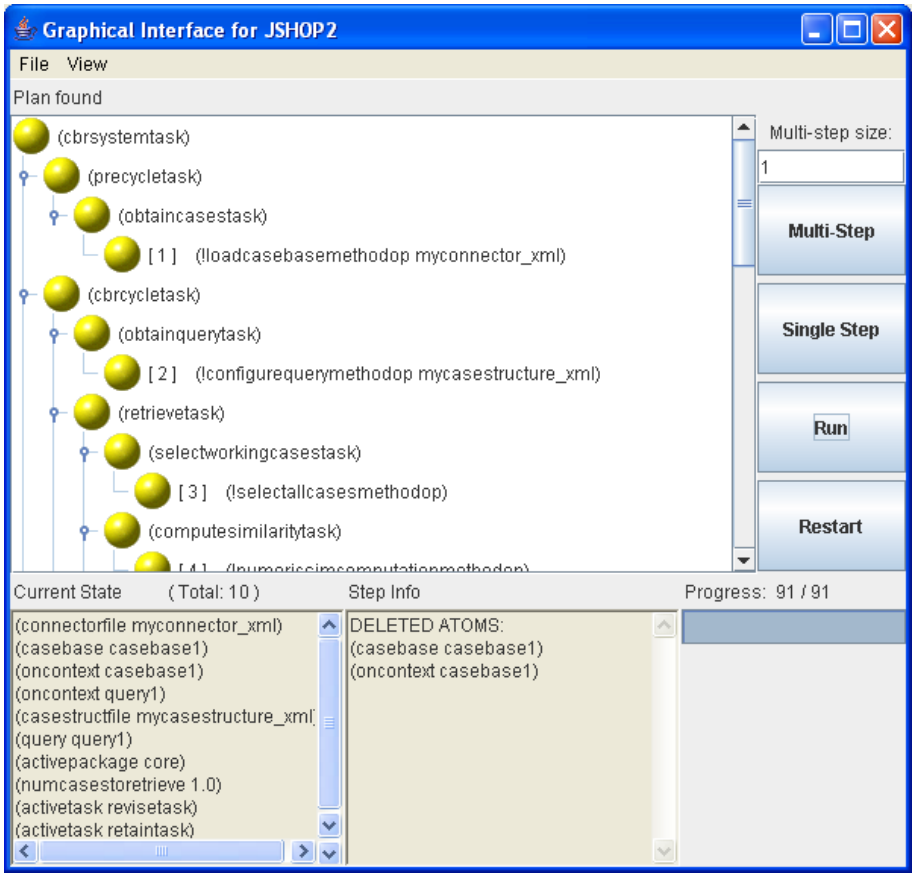

Fig. 4. JSHOP2 interface during the planning process

problem describes the initial state and the goal task. In this approach the goal task is always *CbrSystemTask*, and the initial state constrains the kind of CBR system that we want to build. The initial state contains predicate symbols that describe the active packages, the configuration files, the value of some method parameters, the initial context, etc. Figure 4 is an example of a decomposition process made by JSHOP2 taking as input a complete description of the initial state.

This has the disadvantage of having to describe all the required information in the initial state, in other words, at the beginning of the process. Usually, when the user begins to develop a system, he does not know all the information that is required to build the system from the beginning to the end, and without that information the automatic composition is not possible.

6 Towards the semi-automatic composition

The automatic composition approach only allows to build very simple CBR system because the user has to provide all the required information before the planner execution. In addition, if the user forgets to introduce some information in the initial state, the planner will work as the missing information were false due to the assumption of a closed world. It would be great if the planner could use the current information to explore all the available possibilities, and the same planner could ask the user other information about the initial state as necessary. This way, to build a system, the user only has to answer some questions and the planner will develop the rest.

Kuter et all. [7] use the SHOP2 planner to automatically compose Web Services described using the OWL-S service ontologies (there are several works around using SHOP2 to compound Web Services [1, 20]). They present *ENQUIRER*, an HTN-planning algorithm designed for planning domains in which the information about the initial state of the world may not be complete, but it is discoverable through queries to other Web Services. Our problem is very similar and we plan to

use some ideas of this algorithm, however there is one important difference: in our problem it is a human who is going to answer the questions.

