Industrial-Strength Conversations

Amy Unruh
MCC
3500 W. Balcones Center Drive
Austin, TX 78759
unruh@mcc.com

Marian Nodine
MCC
3500 W. Balcones Center Drive
Austin, TX 78759
nodine@mcc.com

ABSTRACT

The goal of this paper is to describe some of our experiences with implementing long-running tasks in industrial-strength applications that require continuous processing of information and that use agent-based technologies. These tasks typically have involved many heterogeneous agents. Some of these agents take on specialized subtasks that can only be done by the one agent. Other agents fill generalized roles within the task, and at any given instant of time any one of a set of agents that can fill a particular role may be doing it on behalf of a given task.

One further aspect that we consider in this paper is that agent systems are *dynamic*, in that agents may enter and leave the agent system at any time. Thus, at a specific instant in time, there is a specific set of active agents working on different aspects of the task, each fulfilling a specific role. These agents are naturally collaborating towards completing the overall task, and thus must exchange information with each other during the course of their work. These extended conversations among agents encompass a wide range of complexity, and need to overcome obstacles unique to their operating environment.

In enacting extended conversations in an agent-based system, especially ones that operate essentially continuously, several aspects of conversation policy specification are needed. While some of these notions are implemented in an adhoc manner by some agent-based systems, there has been inadequate work on distilling them into policies at the conversation level.

The aspects of conversation policy specification that need to be addressed at the level of conversation types include the following:

Role-based conversation specification: Most conversation managers deal with agent-based conversation specifications. However, since different agents may fill the same role in a conversation at a different times, other agents may need to adapt to the fact that they may be talking to different agents over time, even though those agents fill the same role.

Also, at the level of the agent-based system itself, we have identified the following types of policies related to conversation specification and management:

Agent location and message passing policies: Because existing conversation specification methodologies are primarily agent-based, once an agent is engaged in a conversation, message passing is simplified to be between specific agents. With role-based conversation, execution of a conversation, an agent may need to locate another agent to fill a specific role at any given time. Furthermore, an agent may need to communicate information either to a specific agent or to some agent filling a specific role, but no such agent may be online at the time the communication is initiated.

Control policies for extended conversations: In most existing systems, the timing of the execution of subtasks is directly linked to the timing of the subtask-related messages between agents. However, there are many situations where agents may need to be told not only *what* to do for a given request, but *when* and *how*. Thus, the agent system needs a facility to converse about conversations. For example, agents may wish to take a data-driven approach to execution where an agent executes based on the availability of data and/or a control-driven approach where computation is gated by the presence of certain conditions in the operating environment.

1. INTRODUCTION

In an agent-based system [3], agents cooperate to execute complex tasks. These complex tasks are defined abstractly as a (possibly emergent) set of nested abstract *subtasks*[1]. A complex task is instantiated, based on the request of a user, by assigning specific agents within the agent system to execute specific subtasks using the user request and the information derived from it as parameters. In this case, we say that each agent running some specific subtask takes on the *role* of that specific subtask. In a long-running task, we allow different agents to take on a specific role at different times.

[1]Jennings et.al. [8] use the terms *complex service* and *task* to represent our notions of *complex task* and *subtask*, respectively.

Naturally, to correctly execute a task, the agents taking on the different roles need to converse about the progress of the task. Unlike objects in a distributed object system, agents in an agent-based system interact using complex communication patterns, called *conversations*. Agent conversations fall into two levels, *extended conversations* that encompass all of the interactions among agents and agent types related to some overall task being executed on behalf of the user, and individual, *localized conversations* between pairs of specific agents that form the components of the extended conversations.

Localized conversations occur between pairs of agents, and pertain to the specifics of the particular subtask that the agent is running. Localized conversation types are usually identified by the speech act of the performative that initiated the conversation, e.g., *ask-one* conversation or a *subscribe* conversation. Usually the structure of a localized conversation is well-defined and corresponds to the communication structure required by the underlying abstract subtask.

Extended conversations emerge based on the nature of the task specified to the agent system and the internal operation of the agents that cooperate to execute the task. For instance, as an individual agent may execute parts of a subtask internally, and try to delegate other parts of the subtask to other agents better able to to the work. The form of the extended conversation is affected by what parts executed internally, and by what parts are delegated, and to whom.

