MULTI-PHASE ADAPTATION FOR BROKER AGENTS

M. M. EL-KHOULY, Ph.D.
Helwan University, Faculty of Science
40 Tregenna Avenue, South Harrow, HA2 8QT, U.K.
email: melkhouly@yahoo.com

Abstract - Many important and useful applications for software agents require multiple agents on a network that communicate with each other. We often use the word "agent" to describe people who have a helping or assistive relationship to us – travel agents, secretaries, butlers. The job of such an agent is to act autonomously to satisfy goals that we may have, give us a greater sense of productivity and reduce our workload. In this article we'll use the word agent to describe software that plays a similar role – providing help, advice, and work as a broker who has several software components and can assign one for the user. Such agents can retrieve software components for reuse from their repositories. However, if the keywords of the retrieving disaccords, the retrieval can be failure even if the eligible components exist in the component repository. We propose an information retrieval technique to help in retrieving the software components from the repositories. This technique uses three levels for retrieving components from a repository and can adapt the retrieved components to suite with the required query. The advantage of this technique is that the exact match is not necessary to find a similar software component.

Keywords - Information Retrieval, Reuse, Software Agent, Adaptation, Case-Based-Reasoning.

INTRODUCTION

The fundamental principles of case-based reasoning (CBR) for problem-solving is that new problems are addressed by *retrieving* stored records of prior problem-solving episodes and *adapting* their solutions to fit new situations. Practical experience developing CBR systems has shown that it is difficult to establish appropriate case adaptation rules [1,8]. Currently, agents applying learning and adaptation have shown a great potential in the information retrieval applications [15]. In defining adaptation rules, a key problem is the classic operationality/generality tradeoff that was first observed in research on explanation-based learning [14]: Specific rules are easy to apply and are reliable, but only apply to a narrow range of adaptation problems. In information retrieval research area, research dealing with searching software libraries has principally focused on improving indexing [6,10,13]. Others have much attention towards automated methods for gathering information in response to a query from a user [2,3,7,12].

In our research, first we classify the software components into classes according to their functionality and store them in the repository. Second, we establish a new model containing three levels for retrieving components from a repository. In the first level, the retrieval will be done without adaptation, since it will be exact match or use matched components, however, in both the second and the third levels, the agent will adapt the retrieved components. We give an example of how to apply the above steps in an automatic programming area.

In the next section, we present the retrieval system mechanism, and later we explain the adaptation phases. Finally, the paper will be closed by stating the conclusions.

RETRIEVAL SYSTEM

The element of the retrieval mechanism responsible for the automatic modification of queries is called the Query Process. The query process permits the agent to choose one of three levels of accuracy, *exact match*, *match* and *similar*. "**Find**" and "**Similar**" functions for search had also been built [5]. "**Find**" function is responsible to retrieve *exact match* and *match* software components, while "**Similar**" function is responsible for retrieving *similar* software components. The query process uses the "**Find**" function to search at the name of functions in the repository about *exact match* software components. If no satisfactory component is found, the query process changes the order of parameters to search at the synonym function names and to find *match* software components. If still no satisfactory result is found, the answer returned to the agent is *null*. At that time, the query process replaces "**Find**" function with "**Similar**" function, which uses the data dictionary to find similar software components. The data dictionary contains "instance variables", "operations used", "formula", "similar function" and "definition" for each method in the repository. Retrieving its data dictionary's information expands the selected method. If it needs to be adapted, the agent applies the adaptation rules.

ADAPTATION RULES

Adaptation rules come into play after a software component has been retrieved. Most likely, this software component does not completely fit the requirements specified in the user's query. Some attributes of the retrieved component might differ somehow while others may exactly match the query. According to the differences, the retrieved component must be modified to become better suited to the current query. This adaptation may only need for instance variable (like *matched* components), or for formula (like *similar* components). We will call the adaptation for parameters ***first phase adaptation***, while we will call the adaptation for formula ***second phase adaptation***.

