Knowledge Communication in Intelligent Information Systems

Carson C. Woo[*]
University of British Columbia

Frederick H. Lochovsky[†]
Hong Kong University of Science and Technology

Abstract

An intelligent information system may be composed of hundreds or thousands of entities each of which may possess some part of the overall knowledge of an organization. In such an environment there is a need for these entities to communicate in order to share knowledge and to cooperate in accomplishing organizational activities. In this paper we propose an architecture for modeling intelligent information systems and discuss how this architecture supports the communication of knowledge in such information systems.

1 INTRODUCTION

An intelligent information system (IIS) is an information system in which the entities that comprise the information system possess some capability to store and to process knowledge. An IIS can be composed of hundreds or thousands of such entities which we will refer to as *intelligent agents* or simply *agents* when the meaning is clear. Note that an intelligent agent may be a person, which we will refer to as a *user (agent)*, or a program or application, which we will refer to as a *(software) agent*. As such, IISs are a symbiosis between humans and computers.

A major role of the intelligent agents in an IIS, and in particular that of software agents, is to *encapsulate* knowledge and operations on the knowledge[1]. In this respect, intelligent agents can be thought of as intelligent objects. Intelligent agents may contain knowledge of many different types (e.g., procedural versus problem-solving) and in many different forms (e.g., C programs, Pascal programs, rules, cases, etc.). However, this fact is hidden from agents who only know the capabilities of other agents in an IIS.

Knowledge communication in an IIS deals with the *acquisition*, *dissemination*, and *transformation* of knowledge within and among intelligent agents[2]. In traditional knowledge base development

[*]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 1Z2. Email: woo@unixg.ubc.ca

[†]Author's address: Frederick H. Lochovsky, Department of Computer Science, The Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong. Email: fred@cs.ust.hk

[1]In fact, intelligent agents can be used as a mechanism for encapsulating existing data and applications in an organization.

[2]We use the term *knowledge* here in a generic sense. Knowledge in the context of this paper can be data, programs, rules, cases, etc. For the purpose of the discussion in this paper, we do not think it useful to distinguish among these different kinds of knowledge.

(e.g., expert systems) we often assume the presence of a knowledge engineer whose role is to facilitate knowledge communication. While knowledge engineers may certainly still be present and required for IISs, we also believe that it will not be feasible to employ them for every aspect of knowledge communication. If IISs are to be widely used and easily usable, then appropriate mechanisms will have to be developed to allow the users of such systems, and the agents that comprise such systems, to easily and effectively communicate knowledge without the presence of a human intermediary.

In this paper we propose an architecture, called the micro-organization activity processor (MOAP) architecture, for modeling IISs[3]. A major emphasis of the architecture is on mechanisms for the communication of knowledge among the software agents that comprise an IIS. In the following section we present the types of knowledge communication that we support in our architecture and discuss the shortcomings of existing systems in supporting them. Our proposed architecture for modeling IISs and for supporting knowledge communication is described in Section 3. In Section 4, we discuss how this architecture supports the knowledge communication types outlined in Section 2. Finally, we conclude our discussion and present possible future research directions in Section 5.

2 KNOWLEDGE COMMUNICATION

Knowledge communication can take many forms in IISs. In this paper we are concerned specifically with three forms of knowledge communication – acquisition, dissemination, and transformation. From the perspective of an intelligent agent (user or software), knowledge acquisition is the process of *receiving knowledge* from some other agent. Knowledge dissemination is the process of *making knowledge available* to some other agent. Knowledge transformation is concerned with *changing knowledge* that is of one type into another type. In this section, we discuss these types of knowledge communication and how they may arise in IISs from the perspective of software agents[4]. We progress from those that occur solely within a software agent to those that occur among software agents and finally to those that occur between a software agent and a user.

2.1 Within an Agent

Organizational activities have often been characterized as consisting of a mixture of procedural and problem-solving [32]. Procedural activities are those of a routine nature for which a prescribed step-by-step solution is known. Knowledge of procedural activities (i.e., procedural knowledge) often can be encoded by means of a step-by-step solution in a computer program. Problem-solving activities are novel and non-repetitive, and must be solved with creativity, initiative, and originality [17, 32]. Knowledge of problem-solving activities (i.e., problem-solving knowledge) often is stated as goals and rules for attaining the goals.

In general, most organizational activities, consist of an intermingling of procedural and problem-solving activities. To support organizational activities, an agent may need to contain different types of knowledge (e.g., procedural and problem-solving), may need to keep different types of knowledge in more than one form (e.g., rules, cases, etc. for problem-solving knowledge), and may need to keep the same knowledge as more than one type or form of knowledge (e.g., as both procedural and problem-solving). More importantly, it may become necessary for one type or one form of knowledge to be transformed into another type or form.

[3] The intent of this paper is to provide a *specification* of the functionality provided by the MOAP architecture. Thus, we do not discuss, in general, the implementation aspects of the architecture.

[4] Henceforth, we will refer to software agents simply as *agents* unless the meaning is ambiguous.

For example, consider procedural and problem-solving types of knowledge. It is possible that a procedural activity enters a situation where the existing procedure can no longer deal with the activity. In this case, the activity may have to be handled as a problem-solving activity. Hence, there may be a need to transform procedural knowledge into problem-solving knowledge so that it can be used during problem-solving. Going in the other direction, organizations tend to force organizational activities to be procedural as much as possible. In this way, their outcome is more predictable and it is possible to perform the activity more efficiently. Consequently, the tendency is to move problem-solving activities into a procedural mould. Hence, there is a need to transform probelm-solving knowledge into procedural knowledge. In both of these cases, it may be additionally useful to retain the original form of the kowledge in case it is ever needed again.

As a consequence of these requirements, an IIS needs to allow for multiple representations of knowledge to coexist within agents, for knowledge to be multiply represented, and for knowledge to be transformed from one type and form to another as situations arise that require it.

2.2 Among Agents Within the Same Organization

Agents in IISs are able to acquire and store knowledge independent of other agents. Thus, among the agents there are independent rationales for solutions and, because there is no central control mechanism, there may be multiple, possibly inconsistent, possibly competing, ways of performing organizational activities none of which has a unique claim to validity [16]. On the other hand, in order to solve organizational problems, these agents will have to cooperate and share knowledge amongst themselves [19]. In addition, there may also be consistent knowledge that should be shared by a group of agents or all agents. This knowledge may provide explicit methods for performing certain organizational activities (e.g., how to prepare a weekly work progress report). It also may contain constraints (or policies) for certain actions to be taken by organizational workers that should not be violated regardless of the circumstances (e.g., a full time graduate student cannot work more than 12 hours a week).

An IIS will have to provide mechanisms that allow agents to maintain their own, independent knowledge bases and to guard against acquiring inconsistent knowledge while at the same time allowing their knowledge to be disseminated to and shared with other agents. It also must provide ways of sharing knowledge so that individual knowledge and group knowledge can interact cooperatively and consistently.

2.3 Among Agents in Different Organizations

When we cross organizational boundaries, there may be technical and policy issues that need to be overcome to facilitate knowledge communication. For example, on the technical side, similar types of knowledge may be represented in different ways in two (or more) organizations. On the policy side, issues of competition and disclosure of proprietary knowledge will need to be considered.

An IIS will have to provide mechanisms for transforming one form of knowledge into another, perhaps via a common knowledge representation model [30]. It also will require appropriate security mechanisms to deal with issues of knowledge sharing and privacy among agents in different organizations.

2.4 Between Users and Agents

This is perhaps the most obvious form of knowledge communication. Agents will have to acquire knowledge from users and conversely will have to disseminate knowledge to users. Many questions regarding the nature of the human-computer interface arise. Note that traditional methods of user-system knowledge communication may not be appropriate in this context. Knowledge can take many forms in an organization (e.g., facts, rules, guidelines, policies, etc.) and may best be communicated in its "natural" form, rather than in some artificial way as may be required internally by an agent.