Note that, while the structure of the localized conversations may be fairly deterministic (and thus concretely specifiable), the structure of the extended conversations is more flexible and emergent according to the specific needs of the task and the specific availability of agents to partition the task.

The purpose of a conversation, extended or localized, in an agent system, is to direct agents in the execution of specific tasks that have been input to the agent community. This is evident from the terms that agent communication languages use for communicative messages – "speech acts" and "performatives". Clearly, since we expect the individual messages to effect some action in the receiving agent, so we expect the conversation as a whole to effect some action in the agent community.

Large agent communities with diverse agents of differing capabilities present interesting challenges from a conversation perspective. In long-running applications, this is complicated by the fact that an extended conversation may need to operate continuously in an effective manner, and the localized conversations that support them may need to be similarly long-lived. In a dynamic agent system where agents may come and go at will, these extended conversations may outlive any agents that are executing their related tasks. In addition, a system configuration is not necessarily static throughout the lifetime of a long-running task; sometimes, the tasks may execute more efficiently when their subtasks are executed asynchronously or when related tasks are balanced over a set of agents with similar capabilities. These conditions in turn have impact on both the agents' ability to send and receive messages, and the need to maintain external, persistent state related to the extended conversation.

In addition, the system must have a robust control structure; one that allows for control over *how*, *where* and *when* a subtask is executed (as opposed to *what* the subtask exactly is). To support this, task specifiers and controllers, both human and agent, must be able to determine at any time exactly how far the task has proceeded, what has been done, and how to continue that task from the current moment, to effect correct control.

To illustrate these issues more clearly, let us consider a fairly straightforward business-intelligence type of conversation such as we have implemented in one of the InfoSleuth™ business intelligence applications. One of the standard types of conversations involves requests by a company for articles related to its business interests. The business analyst poses his request to the InfoSleuth system in the form of a "challenge", specifying a classification hierarchy for documents and a specification as to where documents might be found. Once documents are classified against the classification hierarchy, different types of end users can ask questions about the collected documents and data at varying levels of abstraction and sophistication. The exact scope of the questions that can be answered is in fact determined by the set of agents that are available to answer the questions at the time they are posed to InfoSleuth and the analysis that has been done on the documents.

In implementing this application, InfoSleuth agents periodically sweep certain areas of the World Wide Web where documents pertaining to those subject matters are likely to be found. These articles are collected into an intermediate repository. Other sets of agents can classify these articles as to subject matter, as relevant to the company; and can extract various classes of "facts" from the documents. Furthermore, other agents rank these articles according to the certainty of the classification and the relevance of the subject matter to the company. Yet a fourth set of agents looks for publication trends as well as correlations with information from other information sources. Business analysts and corporate planners can query the collected information at varying levels of sophistication; by classified area, by relevance, or merely at the level of noting interesting events related to publication.

The types of agents required to implement a task at this level of complexity is fairly diverse, and the extended conversations used by the agents to mete out each task among themselves is fairly complex. In this paper, we discuss some of the issues that need to be dealt with at the conversational level. These include (1) the need to carry on conversations about tasks; (2) the need to converse about how to manage the mix of tasks currently running within the system; and (3) the need to converse about the agents that are available and able to go to work at a given time.

Current agent communication languages and conversational systems focus on the first of these needs, consider the second to some extent, and seem to have some troubles with the third area.

In the remainder of this paper, we first provide an overview of the InfoSleuth system in Section 2. We give an example of a potentially problematic extended conversation in Section 3

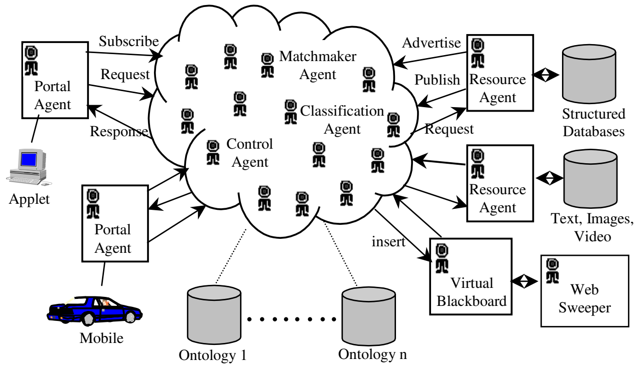

Figure 1: The InfoSleuth agent cloud

and its implementation in InfoSleuth that addresses some of these problems (Section 4. We look towards the future generalization and extension of conversational support for these types of conversations in Section 5. Finally, we provide a discussion of related work in Section 6 and conclude in 7.

