A Multi-Agent Architecture for Knowledge Acquisition

Cesar A. Tacla, Jean-Paul Barthès

CNRS UMR 6599 HEUDYASIC
Université de Technologie de Compiègne, France
{Cesar.Tacla, Barthes}@utc.fr

Abstract

This paper concerns a multi-agent system for knowledge management (KM) in research and development (R&D) projects. R&D teams have no time to organize project information, or to articulate the rationale behind the actions that generated the information. Our aim is to provide a system for helping team members to make knowledge explicit, and to allow them to share their experiences, i.e., lessons learned (LL), without asking them too much extra-work. The article focuses on how we intend to help the team members to feed the system with LL, using the day-to-day operations they perform on desktop computers, and how we intend to exploit the LL by using a case-based reasoning engine.

Introduction

R&D project team members have neither time to organize documentation nor to articulate the rationale behind the set of decisions and actions in a project. At the end of the project, the group has a naturally distributed memory since each member played a different role and consequently has a different view of the joint experiences. Information ends up spread in a multitude of documents on various supports. Most lessons learnt on the project are also distributed among the project team members. Then, since R&D teams change from a project to the next, a great deal of experience can be lost. Thus, the question is how to provide technology to help the actors preserve, organize and improve their use of knowledge, without adding to their workload or endangering the project timetable.

R&D projects are fertile fields for learning new lessons since we can find knowledge intensive tasks often not completely prescribed in the model of tasks of the project. Such tasks are weakly structured, that is, they have neither a well-predefined sequence of sub-tasks, nor a known collection of resources to employ. We can say that they are often at the origin of the LL (we do not mean that lessons result exclusively from knowledge intensive tasks, other kinds of tasks also generate LL). For example, consider a team member writing a technical report, who lacks information concerning a certain subject. She gives herself the task "get information about the subject *x*" and performs a number of operations: search for documents on the WEB, chatting with a colleague, and posting of a question in a discussion forum. When she obtains enough information for finishing her report, she estimates that the task is finished. The set of operations she has done can be used as the basis for a LL record, containing descriptions of the problem (e.g. obtaining information about the subject *x*), of the context, (e.g., the project task in progress and the people involved), and of the results (e.g., the resources employed – documents, specialists – related to the subject *x*).

This article focuses on how we intend to automatically capture the operations performed on a desktop computer by using personal assistant agents, and how we intend to organize them as LL. We are also interested in how agents help users with the sharing of the acquired LLs.

The next section presents the guidelines for the KM system including the principles, the reasons for using a multi-agent architecture, and the functional requirements for the system. Following, we introduce our multi-agent platform that will enable to discuss the KM system, meaning, the kind of agents and their arrangement as a team. Next, we exemplify how the KM system works in practice, and we give more details on the capture of the desktop operations, and on the process of acquiring and reusing LLs. Finally, we present related works, the current development status of the system, and we offer a conclusion.

Principles

In this section, we first present the principles that have guided us in the decisions concerning the proposed architecture. Then, we point out the reasons for using a multi-agent architecture, and finally we present the functional requirements for the KM system.

The very underlying principle is the concept of *Ba* proposed by (Nonaka and Konno 1998). According to the concept of *Ba*, a project may be seen as a shared space where group members perform their work through successive transformations and exchanges of knowledge. In computational terms, the concept of *Ba* may be roughly translated into a system that supports day-to-day work of team members, and helps them to execute the knowledge

management cycle (locating, formalizing, diffusing, and updating knowledge).

The second principle is a practical one. In order to offer possible solutions for KM systems, one has to take into account the reality of the situation, and in particular organizational constraints (e.g. the time available to the team members, the legacy systems), rather than devising elegant but impractical systems.

The third principle is to consider knowledge as a personal item that can be developed individually and maintained locally. Knowledge may be expressed in the information produced, retrieved and exchanged among members of the project. Thus, we try to organize the information automatically and locally in a per user mode. The result is a distributed group memory where each member keeps part of the knowledge of the team.

Choice of Technology

The first reason to employ multi-agents in KM systems is that, like in a team, a multi-agent system is composed of a group of possibly heterogeneous and autonomous agents that share a common goal and work cooperatively to achieve it. A multi-agent system can display an intelligent behavior. For example, using techniques from artificial intelligence, agents can maintain a representation of what is happening and offer spontaneous help (proactive behavior). To do so, they can use past information, e.g., through case-based reasoning techniques. In addition, since agents are developed independently, a multi-agent system can be easily modified by simple addition or removal of agents like in a team where members can join or quit at any time.

