Agent Oriented Programming for Group Performance Support Systems

by

Barry G. Silverman, Christo Andonyadis, Yair Rajwan, Alfredo Morales, and Tom Ericson
Institute for Artificial Intelligence
School of Engineering and Applied Science
George Washington University
Washington DC 20052

March 1996

Abstract

This paper reviews efforts to design an environment for authoring group performance support system (GPSS) agents that can interact with remote internet resources, applications, and users. We review the architecture of the GPSS environment and show how its tutor/critic elements do and don't map into an idealized specification extracted from the agent oriented programming and inter-agent communication literature. The result provides insights into internet information system architectures for life long training and decision aiding, as well as extensions needed in the agent paradigm to accomodate GPSS's critic/tutor agent needs. An actual health care case study is provided with a video simulating use of the agents.

1) Introduction

The advent of the Internet means there are potentially unlimited knowledge and data resources at one's disposal. For example, there are online web-based references, digital gopher libraries, remote applications, news groups, ftp sites, data repositories, and so on. To facilitate access to this heterogeneous set of server resources one accesses viewers (e.g., Mosaic, Netscape, and Gopher Menus) and search engines These viewers and engines are platform independent, yet largely reactive. That is, they largely promote user-directed search, browse, and read operations.

An alternative, more proactive mode is emerging with the advent of forms, scripts, languages, and plug-ins that may all run inside the platform-independent viewers/browsers. This means that browsers can spawn applet and agent processes which can act on behalf of the user to autonomously and proactively seek information (or perform transactions) important to the user. Further supporting this shift is the appearance of number of distributed Object Resource Brokers (e.g., Microsoft's OLE2, the OMG's CORBA, and IBM's OpenDoc) that provide a higher level form of remote procedure calling and distributed computing. Client processes can more readily be connected to remote server applications without the need for user browsing.

The Internet is thus poised for a paradigm shift from user-directed browsing to proactive, anticipatory agent-supported work. This paradigm shift will exploit proactive approaches already existing in non-internet applications, provided those approaches can be "agentified" to work on the internet. The next section introduces a class of proactive approaches called Group Performance Support Systems (GPSSs), and explains some of the difficulties of making them Internet-literate. The rest of this paper presents the agent-oriented programming (AOP) approach and explains a modeling activity that is the first step to understanding how to agentify the GPSS approach. Through careful model construction we have been able to identify both what needs to be added to the GPSS approach, and how the agent oriented programming (AOP) paradigm needs to be extended to accomodate GPSS. Unresolved research issues are included in the conclusions.

1.1) Group Performance Support Systems

One field that has long been concerned with better supporting users in task settings is that of the "group performance support systems" (GPSS) field. This field is the merger over the past several years of computer based training and decision support systems. The reason these two fields merged was the belief that people no longer go through one period of training and then become decision makers. They are engaged in lifelong training on the job (situated) even as they are called on to solve problems and make decisions. For example, MDs must obtain 50 credits per year of continuing education to retain their licenses. Unfortunately, due to the pace of new knowledge in medical specialties, if an MD reads two new articles a night, by the end of the year she will be further behind the state of the art in her specialty. This implies that knowledge assets need to be proactive and context triggered. As a task is being performed, one would like situated tutor/critic agents to notice where training and/or decision support is needed, and to bring the relevant resources to the user. This will save users article search time, plus it will bring material to their attention in context and when their mind is wondering what to do about that topic. Hopefully, the latter will save time of having to backtrack and correct (uninformed) decisions made earlier.

The specific capabilities that a GPSS would provide for a given user can be thought of in terms of training or **informing**, decision **feedback**, and **autonomous action**. Informing procedures are characterized by their presentation of information to the user. Inform procedures include, among others: cueing; tutor via scenario/example/principle; explaining; examining via induction, quizzing, elicitation; and viewpoint argumentation. Autonomous actions are symbiotic with the inform and feedback functions. Thus, search/synthesis preceeds the informing activity, while cueing and explaining let the user know the search/synthesis results. Similarly, learn user preferences is an important precursor to tutoring/arguing with a given user. Feedback procedures primarily involve responding to actions taken by the user. Examples of feedback procedures include: **verify,** alert, remind, critique, and suggest. To provide such feedback, the GPSS must engage in a set of other autonomous action such as: differential analysis; consistency checking; clarity checking; and failure mode checking. Finally, after a user has been informed and given feedback, they may delegate control to the GPSS to implement corrective actions. Examples include: repair; add, replace. The collective set of GPSS procedures thus includes:

- **INFORM (cue, explain, tutor, examine, argue)**
- **FEEDBACK (verify, alert, remind, critique, suggest)**
- **AUTONOMOUS ACTION (search/synthesize, learn preferences, differential analysis, consistency check, replace, add, repair)**

To date, however, much of the situtated GPSS or critic/tutor software exists as either isolated applications or embedded in a single application. It can't go to the Internet and bring in the latest material. Nor can it share its knowledge base and/or problems with other, outside agents. For example, spelling and grammar checkers are replicated and enslaved within each word processor, adverse drug reaction critics and drug education pages can only be called within a given drug reference guide (they can't autonomously read the medical record), and most tutoring systems can only access their internal knowledge base. These critic/tutor agents are "Internet-challenged". Because of this, they under-support their users, waste computing resources, fail to share code or knowledge that could lead to improved training/critiquing, and are difficult to keep current.

To summarize, as the Internet's resources begin to open up to proactive "plug-ins", situated GPSS or critic/tutor technology must transition to open, flexible, adaptive agent architectures that can exploit this opportunity. To rectify this situation, we have begun research into the "agentification" of a critic/tutor authoring environment called COPE: e.g., see Silverman (1992). This paper describes our ongoing efforts and reports initial results, issues and obstacles, and lessons learned thus far.

1.2)Case Study: The Health Object Library Online (HOLON)

To test our model in a real context, we are simultaneously participating in a consortium (including Harvard Medical School, Norwalk Hospital, Time Warner Cable, Oracle, and others) that is building a system in the healthcare domain called HOLON. Historically, the healthcare field has been plagued by the problems of nonintegrated applications that force users to access them through different platforms, operating systems, networks, and interfaces. For example, patient information that is generated and saved in one application must be re-entered for other applications to use it.

HOLON is a standards-based, object-oriented, open-software environment that is developing a suite of middleware utilities that will foster the integration of GPSSs in the healthcare field. The work described in this paper attempts to contribute at the agent layer of the HOLON architecture. As a result, this paper will use healthcare examples to illustrate many of the points being made. There is also a 12 minute video that accompanies this paper and that shows how a (two actors in the roles of) healthcare provider and consumer interact with each other and with (a simulated version of) HOLON's objects and agents.