2. INFOSLEUTH TECHNOLOGIES

The InfoSleuth agent-based system, shown in Figure 1, is a proven technology for ontology-based access to and analysis of multiple, multimedia information sources. Initially designed for distributed query processing over heterogeneous data sources, the InfoSleuth system has grown over the years to incorporate heavyweight processes such as data analysis tools, information extraction tools and document classifiers. The goal of the InfoSleuth agent system is to take a (possibly overwhelmingly large) set of information, from multiple sources, and from that provide each user with the information he wants, at the level of abstraction that he wants, over the time period he wants. This ability to communicate information is predicated on the use of a set of common, focused *ontologies* that define a vocabulary and semantics, sharable among the agents.

In one type of ontology use, resource agents extract information from various information sources and map it into the terms specified in one or more ontologies. This ontologically-represented information percolates its way through a "cloud" of agents that filters, extracts, analyzes, abstracts and/or fuses it as required by the user. Results are presented to the users via specialized portal agents. Each user may view the information via his own applets or using his own view of the information in the ontology.

In another type of ontology use, resource agents such as web sweepers collect whole documents from various locations at the direction of some user and place them in a warehouse or on a virtual blackboard. Classification agents then take these documents and classify them according to the semantic terms defined in one or more ontologies. Other agents within the "cloud" can then analyze and filter information related to the classified documents. Additionally, users can pose queries to retrieve documents pertaining to specific terms within the ontology.

Part of the success of the InfoSleuth system can be attributed to the stable infrastructure that applies generically to all agents and set of core agents that support a broad diversity of applications. The infrastructure supports an internal standard for communication and collaboration. InfoSleuth agents communicate using a standard set of localized conversations, defined in terms of exchanges of KQML messages between the participating.

InfoSleuth agent systems are intended to be dynamic, in that the set of agents available to respond to requests and challenges may change over time. InfoSleuth agents locate other InfoSleuth agents in their community via a system of *matchmakers*. When an agent comes online, it advertises its capabilities to one or more matchmakers in the system. The matchmakers thus collectively serve as a repository for information about all of the online agents in the system, and about their capabilities and the roles that they can fill. Using this information, the matchmakers also provide the ability for an agent to specify a set of capabilities that it needs from another agent, and to receive a set of recommendations of other agents that can fulfill those capabilities.

InfoSleuth agents have varying capabilities and can fill different kinds of information processing and analysis roles. Some of these agents respond to short-term requests, asked once and answered once – "queries" into the information. These pose very little conversational strain on the agent system. Some requests, generated by individual users, are requests to pull information from the resource agents in an ongoing, real-time manner. The user poses a request for a set of information, and the agent system continually monitors the system, feeding the user new information as needed. This kind of request we call "challenges". When multiple users specify similar challenges, the designer of the agent system may multiplex the processing related to these challenges into longer-running "push tasks". Because the push tasks operate on behalf of multiple challenges, their control becomes partly independent of the control of the individual challenges. They also pose the additional problem of requiring individual message exchanges that are themselves a part of multiple extended conversations.

For the remainder of the paper, we will focus on the conversational issues generated only by extended conversations that support challenges.

3. CONVERSATIONAL EXAMPLE

In this section, we illustrate more completely a small subset of the extended conversation we alluded to in the introduction. This example uses *control agents*, rule-based agents which instantiate and interpret specified tasks, to enact the extended conversation within the agent system. It also assumes two types of a class of agents that we call *analysis agents*, web sweepers and classifiers, that execute long-running and potentially computationally intense subtasks.