In this approach, the input of the planner is a domain model and an incomplete-information planning problem. Such a planning problem has a set of ground atoms that are initially known, and a list ASK of logical atoms that are eligible to be queried during the planning process. The ASK list represents the information that the planner can obtain although this information is not available at the beginning of the planning process. The planner may need the same information several times but we only want to ask each question one time, so the planner will use a ANS list to keep all queries that have been answered.

When the planner asks a question, the user can take some time to answer, and this time can be used by the planner to search other possibilities in the search space. When the answer is available, the planner could decide to go back and use that information. We will use an $OPEN$ list to keep the information of the leaf nodes of the search tree generated during the planning process. This information can be stored as tuples (J, T, π), where J is a (possibly) incomplete state, T is a task list, and π is a plan.

The algorithm works as follows at each iteration:

- first check if the $OPEN$ list is empty and in that case return failure (all the possibilities have been explored and the is no solution).
- if there is a new answer we must update the information of each leaf node in the $OPEN$ list keeping the soundness of the plans. In addition we must store the pair (question, answer) in ANS.
- Choose a tuple (J, T, π) of $OPEN$ and remove it. If the task network T is empty we have found a valid plan because all the tasks has been performed. Otherwise, we choose a task t of T and we try to solve it using a method or an operator.
- If there is an action (method or operator) applicable that accomplishes the task t we apply it and we add to $OPEN$ the updated tuple that describes the new leaf.
- If there is no action applicable to solve t there is a precondition p that cannot be satisfied in J. We have to check if p can be asked in the ASK list. If is not we can conclude that p is false, but if p can be asked we must check if p has been queried before and the answer is in ANS.
- If the answer is in ANS, p must be false because p still could not be satisfied in J. If the answer is not in ANS there are two possibilities: p is been answered at this moment or this is the first time we try to ask it.

7 Conclusions

The literature about (semi-)automatic composition of software systems using reusable software components shows a moderately turbulent history [18]. These last years, the boom of Semantic Web Services has promoted the study of planning techniques to make easier the use of this potentially huge library of distributed components [17, 7]. A valuable consequence of these works is the creation of some standard semantic languages (OWL) and ontologies (OWL-S) to annotate the Web Services.

On the other hand, to make easier the use of a framework, different kinds of supporting tools has been developed. Usually these tools are based on documentation techniques like *recipes* and *cookbooks* [11] or even in classic planning techniques [2].

In this paper, we propose to use modern planning techniques like hierarchical planning (HTN), that have been tested usefully in real problems, and rich knowledge representation of the domain, using ontologies and standard semantic languages like OWL, to support the user in the instantiation process of the jCOLIBRI framework.

When a user wants to create a new CBR in jCOLIBRI , he has to define the system behavior compounding reusable software components (PSMs). Actually, this process must be done by hand although jCOLIBRI helps in the process showing only the methods that are applicable at each step. To do it, all the methods in jCOLIBRI have preconditions and postconditions expressed in a description language and using the concepts of CBROnto, an ontology about CBR. We have described the required changes to apply planning techniques to this process and compound the PSMs in a semi-automatic way. Using this approach the planner asks the user for the required

information as needed, and the user only has to answer the questions to get a fully functional system.

In the future work we will implement and explain in detail the algorithm used to get this conversational approach. We also have to resolve the problem related to the communication between the planner and the user, because it is not very intuitive to the user to understand questions with logic formulas and answer using the same formal language. Our goal is to make the process easier for the user, and we should find an intuitive interface of communication, maybe using natural language or graphic schemes. It is also necessary to evaluate if this approach really helps in the development of real CBR system. This study should take into account different kinds of users and different kinds of systems. jCOLIBRI is an open source project available in sourceforge so we will try to get some feedback from users.

