8-25-1995

An Agent Based Architecture ForComponent-Based Software Development

Pinar Kinikoglu
Texas Tech University

Surya B. Yadav
Texas Tech University

An Agent Based Architecture For Component-Based Software Development

Pinar Kinikoglu
Texas Tech University, College of Business Administration, ISQS Area
Lubbock, TX 79409-2101 USA

Surya B. Yadav, Ph. D.
Texas Tech University, College of Business Administration, ISQS Area
Lubbock, TX 79409-2101 USA

I. Introduction

Today's companies are facing major changes in their organizations due to the changing environment in which they operate. They have to decrease the costs, decrease time to market, and improve quality. These imperatives have led to changes in the placement and role of IS department in the organization (Fried, 1995). Together with the recent advances in communication technology and powerful workstations, end-users have become more involved with the application development. Besides, the business processes change so fast that the traditional SDLC is too slow to keep up with these fluctuating requirements in the application domain. The need for rapid application development to respond to users' changing needs, among the other mentioned trends, encourages the use of reusable software components.

In (ATP, 1995), it is stated that at the level of vertical-market products, software design costs are generally $1 million to $10 million with near zero cost of reproducing additional units, and the typical production quantity is one. Reusable software components help organizations recover costs, improve quality through specialization, and develop rapidly from existing components.

1.1 Problem Statement

Reusable software components can be a big competitive advantage for an organization if its domain is well-understood and a systematic software reuse is employed. There are both technical and nontechnical problems to be solved for the successful, systematic use of reusable software components. One of the major problems is to find the component[s] which meets the development needs of the user.

Some ad hoc lists are provided by specific systems like UNIX, or user is required to provide a complete list of directly or indirectly used software components in some systems (Plaice, 1993), or a cataloging schema is developed based on reusability attributes (Chang, 1993).

Most of the current systems assume that the user has medium to high level computer and programming experience. Moreover, the user is required to have the information about the details of each software component. However, the trend is that users will be domain specialists but not necessarily computer experts. Further, the end-users have become more involved with the application development. The component based software development will let the domain specialist end-users use domain specific components and integrate them.

Therefore, a generic software architecture for reusable software components should be developed to effectively design and implement such an environment.

1.2 Research Objectives

The objective of this research is to propose an architecture for the automation of identification and integration of reusable software components. In particular;

1. The components of the architecture will be identified.
2. The functionality of each component is identified.
3. The interaction among the components will be discussed.

II. Proposed Work

In this section, we first identify the requirements of end-users for an automated reusable component based software development system. Then we propose an architecture to satisfy the identified end-user requirements. The components of the architecture will be explained.

2.1 End-user Requirements

The end-user of such a system needs neither be a computer expert neither does (s)he need to be knowledgeable about the details of software components. However, (s)he should have medium to high level of domain knowledge. The user may develop applications as part of a team project or may have one-time queries for some tasks. Most probably, (s)he will be working in a distributed computing environment.

Based on this description of the user; the requirements of the user can be listed as follows;

- transparency to the complexities of distributed system operations,
- transparency to the format, programming language, content, and location of the software component,
- ease of defining his/her needs to the system,
- ease of locating / integrating / debugging the software components,
- evaluation of the system performance,
- security, version control, efficiency.

2.2 Proposed Architecture

Software agents are the best hope for the 90's. They will actively participate in the user tasks. The agent technology is employed in the architecture to satisfy the specified user requirements. The agents are not necessarily just interfaces. Agents are knowledgeable about a process, they are specialized in a certain area. Furthermore, an agent acquires the knowledge it needs to decide when to help the user, know what to do to help to the user with and how to help. Although they are autonomous entities in the system, they will work together with the user and for the user. The user feels comfortable delegating the tasks to the agents.

In order to satisfy the user requirements listed above, the following architecture shown in figure 1 is proposed. In the proposed architecture, each agent has three parts:

i) Attributes: They identify the agent. Among them are the specialization of the agent, owner, success level, life duration, and implementation environment. This list is not exhaustive.

ii) Behavior model: It specifies how the agent operates, and when it terminates. Here, the rules for security, the relationship between the agent and the user and/or other agents are set. These rules also determine how to evaluate the performance of the agent. The agents are autonomous and independent. It is very important to have trustworthy agents. Therefore, the behavior model will ensure that the only agents which are well-behaved and useful continue to operate.

iii) Inference Engine: It operates the agent based on the behavior model.

In Figure 1;

- *User interface* hides the complexities of the system operations. It lets the user define his/her one-time queries as well as long term, relatively static needs.

- *User Profile KB* (UPKB) represents the user's long term requirements. The information about the user interests and needs affects which tasks the domain agent performs. It evolves over time. It is revised continuously with the feedback from the user.