An IIS will have to provide a usable and functional user interface to allow an organizational worker to easily communicate knowledge to the agent. In turn, the agents will need to disseminate their knowledge to the organizational worker in a way that is compatible with the user's capabilities and work environment.

2.5 Related Work

Many efforts at providing facilities for IISs can be cited [2, 3, 4, 9, 10, 11, 12, 20, 23, 24, 31]. Most of these efforts address architectural or knowledge representation issues in IISs. Very few aspects of knowledge communication have been addressed. In this section we briefly review these efforts and discuss how they are related to our own work.

One of the earliest IIS efforts was that by Barber. In [2, 3] a model for representing organizational knowledge, based on the use of a semantic knowledge base and an inference engine for proving the feasibility of goals asserted by the user, is presented. The OMEGA knowledge representation language is used to express goals and constraints. The knowledge representation paradigm is essentially that of a single knowledge base (i.e., knowledge is not partitioned). In addition, knowledge communication is only between a single agent and a user.

The POISE project had as a goal the support of both procedural and problem-solving knowledge [9]. However, the problem-solving knowledge resides with the user and not in the system. Although knowledge transformation within an agent does exist, it can only be transformed from procedural to problem-solving. No facility is provided to transform problem-solving knowledge to procedural knowledge.

A related project to POISE, called POLYMER, emphasized the planning aspect of problem-solving activities [10]. Until recently, the knowledge communication aspects of organizational activities were not emphasized [4].

The lack of support for decentralized planning in POISE was the main motivation for the OFFICE project [31]. In this project, work done in the areas of distributed artificial intelligence and multi-agent planning were applied to support decentralized organizational activities. This work, in some ways, supports some aspects of knowledge communication among agents within the same organization.

The KNO environment, being developed at the University of Geneva, is an object-oriented environment that provides active, nomadic, and adaptive objects [37]. Objects can trigger their own execution, can migrate to new environments, and can acquire new operations dynamically. They negotiate, cooperate, and learn from other KNOs. Communication is through a shared workspace, called a blackboard, that is local to the environment. There are different kinds of KNOs: *user KNOs* which exist for communication with a specific user, and *agent KNOs* which exist to perform some action on behalf of some other KNO (perhaps in some other environment). The KNO environment supports many forms of knowledge communication as discussed in this paper although knowledge transformation is not specifically supported. As a form of user agent communication, one of the user interfaces to the KNO environment is attempting to use animation to illustrate the properties of different agent KNOs to the user [7].

The Ubik project is attempting to facilitate the development of IIS applications [11, 12, 13]. A basic premise of the approach is that an organization consists of multiple, distributed sub-organizations (similar to our micro-organizations – c.f. Section 3.1) each of which is developing its applications with minimum coordination with other sub-organizations. The goal of Ubik is to facilitate the development of these applications and to control their execution. Message passing is the only mechanism for supporting inter- and intra-organization knowledge communication among agents. Although questors [13] are proposed as a construct with the ability to move to different sub-organizations for collecting data (similar to KNOs), other than the mechanism for moving, communication is as straightforward as message passing. Finally, no mechanism is available to support knowledge communication within an agent.

Part of the COKES project [20] has investigated embedding organizational knowledge in a network of distributed cooperating agents (called knowledge-based or expert "assistants" and servers). The agents incorporate both factual and procedural knowledge. Similar to Ubik, the agents, which may reside on different machines, cooperate in solving problems or completing activities by passing messages.

The Object Lens project [23, 24, 27, 28] is an IIS which provides facilities to allow unsophisticated computer users to create their own agents. The system uses the idea of semistructured objects, consisting of semistructured data and associated rules, to represent information in a form that can be processed intelligently by both users and agents. Users can collect these objects into customizable folders for display and can create semiautonomous agents for automatically processing information in different ways at different times. Object Lens thus addresses the problem of knowledge communication between users and agents. Knowledge sharing, however, is only possible via centralized repositories of information. Knowledge communication within or among the semiautonomous agents is not supported.

The need for knowledge communication facilities in organizations is described in [15]. In [22] the way that scientific communities communicate knowledge is studied. In that setting, many scientists work concurrently on similar or dissimilar problems with or without knowledge of the work of other scientists. The scientists can hold conflicting views on the solution to the same problem but, in the long run, certain views become the accepted standard. In general, little in the way of practical approaches to addressing this problem have appeared in the literature.

In general, most of these systems support only some types of knowledge communication as discussed in this section. Knowledge communication *per se* has not been a focus of these systems. Moreover, none of these systems has proposed an architecture for supporting the different types of knowledge communication discussed here. In the next section, we present the MOAP architecture which addresses this deficiency.

3 MOAP: AN IIS ARCHITECTURE

In this section, we describe an architecture for modelling IISs. In Section 4 we discuss how this architecture supports the types of knowledge communication outlined in Section 2. The architecture provides a framework within which one can model IISs. It is *not* an IIS *per se*. While we have implemented, and are continuing to implement, parts of the architecture [25, 29, 40, 43], more research and development is needed for some of the components before we have a *fully* workable system based on this architecture. However, the architecture is an important contribution to the area of IISs because the framework it provides can be used to integrate the different types of technologies currently being researched and developed in this area, particularly in the area of knowledge communication.

3.1 Overview

The purpose of an IIS is to support the activities that occur within an organization. As such, it seems reasonable to assume that the organization of an IIS will, more often than not, parallel that of the organization in which it is embedded. It has often been observed that organizations behave much like a society [19, 22] (i.e., they are composed of many, perhaps specialized, (sub)units which each operate semi-autonomously, but often cooperatively, to accomplish some goal). Therefore, in our approach to modelling IISs, we view an organization as a composition of many units. We call each one of these units a *micro-organization*[5]. This name is chosen because an organizational unit, in this case, behaves much like an organization except that it only contains a small domain of organizational knowledge. Examples of micro-organizations are department head's office, payroll clerks, and word processing advisor. Similar to a society, 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.

Assuming that we are given the structure of an organization in terms of micro-organizations[6], we can embed an IIS within the organization by assigning an *activity processor* to perform all or part of the functions of some of the micro-organizations. An activity 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 system architecture of a micro-organization activity processor (MOAP)[7]. Although the architecture is the same for all the MOAPs, the knowledge between any two of them can be inconsistent (see Section 4.2).

There are six different types of entities in a MOAP: data repository, procedural knowledge, problem-solving knowledge, workspace, activity agent, and activity coordinator. Each one of them plays a distinct role in representing knowledge and supporting knowledge communication as discussed in the following sections.

There are three major reasons for dividing a MOAP into these different entities. First, we want to be able to explicitly represent different types and forms of knowledge within an agent (even though externally this division may not be apparent). We believe that this models how humans represent and use different types and forms of knowledge (e.g., see work on the Model Human Processor [5] and the Goals, Operators, Methods, and Selection rules (GOMS) model [6]). Second, each entity represents the potential use of a different type of technology. For example, the "procedural knowledge" entity can maximize the use of data processing technology to provide efficient processing of activities, while the "problem-solving knowledge" entity can maximize the use of artificial intelligence technology to provide powerful reasoning capabilities. Finally, the division is useful for understanding and modelling the different types of knowledge communication mentioned in Section 2 as will be shown in Section 4.

