CSCLD: A Component for Software Component Library Discovering

SOFIEN KHEMAKHEM
Department of Computer Science
Higher Institute of Technological Studies of Sfax
P.B. 88A 3099 El boustan Sfax, Tunisia
Khemakhem_sofien@yahoo.fr

KHALIL DRIRA
LAAS-CNRS
7, Avenue du Colonel Roche
31077 Toulouse Cedex 4
Khalil.Drira@laas.fr

MOHAMED JMAIEL
LARIS Laboratory
Faculty of Economic Sciences and Management of Sfax
P.B. 1088 Sfax, Tunisia
Mohamed.jmaiel@enis.rnu.tn

Abstract: *Component-oriented software development is a promising solution for complex software systems like internet-based distributed service. The success of this type of development is explained by the re-use of the components. To improve this re-use, it is necessary to implement and maintain well-structured software component libraries and efficient search engine to guarantee the success of the research and the selection of the component, which fulfills the developer's requirements. This paper proposes a new approach to select software components from a library. Without leaving the environment of development, the developer loads Component for Software Component Library Discovering CSCLD, a persistent component that automatically locates and presents a list of software components that could be used in the current development situation. This Component is a specialized search engine that automatically generates a query from developer specification and indexes a database of software components*

Keywords: Software component, discover, CSCLD, non-functional constraints.

1 Introduction

The development of complex software systems is a difficult task. To reduce such a task, it is necessary to elaborate solutions to improve the re-use of the software. After object technology [1], we assist with the emergence of the component technology which proposes solutions for this problem [2, 3]. Component technology extends that of the object while preserving the benefit promised by the latter: reutilisability, productivity, quality and maintainability. In this context, the development of an application consists simply in adapting [4] and integrating the suitable component [5, 6]. Consequently, the re-use of the components is a justifying factor, because it allows to increase in an appreciable way the productivity of the software and to improve development quality.

During the application development, the programmer is faced with a significant number of various components categories. Consequently, the use of a software components library, having a clear structure, is a crucial criteria for the successful of the reuse. This allows the developer to seek and select efficiently the component which meets its need perfectly. To be useful, a library needs to be supported by: tools to create initial information structures, and flexible mechanisms to search and browse the library. In this context, many ap-

proaches have developed software component retrieval systems based on many techniques such as Artificial intelligence, neural network and Agent.

Since the late 1980s, AI-based techniques have been used extensively by component retrieval systems. These techniques have attempted to capture searchers domain knowledge and classification scheme knowledge and effective search strategies. Significant efforts are often required to acquire knowledge from domain experts and to maintain and update the knowledge base [7]. This knowledge base provides components representations and indexes software component library in order to ameliorate the accuracy during the search of components.

The neural network technique, which is a newer paradigm, has attracted attention of researchers in computer science, and in software engineering particulary. This newer technique, has provided greater opportunities for researchers to enhance the information processing and retrieval capabilities of current components storage and search systems.

More recently the software agent technique explores a new approach to locating software components. In fact, it ameliorates, speeds up the search for components and infers the search goal of software developers by observing their browsing actions and delivers components that closely match the inferred goal [8].

In this paper, we have developed a tool which help the developer to find the adequate component and an ontology for the component description. This tool and a component description library are incorporated in a software component that can be integrated in several development environments as C++, Vb, DELPHI, VISUAL Java and browse the library to find the appropriate component. By running this component, the developer formulates a query after selecting the criteria that response to his needs.

This paper is organized as follows: section2 and section3 present respectively the architecture and the implementation of the CSCLD. We will devote section4 to compare our approach with related technologies. Section5 deduces the advantages of the CSCLD through a discussion. In conclusion, we will suggest some openings and prospects related to this study.

2 Architecture

During the application development, the programmer sometimes needs a component that meets its current task. Our system provides a persistent and intelligent component that can be loaded in development environment only during the project creation. This component is persistent because it is always ready for execution by the developers, and intelligent because it contains not only the search process but also the software component description library.

