A Micro-Organizational Model for Supporting Knowledge Migration

Frederick H. Lochovsky[*]
University of Toronto

Carson C. Woo[†]
University of British Columbia

Lawrence J. Williams[‡]
University of Toronto

Abstract

One of an organization's assets is the knowledge it has for carrying out its activities in an acceptable way. However, due to the constantly changing environment in which it operates, its knowledge must evolve in order for it to survive and to stay competitive. For this reason, it is inappropriate to hard-code organizational knowledge into a computer-based organizational support system. Supporting the evolution of organizational knowledge, in general, is a very difficult problem because the domain knowledge needed to support it is open-ended. However, certain forms of organizational knowledge evolution, where the knowledge needed for the evolution already exists in some form in the organization, can be supported. We call this type of knowledge evolution *knowledge migration*. To evolve, in this case, is to recognize the need for change, and to rearrange or acquire the knowledge (possibly located in several different locations). In this paper, we discuss an approach for supporting this type of knowledge evolution.

1 INTRODUCTION

In organizations, individual organizational workers possess a set of knowledge that describes some aspect of how they do their work (tasks). This knowledge might be procedural knowledge, for accomplishing some well-defined organizational task, or problem-solving knowledge, which supports the organizational worker in accomplishing some less well-defined organizational task. In addition to knowledge that is possessed by individual organizational workers, some organizational knowledge can be said to reside with the organization. This type of organizational knowledge is usually in the form of goals to be achieved by and/or outputs required from tasks. In this case, the organization's knowledge imposes constraints (policies) which the specific actions of the organizational workers performing a task must satisfy [WiLo89].

[*]Author's address: Frederick H. Lochovsky, Computer Systems Research Institute, University of Toronto, 10 King's College Road, Toronto, Ontario, Canada M5S 1A4. Email address: fred@db.toronto.edu.

[†]Author's address: Carson C. Woo, Faculty of Commerce and Business Administration, The University of British Columbia, 2053 Main Mall, Vancouver, B.C., Canada V6T 1Y8.
Email address: Carson_Woo@mtsg.ubc.ca or usercwoo@ubcmtsg.BITNET.

[‡]Author's address: Lawrence J. Williams, Bell-Northern Research, P.O. Box 3511, Station 'C', Ottawa, Ontario, Canada K1Y 4H7.

In structuring and managing the interaction among these different collections of knowledge in an organization[1], the following considerations apply for individual knowledge:

- Individuals need to have access to both procedural and problem-solving knowledge in an integrated way. In this way, the individual is able to draw upon and use both types of knowledge when performing organizational tasks.
- Because of the changing nature of organizations, we need to support the migration of procedural knowledge to problem-solving knowledge and vice versa.
- We want to keep one individual's knowledge separate from another individual's knowledge. In this way, we can support conflicting knowledge. Like Hewitt [Hewi86], we believe that conflict is necessary in organizations to allowing differing viewpoints and ideas to be debated and evaluated.
- Individual knowledge needs to interact with the organization knowledge. This allows the organization to constrain the actions that individuals may take (i.e., to enforce organizational policies).
- The responsibility for organizational tasks should ultimately reside with individuals in the organization.

For organization knowledge, the following considerations apply:

- As for individuals, we want the organization knowledge to be separable, perhaps into well-defined organizational units (e.g., by functional area). Again, in this way, we can support conflicting organization knowledge.
- We want the organization knowledge to constrain the actions of individuals, but not to straight-jacket them. In this way, individuals are free to use whatever means they choose to accomplish their task so long as it meets certain criteria imposed by the organization.
- We want to allow individual knowledge to migrate to become part of the organization knowledge. In this way, individual knowledge that has evolved over time, been tested, and found to be useful, can be made available to other members of the organization.
- We want to make organization knowledge available to individuals. In this way, if an individual does not have the required knowledge to perform a task, he can draw upon the organization knowledge to either find suitable knowledge or to adapt organization knowledge to suit his particular situation.

