Rodríguez et al., 2004, "Using a Multi-Agent Architecture to Manage Knowledge in the Software Maintenance Process", Lecture Notes in Artificial Intelligence, Springer, Vol. 3213: p. 1181-1187.

Using a Multi-Agent Architecture to Manage Knowledge in the Software Maintenance Process

Oscar M. Rodríguez[1], Aurora Vizcaíno[2], Ana I. Martínez[1], Mario Piattini[2], Jesús Favela[1]

[1]CICESE, Computer Science Department, México
{orodrigu | martinea | favela}@cicese.mx
[2]Alarcos Research group. University of Castilla-La Mancha, Escuela Superior de Informática, España
{Aurora.Vizcaíno | Mario.Piattini}@uclm.es

Abstract. In the software maintenance process a considerable amount of information needs to be managed. This information often comes from diverse and distributed sources. However, very few software companies use knowledge management techniques to efficiently manage this information. This work presents a multi-agent architecture designed to manage the information and knowledge generated during the software maintenance process. The architecture has different types of agents, each devoted to a particular type of information. Agents can use different techniques to generate new knowledge from previous information and to learn from their own experience. Thereby, the agents can become experts in the type of knowledge they are responsible for and can communicate with each other to share this knowledge.

1. Introduction

The software maintenance process involves considerable effort and costs. In fact, this process is considered the most expensive of the software development life-cycle [11]. On the other hand, maintenance work requires the management of a large amount of information and knowledge [5, 8]. This information often comes from diverse sources such as the products to be maintained, the people who work in this process, and the activities performed to update and evolve the software. However, very few software companies use knowledge management techniques to manage this information efficiently. Appropriate knowledge management would help software companies improve performance, control costs and decrease effort by taking advantage of previous solutions that could be reused to avoid repeating previous mistakes [5]. This work presents a multi-agent architecture designed to manage the information and knowledge generated during software maintenance. The content of this paper is organized as follows: Section 2 justifies the need for knowledge management in software maintenance. Section 3 presents the architecture designed to encourage and facilitate the reuse of knowledge and previous experience in software maintenance, and an initial implementation of the architecture. Finally, conclusions and future work are presented in Section 4.

Rodríguez et al., 2004, "Using a Multi-Agent Architecture to Manage Knowledge in the Software Maintenance Process", Lecture Notes in Artificial Intelligence, Springer, Vol. 3213: p. 1181-1187.

2. Knowledge Problems in Software Maintenance

Maintenance engineers need different kinds of knowledge to perform their job. In the course of their activities different types of maintenance could be required: corrective, perceptive, adaptive or preventive. Each type of maintenance has its own features but all of them follow a similar process, summarized in Figure 1: The maintenance engineer receives the request for modification. Then, s/he identifies which parts of the system should be modified and the modules affected by this modification. With this information s/he plans the activities to be performed. The engineer, unconsciously, takes advantage of his/her experience to carry out all these tasks. During this process s/he might consult other resources, such as a person who has already solved a similar problem or who has worked with that software before, alternatively s/he will consult documentation related to the software to be modified. But the problem arises when any of these sources of information is not accessible because either the employees with experience have left the organization [3], there is not enough documentation, or this is not up to date [13]. In these cases, the engineer will analyse the source code [6] which often requires considerable effort. In fact, sources of knowledge are sometimes so difficult to find that the maintenance engineer often choose to go directly to the code. So it is important to provide mechanisms to support the compilation and management of the knowledge generated during the software maintenance process, in order to avoid its loss and to foster the reuse of information and lessons learned.