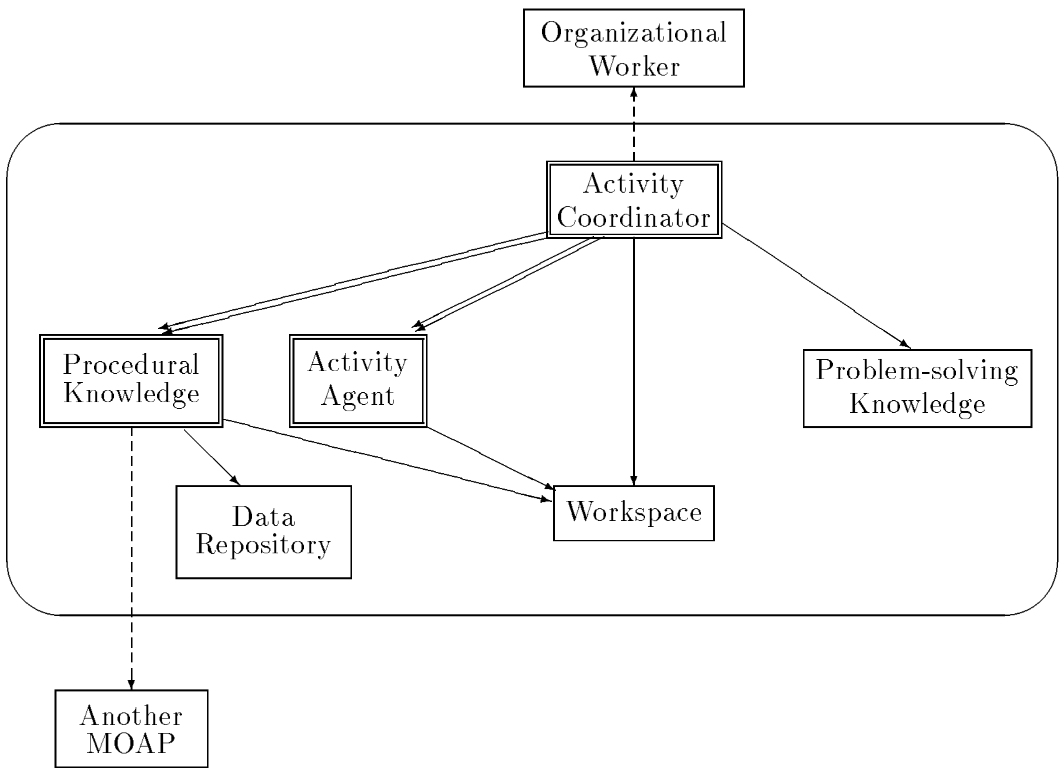
We made two major assumptions in the MOAP architecture. First, we assume that all micro-organizations are supported by a MOAP-based system (i.e., with the same type of entities describe in this section). However, different MOAPs can use different languages or knowledge representation techniques to implement these entities (see Section 4.3). Second, we assume that one MOAP is attached to only one user and a user has at least one MOAP[8]. In this case, what knowledge each MOAP should have depends on what knowledge its owner needs to carry out his responsibilities in the organization. The distribution of responsibility to organizational workers, on the other

[5] A micro-organization is equivalent to a "model" in Ubik [11].

[6] How one might structure an organization in terms of micro-organizations is beyond the scope of this paper.

[7] The architecture of a MOAP is similar to the Hearsay-II expert system [14]. 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).

[8] A user can have more than one MOAP if he plays more than one role in his organization



means supervises the execution of the entities pointed to.
means allowed to access the information in the entities pointed to.
means allowed to pass control to the entity pointed to.

means active entity.

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

hand, is beyond the scope of our work; this issue is dealt with in areas such as management and organizational theory. Nevertheless, we feel that these assumptions are reasonable and they should not cause any major problems to the usefulness of our proposed architecture.

3.2 Data Repository

A data repository stores persistent information. We do not presume any specific representation for data repositories. They can be implemented as data bases managed by a data base management system or they may simply be operating system files. The major criteria for data to qualify for storage in the data repository is that it be reliable, factual data that is amenable to structured storage and retrieval operations and that must be kept for some period of time. Thus, any business data that would normally be kept in data bases or files in an organization would be maintained by the data repository entity of the MOAP architecture.

3.3 Procedural Knowledge

Procedural knowledge contains the well-thought-out, well-organized, well-tested, and well-adapted organizational knowledge. In the MOAP architecture, procedural knowledge is described in terms of *tasks* which can be broken down into a hierarchy of subtasks. To qualify as procedural knowledge, these tasks should be able to be executed successfully in a specified sequence without encountering unexpected circumstances. Each task is modelled using the following seven components:

1. A set of preconditions which determine under which circumstances the task should be executed (e.g., on the arrival or creation of certain data).
2. A program p for executing the task. We do not assume any particular representation for this program; it can be in any programming language amenable to execution by a computer.
3. A function $f : I^m \longrightarrow O^n$ for predicting the outputs, O^n, of the task given a set of inputs, I^m. For example, for a room booking task the inputs might be desired times and a required seating capacity; the output might be a set of rooms available at those times with the required capacity.
4. A function $f^{-1} : O^n \longrightarrow I^m$ which is the inverse of f.
5. A set of postconditions (i.e., things that must be true after the program p executes).
6. A list of tasks that will "undo" the side effects produced by p when the postconditions are not satisfied. For the room booking task, the inverse function would be one that freed the room just booked.
7. A list of other tasks to invoke at the end of program p's execution (i.e., a specialized set of postactions). Invoke, in this case, means (1) notify the tasks (if any) that need to perform work after this task has finished its work, and (2) put its outputs in a designated area in the workspace for the tasks it invokes to pick up.

So far, this view of procedural knowledge assumes no interactions among different MOAPs (i.e., work is done by only one MOAP). On the other hand, a task might require the participation of several MOAPs. The MOAP architecture models this kind of task using the dotted arrow shown in Figure 1. That is, a task can have a MOAP attached to it. The program component of the task, in this case, contains all the necessary information for passing control to the attached MOAP and the action of invoking the attached MOAP. Note that this kind of control passing can only be used for activities that are structured and well-understood (i.e., we know the exact content of what needs to be passed and who to pass it to).

We will not elaborate further on the representation of procedural knowledge since that is not the emphasis of this paper. We will only remark that some organizational activities clearly are of a procedural nature (i.e., make use of procedural knowledge) because of their well-understood nature.

3.4 Problem-solving Knowledge

Problem-solving knowledge describes strategies and serves as advice to the activity coordinator (i.e., it provides assistance to the activity coordinator when the procedural knowledge is insufficient to perform an organizational activity).

Problem-solving knowledge can be grouped into three major categories:

1. *Knowledge related to facts.*
   Normally, facts should be available in the data repository (see Figure 1). However, some data might be unavailable at the time when they are needed (e.g., the sales figure for the current month), some data, when getting it, might have unwanted side effects (e.g., need to pay money for each access), and so on. Under these circumstances, approximate values or some guidelines for predicting the values can be made available in the problem-solving knowledge.

2. *Knowledge related to the missing parts of procedural knowledge.*
   Some of the components of a task can be missing because their exact value or close approximation is not available. In some of these cases, however, some specific cases of these components are known due to past experience [33]. For example, we might not know the exact formulae to calculate the number of days given the year, but we might know that there are 365 days in the year 1990. For these cases, the problem-solving knowledge can provide assistance to fill in the missing parts. The problem-solving knowledge, under this category, can be one of the following:

   (a) The function f of a task.
   (b) The function f^{-1} of a task.
   (c) The special set of postactions of a task (see point number 7 in Section 3.3).

3. *Knowledge playing the role of procedural knowledge.*
   The problem-solving knowledge in this category is a task which has not yet been proven to be stable. It consists of all the components of a task. Knowledge such as heuristics and alternatives for solving a problem also belongs to this category. If performing an organizational activity requires knowledge from this category, then the activity coordinator will plan an execution sequence for all the tasks (including those in this category) and make sure that the estimated final result is acceptable before physically executing them. Planning is necessary in this case because, since these tasks are not yet proven to be stable, they may often fail. By using them in a planning mode, we avoid having to frequently undo actions performed by the program p in these tasks. This is the major difference between placing the knowledge here rather than in the procedural knowledge entity.

The representation for problem-solving knowledge is implicit in the preceding paragraph. That is, for the first category, facts are represented the same way as data in the data repository entity. For the second category, problem-solving knowledge is just the components of a task. For the last category, it is represented exactly the same as a task.