We note that computerizing organizational knowledge, in general, is very difficult because of its inherent characteristics (e.g., open-ended and changes over time). However, if one studies how organizational workers perform their tasks, it is not too difficult to discover that often they handle some cases of the evolutionary aspect of their tasks using their own experience and by learning from each other. Two examples illustrate this point:

- Each department of a firm may have a different policy on employees taking holidays. From experience, the policy used by department Ω has the least complaints from its employees and does not affect the overall effectiveness of the firm. Let's assume that ω is a subset

[1] We will refer to the collection of *all* knowledge in an organization as *organizational knowledge* and that knowledge that is not *individual knowledge* as *organization knowledge*. Our meaning should be clear from the context.

of the rules used in Ω's policy for taking holidays that is relevant to the entire firm. To resolve the large amount of complaints from the other departments, the Vice President of the personnel department decides to adopt ω for the entire firm. In this case, the knowledge of how employees can take holidays in each of the other departments evolves from its old policy to ω plus the specific rules which are applicable only to that department.

- An organizational worker α encounters a task that he has never done before and does not know how to do. One of his colleagues β has the required knowledge and teaches him how to do it. Note that α does not necessarily have to know β directly. Furthermore, β can be an organizational worker working for a different firm located in a different city (e.g., he could be a consultant). In this case, the knowledge of α evolves to include the knowledge he learned from β.

A major characteristic of this type of knowledge evolution is that the information needed for the evolution already is being used by someone else in the same or a different organization. As a result the knowledge evolution involves acquiring the knowledge and incorporating it with pre-existing knowledge. We call this type of knowledge evolution *knowledge migration.*

In the following section, we describe a model for supporting organizational activities [WoLo88]. This model originally was proposed to support individual knowledge in an organization (i.e., to support knowledge about the common, individual organizational activities such as data processing, problem solving, and communication). However, as we will demonstrate, it is easily adapted to include support for organization knowledge and in particular, knowledge migration. In section 3, the concepts of agent and organizational unit are introduced. These concepts are useful for partitioning organizational knowledge. Using the model discussed in Section 2 and the knowledge partitioning concepts discussed in Section 3, the facilities for supporting knowledge migration are presented in Section 4. Finally, we conclude our discussion in section 5.

2 THE MOAP ARCHITECTURE

A model for representing organizational knowledge is presented briefly in this section [WoLo88]. In the MOAP (micro-organization activity processor) model, an organization is composed of many units called "micro-organizations." Each micro-organization behaves the same as an organization except that it only contains a small domain of organizational knowledge. Examples of micro-organizations are department head's office, payroll department, and word processing center. Work done in an organization is a result of the activities that happen inside some of the micro-organizations, as well as the interactions between them.

To support organizational work in a computer-based system, an activity processor is provided for each of the micro-organizations. This processor, in addition to coordinating the execution of the activities within itself, can communicate with other micro-organizations when it is necessary to do so. Figure 1 shows the architecture of a MOAP[2]. Although the model is the same for all the MOAPs, the knowledge between any two of them can be inconsistent.

There are six different types of entities in a micro-organization activity processor: data bases, procedural knowledge, activity agents, activity coordinator, workspace, and problem-solving knowledge. Each one of them plays a distinct role. Data bases store persistent data. Procedural knowledge is used to automate structured and routine organizational tasks. Problem-solving knowledge

[2]The architecture of a MOAP is similar to the Hearsay-II expert system [EHLR80]. This architecture was modified to include procedural knowledge and activity agents, the coordination needed to control the execution of the entities, and the interaction among entities (including activity agents from other MOAPs).