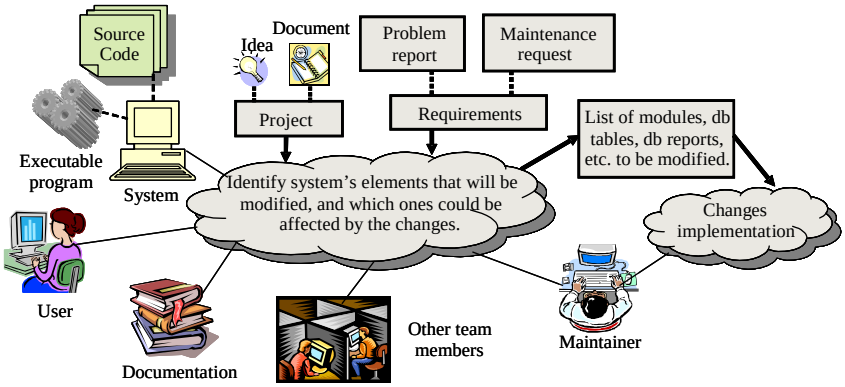

Figure 1. Knowledge sources that help the maintenance engineer to do his/her job.

Frequently, information sources are often not consulted because of people ignore their existence or location. Moreover, sometimes the organization itself is not aware of the location of the pockets of knowledge or expertise [10]. This is the number one barrier to knowledge sharing [14]. We observed this problem in two case studies carried out in two software maintenance teams. The study showed that on many occasions, organizations have documents or people with the information or knowledge necessary to support or help the maintenance engineers to do their activities, but either the latter did not know that other documents or people could have provided useful information to help them to complete the assignment or the people with useful information did not know what the latter was working on.

Rodríguez et al., 2004, "Using a Multi-Agent Architecture to Manage Knowledge in the Software Maintenance Process", Lecture Notes in Artificial Intelligence, Springer, Vol. 3213: p. 1181-1187.

In order to address these problems, we are designing a multi-agent system that acquires and manages information generated during the software maintenance process. The multi-agent architecture of the system is described in the next section.

3. The Multi-Agent Architecture

There are several reasons why agents are good technical alternatives for knowledge management software [15]. First of all, agents are proactive. This means they act automatically when it is necessary. One of the obstacles to implementing knowledge management in software organizations is that employees do not have time to introduce or search for knowledge [8]. During their daily work, different kinds of knowledge and experiences are created, but not captured in a formal way, they are only in the engineer's head. In order to reduce the loss and waste of knowledge in software maintenance, it is important to avoid this problem, but without increasing the maintainer's work. Agents, because they are proactive, can capture and manage information automatically. For example, acting like a personal assistant that knows the engineer's profile and identifies the needs of knowledge and search for sources that can help the engineer to fulfill his/her job [9].

Moreover, agents can manage both distributed and local knowledge. This is an important feature since the software maintenance knowledge is generated by different sources and often from different places.

Another important issue is that agents can learn from their own experience. Consequently, the system is expected to become more efficient with time since the agents have learnt from their previous mistakes and successes [9].

Finally, in a multi-agent system each agent may utilize different reasoning techniques depending on the situation. For instance, they can use ID3 algorithms to learn from previous experiences and case-based reasoning to advise a client how to solve a problem.

3.1 Architecture's Description

In order to design the multi-agent architecture, we have followed MESSAGE, a Methodology for Engineering Systems of Software Agents [4]. MESSAGE proposes different levels of analysis. At level 1 analysis focuses on the system itself, identifying the types of agents and roles, which are described in the next paragraphs.

The architecture has five main types of agent (see Figure 2): staff, product, client, project and directory agents.

The *staff agent* is a mediator between the maintainer and the system. It acts like an assistant to the maintenance engineer (ME). The rest of the agents of the system communicate with the ME through this agent. The staff agent monitors the ME activities and requests the KMA to search for knowledge sources that can help the ME to perform his/her job. This agent has information that could be used to identify the ME profile, such as which kinds of knowledge or expertise s/he has or which kinds of sources s/he often consults.

Rodríguez et al., 2004, "Using a Multi-Agent Architecture to Manage Knowledge in the Software Maintenance Process", Lecture Notes in Artificial Intelligence, Springer, Vol. 3213: p. 1181-1187.

The *product agent* manages information related to a product, including its maintenance requests and the main elements that integrate the product (documentation, source code, databases, etc.). The main role of this agent is to have updated information about the modifications carried out in a product and the people that were involved in it.