The second reason is that most of the currently available technological solutions for KM systems are based on groupware tools. In general, such tools are efficient for project management and provide means for team members to coordinate their actions. However, in relation to the KM, they mainly support combination of explicit knowledge (e.g. automatic information delivery accordingly to user profiles) and, at some degree, externalization of tacit knowledge through dialogues in virtual spaces (e.g. discussion forums). We think they do not support the process of knowledge acquisition adequately. For example, the way users perform their computer-supported tasks is not effectively taken into account at present time, though this may be an important source of knowledge to be reused. This means that groupware tools are not well suited to learn by monitoring the user actions. However, we believe that groupware tools can supply basic functions (e.g. communication, document storage, project management), which constitutes the foundation for knowledge management software.

Functional Requirements

The KM systems we intend to derive from the proposed architecture must be able to provide the following functions:

- Organize documentation automatically, and support the team members in retrieving and sharing documents.
- Aid team members to articulate their tacit knowledge and share formalized knowledge. We have employed lessons learned as a mean to represent formalized knowledge, although other representations could be used like decision trees for representing the project rationale.
- Support the collaborative work by:
  - Managing the team tasks.
  - Supporting asynchronous and synchronous communication among team members.
  - Providing a persistent and reliable environment.

In the next sections, we discuss mainly the first two requirements, given that groupware tools mostly provide the third one. The next section introduces the multi-agent platform that has been used. The presentation of the platform will enable the discussion about a KM system satisfying mainly the first two requirements.

OMAS: A Multi-Agent Platform

In our laboratory we have been developing agent systems for a long time, in particular in the domain of complex design. (Monceyron and Barthès 1992) developed a blackboard design environment, EXPORT, applied to harbor design, and more recently (Shen and Barthès 1996) developed a generic multi-agent platform, DIDE, for mechanical engineering design following the work of (Scalabrin, et al. 1996) on cognitive agent platforms. This led to the development of the OMAS platform (Open Multi-Agent System) the main features of which are summarized in the following paragraphs.

Cognitive Agents

Among the many types of agent models and systems that exist, we chose to work with cognitive agents. Such agents contain “an explicitly represented, symbolic model of the world and in which decisions are made via symbolic reasoning” (Wooldridge 1995). The main advantage of cognitive agents is the possibility of designing intelligent behaviors by means of a set of skills. Moreover, such agents are autonomous, running independently of any particular task. It allows them in particular to display a proactive behavior, which makes them very different from client-server systems. A large number of papers and books have been published about agents and the reader is referred to (Jennings, Sycara, and Wooldridge 1998) for a review of the domain or to (Shen et al. 2001) for a survey of multi-agent systems in design and manufacturing applications.

The platform allows the use of two types of agents:

a) **Service agents (SAs)** that provide a particular type of services corresponding to specific skills (one of the services is the interface to foreign platforms).

b) **Personal assistants (PAs)** that are in charge of interfacing humans to the system. Their particular skills are devoted to understanding their masters and presenting the information in a timely way.

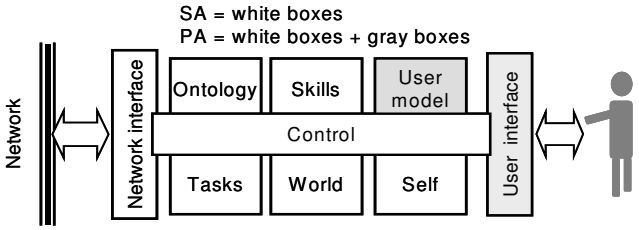

Figure 1: The internal structure for personal assistant (PA) and service agents (SA).

The internal structure of an agent is given on Figure 1. It can be seen that a PA is a SA augmented with a user interface and a model of the user (the gray area). Following, we briefly describe the components of an agent:

- **Net interface** implements the communication protocols.
- **Control module** is responsible for the agent behavior.
- **Skills** contain the set of services of the agent. A particular skill can be expressed as procedures or rules. If the skill is complex, then a plan for executing it can be created, and the derived tasks are spawned and broadcast on the network.
- **World** contains a representation of the other agents (their skills) and of the environment.
- **Tasks** are a representation of what the agent is currently executing.
- **Ontology** deals with agent ontology and domain ontologies. The agent ontology is employed to represent the agent skills, the tasks, and the skills of the other agents.
- **Self** contains the representation of the agent's skills, data stored in its memory, and possibly its own goals.
- **User model** contains user preferences and a model of the dialog with the user.
- **User interface** allows the PA to dialog with the user.