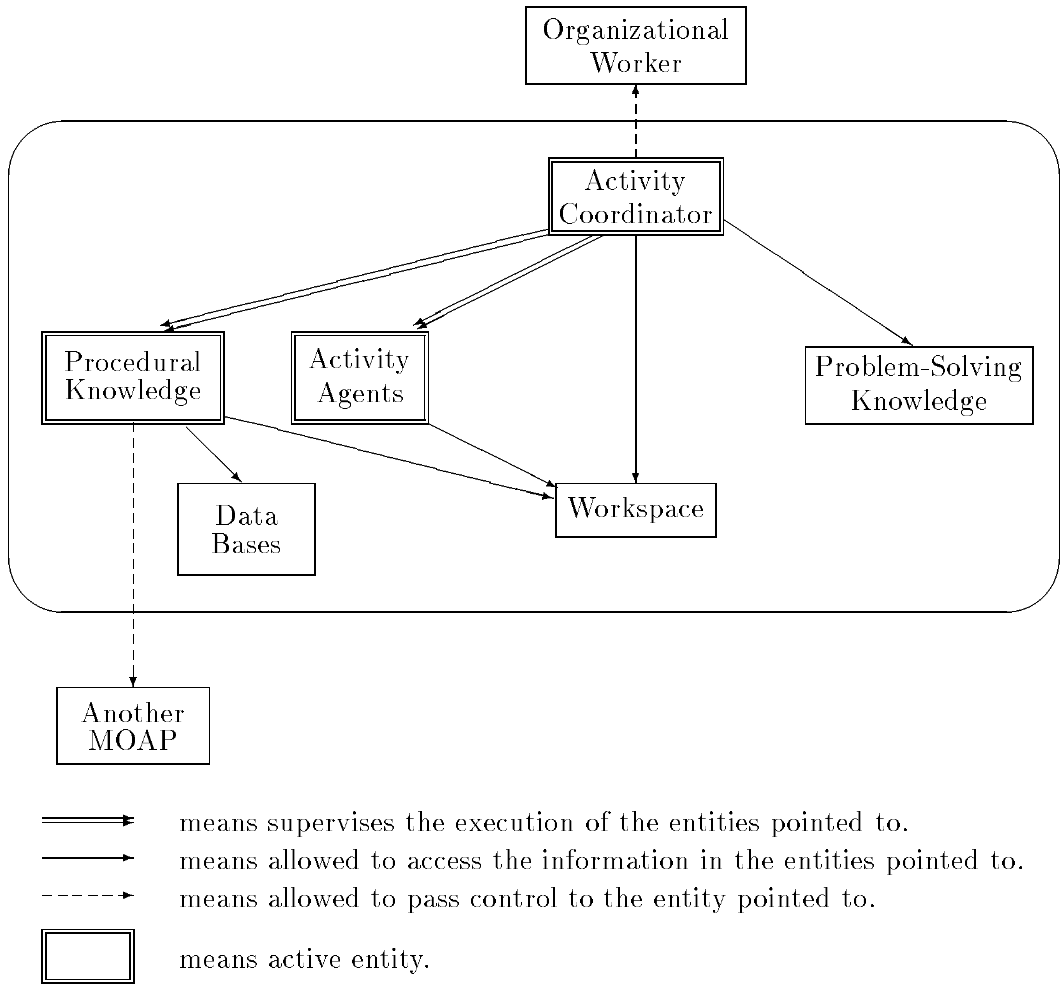


Figure 1: A micro-organization activity processor (MOAP).

describes strategies and serves as advice to the activity coordinator. A workspace serves as the communication medium for the active entities (i.e., procedural knowledge, activity agents, and the activity coordinator) in that information put into a workspace can be picked up by other active entities at a later time. The function of the activity coordinator is to control the execution of procedural knowledge and activity agents, and the use of problem-solving knowledge within its micro-organization jurisdiction. The dotted arrow in Figure 1 shows that a MOAP can be related to other MOAPs (e.g., for tasks that require cooperation and coordination). Activity agents are used to communicate and collect information in the distributed organizational environment. Descriptions of the capabilities, usage, and a prototype implementation of the model and its entities can be found in [WoLo88, Woo88].

3 AGENTS AND ORGANIZATIONAL UNITS

To both partition and share knowledge among the members of an organization, we distinguish between two types of micro-organizations: agents and organizational units.

An *organizational unit* (or simply *unit*) is some subset of the organization[3]. Examples of units in an organization could be marketing division, word processing center, and sales department. In general, a unit can be composed of other units. For example, the marketing division may be composed of the sales department and the advertising department. A unit may also be part of more than one other unit. For example, a project planning group may be part of several departments in an organization. Each unit contains knowledge that pertains to how it relates to other units in the organization as well as unit-specific knowledge (e.g., policy on taking holidays).

An *agent* is the smallest possible unit in an organization. It corresponds to a single individual within an organization. An agent can be part of one or more other units. This separation of units and agents provides flexibility in assigning individuals to units and allows the individuals that are a member of a unit to change without affecting the unit-specific knowledge. In addition, units are useful for performing interactions with other organizational workers without having to know the agent(s) who are part of that unit at a particular time. For example, one can apply for graduate studies in a department of a university by writing to the "department" (instead of a particular individual within that department).

