Cooperative Agent-Based Support For Reusing Software Components

Mahmoud M. El-Khouly, Behrouz H. Far, Zenya Koono

{elkhouly,far,koono}@cit.ics.saitama-u.ac.jp

Dept. Of Information & Computer Sciences, Saitama Univ.,

Abstract

Many important and useful applications for software agents require multiple agents on a network that communicate with each other. 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.

In this paper, we present a new model for retrieving, and show how it is useful in automatic software design.

1. Introduction :

Research dealing with searching software libraries has principally focused on improving indexing [1,2,3]. Others have much attention towards automated methods for gathering information in response to a query from a user [4,5,6,7]. In our research we established a new model contains two levels for retrieving components from repository. The first level, retrieve it according to the component specification (name, similar function, ...) and we used a frame based representation to inherit the supper-class characteristics. The second one, used semantic network for checking the semantic of the retrieved component with the semantic required.

In section (2) we will describe our problem, in section (3) we present one of applications with a complete example, and give a discussion at section (4), and finally, section (5) presents the conclusion.

2. Problem Description :

The failure of component retrieving is mainly caused by the disaccord of component designers and other agents who want to reuse it. Each component has the corresponding specification (e.g. name, class, semantic, ...). Usually, the agents retrieve the appropriate components according to the name or functionality. However, if the keyword of the components at repository and the agents disaccords, the retrieval can be failure even if the eligible components exist in the component repository.

An item in our repository is a “class”, in the object-oriented sense. A class consists of a set of “methods” which define its functionality. Each Method has a set of “instance variables”, “formula”, “similar names”, “similar function” and “semantic”, as shown in figure 1.

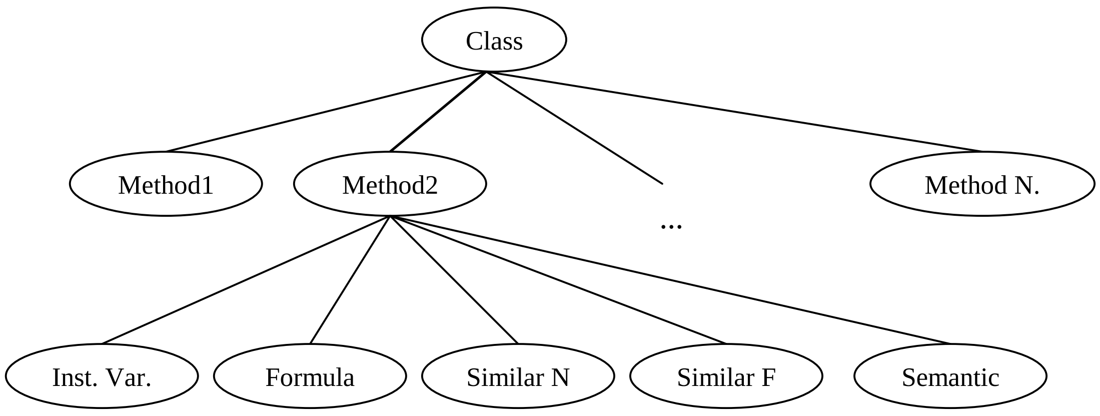


Figure (1) : Class hierarchy

In this approach, if the retrieving request dose not contain semantic check, then the agent scans the methods names from top to bottom and selects the first one that seems of sufficient interest. The selected method is expanded to allow an assessment of its functionality by a closer inspection of its variables and formula. If it needs to adapt, we generalize that method, by going up to its class, and then turn to other similar method in that class, which is more suitable.