Communication

In our platform the agents are tightly linked within groups called *coteries*. Each agent inside a coterie receives all messages, allowing it to update and maintain its own internal representations, like a person in a small group hears all conversations within a certain distance. A coterie however can participate in higher-level interactions, that is, coteries can be linked by means of *transfer* or broker agents. Inside a coterie there is no control structure, and there is no concept of group nor of social structure. There can be however an organizational structure among people using the agents. The inspiration for the concept of coterie comes from the notion of *Ba*.

Communication within a coterie occurs asynchronously in broadcast mode (messages can be directed, e.g., point to point, but communication mode is broadcast). This probably will appear surprising to some readers and interpreted as a waste of bandwidth. In practice, since the agents of a coterie share the same local network, each agent listens to all the messages at the lowest level. As messages are usually filtered out in a point-to-point communication, broadcast, implemented at a low level (e.g., by means of a UDP protocol), is essentially free.

At a higher level, we must make use of some communication protocols. For processing tasks, we extended the Contract-Net protocol (Davis and Smith 1993). The resulting version, the B-Contract-Net, let the agents run at two levels of priority and work on contracts even when they were not selected. The B-Contract-Net is quite complex and out of the scope of this paper. Note however, that although there are a number of exchanges during the first phase of Contract-Net type protocols, we do not consider such protocols as negotiation protocols, but as a coordination method for task allocation, agreeing with Jennings (Jennings, Sycara, and Wooldridge 1998).

In relation to the content language, ontologies play an essential role. Domain ontologies are used as primitives of the agent content language, assuming a KQML-like message structure (Finin et al. 1993) and a content language resembling SL0, SL1 or KIF (Geneserth, and Fikes 1992). SAs use ontologies to interpret the incoming messages and thus to provide the required service. Ontologies are also used by the SAs for building and modifying local representations. The PAs employ the ontologies in the same way and additionally for communicating with the users.

An interesting point of our platform is that it does not contain a directory service (as recommended by the FIPA standard) that provides a location where agents register their descriptions (name, locator, and eventually other descriptive attributes as the services offered by the agent). A new agent, using the internal structure depicted in Figure 1, for advertising its services to the other agents within a coterie may either broadcast a message or simply answer the calls for bids since they are all broadcast within the coterie. An agent not presenting the same internal structure as the one provided by our platform wishing to communicate with an agent inside the coterie must send messages to a special SA, the Transfer Agent. This agent translates the received messages and broadcasts it to all elements belonging to the coterie. Inversely, an agent inside the coterie must send messages to the Transfer Agent to interact with an agent outside the coterie on a different platform. Hence, the Transfer Agent provides facilities for communicating with other coteries or external (FIPA compliant) platforms.

The KM System Architecture

This section describes the KM system architecture built using the OMAS platform. We point out the agents

forming our architecture and how they are arranged into a coterie.

The Agents

In our architecture, the **PAs** play a major role in the KM system. Firstly, they are in charge of all exchanges of information among team members. For example, if a team member wishes to send an email to a colleague, he asks for his PA that knows which SA is capable of effectively sending it. Secondly, the PA is able to organize the documentation of its master always with some help from the SAs. Finally, as R&D members have to deal with knowledge intensive tasks, they are supposed to construct their own work methods, and in this process they should remember their past experiences, and if possible have access to other members' past experiences. Hence, the PAs must capture and represent the team members' operations helping them in the process of preserving and creating knowledge.

In fact, a PA works jointly with the **Organizer Agent**, a service agent capable of categorizing documents using the concepts from the following ontologies:

- The domain ontology, containing the main concepts of the application domain of the project.
- The document ontology, holding the concepts concerning the documentation domain (e.g. types of documents such as report or memo).
- The R&D tasks ontology, containing the R&D basic activities (e.g. write paper, justifying a decision).
- The project tasks ontology, holding the tasks for the current project (e.g. requirements definition, prototype development).

We have stated that documents are locally organized according to the user preferences. Actually, the user preferences are expressed using concepts coming from the ontologies. Team members may extend the ontologies, expressing, in this way, their preferences. For instance, the concept "report" from the documentation ontology can be specialized to "final report" or "current status report".