The agent and the organizational unit concepts are useful for partitioning organizational knowledge. Individual knowledge is associated with agents. Organization knowledge is associated with units. However, there is also a need to share knowledge between the two because the knowledge used by individuals in an organization often is closely related to the part of the organization (unit) of which they are a member. In the next section, we will see that the MOAP model introduced in Section 2, can be used to both encode the knowledge of agents and units, and to facilitate the migration of knowledge among these entities.

4 SUPPORTING KNOWLEDGE MIGRATION

4.1 Structuring Organizational Knowledge Using MOAPs

We have argued that there are two major collections of organizational knowledge: *agent* and *unit*. Agent knowledge is only accessible to the organizational worker who owns it. This type of knowledge has been personalized by, and in a sense optimized for, an individual organizational worker. Inconsistent agent knowledge, used by different organizational workers, represents different opinions on common tasks [WoLo86]. On the other hand, unit knowledge often is used by two or more organizational workers. This type of knowledge has evolved through the experience of organizational workers to obtain standardization or to deal with the environment in which the organization operates (e.g., federal reporting laws).

In our representation of organizational knowledge, a MOAP is associated with each agent and unit in an organization. An agent MOAP is owned by only one organizational worker and encodes an organizational worker's individual knowledge. Unit MOAPs are not owned by any organizational worker and encode organization knowledge that is shared among the organizational workers associated with the unit.

Figure 2 provides a simple example of how agent and unit MOAPs can be used to partition and share organizational knowledge. In the figure, the rectangular boxes are agent MOAPs and are owned by the organizational worker whose name is inside the box. The ovals are unit MOAPs which contain knowledge that can be used by agent MOAPs associated with them. For example, the common, organizational-related sales knowledge used by "Jill" and "Sam" is associated with

[3] Note that the organization as a whole is also considered to be a unit.

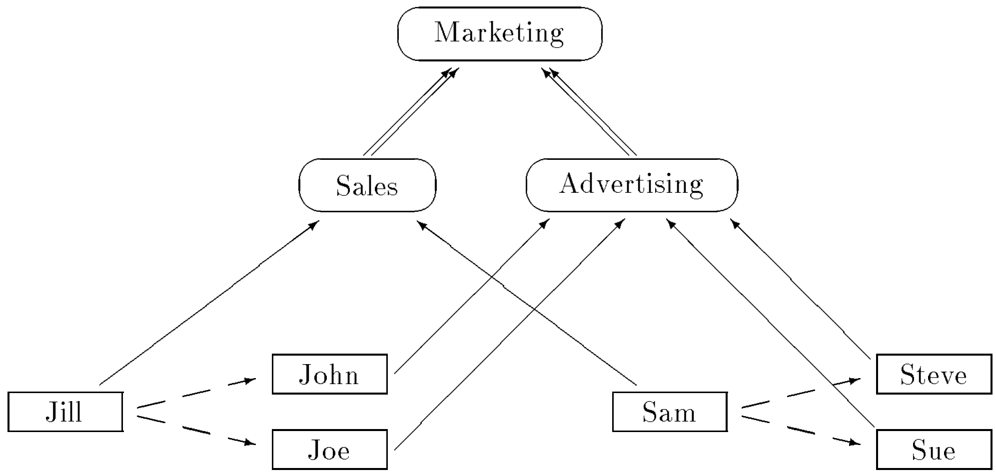


Figure 2: An example of agents and units

the unit "Sales" which is part of the unit "Marketing[4]."

The dotted arrows in Figure 2 are the same as the dotted arrow in Figure 1. That is, for example, Jill can request John's MOAP to perform some task for her. John, in this case, will deal with Jill's request using the knowledge in his agent MOAP and in the units of which he is a member. He is not required to pass his personal knowledge of how he accomplishes his task to Jill or to other agent MOAPs.

All organizational activities are initiated in agent MOAPs. Unit MOAPs are used only to structure, partition, and store organization knowledge. In this way, responsibility for organizational tasks resides ultimately with some agent MOAP (i.e., some organizational worker).