- FIRST PHASE ADAPTATION

We give here an example for Producing Program Automatically (PPA) [4]. Consider a student file which contains the name of students, their numbers, their degrees, etc. and we simply want to print out each class with its performance, and the total file performance at the end. So, the manager of the school, the user, will give to the trainee programmer just the input data file (which contains, student number, student name, class name and students' degrees), and the output list required (which contains some fields that do not appear at the input data file, i.e. class performance). The programmer will design a program data structure (as shown in Figure 1-b). Then, he/she supplies it to the system in the form of two dimensional array "node (row, column)" in which "node (1,1)" means the first node in the chart and "node (2,3)" means the third node at the second level in the chart. The system scans those nodes (except those on a mainstream of the JSP graph) to check whether they contain components which are known to the system or not? If not, then the system acts according to the following algorithm:

1- break the node's name (if necessary) to variable names.
2- if one of those names is related to the mainstream of the JSP graph, then it will be added to the variable names table which contains (level number, node number, the name of the mainstream, the node's parent and the child's name).
3- if not, then the system consults the software components repository using "**Find**" function to find the formula for the required function.
4- the system adapts this formula using child's name of the node (as shown in Figure 1-c).
5- step 1 to step 4 are repeated until scanning all nodes.

The result of the above algorithm is a complete program data structure, which then will be converted to the pseudo code list using the "**Converter**" function as shown in Figure 1-a [4]. However, if there is incomplete program data structure, the system consults the user to complete it. At this point, we are now developing a multi-agent-based system, which consults other software agent's repositories to complete the program data structure instead of consulting the user.

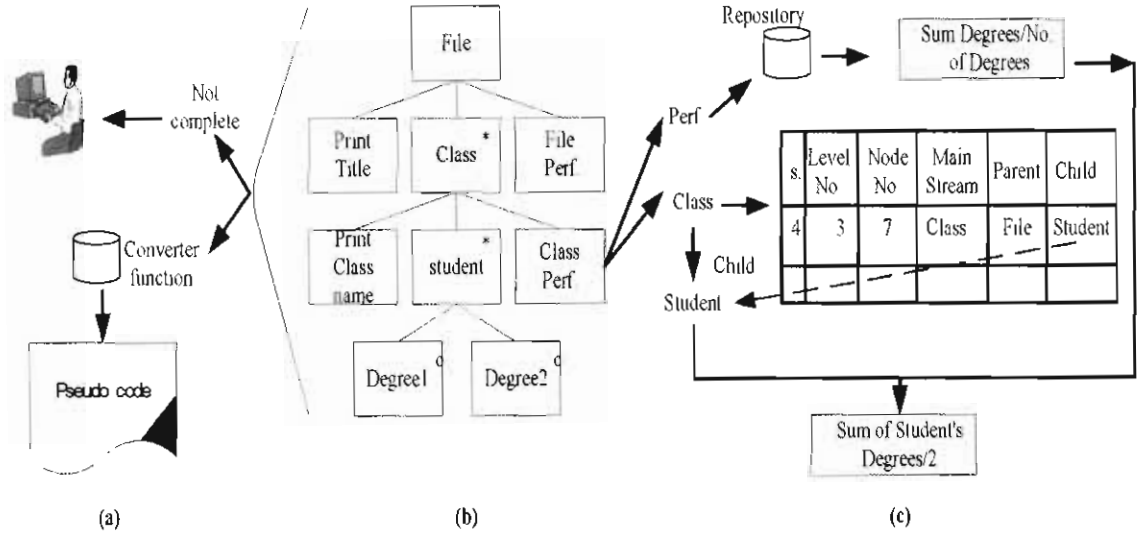

Figure 1: Producing program automatically.

- SECOND PHASE ADAPTATION

This phase will take place if the agent fails to find exact match or matched function, but only finds the *similar* function which needs to be adapted. This phase is concerned with adapting the formula of one function to be able to behave like the function the customer requires. This will be done by storing some relations using frame, these relations describe how one function's formula can be adapted to another formula.