In this extended conversation, the user (a business analyst) introduces a challenge to the agent community. This challenge consists of a classification hierarchy and directions on how to locate and classify a set of documents. The business analyst poses this challenge to a *portal agent*, which forwards it to some control agent that knows how to set up a classification task. One element of the task includes notifying

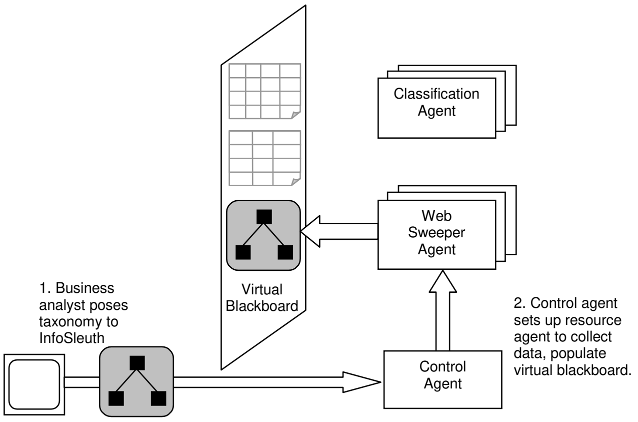

Figure 2: Sweeping setup

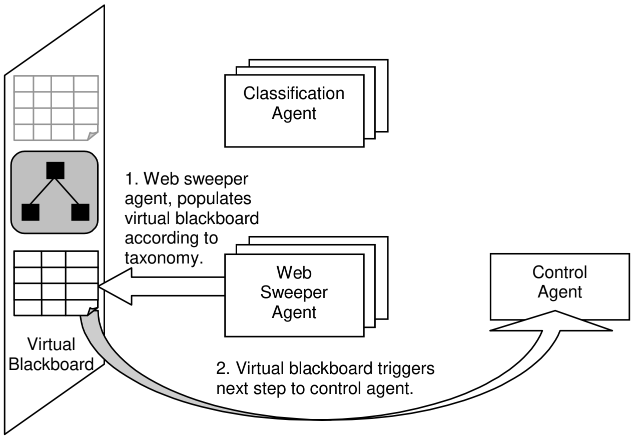

Figure 3: Analysis trigger and setup

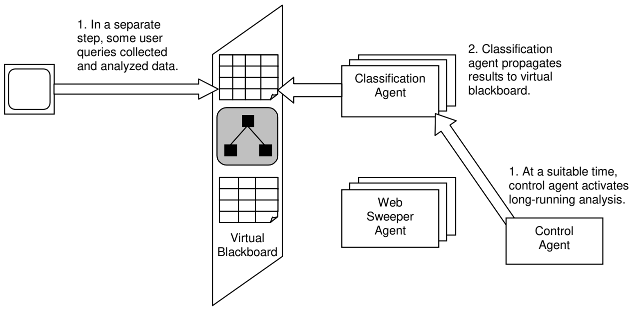

Figure 4: Analysis activation

multiple web sweepers of the areas of the World Wide Web to look for documents. A web sweeper places information about its gathered documents on a virtual blackboard, or VBB. The virtual blackboard is a collection of one or more resource agents that persistently stores information generated during the execution of the task. InfoSleuth's query integration capabilities allow the set of resource agents to be transparently viewed as a single "warehouse" by the analysis agents, to which they post can post persistent state information and (sub-)task results.

An activated web sweeper begins traversing the web and locally storing relevant pages. The web sweeper agent, part of a general class of service resource agents, copies documents from the specified areas of the web to the virtual blackboard. Initially, these documents are unclassified. The sweeping effort is asynchronous, and can be done by different web sweepers at different times or over different segments of the web. This is shown in Figure 2.

Once documents have been accumulated in the virtual blackboard, they are ready for classification. Unlike web sweeping, which is a fairly rapid process on a per-document basis, classification is cumbersome because it involves analyzing the semantics of the document against the semantics of a classification hierarchy. Furthermore, classification may be offline – no direct queries involving inline classification are expected. Therefore, the classifiers are invoked as needed *and* as computing resources become available to execute this subtask. The system achieves this by having the web sweepers inform the control agent when there is information to be classified, effectively triggering the task in the control agent. The task will later be actually activated by the control agent at a propitious time. This is shown in Figure 3.

At some point in time, the control agent notes that the classification task is triggered and that resources are now available for running the classification agent. The control agent then activates the classification agent, which autonomously locates the documents, classifies them against the ontology that was presented by the user, and propagates the results back to the virtual blackboard. This is shown in Figure 4.

At this point, the ontology initially posed as a part of the initial challenge has been populated with documents, and those documents are ready for inline querying by the users. The web sweepers continue to collect more documents, and the classification agents continue to classify the documents to the concept hierarchy, so more and more relevant information becomes available.