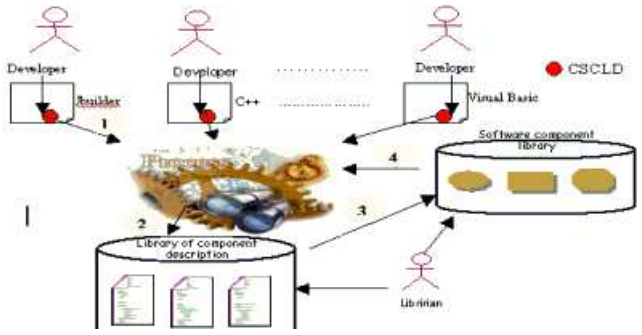

Figure 1: System architecture

Once CSCLD is running, the developer specifies the query by selecting the adequate criteria. An approximate comparison between the specified query and the description of components in the component description library is made by the *compare_query_description()* function. If there is a positive result, the system indexes the obtained description(s) to software component library. So the *search_component(ref_component[])* function retrieves the appropriate component(s), where *ref_component[]* is the list of the components references to retrieve. Then, the developer uses an application programme interface (API) to integrate the desired components in the current project.

The librarian can also manipulate the CSCLD in order to manage the two libraries. He can add, modify and delete the component or/and the component description. Component description is an ontological description that contains:

-Functional properties such as methods and communication objects.

-External properties such as type and author.

-Static non-functional properties such as Capacity and QoS.

-Dynamic non-functional properties such as reliability and availability.

The value for each criterion is also managed by the librarian. The developer has not this privilege; all he can do is to read and select the criteria or the adequate component.

3 Implementation

CSCLD is a component responsible for the location of re-usable components. It executes the specified query and retrieves and presents relevant components. CSCLD requires no loading from software developers in development environments. In current development practices, the developer clicks on the CSCLD icons. Next, he chooses the non-functional constraints and the functional information which meets his needs. A dynamic query would be formulated and then executed automatically in order to deliver the adequate component.

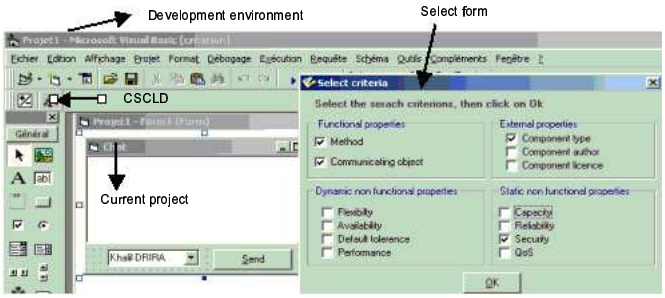

Figure 2: Search criteria

Figure 2 shows the development of a simple Chat application by using Visual BASIC. The Listbox component contains the identity of the recipient, the Text component contains the content of the message. The developer needs an ActiveX component that contains a method having two input parameters of string type that correspond to the address of the recipient, and the message. This component must also ensure the security of the message and establish the communication with the mailing server. To find this component, the developer selects the following search criteria: method and communicating objects as functional properties, component type as external property and security as static non-functional property. In the following step, a value is applied to each criterion previously selected. Thus, the developer can associates the label ”Send” or any other synonym label to the criterion method. A query will be then formulated automatically when clicking on the search button. The components-result of the research will be sorted according to the degree of the similarity with the query. The developer can read the description of each component to understand their functionality detail. Thus, the developer loads the adequate component -that approximately answers its need- in the chat application.

The retrieved re-usable components are presented in the search result form in decreasing order of similarity value. Each component is accompanied by its degree of similarity, name, access path and a short description. Developers who are interested in a particular component can load it in the tool bar, by a mouse click. Finally, the developer can adapt it to the task or re-use it, as it is in the program, depending on the value of the similarities degree. CSCLD provides also an interface for the librarian in order to add, remove components and modify the components information. We formulate many queries without introducing the non-functional constraints. We notice that in many cases the delivery component does not meet the developer needs. In fact, the non-functional constraints play a decisive role in the search quality.

4 Comparison of CSCLD with related technologies

Existing research techniques in software libraries adopt different retrieval methods. Based on a variety of technologies, they can be divided into agent, artificial intelligence, and neural network technology.