When the product agent receives a maintenance request sent by a client, it creates a new project and proposes the tasks that must be done in order to fulfill the request. The agent also proposes the most suitable people to perform those tasks and sends the proposal to the staff agent in charge to assist the ME that plays the role of project manager. The staff agent informs the ME of these proposals, and s/he decides if the proposal is accepted or modified. Once the proposal has been accepted, the project agent starts to work.

Each project is managed by a *project agent*, which is in charge of informing the ME's involved in a project about the tasks that they should perform. To do this, the project agents communicate with the staff agents. The project agents also control the evolution of the projects.

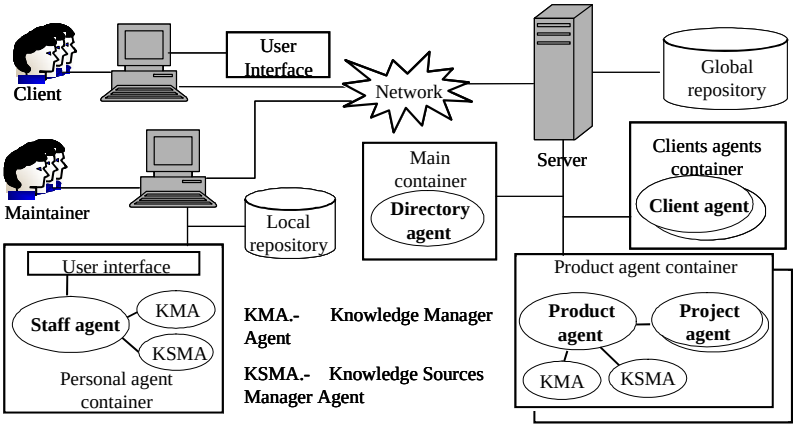

Figure 2. Agent based architecture for a software maintenance knowledge management system.

The *client agent* manages information related to the maintenance requests or error reports performed by a client. There is one agent of this kind per client. Its main role is to assist them when they send a maintenance request, directing it to the corresponding product agent. Another important activity of this agent is to inform the client about the state of the maintenance requests sent previously by him/her, by consulting the project agents in charge of this request.

The *directory agent* manages information required by agents to know how to communicate with other agents that are active in the system. This agent knows the type, name, and electronic address of all active agents. Its main role is to control the different agents that are active in the system at each moment.

Two auxiliary types of agents are considered in the architecture, the *Knowledge Manager Agent* (KMA) and the *Knowledge Source Manager Agent* (KSMA).

The KMA is in charge of providing support in the generation of knowledge and the search of knowledge sources. This kind of agent is in charge of managing the

Rodríguez et al., 2004, "Using a Multi-Agent Architecture to Manage Knowledge in the Software Maintenance Process", Lecture Notes in Artificial Intelligence, Springer, Vol. 3213: p. 1181-1187.

knowledge base. The staff KMA generates new knowledge from the information obtained from the ME's in their daily work. For example, if a ME is modifying a program developed in the Java language, the KMA can infer that the ME has knowledge of this language and add his/her name to the knowledge base as a possible source of knowledge about Java. On the other hand, the product KMA generates knowledge related to the activities performed on the product. It could identify patterns on the modifications done to the different modules. For example, it could detect that there are modules or documents that should be modified or consulted when a specific module is modified, and in this way, it could indicate which modules or programs can be affected by the changes done on others.

Finally, the KSMA has control over the knowledge sources, such as electronic documents. It knows the physical location of those sources, as well as the mechanisms used to consult them. Its main role is to control access to the sources. The documents located in the workspace of the ME's, or those that are part of a product, such as the documentation of the system or the user documentation, are accessed through this agent. The KSMA is also in charge of the recovery of documents located in places different from its workspace. If those documents are managed by another KSMA, the first KSMA should communicate with the other to request the documents.

3.2 Implementation of the Architecture

To evaluate the feasibility of the implementation of the architecture, we have developed a prototype. The requirements were obtained from scenarios identified in the two case studies previously mentioned.

The information managed by the prototype was obtained from one of the organizations where the case studies were performed. The prototype was tested specifically following the scenario described next.