This conversation example brings out several of the issues that we addressed in the Introduction. First, it is a long-running conversation – once the user poses the challenge, it remains in effect until the user cancels it. Different web sweeper agents and different classification agents may occupy those respective roles over time. The web sweeper agents need to post documents for the classification agents, but the classification agents may not be online when the web sweeper agents actually locate the documents. Control agents guide the overall task.

4. IMPLEMENTATION ISSUES

In this section we specifically address not conversation implementation issues, but implementation issues of the agents in the system supporting an extended interaction or conversation, that requires longevity that surpasses the longevity of the agents that participate in the conversation. In this situation, agents must consistently support a given agent location, message passing, and control policy within each given task.

As our practical experience centers around using InfoSleuth for implementing these tasks, this discussion is biased towards the solutions we have implemented within the running InfoSleuth applications. There are four general classes of issues that must be addressed in InfoSleuth, with respect to what needs to be implemented to support robust and persistent extended conversations, or system-wide interaction patterns.

1. What are the localized conversation policies that the system has available to it?
2. What should the capabilities of the agents include, especially those executing long-running subtasks? What information do they need to maintain?
3. What kind of information do the agents need to advertise?
4. What should be explicitly specified by control agents at the level of the extended conversation?

Below, we discuss approaches to addressing each of these issues, and for each approach, point to those aspects of our overall goal of robust and persistent interactions, that it supports.

4.1 Conversation Capability Tradeoffs

In an operating agent system, there is a tradeoff between the capabilities of the localized conversation management and the capabilities of the agents to adapt those conversations for their use within the extended conversations. In particular, the combination of the agent system's available localized conversation policies and the agent capabilities constrain the ways that the system's extended conversation policies can be constructed. The more sophisticated the localized conversation policies, the less the burden placed on application-level support for extended conversations. For example, if suspension of subscriptions is supported by a localized subscription conversation policy, then agents may safely ignore certain types of subscription failures. If subscription suspension is *not* supported by a localized subscription conversation policy, then the agents may deal with subscription failure via a conversation control policy.

4.2 Agent Capabilities

In order to support persistent task management over localized conversation or agent failure, agents carrying out long-running subtasks need to post their results and their processing state, while they are running. In addition, they need to support "start task", "stop task" and "task status" requests from other agents.

If agents not only post their final results to the system's virtual blackboard, but persistently maintain certain intermediate results and processing state information while they are working, then this has several related benefits. First, the consistently writing their result state back to the virtual blackboard enables the use of decoupled task management, as described below in Section 4.4. Second, other agents can ascertain the agent's role and status in a task by accessing the VBB. If this information is recorded properly, it also provides the potential for concurrent work on a task.

In addition, if the agents are able to implement stop- and start-task methods over long-running localized conversations in a manner consistent with the way in which they record persistent information, they then can receive stop/start task commands from other agents without requiring those agents to know the task progress. This means that if, e.g., a control agent must be restarted, it can send a command to a classification agent to re-initiate one of its long-running subtasks, without needing to have explicit access to the subtask status. This allows sub-tasks to be completed across agent failure. Finally, if agents are also able to answer simple status queries about which tasks they are working on, this information can be used to do application load balancing.

4.3 Agent Advertisements

The InfoSleuth agents should advertise and make explicit the information about themselves that is needed by the tasks the agent system can execute. To the extent to which this information can be made explicit to the agent(s) implementing the interaction policies, these policies can be made open, robust and extensible.

Ideally, resource advertisements should include:

1. Information about what tasks they can support, and about task complexity and estimated resources needed.
2. Whether or not they support concurrent access of the same task by multiple agents filling the same role.
3. Information about the task types they support, including inputs, preconditions, results and effects.
4. The tasks they are currently working on.
5. Information about the control strategies they can implement.

In our own work, the control agents currently maintain most of this information, rather than the matchmakers, as the control agents are explicitly responsible for managing these long-running tasks. In addition, the control agents may query other InfoSleuth agents directly for information about the tasks they are working on. In the future, it will be preferable to have the agents advertise this information, then access the matchmaker to reason about advertised process constraints.