With respect to the entities available in the MOAP model, agent and unit MOAPs emphasize different capabilities of a MOAP. Agent MOAPs make use of all of the entities described in Section 2. Unit MOAPs use primarily the procedural knowledge and data bases. To a lesser extent they also make use of problem-solving knowledge and the activity coordinator. The workspace and activity agents, while present, are not used in the normal course of organizational activity processing since all activities involving these entities are initiated by agent MOAPs. In this respect, unit MOAPs serve more as repositories of knowledge and data rather than as active processing entities.

4.2 Knowledge Use and Interaction

The procedural knowledge and the problem-solving knowledge are the main ways in which agent MOAPs and unit MOAPs interact when performing organizational tasks. We first will discuss how the problem-solving knowledge is represented and used in agent and unit MOAPs and then indicate how procedural knowledge differs from problem-solving knowledge.

Both agent and unit MOAP problem-solving knowledge is encoded as a set of domain rules (DR) and meta-level knowledge for use when modifying the DR. In addition, they contain a set of constraint rules (CR) which, for unit MOAPs, define the policies that the organization has placed on the actions of the organizational workers performing a task, and meta-level knowledge that is used to modify the CR.

[4] We note that implicitly all agents are part of the unit called "Organization" which contains knowledge common to all units in the organization.

The domain rules are specified in an IF-THEN format [vMSB84] with the conclusion stating the task that is accomplished once all of the conditions in the premise are satisfied. The following domain rule encodes the policy that if a purchase of less than $1,500 is made for travel, then the purchase can be approved.

purchaseApproval:
IF PurchaseAmount(Purchase) $\leq$ 1500 AND
PurchaseCategory(Purchase) = travel AND
Boss(Person Boss) AND
SendCopy(Person Purchase Boss)
THEN approvePurchase(Person Purchase)

Simple facts can also be encoded by assertions of the form *factName*: predicate(parameters). Thus the assertion *holiday1*: holiday(December 25) could be used to represent the fact that December 25 is a holiday.

There will be some predicates, called actions, that do not require the use of other rules or facts in order to be satisfied. Actions are declared using assertions of the form *Action*(PredicateName). A simple example of an action would be the sending of an electronic copy of a document to another organizational worker. This class of predicate could also be used in situations where the intervention of the organizational worker is required, such as the filling in of a form, in order to determine the values of some variables.

The constraint rules are meta-constraints or meta-level rules [DaBu84] that make demands on the actions that are to be performed in a particular situation. These rules are also expressed in an IF-THEN format very similar to the domain rules. A builtin predicate, *proving*, is used to test whether the parameter to this predicate, which is itself a predicate, was proven during the planning (see below). The conclusion specifies actions that must be present in the plan. The following constraint rule defines the policy that when a D level manager is approving a purchase order, a copy must be sent to the manager of this manager and a copy must be sent to the accounting department.

purchaseApprovalConstraint:
IF *proving*(approvePurchase(Manager, Purchase)) AND
ManagerLevel(Manager, d) AND
Boss(Manager, Boss)
THEN SendCopy(Manager, Purchase, Boss) AND
SendCopy(Manager, Purchase, accounting).

The planning process controls the formation of an agent MOAP's method for solving a problem. During the planning process, the planner can use domain rules and facts from the particular agent MOAP that is involved as well as the unit MOAPs of which it is a part. This also holds true for the constraint rules. Thus the agent MOAP appears to "inherit" both the domain rules and the constraints of the applicable unit MOAPs. Given the dynamically changing contents of the rule bases and the changing structure of the organization, simple static inheritance by the agent MOAPs is not possible. Instead, the unit MOAPs must accept requests from the agent MOAPs about the current contents of their DR and CR.