A service agent called **Project Agent** keeps all the mentioned ontologies. The goal of such an agent is to hold all the values shared by the group, and among them the ontologies. Hence, the first action a personal assistant has to do when it integrates a project coterie is to ask for the ontologies.

One of the requirements for the architecture is to be reliable, secure and persistent. Thus, we integrated a service agent called **Repository Agent** that encapsulates a groupware tool or a database in order to take advantage of the qualities (they are proven technology) and the services such tools do provide (e.g. information control access, information persistence). Currently, the user can assign one of the following sharing levels to his documents: private, project, and public (a well known position). The Repository Agent must be able to translate such levels of confidentiality into the levels offered by the encapsulated tool. In addition to information control access, the Repository Agent offers services for saving and retrieving documents, and, depending on the tool it encapsulates, can offer other kinds of service like WEB search, or e-mail management.

The KM Coterie

In our architecture the service agents of a coterie have two scopes for action. Some of them are dedicated to a single user. Others, working in the scope of the project, serve all agents belonging to the coterie.

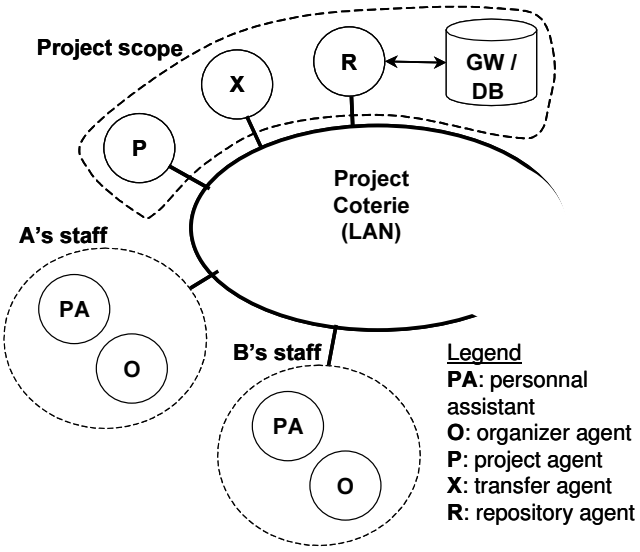

Figure 2: The multi-agent architecture for KM systems.

A PA agent and the set of service agents dedicated to a single user form a staff (see Figure 2). Such service agents cannot serve other users. For example, the Organizer Agent of user A cannot organize the documents of user B, although it can answer queries from B. Agents within a staff run usually on the same computer, often in the same environment.

In the project scope, we find one Transfer Agent, one Project Agent, and one Repository Agent. Other SAs may be added or removed at any time according to the needs of the team or of a specific user. Service agents in the project scope usually run on different computers.

We have proposed three kinds of agents to be part of the KM system: personal assistant, organizer, and project. A fourth one, the **Transfer Agent,** may be present whenever we wish to connect a coterie with another coterie or with a different (FIPA compliant) platform. Further details about the internal functioning of each kind of agent are given in the next section.

The KM in Practice

This section describes a simple case, giving further details on the internal functioning of the agents as well as on their interaction. We illustrate how the agents help the users to

gradually formalize their knowledge and to share it. For the sake of clarity, we use the following example.

Fred is an employee of an enterprise that wants to develop KM. He is working on a project called *First Approach to KM*. We assume that he is currently working on the project task *prototype building* and that the prototype will be developed using groupware. Fred is a novice user in the program language of the groupware tool, and he needs detailed information on, say, refreshing automatic computed fields inside a form. Thus, he performs a number of operations. Firstly, he searches for a Web tutorial, finds an interesting one, and saves it. Next, he chats with Sheila for getting detailed information on the problem, and finally he sends an email to Peter asking for other info sources. As he has put a lot of effort in gathering all this information, he believes that it could be useful to keep it in a lesson-learned format. But, as he has no time to formalize this experience, all the gathered information will probably be scattered and lost after a few days. The following paragraphs explain how the KM system helps Fred to preserve the experience and how other team members can reuse it.

Capturing and Representing Desktop Operations

While Fred is searching for information and dialoging with his peers, the PA captures the operations he carries out on his computer. Such operations are mainly related to communication (e.g. sending e-mails), and documentation (e.g. searching for documents). They can be activated from the user interface provided by the PA. The PA builds a representation for each operation as illustrated on Figure 3 for a save-document, chat, or send-email operation. Note that there is always a document associated with an operation, for example, operation 2 has the textual file *TipsSheila.txt* that contains the dialog between Fred and Sheila.