4.4 Extended Conversation Control Policies

The InfoSleuth *control agents* are rule-based agents that support instantiation and execution of declaratively-specified *tasks* sent to them by other agents. The declaratively-specified information about a task includes not only the task structure, but an associated rule set. A control agent does not have a pre-defined activity – its actions are determined by the tasks and instructions sent to it by the other agents in the system, or by users via a *portal* agent.

A task consists of *actions*, and each action is defined to have an associated Java procedural attachment. A control agent uses knowledge in its rule base to select and instantiate the specified task fragment(s) appropriate to the current context. As a task is executed, the agent may also access its rule engine from Java during an action implementation, both to assert data into its rule-engine knowledge base, and to employ event detection/derivation rules as part of an action. The task selection, instantiation, and execution model maps roughly to that of an HTN planner.

The control agents are the primary repositories of an InfoSleuth system's knowledge of the global structure of the long-running tasks an application supports. They support extended conversation policies by encoding information via policy rules, task specifications, and/or facts about the domain and application.

Most systems will have more than one control agent, each working on some subtasks of the currently-running tasks. The control agent's policy will not be encoded as a monolithic object, but rather will be emergent from the agent's rule and fact sets. Similarly, the global system interaction patterns for the application can be viewed as emergent from the policies of all the control agents.

In the example application, we cast the control policy in terms of 'data triggers' and subtask preconditions whenever possible. Each subtask can be started independently. Each subtask is triggered by the existence of certain data in the application's virtual blackboard, and each writes its results to the virtual blackboard. This requires more bookkeeping during subtask execution, but enables the system to be restarted seamlessly, allows independent subtasks to run concurrently, and decouples the task "pipeline" so that pieces of the problem can be worked on independently as resources become available. Since the extended conversation can be viewed primarily in terms of data flow, each activity defined in terms of the data it needs rather than control information, it is more robust to temporary failures and load balancing problems. Additionally, triggering knowledge can be separated from scheduling and control knowledge. However, it is worth noting that just casting the problem in terms of data-driven as opposed to process-driven subtasks is not sufficient for implementing all decoupled and robust applications, as there may be transactional requirements for multiple subtasks to execute as atomic units.

All of the following types of information are used by a control agent in order to select and compose extended conversation policies. The greater extent to which each type of information is represented explicitly, the more robust the policy will be.

Conversation policy information: What system-level conversation policies are possible, and rule sets to support each. The rules sets include information about how to construct the conversation as well as conversation-specific control policies.

Task decomposition and role assignment methods: The issues of loci of control across an application, and task decomposition methods, are beyond the scope of this paper, though obviously they impact the system's emergent extended conversations.

Task trigger information: For each agent role or type, the control agent needs information about its trigger conditions in the context of the application, and/or information about whether or not a given agent can autonomously determine *itself* when its trigger conditions are met.

Scheduling and selection rules: For a an agent role whose trigger conditions are satisfied, the control agent needs information about when to activate it. A control agent must determine when and how many instances of the agent type should be used to work on the task. This set of rules can include error condition rules: what actions to take when an agent or conversation dies. This set of rules will also leverage knowledge of agent types/roles and type capabilities; information about agents' load and activities; and domain knowledge. Queries to agents about their current activities ("status" queries), as well as agent negotiation/bidding, can occur in this context.

Control flow specification: When it is not possible to cast the problem as decoupled subtasks, the control agent uses sub-task specifications to tell it what action should follow another in an interaction, and how the inputs of an action bind to the results of previous actions.

5. SPECIFICATION / POLICY ISSUES

Our application and its implementation from the previous sections have led us towards a more comprehensive view of the different issues that surround the specification of robust extended conversations in a dynamic environment. In this section, we discuss some of the directions we have examined with respect to extended conversation specifications and policies. These directions extend those we have implemented in the previous example, as well as other InfoSleuth applications.

In [6], Greaves *et.al.* discuss multiple areas in which (extended) conversation policies need to be developed, and a particular multi-level view towards the instantiation of these extended conversations. This section follows in the same spirit, addressing areas of conversation specification and levels of development need to instantiate a specific extended conversation to support a large and long-running task partitioned among many agents.

We have mentioned three areas where policy specification is important. These are (1) role-based conversation specification, (2) agent location and message passing policies and (3) control policies for extended conversations. The first area is specific to the type of task that has been posed by the user, and may be developed by piecing together role-based interaction pattern specifications. The latter two can either be specified at the level of the agent system, or be attached to specific conversations, depending on the needs of the agent system.