3.5 The Workspace

The 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 the workspace can be picked up by other active entities at a later time. In addition, activity agents from other MOAPs can access the workspace. The workspace is divided into five areas:

1. *Data passing area.*
   This is used by tasks in the same MOAP to pass data among themselves.
2. *Bulletin board area.*
   As the name implies, this area is accessible to any entity internal to a MOAP which is interested in reading the posted information. This area provides a very useful mechanism for problem solving because it allows an entity to put information into it without knowing which other entities will pick it up for their use. In this case, an entity in a MOAP does not have to know the name (or even the existence) of another entity in order to communicate with it.
3. *External communication area.*
   A visiting activity agent uses this area to communicate with the activity coordinator of the MOAP (see Section 3.6).
4. *Reasoning working area.*
   This is used by the activity coordinator when reasoning (i.e., planning).
5. *Temporary knowledge storage area.*
   This is used to temporarily save the knowledge brought back by an activity agent (see Section 4.2).

The workspace entity provides the mechanisms for accessing these different areas.

3.6 The Activity Agent

An activity agent is a representative of a MOAP which specializes in communicating with and collecting information from other MOAPs (i.e., external communication). An activity agent consists of the following parts:

1. *A MOAPs list.*
   This identifies the MOAPs yet to be communicated with.
2. *Local data.*
   This is used to store data gleaned from different MOAPs and to do any intermediate computation.
3. *Interaction consultant.*
   This guides the activity agent in its interaction when communicating with a MOAP.
4. *Next MOAP selection guidelines.*
   These guide the selection of the next MOAP to communicate with (i.e., the ranking of the MOAP entries in the MOAPs list).
5. *Termination conditions.*
   These determine when the activity agent will report back to its MOAP.

All communication between a visiting activity agent (say from MOAP α) and the visited MOAP (say MOAP β) is done via the *external communication area* of the workspace in MOAP β. More specifically, the activity agent puts its request into the workspace, and the activity coordinator in MOAP β picks up the request and tries to satisfy it using the knowledge available to it. When the activity coordinator in MOAP β satisfies the request, it puts the response into the workspace for the activity agent to pick up. If MOAP β cannot satisfy the request using the knowledge available to it, it can suggest other MOAPs who might be able to satisfy the request, and/or it can appeal to the user who owns it. Alternatively, the activity agent can simply time-out on waiting for a response and communicate with another MOAP.

The most crucial component of an activity agent is the *interaction consultant*. For this component to be useful, a good theory of interaction or communication is required so that, using a high-level language, the user can specify the application dependent part and the system can automate the rest. The work we have done in [41] on applying Searle's speech act work [34, 35, 36] to automate semi-structured communication can be applied to developing such an interaction consultant component.

3.7 The Activity Coordinator

The function of the activity coordinator is to control the execution of tasks and activity agents, and the use of problem-solving knowledge within its micro-organization jurisdiction. It also responds to inquiries by activity agents from other MOAPs.

In general, the bias of the MOAP architecture is to try to use procedural knowledge whenever possible since it is generally more efficient to do so. Therefore, the activity coordinator enforces the following execution sequence for the different entities:

1. *Procedural Knowledge.*
   Given an activity, the activity coordinator attempts to find a task to carry it out. Procedural knowledge is used first because it performs organizational work most efficiently.
2. *Problem-solving knowledge.*
   If a task cannot be found or if the task runs into "difficulties" trying to accomplish the activity (e.g., the set of postconditions mentioned in Section 3.3 was not satisfied), the activity coordinator attempts to find appropriate problem-solving knowledge to carry on with the activity.
3. *Activity agent.*
   If problem-solving knowledge cannot be found or if the problem-solving knowledge runs into "difficulties" trying to accomplish the activity (e.g., the activity coordinator cannot use the problem-solving knowledge to successfully plan an execution sequence as mentioned in Section 3.4), the activity coordinator attempts to find an activity agent to carry on with the activity. Problem-solving knowledge is consulted before sending out an activity agent for external communication because problem-solving knowledge is located locally in the MOAP and represents the organizational worker's knowledge for performing activities. It, therefore, has higher priority than activity agents. Also, problem-solving knowledge should take much less execution time because it is much more easily accessible as compared to an activity agent which is required to communicate externally.
4. *User.*
   Finally, if an activity agent cannot be found or if the activity agent runs into "difficulties" trying to accomplish the activity, (e.g., it ran out of MOAPs with which to communicate), the activity coordinator consults the human agent to carry on with the activity.

For an entity to run into "difficulties" means that it cannot accomplish the activity assigned to it. For procedural knowledge, this might mean that an appropriate task could not be found. For problem-solving knowledge, this might mean that a plan could not be formulated. For an activity agent, this might mean that it could not find the information it was sent out to gather. In each case, the MOAP architecture attempts to carry on by passing control of the activity to the next entity in the sequence discussed about. When doing so, there may be a need for knowledge communication of various forms as will be discussed in Section 4, particularly for knowledge transformation so that the succeeding entity can carry on with the activity.

The activity coordinator specifically provides user-agent knowledge communication facilities for:

- querying information such as the execution status of tasks, what problem-solving knowledge has been used, and the content of the workspace;
- acquiring knowledge such as procedural knowledge, problem-solving knowledge, and activity agents.

3.8 The Role of the User in the MOAP Architecture

It should be implicit from the discussion of the activity coordinator in Section 3.7 that a MOAP will perform activities automatically if it has the knowledge to do so. This includes automatically sending requests to other MOAPs and automatically handling incoming requests. In other words, if a MOAP has complete knowledge for handling all activities, then it can replace its user owner. However, we believe that this is unlikely to happen unless a MOAP is intelligent enough to learn and to evolve automatically.

The role of the user is to handle those activities that his MOAP cannot perform (or those activities that the user does not want his MOAP to perform), and to enter knowledge into the MOAP for activities that he wants the MOAP to perform. For activities that a MOAP cannot perform, it can provide assistance to its user such as that mentioned at the end of Section 3.7.

In the discussion of modelling organizational activities in this paper, we generally do not distinguish a user from the knowledge in his MOAP. In most contexts, we use the term "the knowledge in a MOAP" instead of "the user", and assume that if the MOAP does not have the knowledge, then its user will perform the activity.

4 SUPPORTING KNOWLEDGE COMMUNICATION

Conceptually speaking, there is not much difference between communicating knowledge and communicating data. Knowledge can also be considered as data. For example, knowledge in expert systems is treated as data. However, in order to communicate knowledge, it must be represented in a way that can facilitate its communication. The MOAP architecture described in the previous section is designed specifically for this purpose. In this section we will show how the architecture supports the types of knowledge communication discussed in Section 2. Our discussion will refer to Figure 2. Operational issues such as security are similar to those in data communication, and are beyond the scope of this paper.

4.1 Within an Agent

Recall from Section 2 that an agent may keep many types and forms of knowledge. The two major types of such knowledge are procedural and problem-solving. Within each of these types, knowledge may itself be kept in more than one form. While knowledge may be transformed from one form into another, the most important form of knowledge communication that we see arising is that between problem-solving and procedural knowledge.