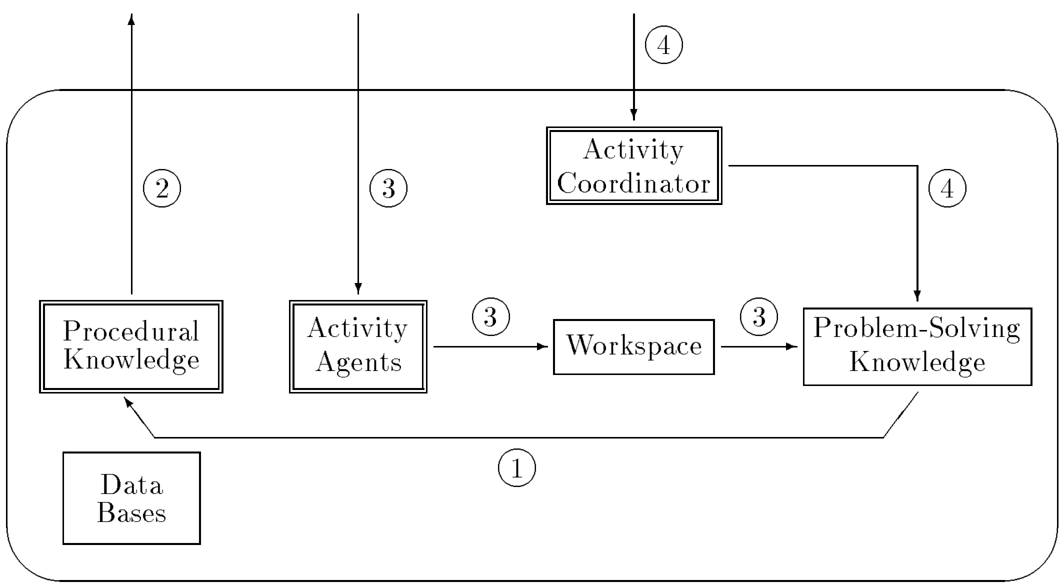
The plans generated may be incomplete due to their dependence on information supplied by the organizational worker, such as values filled into a form [Crof87]. This can be detected in the planning process as an action predicate that has some unbound parameters. This action will provide information that is needed to continue the planning, and thus the action must be executed during the planning procedure. If this action succeeds, the planner continues attempting to create the rest of the plan.

The resulting plan contains a list of actions that must be executed, along with a list of predicates that were proved. This plan is then compared with the applicable constraints to determine if it is valid. If a constraint is violated, the planning procedure will be re-invoked to begin backtracking.

For efficiency, procedural knowledge is encoded in a form that does not require planning to execute the task. Instead, a representation of a valid plan to execute the task is saved by the agent MOAP. During execution of the procedural task, if any of the constraints for the procedure are violated, then we need to replan execution of the task (i.e., it becomes a problem-solving task).

4.3 Different Types of Knowledge Migration

The MOAP model can support three different types of knowledge migration. These are shown in Figure 3 and will be discussed in this section individually.



1 Internal standardization

2 Macro-standardization

3 Acquiring external knowledge using activity agents

4 Acquiring external knowledge from its owner

Figure 3: Different Types of Knowledge Migration

4.3.1 Internal standardization

When a new task or a new situation arises, the organizational worker might not have enough knowledge to handle it. Under this situation, he must treat the task as a problem-solving one, and experiment with possibly several alternatives. In this case, the agent and unit MOAPs and their problem-solving knowledge can assist him to accomplish the task. However, organizational workers learn from experience. As they know more about a particular task, the task can become very routine and structured. In this case, the task should be handled by procedural knowledge rather than problem-solving knowledge in order to gain better performance. Internal standardization is thus the process of migrating agent MOAP problem-solving knowledge to procedural knowledge[5].

4.3.2 Macro-standardization

Macro-standardization is the migration of procedural knowledge from an agent MOAP to a related unit MOAP, or from one unit MOAP to another related unit MOAP. The process involves inserting domain rules, appropriate constraint rules, and a valid procedure into the target unit MOAP and removing the same domain rules and procedure from the source agent or unit MOAP (see [WiLo89] for details). The need for this kind of migration can happen in two ways:

1. some higher authority decides to eliminate the inconsistent knowledge used by different MOAPs in performing a task, or

2. it is recognized that the same knowledge exists in several MOAPs and it is decided to consolidate this knowledge and make it available to a larger group of organizational workers.

The actual migration of the procedural knowledge can be done automatically as indicated above, but the determination of which procedural knowledge to migrate requires the participation of one or more organizational workers.