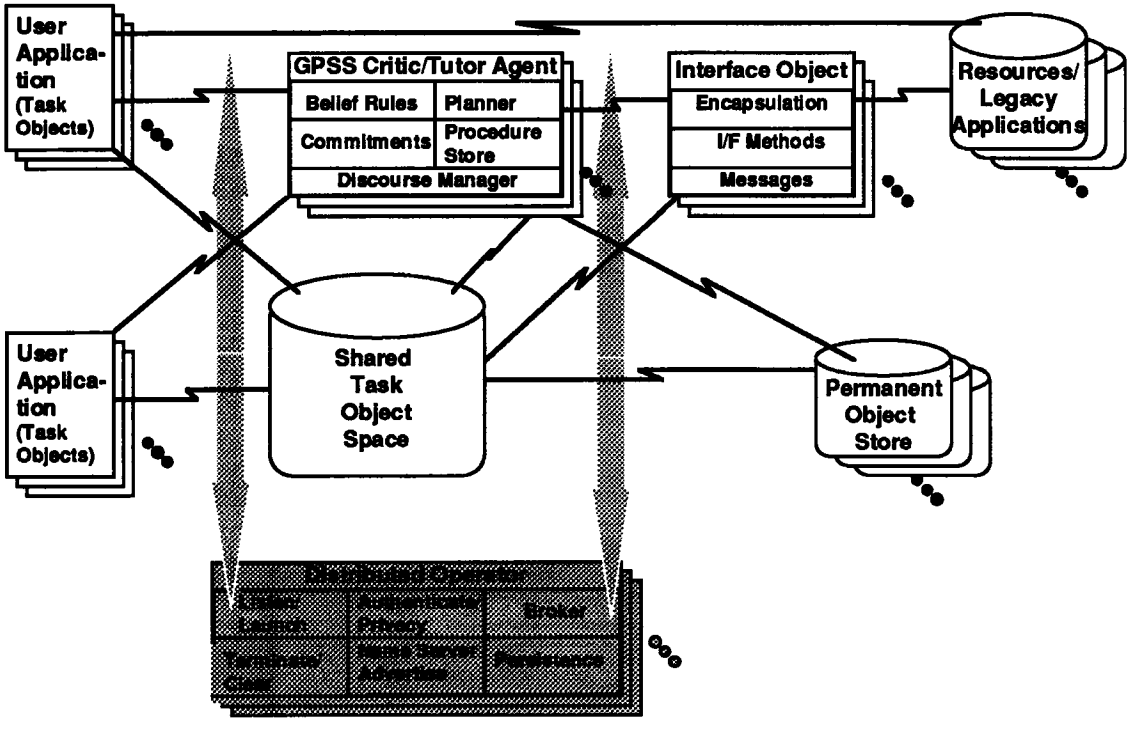
2) Idealized Architecture

Our present efforts involve modeling an idealized architecture that shows the structural, behavioral, and process dynamics views needed to provide lifelong training/performance support in an Internet environment. This modeling activity is the first step to understanding how to agentify the GPSS approach. Through careful model construction we have been able to identify both what needs to be added to the GPSS approach, and how the agent oriented programming (AOP) paradigm needs to be extended to accomodate GPSS. This modeling activity has also helped us to understand what topics in the AOP field need further research, a subject that will be discussed in the conclusions.

Figure 1 introduces the top level view of the agent-oriented model of GPSS. Users (left side) may work collaboratively over a shared object task space, via their applications, browsers, viewers, etc.. The shared space (center box) will support private and public task items, and it may have behavioral elements that handle task artifact behavior (eg., annotations, version control, permanent object store, etc.). Portions of the task space may be resident on user platforms, while other portions exist elsewhere. An example might be a Health Maintainance Organization which has distributed doctors and patients that asynchronously collaborate through shared (multimedia, virtual) patient folders about chronic conditions, status, complaints, appointments, health education materials, and so on.

To make the the object space more proactive, a number of inform/feedback agents are added that can interact with items on the task object spaces (public, private, shared, local, permanent) as well as with remote applications and resources. The idealized agent architecture adpoted here is based on the work of Shohan (1993), Finin et a. (1993), Davies & Edwards (1995), Barbucceanu & Fox (1995), and Silverman & Owens (1996), among others. In the tradition of these authors, the idealized agent may be conceptualized as a set of belief rules where the antecedent matches observed events and messages (patterns indicating performance support needs) and the consequent specifies commitments (for training/problem support) the agent intends to honor. Commitments are made after checking for suitable plans and procedures that will carry them out. Once made, commitments are communicated to other agents (or the users) via speech acts, potentially according to the KQML specification. Likewise, the procedure store may invoke speech acts with remote resources/applications in order to carry out the commitments. Each agent thus has a set of conversation policy objects it instantiates and tracks during the course of fullfilling a commitment.

Figure 1 - Idealized Architecture for Agentification of a Group Performance Support System



This "definition" of an agent is idealized and it could be implemented by any of a number of approaches ranging from expert systems that actually use belief rules to procedural programs that seem to exhibit the behavior described here. The precise implementation is unimportant. What is important to our definition of an agent is that it semi-autonomously exhibits "mental states," engages in "communication acts," and is capable of limited degrees of planning, adapting, and problem solving in achieving its goals.

In order for GPSS Critic/Tutor agents to tap remote resources with training, problem solving, and decision support value, "Interface Objects" may be needed. Some of these agents may be existing search agents and meta-crawlers for online libraries. More often this will be an object layer encapsulating specific legacy resources/applications to give them a variety of communicative features.

The idealized architecture also includes a "general operator" functionality (bottom of Figure 1) that listens at each client and server for agent requests, and that handles agent authentication, object space security, naming, resource allocation/protection, agent management (launch, run, terminate, garbage collect), state persistency, and shared cache and repository for the agents. This is not a single operator, but is a distributed entity consuming a small amount of resources on each client and server in the network. It sits atop other layers such as ODBCs, ORBs, transport protocols/APIs, and so on. This is depicted by the shadowed-in arrows on the diagram.

2.1) Idealized Agent Architecture

This section looks more closely at the internals of the agent architecture and how it applies to and is modified by GPSS needs. The agent architecture, as depicted in Figure 1, maps to AOP in the following

way: current Beliefs and Commitments are in the Working Memory; Commitment Rules are in the Belief Processor; and Private Actions are in the Procedure Store.