When a new activity arises, a user might not have enough knowledge to handle it. Under this situation, he must treat the activity as a problem-solving one, and experiment with possibly several alternatives. In this case, the MOAPs and their problem-solving knowledge entity can assist him

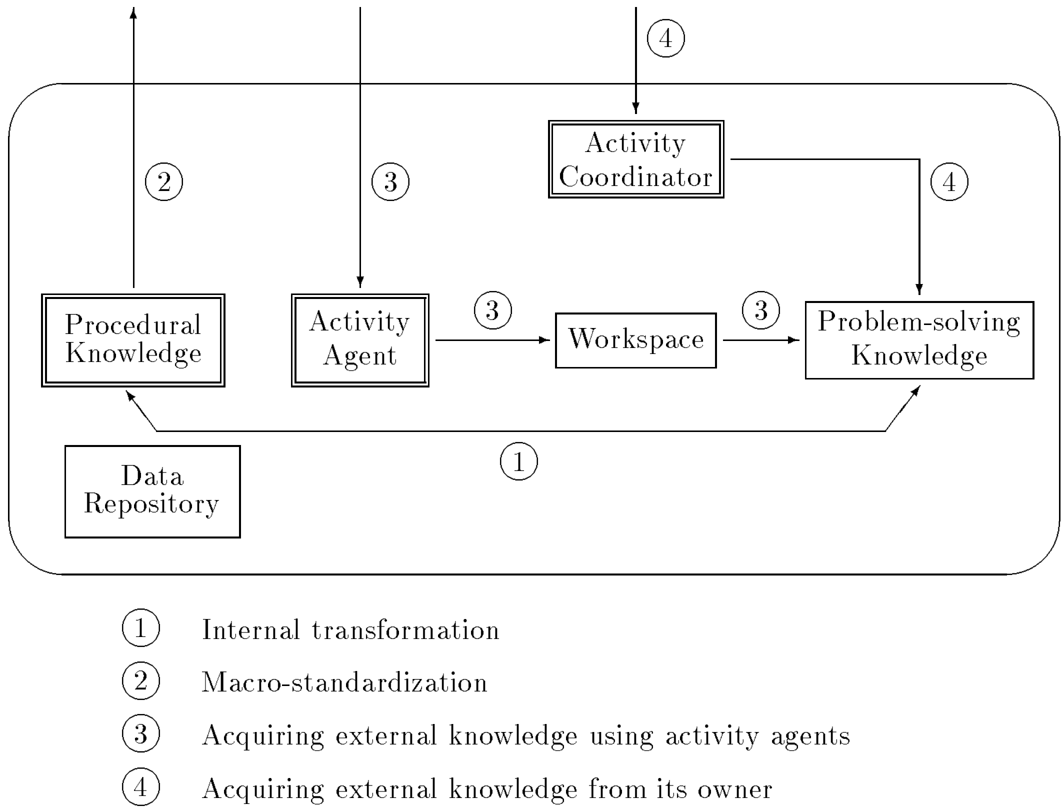


Figure 2: Different types of knowledge communication.

to accomplish the activity. However, users (and MOAPs) learn from experience. As they know more about a particular activity, the activity can become very routine and structured. In this case, the activity should be handled by the procedural knowledge entity rather than the problem-solving knowledge entity in order to gain better performance. We refer to this type of knowledge communication as *internal transformation*.

There are two major types of internal transformation. First, knowledge related to the missing parts of the procedural knowledge entity (see the second category of problem-solving knowledge in Section 3.4) can be transformed. Transformation, in this case, means to place the missing part into the procedural knowledge entity and remove the knowledge from the problem-solving knowledge entity. For example, suppose that a source of information for a task has now stabilized and can be found in an organizational data base. Then, a former call to the problem-solving knowledge entity to acquire this information can now be replaced by a data base query embedded directly in the task itself.

Second, knowledge playing the role of procedural knowledge (see the third category of problem-solving knowledge in Section 3.4) can be transformed. Transformation, in this case, also means to remove the knowledge from the problem-solving knowledge entity and place it in the procedural knowledge entity. For example, suppose that a plan generated by the problem-solving knowledge entity for a particular activity is found to have stabilized (i.e., the same plan is generated each time the activity is carried out). Then this plan can be replaced by a task which always executes the plan; there is no longer a need to reform the plan each time to accomplish the activity.

In both cases, the user has to initiate the actual knowledge communication (i.e., to instruct the activity coordinator to transform a particular piece of knowledge). A MOAP, on the other hand, can be implemented so that it will assist the user in recognizing the need for such knowledge communication. For example, the activity coordinator can save the successful schedule for each planning result and, if there is an indication of a repeated success for a particular plan, the user can be notified.

Conversely, it is possible that a well-understood activity enters an unexpected situation where there is no *a priori* knowledge to handle it. In this case, the activity should be handled by the problem-solving knowledge entity. The same two major types of internal transformation also apply here with the exception that transformation is from the procedural knowledge entity to the problem-solving knowledge entity.

4.2 Among Agents Within the Same Organization

The MOAP architecture, by partitioning knowledge among different MOAPs, is designed specifically to allow knowledge to be decentralized. We want to keep one agent's knowledge separate from another agent's knowledge. In this way, the MOAP architecture can support conflicting knowledge [18]. Different MOAPs can be related via task hierarchies as discussed in Section 3.3. This allows MOAPs to cooperate when executing tasks, but does not allow them to share knowledge. Thus, explicit knowledge communication mechanisms to support knowledge sharing among the agents are required.

To support knowledge decentralization and sharing, we make use of a *knowledge inheritance hierarchy*. First, we categorize the knowledge contained in agents into two major collections: *user* and *organizational unit* [25]. Then, we relate these collections of knowledge in an inheritance hierarchy according to the structure of the organization. Figure 3 provides a simple example of how user and organizational-unit MOAPs can be used to decentralize and share knowledge. In the figure, the rectangular boxes are user MOAPs and are associated with the user whose name is inside the box. The ovals are organizational-unit MOAPs which contain knowledge that can be used by user MOAPs associated with them.

We categorize knowledge into user and organizational unit to support decentralized, private and shared knowledge. User knowledge is only accessible to the user who owns it. This type of knowledge has been personalized by, and in a sense optimized for, an individual user. Inconsistent user knowledge, used by different users, represents different opinions on common activities [42]. On the other hand, organizational-unit knowledge often is used by two or more users. This type of knowledge has evolved through the experience of users to achieve standardization or to deal with the environment in which the organization operates (e.g., federal reporting laws).

In our representation of knowledge, a MOAP (i.e., an agent) is associated with each user and organizational unit in an organization. A user MOAP is associated with only one user and encodes a user's individual knowledge. For example, in Figure 3 "Jill," "John," "Joe," etc. are user MOAPs. Organizational-unit MOAPs are not associated with a specific user[9], but encode organizational-unit knowledge that is shared among all the users associated with the organizational unit. Thus, in Figure 3, "Sales," "Advertising," "Marketing," and "Organization" are organizational-unit MOAPs.

We place knowledge in an inheritance hierarchy to facilitate knowledge sharing. In Figure 3, the directed arcs indicate the MOAP(s) from which a given MOAP inherits knowledge. Inheritance is in the direction opposite that of the directed arc. For example, the organization's common marketing knowledge is associated with the organizational-unit MOAP "Marketing" and is inherited by the organizational-unit MOAPs "Sales" and "Advertising." In turn, the organization's common sales

[9] However, there may be a specific user that has responsibility for the organizational-unit MOAP.

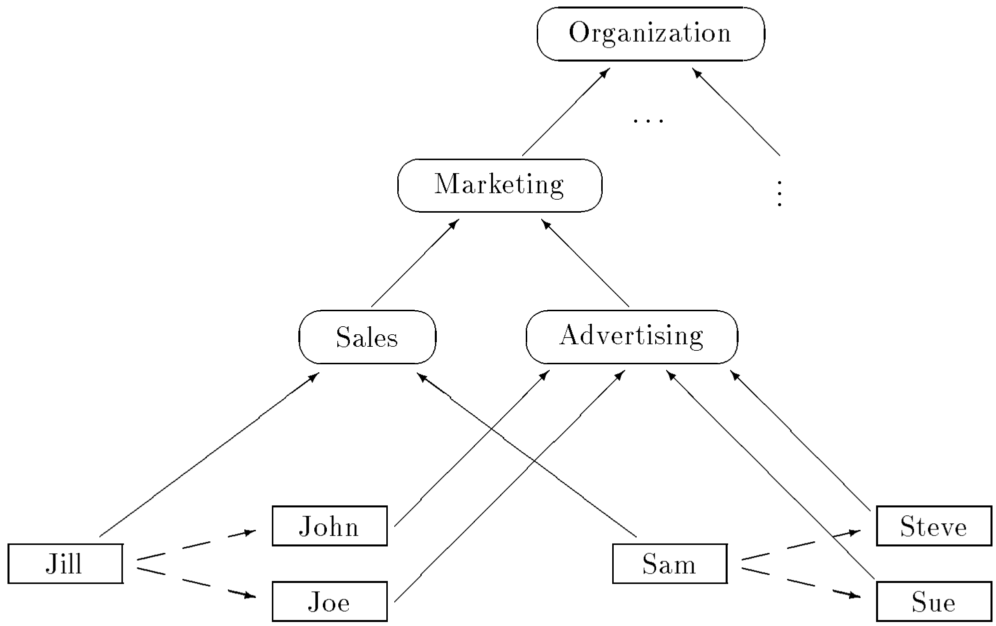


Figure 3: An example knowledge inheritance hierarchy of user and organizational-unit MOAPs.