We choose not to migrate problem-solving knowledge in this way in our approach. While we could do so, we believe that it is not appropriate to do so. Procedural knowledge is that knowledge that has been tested by organizational workers and found to be useful for a large class of situations. Problem-solving knowledge, on the other hand, is largely untested and/or applies in only a few situations. As such, we have taken the approach of waiting until problem-solving knowledge becomes procedural knowledge before allowing it to be migrated in this way. Note that in certain cases, problem-solving knowledge can still be migrated as discussed in the following section.

4.3.3 Acquiring external knowledge

The macro-standardization described in the previous section can only share knowledge if the MOAPs are directly related. Unfortunately, MOAPs are not always directly related. Sometimes, the knowledge needed by a MOAP resides in an unrelated part of the organization or with the user. Sometimes, the usefulness and correctness of the knowledge for performing a task requires more testing. Under these circumstances, we cannot create a unit MOAP to share the knowledge. However, we can copy the knowledge from one MOAP to another one or ask the user to supply missing knowledge. This is the last type of knowledge migration (i.e., acquiring external knowledge) we will discuss in this section [TFGN87].

This type of knowledge migration is achieved in one of two ways. First, we can have an agent MOAP's activity agent collect relevant knowledge from other MOAPs (agent or unit) [Woo90]. The activity agent will travel to different MOAPs according to the recommendations made by the sending MOAP and the visited MOAPs. Knowledge that appears to be relevant or useful will be brought back and placed in a workspace of the sending MOAP. The sending MOAP together with its owner (organizational worker) will decide (i.e., not the activity agent) which pieces of knowledge are relevant or useful. The migration path in this case is therefore:

[5] As indicated in the previous section, this involves identifying and saving a representation of a valid plan that can be used to execute the task in the future. One of the major problems in supporting internal standardization is in the identification of a plan that is worth saving. See [Woo88] for some possible approaches to dealing with this problem. In all cases, the ultimate decision of which knowledge to migrate and when to migrate it rests with an agent MOAP's owner.

$$\text{Outside MOAP} \longrightarrow \text{Activity Agent} \longrightarrow \text{Workspace} \longrightarrow \text{Problem-Solving Knowledge}$$

Note that if the knowledge turns out to be not useful, the migration will stop at the workspace (i.e., since the workspace is a temporary working memory, the knowledge will be deleted together with the workspace when the task is done). On the other hand, if the knowledge is useful, then the sending MOAP will place the knowledge in its problem-solving knowledge entity.

Note that the knowledge brought back by an activity agent will be treated as problem-solving knowledge. We handle this knowledge in this way because it may require further testing before it can be considered procedural knowledge in the sending MOAP. In the case where acquired knowledge is represented as procedural knowledge in another MOAP, it will be converted to problem-solving knowledge[6].

The second way to acquire external knowledge is from an agent MOAP's owner. The migration path in this case is:

$$\text{MOAP's Owner} \longrightarrow \text{Activity Coordinator} \longrightarrow \text{Problem-Solving Knowledge}$$

In this case we do not place the knowledge in the workspace because we assume that knowledge supplied by a MOAP's owner is useful knowledge and will be retained by the agent MOAP.

5 CONCLUDING REMARKS

In this paper we have described an approach for knowledge migration in organizations. We presented the MOAP model for structuring knowledge and distinguished between agent and unit MOAPs. Finally we described various ways in which knowledge can evolve and migrate in our model. We believe that our approach to knowledge migration meets the criteria for structuring and partitioning knowledge outlined in Section 1.

Procedural and problem-solving knowledge are supported in a uniform way in our approach allowing problem-solving knowledge to migrate to become procedural knowledge and vice versa. Knowledge is partitioned and separated allowing conflicting knowledge to co-exist in the same organizational support system. At the same time, cooperation is supported by allowing agent MOAPs to interact with other agent MOAPs in both structured (delegation via requests to other MOAPs) and unstructured ways (gathering of knowledge via activity agents). In addition, knowledge that is useful to more than one individual in an organization can be shared by migrating it to unit MOAPs and organizational policies can be enforced by the association of agent MOAPs with unit MOAPs.

To date, we have implemented two prototypes that demonstrate the feasibility of our approach to organizational knowledge representation and migration. The MOAP architecture described in Section 2 has been prototyped without the knowledge migration part [Woo88]. A knowledge sharing system which supports the macro-standardization type of knowledge migration has also been prototyped [WiLo89]. We are currently designing a prototype that will combine the functionality of these two systems and will incorporate internal standardization, macro-standardization, and external knowledge acquisition facilities.