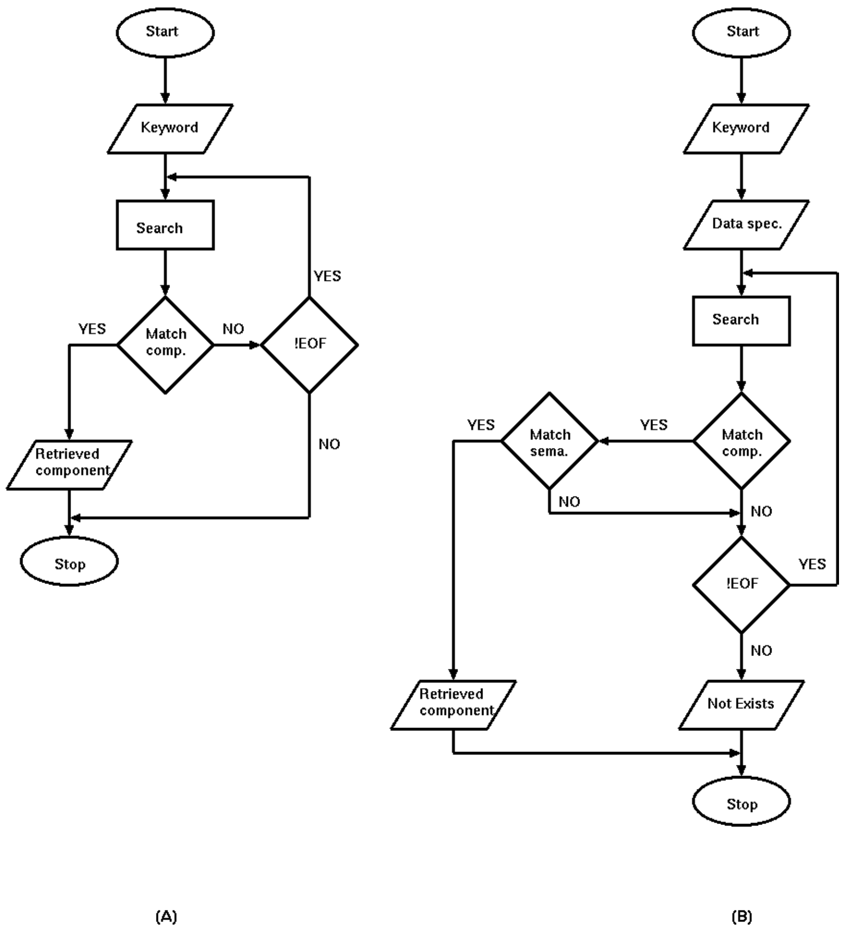
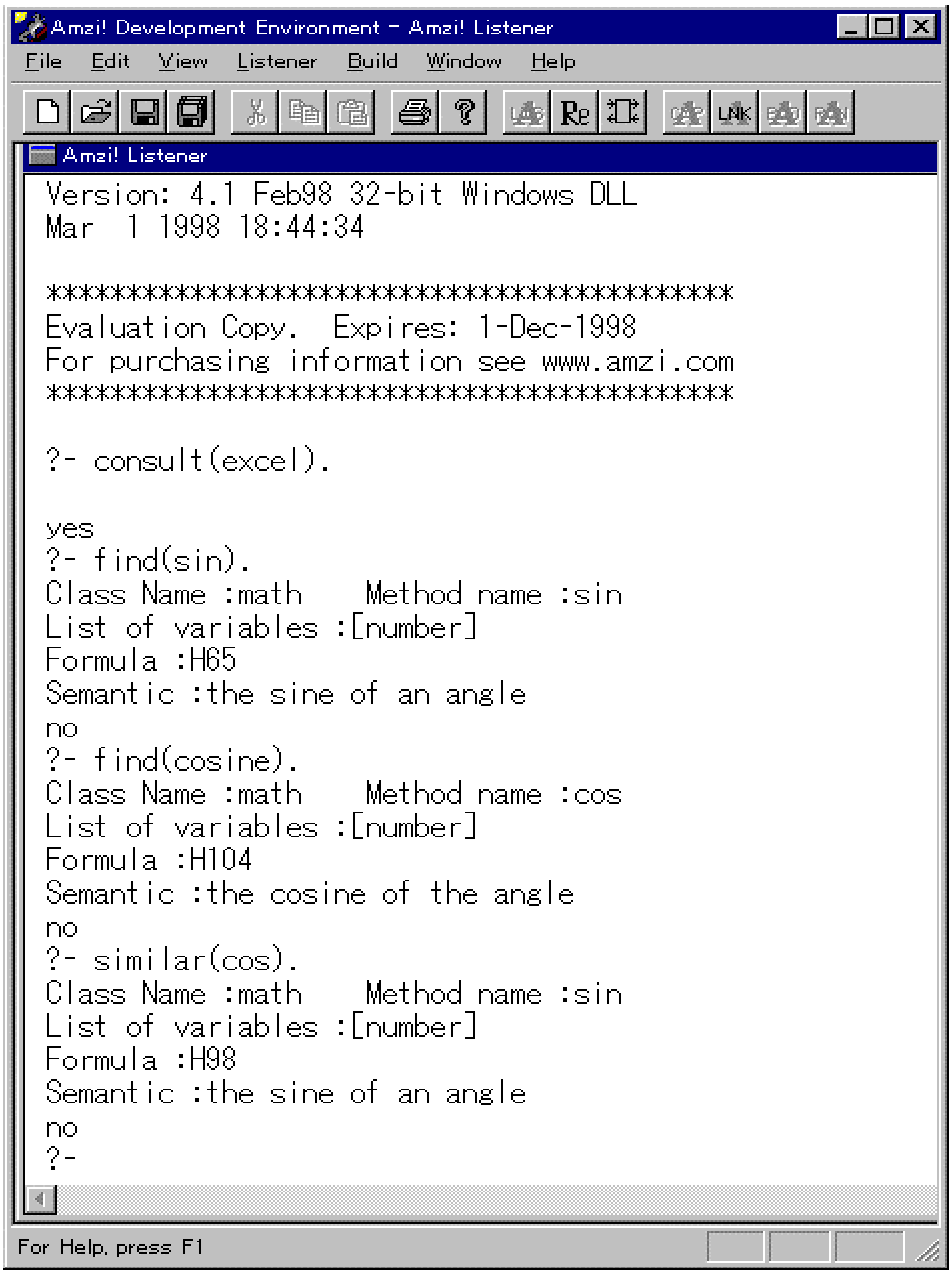