Agent technology is a recent technology used in many retrieval systems to facilitate the component search in the library.

In some approaches, the agent runs continuously in the background of a development environment and monitors the developer’s actions. The CodeBroker agent [8] infers software developers needs for components by comparing the contents of doc comments with methods signatures. It supports ”context-aware browsing” and measures the similarity between the specified query which is a comment in the emacs editor and the documentation of the components. The comparison distance is based on a probabilistic calculation. The information type in the documentation of each compo-

nent is functional. Whereas in [9] the agent infers the search goal of software developers by comparing their browsing actions and delivers components that closely match the inferred goal. In this approach the specification level is limited to the methods signatures and input/output parameters. The comparison distance is approximative.

In other approaches, agent is directly running by the developer, after query specification. Agora system[10] use an independent JavaBeans and CORBA agents in conjunction with AltaVista search Developer's Kit for indexing and retrieving component data. The Agora searches only on component interfaces, covering solely the component connectiveness problem. The component name and type are preserved as fields to enable searches by name and component type. Indexed search is the search style adopted. The functional information is the information type used in this approach.

AI-based technologies uses knowledge representation to represent and index components. The goal is to give semantic meaning to the representation in order to ameliorate the accuracy during the search of components.

CodeFinder [11] as well as LaSSIE [12] represent components as frames and have used the inferencing capabilities of inheritance and classification to retrieve information with Kandor representations. Kandor is the frame-based knowledge representation language that is used by CodeFinder to index components and create a frame-based hierarchy. CodeFinder organizes those frames into an associated network and uses spreading activation to find components whereas Frames in LaSSIE are organized into hierarchical, taxonomic categories. LaSSIE allows the use of natural language queries that can be converted into query frames. This query frame is placed by the search algorithm in the frame hierarchy and then matches all of the instances that a query frames subsumes.

The Knowledge-base in [13] is a base of frames where each software component has a set of associated frames. Those frames contain the internal representation of components description and other associated information (source code, executable examples, re-use attributes, etc). The same mechanism applied to the software descriptions is used to convert a query into a free text of an internal representation. The set of frames generated for the query are used by the research process to match similar frames in the Knowledge-base. An approximate comparison distance is adopted by comparing the internal representation of the query with the software components descriptions in the Knowledge-Base. The research style is automatically indexed. The system uses a functional information to describe components.

The neural network technology was the most used in the mid 1990's in the search process. It was used to structure a library of re-usable software according to their semantic similarities in order to facilitate the search and to optimize the retrieval of similar repetitive queries. Neural networks are considered as associative memories in some approaches in support of imprecise queries.

The work of Clifton and Li [14] can be considered as instances of information retrieval methods. In this approach, conventional abstractions are used to describe software. Clifton and Li use design information as abstraction and propose neural network technology to accomplish the match.

The approach [15] employs neural network to extend and to ameliorate the traditional methods where the query should contain exact information about the component in the library. The motivations behind using neural networks is to use relaxation and retrieving component based on approximate matches to optimize the search of similar repetitive queries and to retrieve component(s) from a large library.

Zhiyuan [16] proposes a neural associative memory and bayesien inference technology to locate components in a library. For each component, there are ten facets to represent a component(Type, domain, local identifier, etc.). The neural associative memory stores the relationship between components and facets values. During the search stage, the described component representation is mapped into facets. The value of each facet is fed into its dedicated associative memory to recall the components that have the same value for this facet. In this approach, the comparison distance is exact and the information type is functional.

5 Discussion

In most of the approaches, the description of the non-functional aspects is neglected. The functional and

the non-functional aspects must be considered during the specification, the design, the implementation, the maintenance and the re-use. In the phase of the re-use, and if our research is based only on the functional aspects, the selected component can not support the non-functional constraints of the environment. In several cases, the non-functional constraints play a decisive role in the choice of the most powerful component(s).

In agent technology, the agent runs continuously in the background of a development environment, which weighs down the execution of the system and causes a wasting of the resources. Moreover, the agent can be integrated only into one environment of development.