Working Memory Cache: The working memory cache gives the agent persistent knowledge of the environment in which it is operating. The components of this memory include: a user model which describes the current user in terms of attributes and preferences; a model of the work artifact which provides the agent with visibility of the work being performed; an intention stack which keeps the agent from forgetting commitments that have not yet been acheived; a set of beliefs which describe what the agent knows to be true; and an incoming message stack which serves as an inbox for the agent. The agent could commit to any number of different actions depending/based on the state of the contents of the working memory. The user model can be modified as a result of user actions, as well as by actions of agents. Actions taken locally by the user in agentified applications will update the agent's model of the artifact in working memory. On instantiation the agent object would be devoid of intentions, but thereafter, the intention stack would collect all ongoing commitments. The message stack would buffer incoming speech acts received from other agents engaged in conversation.

In HOLON the work artifact could be a medical record, or a prescription, or a lab result. The user could be a doctor, or a patient, or a lab technician. The action taken by the user could be prescibing a medication, or having a new medication prescribed, or processing samples sent to the lab. The agent could believe that a prescription is appropriate, or that a patient needs information, or that a lab result is pending. The agent could intend to establish that a prescription is appropriate, or to inform the patient about a medication's side-effects.

Belief Processor: The belief processor consists of a collection of commitment rules. These rules determine the conditions under which an agent would commit to action. Changes in the state of the agent's beliefs and/or intentions could trigger a rule to fire, as could messages received from other agents. Generally, a combination of the various different types of conditions would have to be satisfied for a rule to fire and a commitment to be made.

Commitment rules for GPSSs take the following general form:

IF: Belief
Trigger Events - Work Artifact Related
- Environmentally Precipitated (time triggers, sensory input)
Speech Act Events
Content-Related Omissions
Content-Related Actions - OnClick
- OnChange
- OnClose
Task-Specific Omissions
Application Patterns - learned from user actions
User Model.Settings
User Model.Preferances

THEN:commit to-
Do private actions
Do communicative actions
Inform/Tutor
Feedback/Critique
Direct/Autonomous Action

In the HOLON case, example commitment rules could look like this:

IF Trigger Event = a prescription is added to the medical record
and Belief = the medication is new (not yet taken by the patient)

and User Model = user (doctor) has authority
THEN commit to Verify prescription

IF Trigger Event = a prescription has been added to the medical record
and User Model.type = patient
and User Model.interface model = Influence
and Belief = the medication is new
THEN commit to Inform user about new medication

Procedure Store: The store contains the collection of procedures available to an agent. This store includes procedures that invlove both private actions and/or communicative actions. The range of communicative actions would include the on-line services currently avaiable (FTP, GOPHER, Netscape). The procedures generally fall into one of the following categories mentioned in Sect. 1.1: **Feedback, Inform** or **Autonomous Action.**

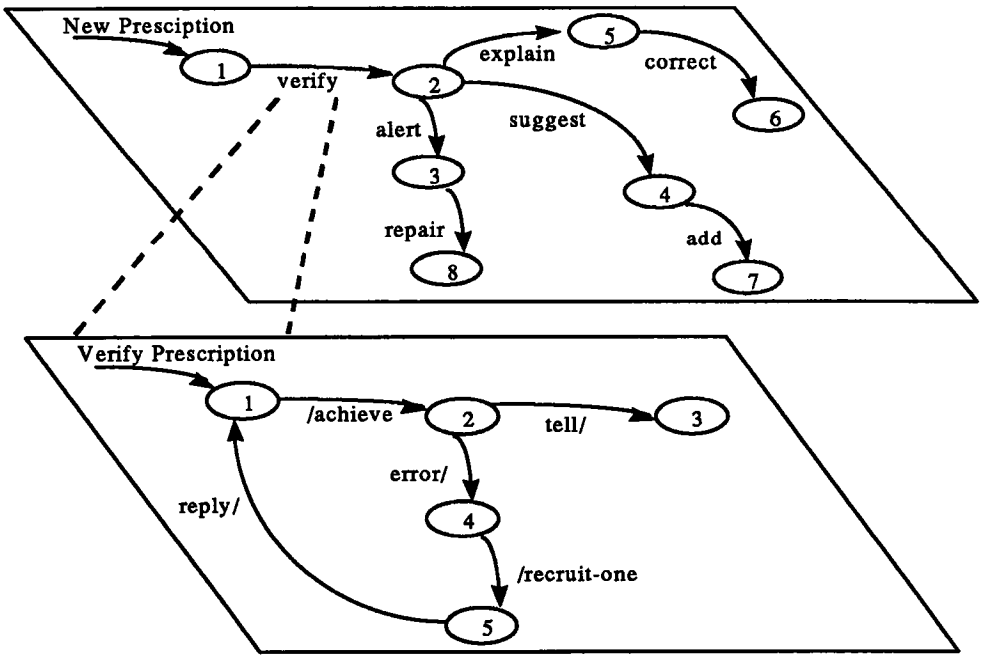
In HOLON, the procedure store could include, among many others:

- Verify Prescription - checks for contra-indications (allergies, drug interaction, side effects) by referencing the remote medical record and an online Physicians Desk Reference (PDR)
- Suggest Prescription - returns prescription based on medical record (diagnosis, conditions, medication list) and PDR
- Alert - alerts doctor of contra-inications found while Verifying
- Remind - reminds the patient about dosage schedule
- Tutor/Inform - Educate the patient about taking the new medication
- Explain - provides justification and background
- Examine - elicit conditions, allergies, and medical history from the patient

Planner: The planner operates at two distinct levels. Upon receipt of a commitment from the Belief processor, the planner must first determine which of a potentially large collection of decision networks to invoke. This determination is made in accordance with the agent's mental state and beliefs, as maintained in the working memory. The top layer of Figure 2 depicts one such network. The commitment in this case is to Verify a prescription. The user must be a doctor, or someone else with the authority to prescribe medications. The decision network for a patient user certainly would not include the correct, add or repair procedures. Patients do not have that authority, so neither would their agents (in decision networks). In traversing this network the planner deals with the second layer of Figure 2, the conversation policy. In this layer, speech acts in the form of KQML performatives, as in AGENTK, are issued and received. There could be collection of conversation policies that the planner can choose from in order to Verify a prescription. One such policy might involve only private acts. Another might converse with a known and trusted agent. While yet another might recruit an unknown agent. As seen in the use of the recruit-one performative, speech acts can be constructed with reference to capability, in addition to reference by name.

Discourse Manager: The GPSS critic/tutor agent needs to communicate with both the human user as well as other machine agents. This module manages both type of conversations in the form of speech acts. Speech acts issued by the planner are passed to the discourse manager, which processes them based on established criteria. If a given speech act is requesting a new conversation, the discourse manager will instantiate a new conversation object. Subsequent speech acts reference the conversation object. The discourse manager can affect changes to the user environment.

Figure 2 - Illustration of the Planner's Two Level Network of Decision Paths and Conversation Policies



Top Layer - Decision Networks (arcs are procedures)
Bottom Layer - Conversation Policy (arcs are speech acts <received>/<sent>)

In the HOLON example, if a doctor (user) attempts to prescribe a drug that is inappropriate for a given patient and medical record, the agent will commit to alerting the user. This commitment is passed as a speech act to the discourse manager, where it is acheived by displaying a red alert message to the user's screen.

2.2) Distributed Operator

In order for agents to run on and communicate across computers, it is necessary for each participating computer to have a resident "operator". This section describes the distributed operator. Specifically, this section describes the distributed operator's modules for inter-operator conversation policies (communication), coordination (synchronization), naming facility, agent management, resource allocation, deadlock detection, resource protection, conversation security, agent authentication, and persistency. In addition to traditional distributed operating systems services the distributed operator needs to take care of the following situations (in HOLON): an event or a change to a work artifact could trigger the agent to make a commitment. This in turn needs the services of remote event notification, conversation-flow handling (transaction service operations such as commitment, concurrency control and recovery after failures), translation between agents within a group, and migrating agents or external objects. In the following we elaborate a few of these facilities.

Communication -- The interoperator communication facility is defined in terms of communications between distributed operators, and in terms of two abstractions: links between specific agents/entities and ports (direct and indirect communication) between locales. When an agent needs to interact with objects on the "Shared Task Object Space" as well as with remote applications and resources, the agent needs to obtain a "string value" from an object reference, and acquire an object handle (object id) to set up the link.

AllocatePort	Creates a new port, allocates space for its queue of messages, and returns a local port name.

DeallocatePort	Destroys a port, if it both owns and has receive access to it.
MakeLink	Construct a new link with the creator holding both ends.
DestroyLink	Destroys the link one end, and notifies the other end.