- *Task Domain KB* (TDKB) includes specific guidelines, principles, necessary information about a certain task. It the user.

- *Application Database* keeps the software component lists of different applications developed so that the testing, user change requests, and integration with the newer versions can be handled.

- *Domain Agent* (DA) works on behalf of the user. It is responsible to represent user's requirements in the UPKB as well as to build the TDKB based on feedback from the user. It formulates the component specifications from the user requirements and delegates the authority to the software component agents mediator to find the matched software components. It evaluates the performance of software component agents mediator based on how satisfactorily it finds the software components. Furthermore, based on the patterns of change in the user requirements, it updates the UPKB. Finally, whenever it needs information which is out of scope of TDKB, it communicates with other domain agents through domain agents mediator to complete its work for the user.

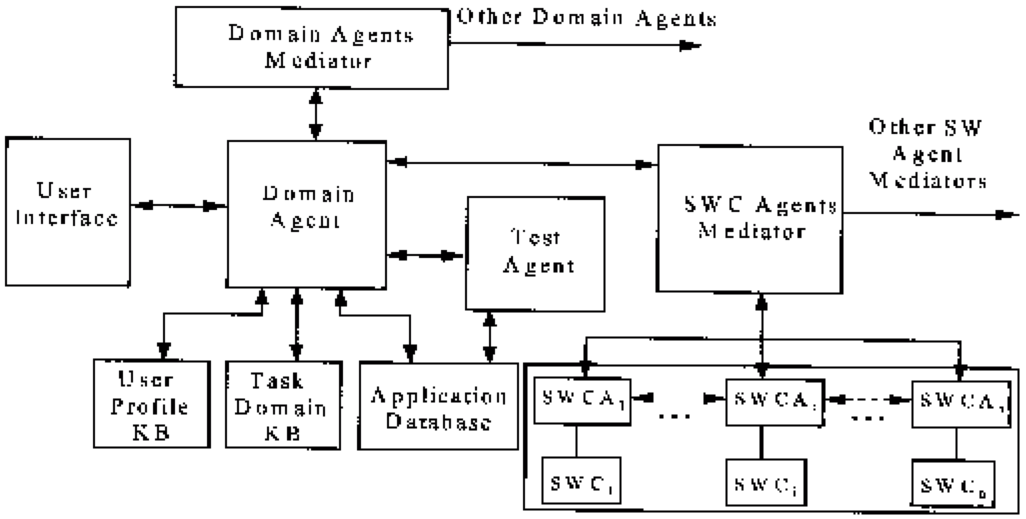
- *Software Components Agents Mediator* (SWCAM) is an agent specialized on software components. The idea is borrowed from Genesereth, 1995. SWCAs communicate with each other through SWCAM. However, it has the authority to let the SWCAs talk to each other directly if it is more feasible. SWCAM is needed since there may be communication problems for SWCAs developed at different platforms. It also negotiates with SWCAs to get their services. Moreover, it evaluates the performance of individual SWCAs. When it needs the services of other SWCAs which are not available at the software agents library, it negotiates with other SWCAMs in the system.

- *Software Components Agents* (SWCA) represents the reusable software components. Software components can be written using different tools at different platforms. SWCA will provide the services of the software component that it represents to other SWCAs and at the same time it can request their services.

- *Domain Agents Mediator* provides communication service to DAs since they can be implemented and placed in a heterogeneous distributed environment.

- *Test Agent* produces the test data to test the integrated components of the application using application database and performs the integration test. It communicates the problems with the application to the domain agent.

All of the agents should be self contained and encapsulated so that security can be achieved to a certain extent although more security measures are needed to avoid the unauthorized uses.



III. Conclusion

This paper proposes an agent based architecture for automated software development. This architecture does not need experienced programmers or system developers. The users are not required to provide a list of modules. Besides, they do not have to keep track of all software components since the application development is automated through the use of agents.

Selected References

Advanced Technology Program - Information Package, Information Package for Component-Based Software, National Institute of Standards and Technology (NIST), 1995.

Chang, Yuk Fung, C.M. Eastman, "An Information Retrieval System for Reusable Software", Information Processing & Management, Vol.29(5), pp.601-614, 1993.

Fried, Louis, Managing Information Technology in Turbulent Times, John Wiley&Sons, Inc., 1995.

Genesereth, Micheal R., "Interoperability: An Agent-Based Framework", AI Expert, Vol.10(3), pp.34-39, 1995.

Lashkari, Yezdi, M. Metral, P.Maes, "Collaborative Interface Agents", in Proceedings of the National Conference on Artificial Intelligence, MIT Press, Mass., 1994.

Plaice, John, W. Wadge, "A UNIX Tool for Managing Reusable Software Components", Software: Practice & Experience, Vol.23(9), pp.933-948, 1993.