5.1 Role-Based Conversations

At a global level, tasks are partitioned among agents in an agent-based system, and the interactions among these agents form interaction patterns. These interaction patterns represent the extended conversations or components of the extended conversations that the agents in the agent-based system will be involved in. A given pattern may apply to an entire task, or to some subtask that is executed coopera-

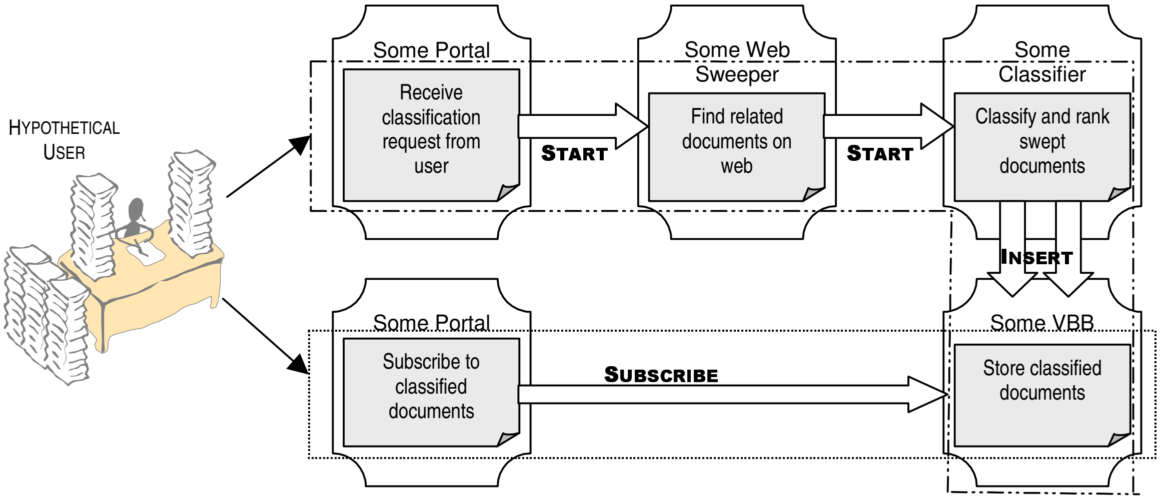

Figure 5: Role-based specification of example extended conversation

tively among a smaller subset of the agents. These patterns themselves may be fairly predictable – the emergence of the overall conversation comes from different ways of composing these patterns together.

The specification of an extended conversation may be based on the the interaction patterns that the agents in the system will naturally follow. In a limited system of idealized, robust agents it may be possible to specify these interaction patterns in terms of localized conversations among a fixed set of agents. However, in a long-running agent system with failure-prone agents, these interaction patterns should more generally (and more properly) be specified as a partial ordering of localized conversation types between general agent roles. From our example, this might look as shown in Figure 5. Here, we have localized conversation types (start, subscribe) among agents filling different roles (portal, web sweeper, classifier, VBB).

In addition to the aspect of interaction patterns being defined among agent roles, there is also the aspect that multiple interactions get composed together to form the extended conversation. In the extended conversation in the figure, we show one interaction pattern related to starting the task (represented by the dashed line) and a second interaction pattern relating to subscribing to the classified documents. One could envision other interaction patterns in the context of the same extended conversation; for instance, making specific queries over the classified documents, starting another analysis or extraction task related to the classified documents, and even the inevitable stopping of the classification task. This composition may be as loose as having the interaction patterns run in parallel, assuming the other tasks are properly executing, through concatenation of the interaction patterns to embedding one interaction pattern within another.

5.2 Agent Location and Message-Passing

Given that our conversation policies are specified at the level of a role, the agent-based system itself needs to implement policies for locating agents to fill specific roles. The location of these agents may occur once for each conversation for its entire lifetime, or multiple times as different agents go online and offline. Note that there are several conditions that may hold upon attempting to locate an agent, including:

1. There may be an agent online that you can contact immediately.
2. There may be an agent (or a role) offline that you can leave a message for.
3. There may exist a matchmaking service that helps locate agents by name or by role.
4. The requesting agent may be able to start an agent to fill the role.
5. There may be no way of finding any agents to fill the role.