References

1. T.-C. Au, U. Kuter, and D. S. Nau. Web service composition with volatile information. In *International Semantic Web Conference*, pages 52–66, 2005.
2. M. R. Campo, J. A. D. Pace, and F. U. Trilnik. "computer, please, tell me what i have to do...": an approach to agent-aided application composition. *J. Syst. Softw.*, 74(1):55–64, 2005.
3. B. Díaz-Agudo and P. A. González-Calero. An architecture for knowledge intensive cbr systems. In *EWCBR*, pages 37–48, 2000.
4. B. Díaz-Agudo and P. A. González-Calero. Cbronto: A task/method ontology for cbr. In *FLAIRS Conference*, pages 101–105, 2002.
5. K. Erol, J. A. Hendler, and D. S. Nau. UMCP: A sound and complete procedure for hierarchical task-network planning. In *Artificial Intelligence Planning Systems*, pages 249–254, 1994.
6. A. Gómez-Pérez. Knowledge sharing and reuse. In Liebowitz, editor, *The handbook on Applied Expert Systems*. CRC Press, 1998.
7. U. Kuter, E. Sirin, B. Parsia, D. S. Nau, and J. A. Hendler. Information gathering during planning for web service composition. *J. Web Sem.*, 3(2-3):183–205, 2005.
8. B. Morisset and M. Ghallab. Learning how to combine sensory-motor modalities for a robust behavior. In *Revised Papers from the International Seminar on Advances in Plan-Based Control of Robotic Agents,*, pages 157–178, London, UK, 2002. Springer-Verlag.
9. H. Muñoz-Avila, D. W. Aha, D. S. Nau, R. Weber, L. Breslow, and F. Yaman. Sin: Integrating case-based reasoning with task decomposition. In *IJCAI*, pages 999–1004, 2001.
10. D. S. Nau, T.-C. Au, O. Ilghami, U. Kuter, J. W. Murdock, D. Wu, and F. Yaman. Shop2: An htn planning system. *J. Artif. Intell. Res. (JAIR)*, 20:379–404, 2003.
11. W. Pree. *Design patterns for object-oriented software development*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1995.
12. J. A. Recio, B. Díaz-Agudo, M. A. Gómez-Martín, and N. Wiratunga. Extending jcolibri for textual cbr. In *ICCBR*, pages 421–435, 2005.
13. J. A. Recio-García and B. Díaz-Agudo. An introductory user guide to jcolibri 0.3. Technical report, Dep. Sistemas Informáticos y Programación, Universidad Complutense de Madrid, Spain, 2004.
14. J. A. Recio-García, A. Sánchez-Ruiz-Granados, B. Díaz-Agudo, and P. González-Calero. jcolibri 1.0 in a nutshell. a software tool for designing cbr systems. In M. Petridis, editor, *Proccedings of the 10th UK Workshop on Case Based Reasoning*, pages 20–28, University of Greenwich, 2005. CMS Press.
15. J. A. Recio-Garía, B. Díaz-Agudo, and P. A. González-Calero. A Distributed CBR Framework through Semantic Web Services. In M. Bramer, F. Coenen, and T. Allen, editors, *Research and Development in Intelligent Systems XXII (Proc. of the Twenty-fith SGAI Int. Conf. on Innovative Techniques and Applications of Artificial Intelligence, AI 2005)*, pages 88–101. Springer, 2005.
16. S. J. J. Smith, D. S. Nau, and T. A. Throop. Success in spades: Using ai planning techniques to win the world championship of computer bridge. In *AAAI/IAAI*, pages 1079–1086, 1998.
17. B. Srivastava and J. Koehler. Web service composition — current solutions and open problems. In *ICAPS 2003*, 2003.
18. S. van Splunter, N. Wijngaards, F. Brazier, and D. Richards. Automated component-based configuration: Promises and fallacies. In *Proceedings of the Adaptive Agents and Multi-Agent Systems workshop at the AISB 2004 Symposium*, pages 130–135, 2004.
19. D. E. Wilkins. *Practical planning: extending the classical AI planning paradigm*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988.
20. D. Wu, B. Parsia, E. Sirin, J. A. Hendler, and D. S. Nau. Automating daml-s web services composition using shop2. In *International Semantic Web Conference*, pages 195–210, 2003.