First, the maintenance engineers see a list of the projects they are assigned. These are shown by the staff agent through its GUI. When an engineer selects one project, an event is triggered and captured by the staff agent, which obtains the information of the project, identifies knowledge topics (system and module where the problem appeared, kind of problem, etc.) and generates some rules to request the KMA to search for knowledge sources. To create the rules, the staff agent tries to identify the knowledge that the engineer would need to carry out the assignment. Also the agent considers the types of sources the engineer consults, assigning more relevance to the sources that the engineer consults most frequently. When the search has finished, the KMA sends a message to the staff agent informing it about the sources found. The staff agent displays a message with the number of sources found in order to inform the engineer. Finally, if the maintenance engineer wants to look for the sources found, s/he chooses a button in the staff agent GUI, and the agent will display a window with the list of sources grouped by kind (see Figure 3). When the maintainer selects one source from the list, the window shows some information related to that source: location, knowledge that it has, etc.

JADE was chosen as the platform for implementing the multi-agent prototype, since it is FIPA compliant, and provides mechanisms which define ontologies and

Rodríguez et al., 2004, "Using a Multi-Agent Architecture to Manage Knowledge in the Software Maintenance Process", Lecture Notes in Artificial Intelligence, Springer, Vol. 3213: p. 1181-1187.

content languages that make it easy to develop the language for agent communication. Moreover, JADE has applications which monitor some of the agent behaviours [2].

As shown in the architecture presented in Figure 2, the prototype has two types of repositories of information. One is where local information related to specific tasks is stored and the other is a global repository where more generic knowledge is stored. Data are represented as XML documents and managed by XINDICE, a XML database [1]. The data are classified following an ontology for software maintenance proposed in [12], which is an extension of that of [7].

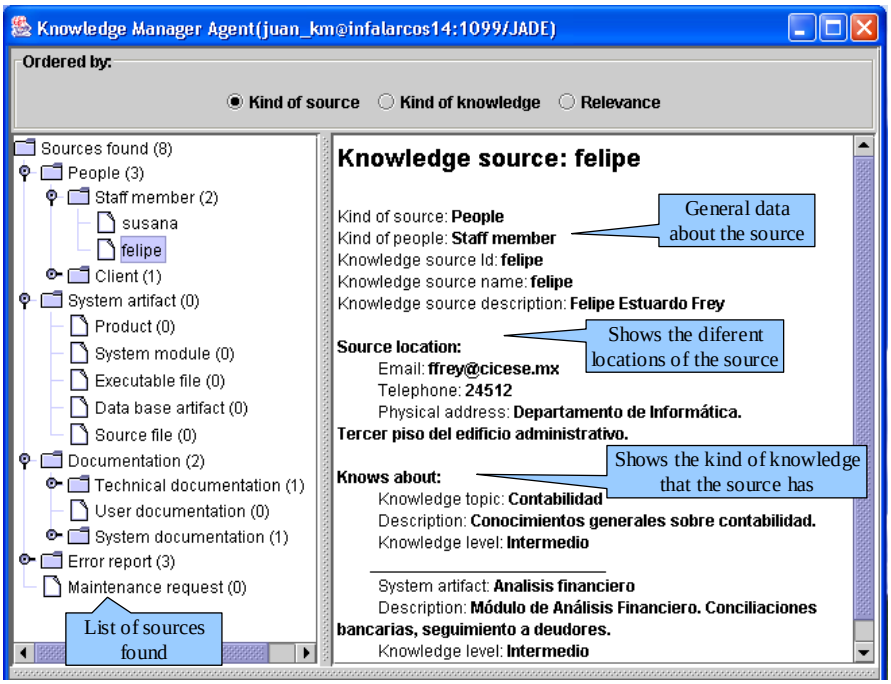

Figure 3. Window that shows the list of knowledge sources found.

4. Conclusions and Future Work

This paper presents an architecture for a multi-agent system in charge of storing and managing knowledge, expertise and lessons learned generated during the software maintenance process. The architecture of the system is formed from different kinds of agent in charge of managing a specific type of knowledge, thus they can become expert in a particular kind of knowledge and share it with others when they need it.