knowledge is associated with the organizational unit MOAP "Sales" and is inherited by both "Jill" and "Sam" user MOAPs. In general, a MOAP lower in the hierarchy inherits knowledge from all MOAPS to which it is related higher in the hierarchy[10]. Thus, implicitly all user MOAPs and organizational-unit MOAPs inherit the knowledge of the "Organization" organizational-unit MOAP (the root of the hierarchy) which contains knowledge common to all organizational units and users in the organization.

The dotted arrows in Figure 3 are the same as the dotted arrow in Figure 1. That is, for example, Jill's MOAP can request John's MOAP to perform some activity for her. John's MOAP, in this case, will deal with Jill's MOAP's request using its own knowledge and in the MOAPs of the organizational units to which it is related in the knowledge inheritance hierarchy. It is not required to pass its knowledge of how it accomplishes its activity to Jill's MOAP or to other user MOAPs.

All organizational activities are initiated in user MOAPs. Organizational-unit MOAPs are used mainly to structure, partition, and store knowledge. In this way, responsibility for organizational activities resides ultimately with some user MOAP (i.e., some user). Thus, all the MOAPs discussed so far have been user MOAPs. In particular, the MOAPs used to compose the hierarchy of tasks mentioned in Section 3.3 are user MOAPs.

With respect to the entities available in the MOAP architecture, different MOAPs may choose to emphasize different capabilities of a MOAP. For example, user MOAPs would probably make use of all of the entities described in Figure 1. On the other hand, organizational-unit MOAPs may be used primarily to store procedural knowledge and thus make more use of the procedural knowledge and data repository entities. Some may specialize in problem-solving knowledge and thus make more use of the problem-solving knowledge and activity agent entities. Regardless, organizational unit MOAPs will store knowledge that is shared while user MOAPs will store knowledge that is private.

[10]Note that inheritance is not necessarily strictly hierarchical as multiple inheritance is allowed.

Since user and organizational-unit MOAPs may be used for different purposes, it might be better to optimize the workings of the organizational-unit MOAPs with a different (i.e., simpler) architecture. However, in order to support the types of knowledge communication discussed in this paper, the architecture used must facilitate the migration of knowledge from one type of MOAP to the other. For this reason, the same architecture is used for both user and organizational-unit MOAPs. This uniformity of representation also simplifies the knowledge communication mechanisms.

Conceptually, a user MOAP inherits all the procedural knowledge in the organizational-unit MOAPs with which it is associated. However, the owner of the user MOAP can modify any of the components of the procedural knowledge except the pre- and post-conditions. In this way, the user MOAP has the freedom to perform its work in its own way (or what it believes to be a better way) provided that the constraints set by the organizational-unit MOAPs are satisfied. We allow this flexibility because we want to support different knowledge about the same activity as mentioned in Section 2.3.

In general, the knowledge between any two MOAPs can be inconsistent. However, knowledge among an organizational-unit MOAP and its associated organizational-unit/user MOAPs is *always* consistent. This is because of the knowledge inheritance hierarchy. Thus, every MOAP is always *internally consistent*. However, MOAPs may be *externally inconsistent*. That is, if two MOAPs are not related in the knowledge hierarchy, then they can contain inconsistent knowledge.

If two MOAPs do not need to interact or work together to accomplish an activity, then we do not have to worry about the inconsistency between their knowledge. Even in the case where two inconsistent MOAPs are required to work with each other, but the inconsistency does not create any problem, then nothing needs to be done either. For example, given some fixed resources, each MOAP might have a different method for calculating what it should get and what other MOAPs should get. The method might result in a different allocation of the resources for different MOAPs. However, if every MOAP uses its own method to get the resources it wants, and there are enough resources to satisfy every MOAP, then it may not be worthwhile to resolve the inconsistency.

Otherwise, inconsistencies must be resolved. This can be detected when two MOAPs, interacting together, violate some global constraints (e.g., total money available to them). Using the same example given above, if the overall requests by the MOAPs add up to be more than what is available, then the inconsistent knowledge for allocating the resources must be resolved. Resolving inconsistencies in general is a challenging research area. For example, Hewitt [18] proposes to resolve inconsistencies via debate and negotiation. While there are no good techniques currently available to automate this process, a MOAP can provide assistance to its user by narrowing down where the inconsistencies occur [8].

Technically speaking, there is nothing special (or difficult) about allowing inconsistent knowledge among different MOAPs. It just means that we do not need to check for consistency which, in turn, means less work. On the other hand, allowing inconsistent knowledge means that different MOAPs can have different knowledge for dealing with the same situation. A contribution of the MOAP architecture is therefore to allow some types of inconsistent knowledge to exist in different MOAPs and to provide mechanisms, such as the activity agent (see below), for combining knowledge when there is a need to do so.

4.2.1 Macro-standardization

The knowledge inheritance hierarchy discussed in the previous section facilitates knowledge communication from organizational-unit MOAPs to user MOAPs (i.e., top-down). However, it does not deal with knowledge communication from user MOAPs to organizational-unit MOAPs (i.e., bottom-up). For instance, knowledge that is known by one user MOAP may be useful to other user MOAPs. At some point in time, this knowledge may be useful to a large enough group of user MOAPs that it should be made known to them on a group- or an organization-wide basis. Therefore, we may need to communicate knowledge from a user MOAP to a related organizational-unit MOAP, or from one organizational-unit MOAP to another related organizational-unit MOAP. We refer to this type of knowledge communication as *macro-standardization*. For reasons mentioned below, this type of knowledge communication is restricted to procedural knowledge only.

The need for this kind of knowledge communication can happen in two ways:

1. some higher authority decides to eliminate the inconsistent knowledge used by different MOAPs in accomplishing an activity, 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 users.

Similar to internal transformation, the actual knowledge communication can be done automatically (see [39] for a description of the method), but the determination of which procedural knowledge to communicate requires the participation of one or more users.

In our approach, we choose not to communicate problem-solving knowledge in this way for the following reason. Procedural knowledge is that knowledge that has been tested by user agents 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 communicated in this way. Note that in certain cases, problem-solving knowledge can still be communicated as discussed in the following section.

4.2.2 Acquiring external knowledge using activity agents

The macro-standardization described in the previous section can only communicate knowledge if the MOAPs are *directly related* in the knowledge hierarchy. Unfortunately, this is not always the case. Sometimes, the knowledge needed by a MOAP resides in an unrelated part of the organization [37]. In this case, we can have a user MOAP's activity agent collect relevant knowledge from other MOAPs (user or organizational-unit). Knowledge that appears to be relevant or useful will be brought back and placed in the MOAP's workspace in the *temporary knowledge area*. The MOAP, together with its user (i.e., not the activity agent), will decide which pieces of knowledge are relevant or useful. The communication path in this case is therefore (see Figure 2):

$$\text{Outside MOAP} \longrightarrow \text{Activity Agent} \longrightarrow \text{Workspace} \longrightarrow \text{Problem-solving knowledge}$$

Note that if the knowledge turns out not to be useful, the communication will stop at the workspace (i.e., it will be deleted from the workspace). On the other hand, if the knowledge is useful, then the MOAP will place the knowledge in its problem-solving knowledge entity.