The majority of approaches use neural network technology to optimize the retrieval of similar repetitive queries and to retrieve components from a large library. Although this method is fast, it has disadvantages for the developers and especially the beginners. In fact, the system requires much time to apply to beginner actions.

The problem with AI-based approaches is the difficulty of getting enough knowledge about a given domain. The graph nature classification of components can be computationally expensive. Therefore, the poor response times are one of the major problems with this technology.

Compared to other retrieval systems, our approach is unique in the following aspects:

(1) The first attempt to locating components is based on a software component.

(2) The developed system can be integrated in many software development environments.

(3) The whole system (The software development environment, the application and the search tools)is software component-oriented.

6 Conclusion

Software component retrieval research has been advancing very quickly over the past few decades. Researchers have experimented techniques ranging from AI-based approach to the neural network-based approach and the recent Agent techniques. At each step, significant insights regarding how to design and implement more useful software component search systems have been gained.

In this paper, we present a persistent component for discovery of software component library that delivers re-usable components into the current working environment of developers so that they can readily access task-related and user-specific reusable components. The query contains non-functional constraints and functional information.

In the future, we plan to evaluate CSCLD and to use it in our teaching so that our students can easily integrate the components they need in their different tasks. We also plan to extend this approach to exploit the search and the description of web services. In fact we will develop a web service to web service library discovery with the same presented architecture system. In addition, we plan to use WSDL as a language to describe web services and introduce other important non-functional constraints like response time.

References

[1] A. Ben Hamadou and F. Gargouri. *Développement de logiciels: l'approche objet*. Ed.CPU, Tunisia, 1998.

[2] C. D. T. Cicalese and S. Rotenstreich. Behavioral specification of distributed software component interfaces. *IEEE Computer*, July 1999.

[3] J. Chauvet. *Corba, ActiveX et Java Beans*, pages 63–65. Eyrolles, 1998.

[4] B. Morel and P. Alexander. SPARTACAS: automating component reuse and adaptation. *IEEE Transactions on Software Engineering*, *30*(9):587–600, September 2004.

[5] L. Bellissard. *Construction et configuration d'application reparties*. PhD thesis, Institut National Polytechnique de Grenoble ENSIMAG, 1997.

[6] S. Lionel. *Distributed Component Object Model DCOM*. Software Engineering Environments, 2000.

[7] H. Chen. Machine learning for information retrieval: Neural networks, symbolic learning and genetic algorithms. *Journal of the American Society for Information Science (JASIS)*, 46(3):194–216, 1995.

[8] Y. Yunwen and G. Fischer. Context-aware browsing of large component repositories. *Proceedings of 16th International Conference on Automated Software Engineering (ASE'01), Coronado Island, CA*, pages 99–106, Nov 2001.

[9] C. Drummond, D. Ionescu, and R. Holte. A learning agent that assists the browsing of software libraries. Technical Report TR-95-12, Computer Science Dept, 1995.

[10] R. Seacord, S. Hissam, and K. Wallnau. Agora: A search engine for software components. *IEEE Internet Computing*, pages 62–70, November/December 1998.

[11] S. Henninger. An evolutionary approach to constructing effective software reuse repositories. *ACM Transactions on Software Engineering and Methodology*, 6(2):111–140, 1997.

[12] P. Devanbu. Lassie: A knowledge-based software information system. *Communications of the ACM*, 34(5):34–49, 1991.

[13] R. Girardi and B. Ibrahim. A similarity measure for retrieving software artifacts. *Proceedings of Sixth International Conference on Software Engineering and Knowledge Engineering (SEKE'94)*, pages 478–485, June 21-23 1994.

[14] C. Clifton and L. Wen-Syan. Classifying software components using design characteristics. *Proceedings. The 10th Knowledge-Based Software Engineering Conference (Cat. No.95TB100008)*, pages 139–146, 1995.

[15] K. Srinivas D. Eichmann. Neural network-based retrieval from software reuse repositories. *Neural Networks and Pattern Recognition in Human Computer Interaction*, pages 215–228, 1992.

[16] W. Zhiyuan. *Component-Based Software Engineering*. PhD thesis, Faculty of New Jersey Institute of technology, May 2000.