Figure (2): a- Retrieve without semantic check,
b- Retrieve with semantic check.

But, if the retrieving request contains semantic check, then we extract those methods which seem of sufficient interest, and then apply some kind of natural language processing method to check its semantic, to find a most suitable method for accurate retrieve.

Figure 2, shows the flow chart of search technique. While figure 3 shows the implementation of first level using AMZI PROLOG.



Amzi! Development Environment - Amzi! Listener
File Edit View Listener Build Window Help
Amzi! Listener
For Help, press F1

```
Version: 4.1 Feb98 32-bit Windows DLL
Mar  1 1998 18:44:34

********************************************
Evaluation Copy.  Expires: 1-Dec-1998
For purchasing information see www.amzi.com
********************************************

?- consult(excel).

yes
?- find(sin).
Class Name :math    Method name :sin
List of variables :[number]
Formula :H65
Semantic :the sine of an angle
no
?- find(cosine).
Class Name :math    Method name :cos
List of variables :[number]
Formula :H104
Semantic :the cosine of the angle
no
?- similar(cos).
Class Name :math    Method name :sin
List of variables :[number]
Formula :H98
Semantic :the sine of an angle
no
?-
```

Figure (3) : Results in AMZI PROLOG

3. Automatic Software Design :

We present here one of the above method's applications, which is useful in automatic software design. Since, the recent explosion in networked information resources has been exhilarating reuse of the software area, such that, generate a program source automatically, by reusing similar parts of the required program from other agents over the network. This assists programmers in terms of reducing the cost and time; instead of building their programs from scratch.