Once the PA has partially represented an operation, it transfers it to the Organizer Agent that will complement the representation by identifying the relevant terms of the associated document. To compute the relevance of a term the organizer employs a simple measure based on the term frequency: the more often a term occurs in a document the more important it is. The organizer computes the measure only for terms related to the concepts of the domain ontology.

In a previous approach we used the TF/IDF measure (Term Frequency, Inverse Document Frequency) as proposed by (Salton 1989). TF/IDF states that the relevance of a term in a document is in direct proportion to its frequency in the document, and in inverse proportion to its incidence in the whole collection of documents under consideration. As documents arrive in an incremental way to the Organizer Agent we had to abandon TF/IDF for two reasons. Firstly, the relevant terms must be recalculated every time a new operation is done because the collection of documents changes. Secondly, it affects the clustering algorithm (discussed in the next paragraph) as operations are grouped accordingly to their relevant terms. Hence, it was computationally too expensive to recalculate all relevant terms and clusters.

Clustering Operations into Tasks

After a certain time, Fred's Organizer Agent has gathered a number of operations together with the associated documents. We recall that the goal of such an agent is to organize Fred's documents, and that one of the goals of the system is to help Fred to articulate tacit knowledge. Hence, the Organizer clusters operations in an attempt to correlate them with an implicit intention. I.e., the documents are correlated because they are relevant for satisfying a goal that is not explicitly represented. In our case, Fred assigned himself a goal but he did not declare it to the system. Maybe, he has mentioned the goal to his colleagues in an informal way, but this may be lost. Thus, clustering of operations has a twofold purpose: organize Fred's documents by the similarity of their relevant terms, and help Fred to make his intentions explicit.

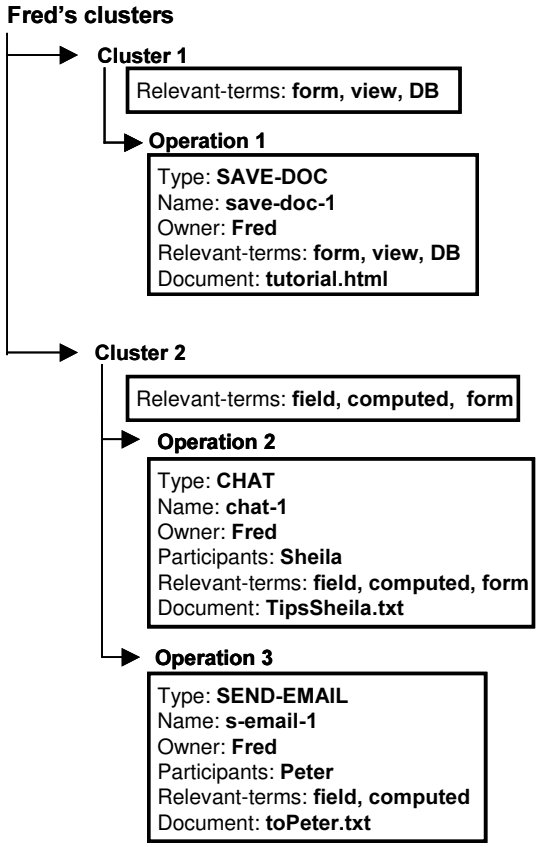

Figure 3: Clustering of Fred's operations.

The Organizer Agent keeps the operations in a frame representation. For each new operation, it looks for a cluster that best fits the operation using COBWEB, an incremental unsupervised learning algorithm by (Fisher 1990). The general idea of the algorithm is to create partitions or classes over a set of objects, using a category utility measure having two components: the intra-class

similarity and the inter-class similarity. The first component states that the greater the proportion of class members sharing a specific value, the more predictable the value is for the class. The inter-class similarity says that the fewer the objects that share a specific value, the more predictive the value is for the class. In our case, the objects are the operations and the specific values are the relevant terms in the documents. Figure 3 shows clusters built automatically by the system, based on the relevant terms of the three operations Fred has performed: finding a tutorial, chatting with Sheila, and sending an e-mail to Peter.

Note that although we know that the three operations of Figure 3 should be clustered together, the operation named *save-doc-1* is isolated, because the relevant terms do not match the terms of the other two operations. In this case, the user may modify the proposed cluster, moving the *save-doc-1* operation. If the user does not rearrange the operations, at least the system has organized the documentation through automatic indexing.