Coordination -- Basic coordination (synchronization) activities in this module never block. The Send and Receive services initiate communication, but do not wait for completion. This implies that an operator can issue Send or Receive requests on many links without waiting for any to finish. Operators can perform work while communication is in progress. Issuing the Send and Receive calls is synchronous; a process knows at what time the request was issued. On the other hand, completion is asynchronous; the data transfer may occur at any time in the future.

Send	Posts a send operation on a given link end.
Receive	Posts a receive operation on a given link end
Cancel	Attempts to cancel a previously-started Send or Receive operation.
Wait	Waits for an operation to complete.
GetResult	Asks for the same information returned by Wait, however does not block if the operation has not completed.

Naming facility -- This facility ideally offers directory/advertising, lookup/reference, and interrogation services. Internet agents and resources are expected to publish their "model" (e.g., ontology, services, ACL vocabulary/speech act set) with a broker/name server. The directory should include the following: what types of training materials do I offer, what types of errors can I detect, what is my internal language, what is my full set of ACL vocabulary/speech acts and what translation (thesaurus) services can be arranged. Agents can use this service to lookup the resources that will satisfy their commitments and intentions. Specifically, the lookup should be via direct name call or via capability request. Thus an agnet that neded to have a prescription checked for interaction effects could find any of several possible drug reference guides via the naming facility.

Subscribe	Register an Agent with its relevant advertising information.
Unsbscribe	Take out an agent advertising information.
LookUp	Lookup for an agent by name, roll or capability.

Agent Management -- The agent management module has two distinct services: operations on agents and agent migration. An agent life-cycle specification, which describes how agents are created (MakeAgent), launch (StartAgent), destroyed (TerminateAgent), is the responsibility of the agent management module and part of operations on agents. Furthermore, it exercises control over the created agent by: PeekAgent, PokeAgent, InspireAgent, ExpireAgent, StatusAgent, SuspendAgent and ResumeAgent. Agent migration is performed by two cooperating distributed operators by MigrateAgentOut, MigrateAgentIn and CancelAgentMigrate.

MakeAgent	Create an agent with the memory requirements, where it starts in physical memory and how long it is.
StartAgent	Launch an agent.
TerminateAgent	Terminate an agent, destroy all links owned by the terminated agent, and perform garbage collection.
MigrateAgentOut	Migrate an Agent to a distributed environment.
MigrateAgentIn	Approve/disapprove a migration attempt after negotiating it with the operator that controls the source agent.
CancelAgentMigrate	Tries to retroactively terminate a MigrateAgentOut or MigrateAgentIn request.

Resource Allocation -- For each resource in the system, the resource manager maintains an entry in its Resource Table, containing information necessary to manage the resource. This table is searched by the resource manager when it tries to satisfy incoming ResourceRequest, ResourceRelease or ResourceConfig messages. Load balancing is part of the broad problem of resource allocation. The problem of load balancing (referred to also as distributed scheduling) is how to distribute agents among processors connected by a network to equalize the workload among them. While load sharing uses agent migration only when there is an idle processor, balancing the workload may require migration even when no node is idle.

ResourceRequest	Requests a resource to be allocate.
ResourceRelease	Release a resource allocation.
ResourceConfig	Resource (e.g. environment, network and files) configuration.

3) Implementation Challenges and Issues

A number of challenges need to be faced if situated GPSS critics/tutors are to provide agent services in organizational settings. We have not yet worked out how our system should handle all these issues. Here is a sample:

naming facility -- If a resource/agent does not publish its verb set, ontology, and services in the name server's directory, a desired service is to interrogate external resources, and learn, remember, and catalog (index) them. Since accessing, interrogating, and processing information can be very costly (and sometimes infeasible), the operator would seek the best performance with the resources available. This means that when the operator is not processing a conversation, it gathers information to aid in future naming retrieval requests. It is an open research issue to determine the necessary and sufficient advertising that an agent needs to publish to obtain knowledge sharing as well as the methods available to operators to learn about the contents of the information sources or the agent specific-domain [Hsu and Knoblock, 1993].

function access -- many resources/applications one would like to access may not open their funtion calls to outside agents. For example, in OLE, the Dynamic Data Exchange protocol is only available for a few function arguments of many proprietary systems. Until those vendors decide to open up their systems, little can be done to foster integration into a federated architecture such as is modelled here.