In this application, we integrate the properties of agent technology (e.g. negotiation, adaptation, ... etc.) with the properties of software engineering methods (e.g. hierarchical structure) to find a model to generate program source automatically. We used Jackson System Process (JSP) [8] as a systematic way of design, and because it is easy to understand from normal user (JSP's notations are shown in figure 4).

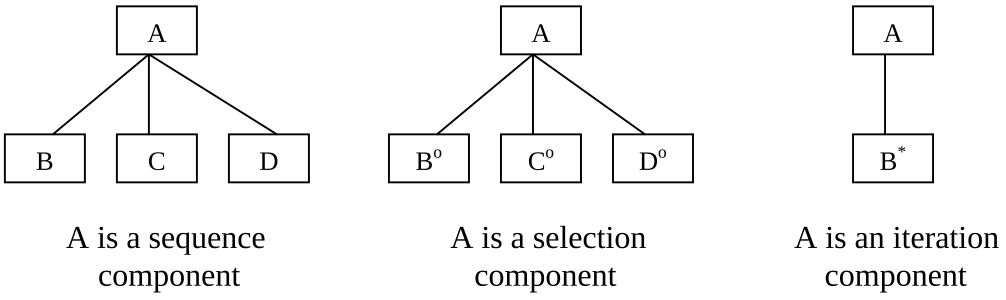

A
B
C
D
A is a sequence component
A
B^o
C^o
D^o
A is a selection component
A
B^*
A is an iteration component

Figure (4) : JSD's notations

3.1 Example :

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.

a) Assumptions :

We consider the following assumption for agent A :-

- Agent A is responsible for making pseudo code program for one problem, given input and output data structures in form of JSP for a specific problem.
- Agent A searches within its data dictionary about similar cases that match the current problem, in order to construct the program data structure.
- if agent A cannot construct program data structure successfully, it needs to know the function required to transfer that input to the required output, at that time agent A begins to negotiate with other agents using KQML language [9].
- After constructing program data structure, agent A uses converter rule base (for pseudo code) to produce the source program.

We consider the following assumption for other agents (for simplicity, we'll use one agent called Agent-B):-

- Agent B is a software house' dealer which sells software modules or functions to customers;
- As agent B wants to gain more customers. It has always a secondary goal, which is to make a customer satisfied, i.e. their program is correct. That will done by selling the only suitable module/function to the customers, and to apologize if the required function is not available at that time, or advice to purchase it from another agent(s);
- The profit here, is the customer satisfaction.

b) Approach :

First : we will provide input data structure and the output data structure to agent A.

Second : Agent A, will search in his repository about the suitable functions which required to transform the input data to the output data, as shown in figure (5).