From a Cluster to a task or to a LL

When Fred decides that a cluster is well formed, he may decide to transform it into a task or LL. The Organizer Agent is not able to trigger this transition automatically because it cannot determine whether the cluster is a task or a LL. In both cases, the PA will aid Fred to reuse the information from the cluster in the construction of a task or a LL

In the first case, Fred would select some model of task and fill in the respective slots. For example, it could select the task *write-paper-for-event* from the R&D tasks ontology having the following list of slots: textual description, event, language, project tasks, and operations. The operations come directly from the cluster. The other slots must be filled in manually.

In the second case, when the cluster invokes a LL, the work is quite similar. Suppose that Fred has grouped the three operations in cluster 2, and has chosen *solving-a-problem* as the kind of LL that best fits the cluster. The models of LL are defined in the R&D tasks ontology as well as their slots. Fred is asked to fill in the slots the system cannot obtain automatically from the selected cluster like, for example, textual descriptions. In turn, the system is able to give the solution part of the LL filling it with the relevant resources (involved people and documents) referred to in the clustered operations.

After that, the system, using the domain ontology, performs a lexical analysis of the textual descriptions entered by Fred, and extracts terms that will index the LL. Such indexes in conjunction with other LL slots (e.g. project task, involved people) will enable Fred and other team members to retrieve the LL information.

Exploiting and Sharing the Lessons Learned

Over time, team members accumulate experiences and their organizer agents arrange them as LL, tasks, and operations. The sharing of such elements occurs when:

- A user makes a request to his PA.
- A PA makes a request to other PAs.
- A PA pro-actively suggests a past experience to the user.
- A PA "hears" other PAs exchanging messages.

The first situation includes the second one, thus it will not be described.

Explicit user's Request to the PA. Some time past after defining the LL, Fred wants to retrieve it. Thus, he sends a query to his PA that transfers it to the Organizer Agent. The Organizer traverses the past experiences, evaluating their similarity with Fred's query.

In addition to the direct comparison between values and slot values, we plan to implement a textual case based reasoning engine as proposed by (Burke et al. 1997) in the FAQ Finder System. Burke proposes a similarity measure between the query and the question part of a question-answer pair formed by three components: TF/IDF, the semantic distance (based on the domain ontology), and the coverage. In our case, the question part is equivalent to the task and LL descriptions. Probably, we will abandon the IDF component for the same reasons we presented before in the clustering of operations. When there is no textual description for a task, or for an isolated operation, we use the relevant associated terms.

If the Organizer Agent has found a LL, task, or operation, it returns the item to the PA that displays it to Fred. Otherwise, the Organizer broadcasts a call for bids (contract-net protocol) to the other organizer agents within the coterie. The interested organizers make offers (e.g. time estimate to answer, quality of results), and the caller grants the contract to all of them because we are using the contract-net as a coordination protocol. When all answers have been received or the time limit for answering has been reached, the caller sorts the answers according to Fred's profile and transmits them to the PA in charge of displaying them.

Finally, the acquisition of LL has an interesting side effect: we can acquire knowledge about the competences of the team members. The solution part of a LL contains references to people who contributed to solve the problem. Hence, we can link this people to the domain ontology concepts used for indexing the LL.

Pro-active Suggestion. Fred's PA may trigger the suggestion mechanism whenever a cluster receives most part of operations. Thus, the organizer identifies the relevant terms for the cluster, builds a query using the terms, and executes the process described in the first two situations. When the organizer has obtained the answers, it notifies the PA that can display the information to Fred.

Listening to the Others. Suppose that while Fred is looking for information to solve his problem, two colleagues exchange messages about the same subject. Fred's PA, based on the most active cluster, can capture the message that may serve to make Fred aware that other colleagues are working on the same subject or know something about it. Of course, the problem of access

control to the information must be addressed for this kind of monitoring. The concept of *Ba* (Nonaka, and Konno 1998) has inspired us in the creation of the process of learning from the others agents.

Important points

Firstly, in the proposed system, information can be gradually formalized (Shipman, and McCall 1997). Thus, Fred can work with isolated operations and documents, tasks, or LL. The clustering of operations aims at making the creation of tasks and LL easier. The creation of tasks and LL reduces the information overload and improves the accuracy of the returned answers, when later querying the system. Besides, in a more general perspective, it is a tentative to help team members to articulate part of their tacit knowledge and to stimulate them for sharing it.