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 as a structured and routine task in the sending MOAP.

A protocol will facilitate the inter-agent knowledge communication. We believe that this protocol can be constructed by applying research results in speech act theory [1, 34, 35]. However, the development of a general purpose protocol is a complex task. We have taken the approach of developing different protocols for different purposes initially. For example, we have developed a protocol for collecting and disseminating data [43], and another one for negotiation [8]. We hope to gain experience with the usage and validity of these protocols, and eventually integrate them into a general purpose protocol.

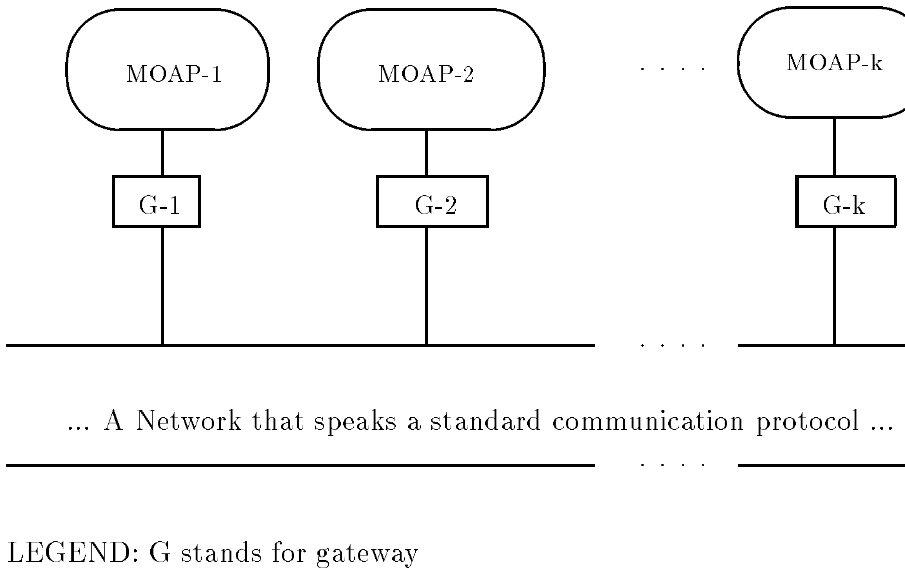


Figure 4: A MOAP network.

4.3 Among Agents in Different Organizations

The difference between intra-organizational and inter-organizational communication is similar to that between local area networks and wide area networks. For example, we expect different organizations to have different protocols for intra-organizational communication. We believe that some solutions used to facilitate communication between different data communication networks can be useful for inter-organizational communication [43] (e.g., the concept of a standard "internet protocol (IP)" and "gateways"). Therefore, a MOAP-based IIS would have a standard inter-organizational communication protocol. Individual organizations are allowed to have their own communication protocol. Gateways are used to perform translations between the local protocol and the standard protocol. Conceptually, the relation between MOAPs, gateways, and the standard communication protocol is that shown in Figure 4.

As an example, consider a MOAP, α, in an organization which wants to communicate with a MOAP, β, in a different organization. α constructs and asks its gateway to send out an activity agent. Construction means to program the application dependent part of the activity agent so that it knows what to say when it meets with the other parties. The activity agent will migrate to β's machine and communicate with β's gateway. When the activity agent communicates with β's gateway, their dialogue will be guided by the standard communication protocol and the application dependent part of the activity agent. The gateway, as part of its task, will act as a translator between the activity agent and β. When the conversation ends, the activity agent will bring the result back to α.

It is a challenging problem for the gateway to perform translation between the activity agent and the MOAP it is representing. However, there is a technique called *universal attachment* [30] which uses artificial intelligence techniques to resolve the problems that arise in the interpretation of many different knowledge representing languages. We believe that this technique can be adapted for solving the inter-organizational knowledge translation problem.

4.4 Between Users and Agents

An agent can acquire external knowledge from its user. The communication path in this case is (see Figure 2):

$$\text{Owner} \longrightarrow \text{Activity Coordinator} \longrightarrow \text{problem-solving Knowledge}$$

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

User-level facilities are required for knowledge communication between users and agents. While these facilities fall outside the "architectural" scope of the MOAP architecture, they nevertheless will be crucial to the eventual success of IISs. It is clear that visual-based, direct manipulation facilities will be an extremely important mechanism for knowledge communication. Recent research on systems such as Object Lens [23] and Polymer [26] which use forms, icons, and hypertext-like facilities point the way to the types of interfaces that may be successful in this domain. OASIS, an object-oriented programming environment for implementing MOAPs, is attempting to provide programming facilities for implementing such user/agent knowledge communication facilities [29].

5 CONCLUDING REMARKS

In this paper, we proposed the MOAP architecture for modelling IISs and for supporting knowledge communication. The MOAP architecture supports the following types of knowledge communication:

1. within an agent,
2. among agents within the same organization,
3. among agents in different organizations,
4. between users and agents.

A properly designed MOAP-based IIS has the ability to introduce an IIS into an organization step by step. Initially all organizational workers can be supported by "shell" MOAPs (i.e., MOAPs that are seeded only with the knowledge they acquire via the knowledge inheritance hierarchy). However, users can slowly add their knowledge (i.e., problem-solving knowledge) into their corresponding MOAPs over time. At any point in time, a MOAP can automatically accomplish an activity if it has enough knowledge to do so. Otherwise, the owner of the MOAP will assist it in accomplishing the activity. Over time, the knowledge communication facilities provided in the MOAP architecture will change a MOAP-based IIS into a useful system for performing many activities for its owner.

To date, we have implemented five prototypes that demonstrate the feasibility of the MOAP architecture. The MOAP architecture itself has been prototyped without the knowledge communication facilities [40]. An efficient implementation of the physical migration of the activity agent has been prototyped separately [21]. In addition, a third prototype for the activity agent has been designed and implemented according to the standard open systems interconnection (OSI) reference model specified by the data communications community [43]. With this third prototype, the activity agent is capable of migrating to and communicating with systems running on different data communication networks. A system which supports the macro-standardization type of knowledge communication has also been prototyped [39]. Finally, OASIS has demonstrated the applicability of object-oriented techniques for the construction of MOAP-based IISs [29]. We are currently designing and implementing a prototype that will combine the functionality of these systems and will incorporate the other types of knowledge communication facilities discussed in this paper.

There are a number of unresolved issues in the MOAP architecture. One of the them is how to support competitive interactions in an open systems environment. This kind of support can be provided at two different levels. At one level, primitive operations for facilitating competitive negotiation can be provided. At another level, tools to actually automate this activity can be provided. A model of speech acts [1], called the struggle model, specializes in competitive and cooperative interactions. Although this model does not suggest methods to automate cooperative and competitive interactions, it does provide the necessary primitives needed in terms of speech act verbs. We plan to investigate the use of these speech act verbs as primitives for cooperative and competitive interaction.

Another unresolved issue is recognizing that there is a need to communicate knowledge. For example, how would a MOAP know that problem-solving knowledge is stable enough for it to be transformed to procedural knowledge? Although automating this kind of knowledge communication is not critical and may even be dangerous for supporting organizational activities, there is a need for a MOAP to inform its owner when the potential for knowledge communication exists. The user would then decide what to do next. We plan to investigate the application of case-based reasoning [33, 38] for this kind of support.

ACKNOWLEDGEMENTS

This research was supported in part by the Natural Sciences and Engineering Research Council of Canada, the Research Grants Council of Hong Kong, and 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 Anderson Consulting, Bell-Northern Research Laboratory, and British Columbia Telephony Company.

References

[1] Ballmer, Th., and Brennenstuhl, W., *Speech Act Classification.* Springer-Verlag, Berlin, 1981.