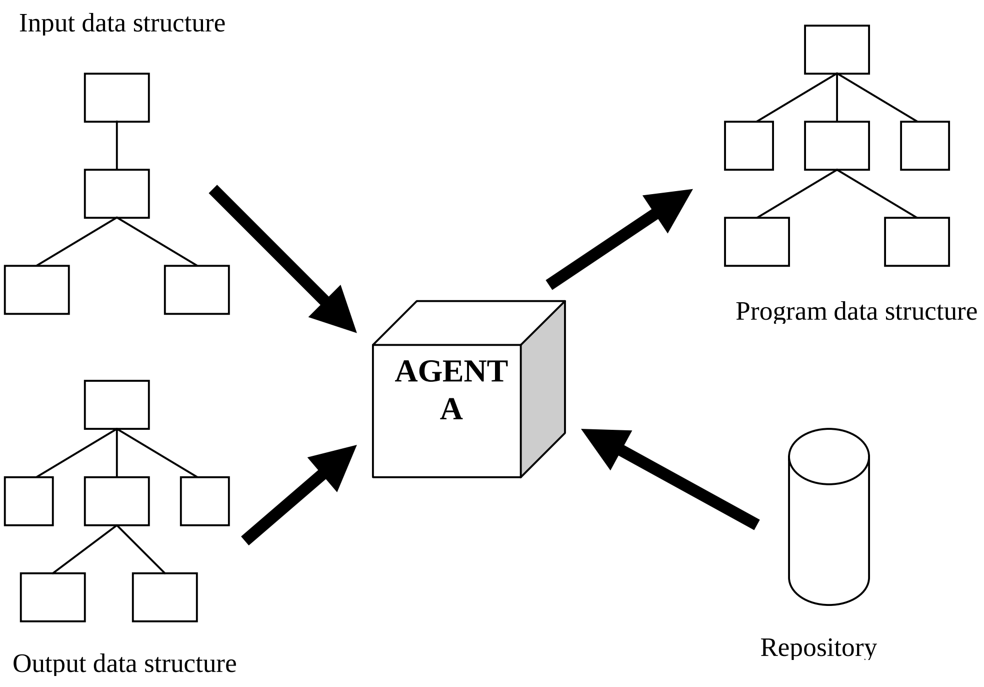


Figure (5) : Producing the program's data structure.

Third : If agent A doesn't complete his target, from his own repository then, it negotiates with other agents. The following scenario may take place :

a) asking broker to suggest one agent at Mathematical field

```
(recommend-one
            :language PROLOG
            :ontology Mathematics
            : sender Agent-A)
```

b) asking that agent about the required function

```
(ask-one
            :language PROLOG
            :ontology Mathematics
            :content (find (average(?x ?y)))
            :sender Agent-A
            :receiver Agent-B)
```

c) suppose that agent hasn't this function

```
(sorry
            :in-reply-to S1
            :sender Agent-B
            :receiver Agent-A)
```

d) repeat (a,b) until we get the following reply

```
(tell
            :language PROLOG
            :ontology Mathematics
            :content(=avg(sum variables/number of variables))
            :in-reply-to S1
            :sender Agent-C
            :receiver Agent-A)
```

Fourth : if they agree, then Agent A confirms program data structure, and then generate pseudo-code by using rules in associated converter rule base. (Figure 6)

Fifth : if not then go to step 3.

4. discussion :

As shown in the above example, only by providing an agent with input and output data structures, we got the pseudo code source program. We used KQML because, KQML is both a message format and a message-handling protocol to support runtime knowledge sharing among agents. KQML can be used as a language for an application program to interact with two or more intelligent systems to share knowledge in support of cooperative problem solving. [10]

At the end, if the agents agreed about some components. Agent A again refers to its repository to adapt what it got from other agents to its own problem..