Secondly, Fred shares his experiences in a transparent way. While he is working, his Organizer Agent is able to answer queries coming from other agent staffs. In the other direction, Fred's staff is able to capture LL from other members or send requests for information to them in order to make suggestions to Fred. Such a mechanism emulates what happens in a real team: each team member performs his own tasks and when one misses some information (or knowledge) to complete the task, one asks his colleagues for help.

Related Work

Among the many issues we are currently addressing, we would like to focus on the capture of actions through PA agents, on the multi-agent systems that support KM systems, and finally on the gradual formalization of the information.

PA applications may range from Internet search, up to collaborative tasks. Pattie Maes (Maes 1994) reports successful application of agents in assisting user with email, filtering news, scheduling meetings and selecting entertainment through social filtering. In (Lieberman et al. 1999) and (Huhns and Singh 1998) a PA is used in WEB search. While the user navigates, the agent proposes new WEB sites analyzing the contents of the sites visited by the user.

Multi-agent systems have been developed for supporting group memories. The CoMMA project (Gandon et al. 2000), Corporate Memory Management through Agents, combines emergent technologies (e.g., XML, machine learning) allowing users to exploit an organizational memory. The FRODO project (Van Elst, and Abecker 2001) provides an agent-based middleware for distributed organizational memories whose main features are scalability and distribution. The works on digital libraries (Dignum 2000) and federated databases as InfoSleuth (Bayardo et al. 1997), where agents are responsible to gather information and give a uniform view of various documents stored in heterogeneous information sources, have some similarities mainly in relation to the document organization and retrieving. Although this has not been yet implemented, the main difference with our work is in the concept of *coterie* where PAs are capable of capturing circulating information whenever they believe it will be useful for their master.

In relation to the gradual formalization of information, the work of Shipman and McCall (Shipman, and McCall 1997) has inspired us. The HOS system (Shipman, and McCall 1998) enables users working in a collaborative space to progressively formalize textual information in identifying objects of the domain and their attributes. An interesting point is that informal and formal information coexists, meaning that formalized information does not take the place of the corresponding informal one. The SHARE project (Toye et al. 1994) has brought a number of interesting services to provide to designers.

Conclusion

We analyze the proposed architecture from the KM and from the architectural perspectives. In conclusion, we give the current status and future directions.

From the **KM perspective**, we expect the proposed architecture to:

- Help the user in articulating tacit knowledge and formalizing it gradually.
- Reduce the information load because the more formalized the information, the more precise the system will be at retrieval time.
- Augment knowledge sharing given that agents are capable of exchanging and capturing LL behind the scene. Moreover, a staff of agents is able to pro-actively make suggestions to the user based on his current task.
- Allow team members to organize documentation according to their preferences by extending the document and domain ontologies.

The main drawback is that LL is just one of the ways of articulating tacit knowledge. Other methods should be considered, like the expression of the design rationale using, for example, decision trees. Concerning the **proposed KM architecture**, we can say that:

- It defines a minimal MA architecture for KM systems composed by agents with two scopes of action, project and user.
- It introduces the notion of an agent staff dedicated to a user and locally available (e.g., on the same computer), which is somehow new.
- It encapsulates a groupware or database providing information access control to project documents. It does not control the access to the system, meaning that malicious agents may enter in a coterie. An interesting result when working with groupware is that users can bypass the multi-agent system and access the project memory trough a portal.

The prototype was first undertaken during another R&D project in the domain of automobile design, when we

could test some ideas like textual CBR for retrieving segments of documents and a mechanism for suggesting similar documents accordingly to what the user types in a document being edited. We are currently developing the multi-agent version for the prototype as described in this paper, more precisely implementing the transformation from clusters to tasks or LLs.

The next step is to develop the cooperative part that will provide means to the team members to share their experiences and to know the other members' competences. Then, we have to examine whether the use of extendible ontologies in per user mode causes problems of communication when the agents share knowledge items.

Acknowledgement

This work was funded in part by CAPES and CEFET-PR, two Brazilian institutions, through a scholarship granted to Cesar Tacla.

References

Bayardo Jr., R.J.; Bohrer, W.; Brice, R.; Cichocki, A..; Fowler, J.; Helal, A..; Kashyap, V.; Ksiezyk, T.; Martin, G.; Nodine, M.; Rashid M.; Rusinkiewicz, M.; Shea, R.; Unnikrishnan, C.; Unruh, A.; Woelk, D. InfoSleuth: Agent-Based Semantic Integration of Information in Open and Dynamic Environments, Proceedings of the ACM SIGMOD International Conference on Management of Data, ACM Press, vol. 26-2, pp. 195-206, 1997.