[6] One may ask why, when a MOAP needs an external piece of knowledge, we do not just communicate with the MOAPs who have the knowledge, and save the effort of sending a representative to travel among different MOAPs? The reason is that organizations are open systems [Hewi86]. That is, the specific other MOAPs that a MOAP might interact with cannot be determined a priori [Tsic89]. This is due to the open-ended and evolutionary nature of organizational work [WoLo88] (e.g., what is true last month might not be true now). Direct communication will limit an agent MOAP to communicate only with those that it initially knows or is explicitly informed of by its owner. On the other hand, an activity agent can dynamically modify its travel path by the information and knowledge available in the visited MOAPs. In this case, it can travel to a MOAP which is initially not known to it or its sender.

ACKNOWLEDGEMENTS

This research was supported in part by the Natural Sciences and Engineering Research Council of Canada, and in part by the Information Systems Research Bureau (ISRB) of the Faculty of Commerce and Business Administration, the University of British Columbia. The current sponsors of ISRB are Arthur Anderson & Company, Bell-Northern Research Laboratory, and British Columbia Telephone Company.

References

[Crof87] Croft, W.B., "Representing office work with goals and constraints," *Proc. IFIP WG8.4 Workshop on Office Knowledge: Representation, Management and Utilization*, 1987, pp. 13-18.

[DaBu84] Davis, R. and Buchanan, B.G., "Meta-level knowledge," Chapter 28, *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project.* Shortliffe, E.H. and Buchanan, B.G., eds., Addison-Wesley, Reading, MA, 1984.

[EHLR80] Erman, L.D., Hayes-Roth, F., Lesser, V.R., and Reddy, D.R., "The Hearsay-II Speech-Understanding System: Integrating Knowledge to Resolve Uncertainty". *ACM Computing Surveys*, Vol. 12, No. 2, pp. 213-253, June 1980.

[Hewi86] Hewitt, C., "Offices Are Open Systems," *ACM Trans. on Office Information Systems*, Vol. 4, No. 3, pp. 271-287, July 1986.

[Tsic89] Tsichritzis, D., "Object-Oriented Development for Open Systems". *Proceedings of the IFIP Congress '89*, 1989, pp. 1033-1040.

[TFGN87] Tsichritzis, D., Fiume, E., Gibbs, S., and Nierstrasz, O., "KNOs: KNowledge acquisition, dissemination, and manipulation Objects," *ACM Trans. on Office Inf. Sys.* **5**(1), 1987, pp. 96-112.

[vMSB84] van Melle, W., Shortliffe, E.H., and Buchanan, B.G. "EMYCIN: A knowledge engineer's tool for constructing rule-based expert systems," Chapter 15, *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project.* Shortliffe, E.H. and Buchanan, B.G., eds., Addison-Wesley, Reading, MA, 1984.

[WiLo89] Williams, L. and Lochovsky, F.H., "Supporting Knowledge Migration in Organizations". *Proceedings of the IFIP Congress '89*, 1989, pp. 259-264.

[WoLo86] Woo, C.C., and Lochovsky, F.H., "Supporting Distributed Office Problem Solving in Organizations," *ACM Trans. on Office Information Systems*, Vol. 4, No. 3, pp. 185-204, July 1986.

[WoLo88] Woo, C.C. and Lochovsky, F.H., "Toward Supporting the Open-Ended and Evolutionary Aspects of Organizational Work". Working paper 88-MIS-022, Faculty of Commerce and Business Administration, The University of British Columbia, Vancouver, Canada, 1988.

[Woo88] Woo, C.C., "An Object-Oriented Model for Supporting Office Work". Ph.D. thesis, department of Computer Science, University of Toronto, 1988. (Published as technical report 88-MIS-025, Faculty of Commerce and Business Administration, University of British Columbia, Vancouver, Canada, 1988.)

[Woo90] Woo, C.C., "SACT: A Tool for Automating Semi-Structured Organizational Communication," *Proc. ACM SIGOIS Conf. on Office Inf. Sys.*, 1990, pp. 89-98.