[2] Barber, G.R., *Office Semantics.* Ph.D. Thesis, Dept. of Electrical Engineering and Computer Sc., Massachusetts Institute of Technology, Cambridge, Massachusetts, 1982.

[3] Barber, G.R., "Supporting Organizational Problem Solving with a Workstation," *ACM Trans. Office Inf. Sys.* **1**(1), 1983, 45–67.

[4] Bhandaru, N., and Croft, W.B., "An Architecture for Supporting Goal-Based Cooperative Work," *Multi-User Interfaces and Applications*, S. Gibbs and A.A. Verrijn-Stuart (Editors), Elsevier Science Publishers B.V. (North-Holland), 337–354, 1990.

[5] Card, S.K., Moran, T.P., and Newell, A., *The Psychology of Human-Computer Interaction.* Lawrence Erlbaum Associates, Hillsdale, NJ, 1983.

[6] Card, S.K., Moran, T.P., and Newell, A., "The Keystroke-Level Model for User Performance Time with Interactive Systems," *Comm. ACM* **23**(7), 1980, 396-410.

[7] Casais, E., "An Object Oriented System Implementing KNOs," *Proc. Conf. on Office Inf. Sys.*, 1988, 284–290.

[8] Chang, M.K., and Woo, C.C., "SANP: A Communication Level Protocol for Negotiations," *Proc. Third European Workshop on Modelling Autonomous Agents in a Multi Agent World*, 1991.

[9] Croft, W.B., and Lefkowitz, L.S., "Task Support in an Office System," *ACM Trans. Office Inf. Sys.* **2**(3), 1984, 197–212.

[10] Croft, W.B., and Lefkowitz, L.S., "Using a Planner to Support Office Work," *Proc. Conf. Office Inf. Sys.*, 1988, 55–62.

[11] de Jong, P., "The Ubik Configurator," *Proc. Conf. Office Inf. Sys.*, 1988, 309–315.

[12] de Jong, P., "Structure and Action in Distributed Organizations," *Proc. Conf. Office Inf. Sys.*, 1990, 1–10.

[13] de Jong, P., "A Framework for the Development of Distributed Organizations," *Proc. Second Conf. on Organizational Computing, Coordination, and Collaboration*, Austin, Texas, February 24-26, 1991.

[14] 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 Comput. Surv.* **12**(2), 1980, 213–253.

[15] Fikes, R.E., and Henderson, D.A., "On Supporting the use of Procedures in Office Work," *Proc. AAAI Conf.*, 1980, 202–207.

[16] Gerson, E.M., and Star, S.B., "Analyzing Due Process in the Workplace," *ACM Trans. Office Inf. Sys.* **4**(3), 1986, 257–270.

[17] Gorry, G.A., and Scott Morton, M.S., "A Framework for Management Information Systems," *Sloan Management Review* **13**(1), 1971, 55–70.

[18] Hewitt, C., "Offices are Open Systems," *ACM Trans. Office Inf. Sys.* **4**(3), 1986, 271–287.

[19] Ho, C., Hong, Y, and Kuo, T., "A Society Model for Office Information Systems," *ACM Trans. Office Inf. Sys.* **4**(2), 1986, 104–131.

[20] Kaye, A.R., and Karam, G.M., "Cooperating Knowledge-Based Assistants for the Office," *ACM Trans. Office Inf. Sys.* **5**(4), 1987, 297–326.

[21] Keong, G., and Woo, C.C., *An Implementation of an Intelligent Communication Tool for Supporting Organizational Work.* Tech. rep. 88-MIS-030, Faculty of Commerce and Business Administration, Univ. of British Columbia, Vancouver, Canada, Nov. 1988.

[22] Kornfeld, W.A., and Hewitt, C.E., "The Scientific Community Metaphor," *IEEE Trans. Systems, Man, and Cybernetics* **SMC-11**(1), 1981, 24–33.

[23] Lai, K.Y., and Malone, T.W., "Object Lens: A 'Spreadsheet' for Cooperative Work," *Proc. Conf. Computer-Supported Cooperative Work*, 1988, 115–124.

[24] Lai, K.Y., Malone, T.W., and Yu, K.-C., "Object Lens: A 'Spreadsheet' for Cooperative Work," *ACM Trans. Office Inf. Sys.* **6**(4), 1988, 332–353.

[25] Lochovsky, F.H., Woo, C.C., and Williams, L.J., "A Micro-Organizational Model for Supporting Knowledge Migration," *Proc. Conf. Office Inf. Sys.*, 1990, 194–204.

[26] Mahling, D.E., and Croft, W.B., "An Interface for the Acquisition and Display of Office Procedures," *Proc. Conf. Office Inf. Sys.*, 1990, 136–141.

[27] Malone, T.W., Grant, K.R., Lai, K.Y., Rao, R., and Rosenblitt, D., "Semistructured Messages are Surprisingly Useful for Computer-Supported Coordination," *ACM Trans. Office Inf. Sys.* **5**(2), 1987, 115–131.

[28] Malone, T.W., Grant, K.R., Turbak, F.A., Brobst, S.A., and Cohen, M.D., "Intelligent Information Sharing Systems," *Comm. ACM* **30**(5), 1987, 390–402.

[29] Martens, C., and Lochovsky, F.H., "OASIS: A Programming Environment for Implementing Distributed Organizational Support Systems," *Proc. Conf. Organizational Computing Systems*, 1991, 29-42.

[30] Myers, K., "Automatically Generating Universal Attachments Through Compilation," *Proc. AAAI Conf.* , 1990, 252–257.

[31] Nirenburg, S., and Lesser, V., "Providing Intelligent Assistance in Distributed Office Environments," *Proc. ACM SIGOIS Conf. Office Inf. Sys.*, 1986, 104–112.

[32] Panko, R.R., and Sprague, R.H., "Toward a New Framework for Office Support," *Proc. ACM SIGOA Conf. Office Inf. Sys.*, 1982, 82–92.

[33] Riesbeck, C.K., and Schank, R.C., *Inside Case-Based Reasoning.* Lawrence Erlbaum Associates, Hillsdale, NJ, 1989.

[34] Searle, J.R., *Speech Acts: An Essay in the Philosophy of Language.* Cambridge University Press, London, 1969.

[35] Searle, J.R., *Expression and Meaning.* Cambridge University Press, London, 1969.

[36] Searle, J.R., and Vanderveken, D., *Foundations of Illocutionary Logic.* Cambridge University Press, London, 1985.

[37] Tsichritzis, D., Fiume, E., Gibbs, S., and Nierstrasz, O., "KNOs: *KN*owledge Acquisition, Dissemination, and Manipulation *O*bjects," *ACM Trans. Office Inf. Sys.* **5**(1), 1987, 96–112.

[38] Webber, B.L., Clarke, J.R., Rymon, R., Niv, M., and Ibanez, M.M., *TraumAID: AI Support for the Management of Multiple Trauma.* Tech. rep. MIS-CIS-90-09, Dept. of Computer Sc., Univ. of Pennsylvania, Philadelphia, PA, 1990.

[39] Williams, L., and Lochovsky, F.H., "Supporting Knowledge Migration in Organizations," *Proc. IFIP Congress '89.*, 1989, 259–264.

[40] Woo, C.C., *An Object-Oriented Model for Supporting Office Work.* Ph.D. thesis, Dept. of Computer Sc., Univ. of Toronto, Canada, April 1988.

[41] Woo, C.C., "SACT: A Tool for Automating Semi-Structured Organizational Communication," *Proc. Conf. Office Inf. Sys.*, 1990, 89–98.

[42] Woo, C.C., and Lochovsky, F.H., "Supporting Distributed Office Problem Solving in Organizations," *ACM Trans. Office Inf. Sys.* **4**(3), 1986, 185–204.

[43] Zeng, T., *An Organizational Communication Protocol Based on Speech Acts: Design, Verification and Formal Specifications.* M.Sc. thesis, Dept. of Computer Sc., Univ. of British Columbia, Canada, July 1990.