Burke, R.; Hammond, K.; Kulyukin, V.; Lytinen, S.; Tomuro, N.; and Schoenberg, S. 1997. Question Answering from Frequently-Asked Question Files: Experiences with the FAQ Finder System. *AI Magazine* 18(2):57-66.

Davis, R.; and Smith, R.G. 1983. Negociation as a Metaphor for Distributed Problem Solving. *Artificial Intelligence* 20(1):63-100.

Dignum, F. 2000. Information Management at a Bank using Agents: Theory and Practice. *Journal of applied Artificial Intelligence* 14: 677-696.

Finin, T., Fritzon, R., McKay, D., McEntire, R. 1993. KQML – a language and protocol for knowledge and information exchange, Technical Report, University of Maryland, Baltimore.

Fisher, D. H. 1990. Knowledge Acquisition Via Incremental Conceptual Clustering. In Readings in Machine Learning, Judde Shavlik and Tomnas G. Dietterich (eds.), Morgan Kauffman Publishers, ISBN 1-55860-143-0, pg. 267-283.

Gandon, F. 2000. Multi-Agent System to Support Exploiting an XML-based Corporate Memory, In Proceedings PAKM'00, 10.1-10.12, Basel, Switzerland.

Geneserth, M., Fikes, R. 1992. Knowledge interchange format, version 3.0, Reference Manual, Tech Report Logic-92-1, Computer Science Department, Stanford University.

Huhns, M. N.; and Singh, M. P. 1998. Readings in Agents. Morgan Kaufman Publishers Inc.

Jennings, N.R.; Sycara, K.; and Wooldridge, M. 1998. A Roadmap of Agent Research and Development. Autonomous *Agents and Multi-Agent Systems*, 7-18.

Lieberman, H. 1999. Let's browse: a collaborative browsing agent, Elsevier Science B. V., 12 (December):427-431.

Maes, P. 1994. Agents that Reduce Work and Information Overload, Communications of ACM, ACM Press, 37(7): 31-40.

Monceyron, E.; and Barthès, J-P. A. 1992. Architecture for ICAD Systems: an Example from Harbor Design. *Revue Sciences et Techniques de la Conception* (1):49-68.

Nonaka, I.; and Konno, N. 1998. The Concept of "Ba": Building a Foundation for Knowledge Creation. California *Management Review*. 3(40):40-54.

Salton, G. 1989. Automatic Text Processing: The Transformations, Analysis, and Retrieval of Information by Computer, Addison-Wesley.

Scalabrin, E. E.; Vandenberghe, L.; de Azevedo, H.; and Barthès, J-P. A. 1996. A Generic Model of Cognitive Agent to Develop Open Systems. In Advances in Artificial Intelligence. Series Lecture Notes in Artificial Intelligence. D.L. Borges, C.A.A. Kaestner (Eds). Springer, 61-70.

Shen, W.; and Barthès, J-P. A. 1996. An Experimental Multi-Agent Environment for Engineering Design. International Journal of Cooperative Information Systems, World Scientific Publishing Company, 2&3(5):131-151.

Shen W., Norrie D. H., Barthès J-P. A. 2001. *Multi-Agent Systems for Concurrent Intelligent Design and Manufacturing*. Taylor and Francis.

Shipman, F.; and McCall, R. 1997. Integrating Different Perspectives on Design Rationale: Supporting the Emergence of Design Rationale from Design Communication, Artificial Intelligence in Engineering Design, Analysis, and Manufacturing (AIEDAM), 11(2): 141-154.

Shipman, F.; and McCall, R. 1998. Supporting Incremental Formalization with the Hyper-Object Substrate, *ACM Transactions on Information Systems*, 17(2):199-227.

van Elst, L., Abecker, A. 2001. Domain Ontology Agents in Distributed Organizational Memories. IJCAI'2001 Workshop on Knowledge Management and Organizational Memories, August, Seattle, Washington, USA.

G. Toye, M.R. Cutkosky, L.J. Leifer, J.M. Tenenbaum and J. Glicksman 1994. SHARE: A Methodology and Environment for Collaborative Product Developement, *The International Journal of Intelligent and Cooperative Information Systems*, 3(2):129-53.

Wooldridge, M. 1995. Conceptualising and Developing Agents, In Proceedings of the UNICOM Seminar on Agent Software, 25-26 April , 40-54.