- - FRAME SYSTEM AND DEFAULT VALUE

A frame is a structure like a database record with slots and fillers corresponding to fields and values. Each frame consists of slots, each of which represents a certain attribute of the object represented. Also, frames can be joined together by links to form frame systems. A frame can inherit values of slots from another linked frame called the parent frame. A default value means that unless there is a contradictory value assigned to the same slot, the slot will take the default value. The use of the default values allows the system to be more economical in terms of the memory used [9,11].

- - NESTED FRAME

Since we found that there is always a need for nested frames to allow the agent to reach the actual adaptation form of the function, we made use of nested frames in this system. The following example can clarify this point.

Suppose we would like to represent the following rule:

$$\text{Sin}(n\theta)=2\cos(\theta)\sin([n-1]\theta)-\sin([n-2]\theta)$$

Using nested frames, it may be shown as Figure 2.

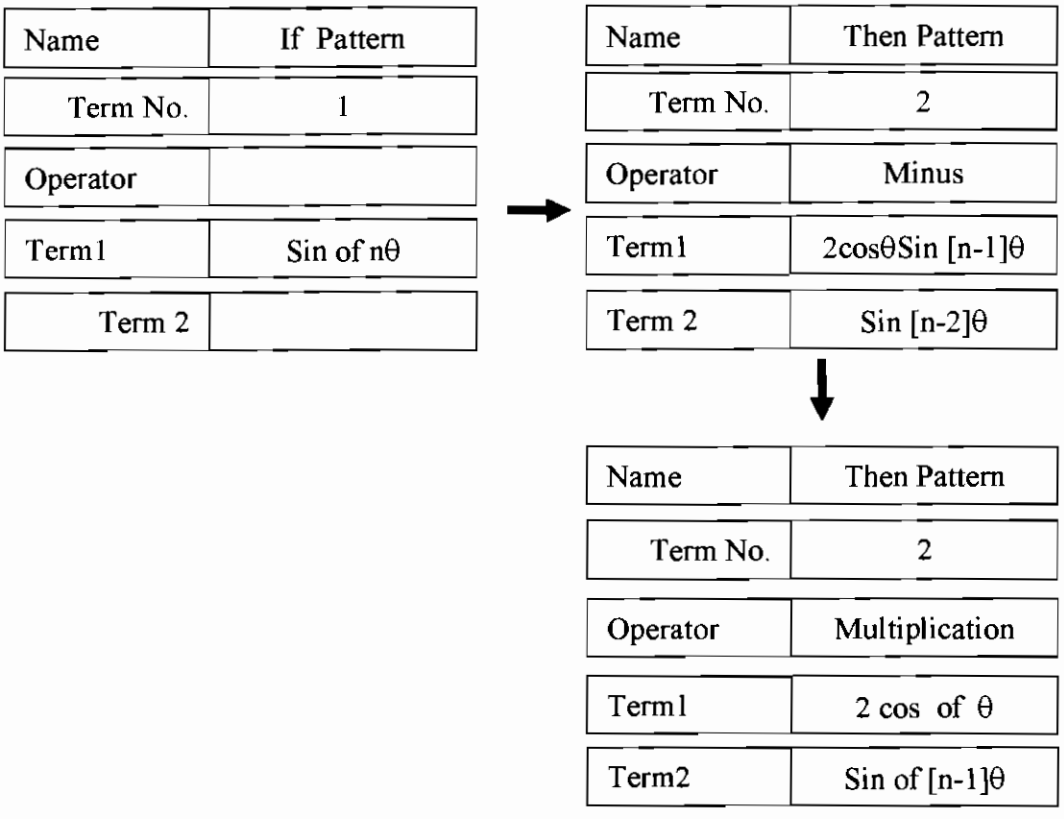

Figure 2: Nested frame for sin (n θ).

When inferring this rules the agent will recognize the following:

1- Subtraction is added to operator slot.

2- Only one term in input pattern, but there are two terms in output pattern.

3- The first term in the output pattern can be further decomposed to be two terms with multiplication operation.