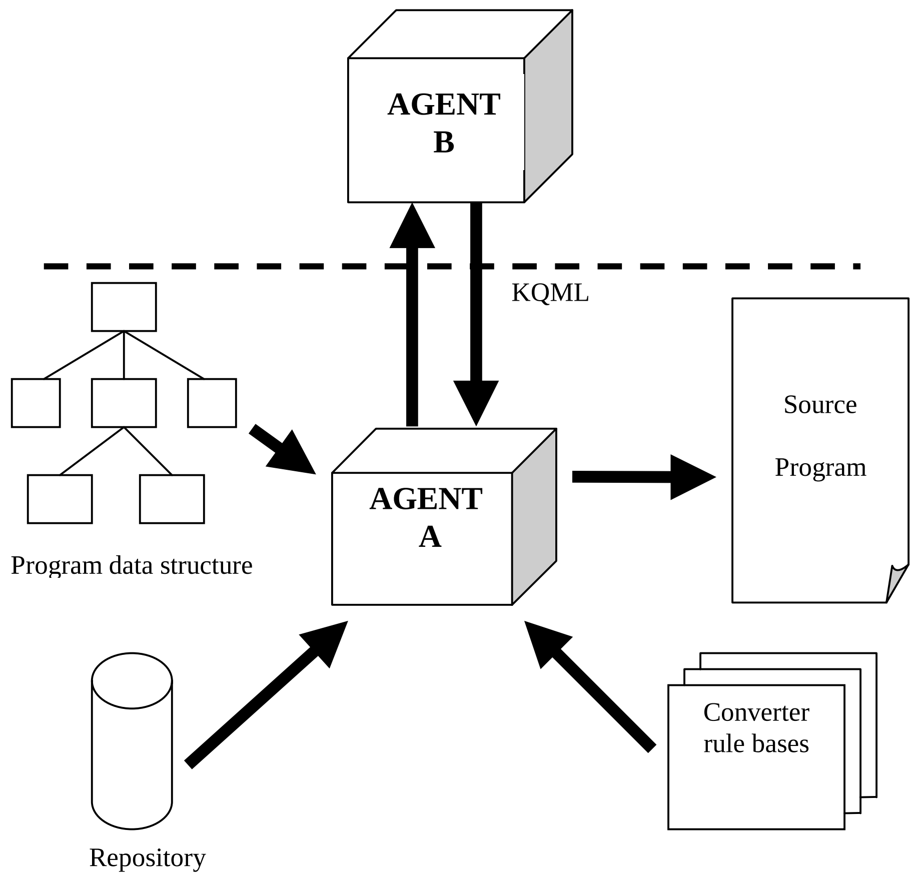


Figure (6) : Producing the source program.

5. Conclusion :

In this paper we described a new model for reusing components in a repository. The advantage of this model is that the exact keywords match is not necessary to find a component, and there is an ability to check the semantic of retrieved components. We also, presented the application of this new model at automatic software design area.

References :

[1] S.D. Fraser, J.M. Duran & R. Aubin, “Software Indexing For Reuse”, Proc. 1989 IEEE International Conference On Systems, Man and Cybernetics 1989, pp 853-858.

[2] R. Prieto-Diaz. “Implementing Faceted Classification For Software Reuse”, CACM Vol 34, 1991, pp 89-97.

[3] Y.S. Maarek, D.M. Berry & G.E. Kaiser, “An Information Retrievel Approach For Automatically Constructing Software Libraries”, IEEE Transactions On Software Engineering, Vol. 17, No. 8 Aug. 1991, pp 800-813.

[4] Arens, Y., Chee, C.Y., Hsu, C., and Knoblock, C.A., “Retrieving and integrating data from multiple information sources”, in International Journal on Intelligent and Cooperative Information Systems, 2(2), 1993, pp.127-158.

[5] M.C. Bowman, P.B. Danzig, U. Manber, and M.F. Schwartz, “Scalable Internet Resource Discovery: Research Problems and Approaches”, Communication of the ACM, 37(8), 1994, pp 98-107.

[6] Tim Oates, M.V. Negendra Prasad, V.R. Lesser, “Cooperative Information Gathering: A Distributed Problem Solving Approach”, Technical Report 94-66, Dept. Of Computer Science University of massachusetts, Amherst, 1994.

[7] Shiger Fujita, Hideki Hara, Kenji Sugawara, Tetsuo Kinoshita & Norio Shiratori, “Agent-Based Support for Reusing Components in Library”, Koweldge-Based Software Eng., P.Navrat and H.Ueno (Eds.) IOS Press, 1998.

[8] Jackson M.A., ``Principles of Program Design'', Academic Press, 1975.

[9] Finin, T., Fritzson, R., McKay, D. and McEntire, R., KQML - An Information and Knowledge Exchange Protocol, in Kazuhiro Fuchi and Toshio Yokoi, editors, Knowledge Building and Knowledge Sharing, Ohmsha and IOS Press, 1994.

[10] A. M. Zaremski & J. M. Wing, ”Specification Matching of Software Components”, ACM Trans. On S.W. Eng. And Math., Vol.6, No.4, October 1997, PP 338-369.