wrapper/translation -- even when function calls are available, when legacy applications are included in the architecture, they may not be suited to agent calling due to terminological, ontological, and definitional differences. Building interface agents able to perform such translation services is often a time consuming, manual task. In the medical field there are some vocabulary servers beginning to surface that could ease this difficulty: e.g., see Gennari et al. (1995). Other domains do not have such advanced capabilities.

team settings -- often people need support from others in performing a task. So the GPSS tutor/critic agents need to support teams, possibly from differing perspectives, when teams collaborate on shared work artifacts. The difficulty is that thiis presumes not only shared work spaces, but also that agents can adapt the critic/tutor services they provide to needs of different users. We have designed our system to support multiple users, but have yet to consider scaling to 100s or 1,000s of users, many with different professional perspectives and support needs.

data warehouse/case based reasoning -- as individuals' problems arise and are solved, these problem-solution pairs become lessons learned for improving future agent performance and for outcome studies. These lessons need to be captured and archived, possibly as case bases that can be shared by other critic/tutor agents across space and time. (Presumes CBR in place for each agent -- longer term goal).

code mobility/platform issues -- to provide individualized support tutor/critic agents may need to move any of a variety of elements into/out of each team members' individual workspaces (e.g., code, data, text, forms, streams, conversation objects, etc.). This raises a number of unresolved issues for the hosting platform such as, but not limited to: security, privacy, resource use, load balancing schemes, migration algorithms, economics, '**when** to migrate **which** agents to **where**', and so on.

implementation --A number of languages, environments, and tool sets are being proposed to facilitate the needed modules. To mention a few, these include Java, Java Script, Telescript Magic, TCL/TK, CyberAgent, Clearwater, Level5 Object, OLE2, and CORBA, among others. Many of these systems are embryonic and rapidly evolving. No one language yet offers all the features one would desire. Further, it is difficult to determine the extent to which they do offer the desired modules/services. Attempts at evaluation ratings published to date are preliminary and are based on vendor literature for the most part. While we currently have installed or are about to install each of these systems, only in a few cases have we yet verified the ratings with the actual code. One of the difficulties of such an evaluation concerns the subtleties that only show up after a larger scale implementation has been attempted. Unfortunately, our resources will not permit us to attempt a thorough evaluation of each of these alternatives.

4) Concluding Remark

This paper has presented an approach that promises to:

- "agentify" group performance support systems (i.e., situated, lifelong training and decision support systems)
- extend agent oriented programming in the direction of group performance support systems (e.g., identified the generic tutor/critic commitment rules, inform/feedback procedure store, and two-layer conversation policy sets for the planner).
- provide new capabilities for Internet information systems so they become anticipatory and so relevant information travels to the user in a proactive fashion.

Thus far the work has involved modeling the approach. The next step is to begin the implementation.

Acknowledgement

The financial support are gratefully acknowledged of NIST ATP II and the Koop Foundation Inc.

References

Davies, W., Edwards, P., "Agent-K: Integration of AOP and KQML," Aberdeen, UK: King's College Tech Report, (http://www.csu.abdn.ac.uk/pubis/agentk.html

Shoham, Y., Agent-Oriented Programming, Stanford U., CS Dept. Tech Report, STAN-CS-90-1335, 1990

Thomas, S.R., PLACA, an Agent Oriented Programming Language, PhD Dissertation, CS Dept. Stanford Univ., 1993.

Finin, T., et al., "DRAFT Specification of the KQML Agent-Communication Language, unpublished draft, 1993 (http://www.cs.umbc.edu/pubs/kqml.html)

Barbucceanu, M., Fox, M.S., "COOL: A Language for Describing Coordination in MultiAgent Systems," Dept. Industrial Engineering, U. of Toronto, unpublished technical report (http://www.ie.utoronto.ca/pubs/cool.html)

Silverman, B.G., Critiquing Human Error, London: Academic Press, 1992

Silverman, B.G., Owens, J., "Enterprise Information Infrastructure," Systems Engg Journal (pending).

Knoblock, C.A., Arens, Y., Hsu, C., "Cooperating Agents for Information Retrieval," Coop Info Sys. Proc., 1994

Gennari, et al.,"HTTP-based Arch. for Interaction with a Vocabulary Server,"CAMIS tech.report'95

Bradshaw et al., "KAoS: A Generic Agent Architecture for Aerospace Applications," CIKM'95, ACM.