- - ADAPTING FORMULA

Now, suppose that the query process searches $\sin^{-1}$ function, but the **similar** function found only sin function. So, the adaptation phase will take place to adapt the formula of sin function to behave like $\sin^{-1}$ function, given equation (1) from the retrieved record by similar function, and equation (2) from the frame slot as shown below:

$$\text{Sin}(x) = \Sigma ((-1)^k x^{(2k+1)}/(2k+1))!;\ k=0 \text{ to } \propto \quad (1)$$

$$\text{Sin}^{-1}(x) = 1/\sin(x) \quad (2)$$

CONCLUSIONS

The effectiveness of the reuse-based approach to software development is strongly dependent on the underlying classification scheme and retrieval mechanism. In this paper, we tried

to cover both, by enhancing the structure of the repository, and in the retrieving point, we described a new model consisting of two levels of retrieving, retrieveal without adaptation and with adaptation. The advantage of this model is that the exact keywords match is not necessary to find similar components. We also, presented the application of this new model to automatic programming area. The technique has the potential to be applied to multi-agent environments.

REFERENCES

[1] Allemang, D., "Review of the First European Workshop on Case Based Reasoning." *Case-Based Reasoning Newsletter*, Electronic Newsletter of AK-CBR, Vol. 2, No. 3, 1993.

[2] Arens, Y., et al., "Retrieving and Integrating Data From Multiple Information Sources." *In International Journal on Intelligent and Cooperative Information Systems*, Vol. 2, No. 2, pp.127-158, 1993.

[3] Bowman, M. C., et al., "Scalable Internet Resource Discovery: Research Problems and Approaches." *Communication of the ACM*, Vol. 37, No. 8, pp. 98-107, 1994.

[4] El-Khouly, M., et al., "A New Multi-Level Information Retrieval Technique for Reuse Software Components." *IEEE Int. Conf. On Systems, Man and Cybernetics, V-773-V777*, Tokyo, Japan, 1999.

[5] El-Khouly, M., et al., "Software-Agent for Reuse Software Components." in: *proceeding of Seventh Int. Conf. on Artifical Intelligence Applications*, pp. 299-306, 1999.

[6] Fraser, S. D., et al., "Software Indexing for Reuse." *Proc. IEEE Int. Conf. on Systems, Man and Cybernetics*, pp. 853-858, 1989.

[7] Fujita, Sh., et al., *Agent-Based Support for Reusing Components in Library, Knowledge-Based Software Eng.* Navrat, P. and Ueno, H., (Eds.) IOS Press, 1998.

[8] Leake, D., "Towards A Computer Model of Memory Search Strategy Learning." in: *proceedings of the Sixteenth Annual Conference of the Cognitive Science Society*, Atlanta, GA, pp. 549-554, 1994.

[9] Lee, F., and Heyworth, R., *Errors Due to Misperception and the Default-Value Model. Advanced Research in Computers and Communications in education*. G. Gumming et al., (Eds.), IOS Press, 1999.

[10] Maarek, Y. S., et al., "An Information Retrieval Approach for Automatically Constructing Software Libraries." *IEEE Transactions on Software Engineering*, Vol. 17, No. 8, pp. 800-813, 1991.

[11] Minsky, M., *A Framework for Representing Knowledge.* in: Branchman, R. J. & Levesque H. J. (Eds). Readings in knowledge representation. San Mateo, California: Morgan Kaufmann, 1981.

[12] Oates, T., et al., *Cooperative Information Gathering: A Distributed Problem Solving Approach.* Technical Report 94-66, Amherst: Dept. of Computer Science University of massachusetts, 1994.

[13] Prieto-Diaz, R., "Implementing Faceted Classification for Software Reuse." *CACM*, Vol 34, pp. 89-97, 1991.

[14] Segre, A., "On the Operationality/Generality Tradeoff in Explanation-Based Learning." in: *proceedings of the Tenth International Joint Conference on Artificial Intelligence Mailan*, Italy, 1987.

[15] Shan, F., et al., "A Programmable Agent for Knowledge Discovery on the Web." *Expert Systems*, pp.79-85, 2003.