A initial prototype has been developed in order to test whether it is feasible to implement a knowledge management system for software maintenance based on the proposed architecture. The prototype enable us to track scenarios that show how a knowledge management tool can help to solve some of the problems detected in the case studies. For instance, how to find experts. Once this prototype is finished we are

Rodríguez et al., 2004, "Using a Multi-Agent Architecture to Manage Knowledge in the Software Maintenance Process", Lecture Notes in Artificial Intelligence, Springer, Vol. 3213: p. 1181-1187.

planning to perform a case study to evaluate how the system is perceived by a software maintenance group and how it can be improved.

Acknowledgements

This work is partially supported by CONACYT under grant C01-40799 and the scholarship 164739 provided to the first author, and the MAS project (grant number TIC2003-02737-C02-02), Ministerio de Ciencia y Tecnología, SPAIN.

References

[1]. Apache-Software-Foundation, "Apache Xindice official site", (2004), http://xml.apache.org/xindice/, consulted at 16-feb-2004.

[2]. F. Bellifemine, A. Poggi and G. Rimassa, "Developing multi-agent systems with a FIPA-compliant agent framework", Software practice &experience, 31, (2001), p. 103-128.

[3]. K. Bennet and V. Rajlich, "Software Maintenance and Evolution: A Roadmap", in The Future of Software Engineering, International Conference on Software Engineering (ICSE'2000), Limerick Ireland, IEEE Computer Society Press, (2000), p. 73-87.

[4]. G. Caire, W. Coulier, F. Garijo, J. Gómez, J. Pavón, F. Leal, P. Chainho, P. Kearney, J. Stark, R. Evans and P. Massonet, "Agent Oriented Analysis using MESSAGE/UML", in Agent Oriented Software Engineering, (2001), p. 119-135.

[5]. T. Dingsoyr and R. Conradi, "A survey of case studies of the use of knowledge management in software engineering", International Journal of Software Engineering and Knowledge Engineering, 12(4), (2002), p. 391-414.

[6]. ISO/IEC, "ISO/IEC FDIS 14764:1999, Software Engineering - Software Maintenance". Secretariat : Standard Council of Canada. (1999).

[7]. B. A. Kitchenham, G. H. Travassos, A. v. Mayrhauser, F. Niessink, N. F. Schneidewind, J. Singer, S. Takada, R. Vehvilainen and H. Yang, "Towards an ontology of software maintenance", Journal of Software Maintenance: Research and Practice, 11(6), (1999), p. 365-389.

[8]. M. Lindvall and I. Rus, "Knowledge Management for Software Organizations", in Managing Software Engineering Knowledge, Aurum, A., et al., (eds.), Springer, Berlin, (2003), p. 73-94.

[9]. P. Maes, "Agents that reduce work and information overload", Communications of the ACM, 37(7), (1994), p. 31-40.

[10]. J. Nebus, "Framing the Knowledge Search Problem: Whom Do We Contact and Why Do We Contact Them?" in Academy of Management Best Papers Proceedings, (2001), p. h1-h7.

[11]. M. Polo, M. Piattini and F. Ruiz, "Using a Qualitative Research Method for Building a Software Maintenance Methodology", Software Practice & Experience, 32(13), (2002), p. 1239-1260.

[12]. F. Ruiz, A. Vizcaíno Barceló, M. Piattini and F. García, "An Ontology for the Management of Software Maintenance Projects", International Journal of Software Engineering and Knowledge Engineering, (2004), Accepted for publication.

[13]. J. Singer, "Practices of Software Maintenance", in Proceedings of the International Conference on Software Maintenance, (1998), p. 139-145.

[14]. G. Szulanski, "Intra-Firm Transfer of Best Practices Project", in American Productivity and Quality Centre, Houston, Texas, (1994), p. 2-19.

[15]. C. A. Tacla and J.-P. Barthès, "A Multi-agent Architecture for KM Systems", in IEEE International Symposium on Advanced Distributed Computing Systems (ISADS 2002), Guadalajara, México, IEEE Computer Society Press, (2002), p. 1-12.