Clearly, agent location and message passing policies should take into account any of these cases that are possible, and have a failure mode if no attempts to locate an appropriate agent can be located.

Table 1 shows two possible agent location policies for an agent-based system. In the first policy, consisting of the pair {L-1 Role, L-1 Agent}, once an agent takes up a specific role in a conversation it is used until the agent fails or goes offline, then a new one is located. Conversation failure occurs when no online agent can fill the role.

In the second policy, consisting of the pair {L-2 Role, L-2 Agent}, the agent-based system also wants to maintain the property that once an agent starts filling a role, it fills it until its death. If the agent goes offline, its messages are delivered to the agent's mailbox on the virtual blackboard. If the agent filling the role fails, a new agent is selected to fill that role. When selecting a new agent to fill that role, the policy is to negotiate among the agents that can fill the role, looking for the best one. Lastly, the policy assumes that it wants to be interacting with a possibly offline, but

POLICY L-1 Role *Assumes addressee is a role* 1. Check your cache to see if you already are communicating with an agent in this role. If so, send the message using Policy L-1 Agent. If this fails, remove the agent from the cache. 2. If step 1 fails, ask the matchmaker for possible agents to fill the role, then try each returned agent in turn using Policy L-1 Agent until one succeeds. Cache that agent. 3. If step 2 fails, terminate the conversation.
POLICY L-1 Agent *Assumes addressee is a specific agent* 1. Try to send the message to the agent. 2. If step 1 fails, get other addresses of that agent from the matchmaker, and try each address in turn until one succeeds. 3. If step 2 fails, return failure.
POLICY L-2 Role *Assumes addressee is a role* 1. Check your cache to see if you already are communicating with an agent in this role. If so, send the message using Policy L-2 Agent. If this fails, remove the agent from the cache. 2. If step 1 fails, ask the matchmaker for possible agents to fill this role. Negotiate using negotiation conversations with the agents recommended by the matchmaker, and select the best such agent. Cache the agent and send the message using policy L-2 Agent. 3. If no current agent is acceptable, start up a new agent. Cache the new agent and send its message using policy L-2 Agent. 4. If step 3 fails, terminate the conversation.
POLICY L-2 Agent *Assumes addressee is a specific agent* 1. Try to send the message to the agent. 2. If step 1 fails, put the message in the agent's mailbox. 3. If step 2 fails, return failure.
POLICY L-3 Role *Assumes addressee is a role* 1. Put the request in the role's mailbox. 2. If step 1 fails, terminate the conversation.

Table 1: Example policy options for role-based agent location

otherwise live agent, so it will start an agent to fill the role if needed. Conversation failure occurs when there is no agent, online or offline, that can fill the role, and no appropriate agent can be started either.

The third policy, consisting of {L-3 Role}, assumes that all message passing occurs via mailboxes in the virtual blackboard. Thus, each role of some conversation is associated with a mailbox on the virtual blackboard, to collect the role's messages in the absence of an active agent filling that role. In this case, the sending agent need know neither which agent is filling the role at the other end of the localized conversation, nor even whether any such agent exists. Conversation failure only occurs when no such mailbox can be located.

5.3 Extended Conversation Control

Orthogonal to the issues of locating and sending messages to agents is the issue of controlling the flow of execution of the extended conversation. Several issues arise even in determining *when* a specific localized conversation is to be initiated, including the following: immediately, when data is available to process, when other conditions are met, and when there is some suitably underloaded agent available that can fill the role. There are also control issues related to the duration and robustness of the conversation, such as whether the control flow is explicitly or implicitly managed, and how extended conversations (e.g., subscriptions) can be suspended by one agent and resumed by another. Our implementation in the previous section followed along the lines of one policy that required the presence of control agents; other InfoSleuth applications have used other policies.

Table 2 shows some example policy options for control of extended conversations. The first policy, C-1, is a non-robust policy that presumes upon the inherent stability of the underlying agent community. The conversation is instantiated, particular agents are located to fill particular roles, and those agents execute the extended conversation to its natural termination. Unexpected agent and localized conversation failures terminate the extended conversation.

Policy C-2 is a more robust policy, intended for 24 x 7 applications. In this policy, the extended conversation carefully maintains the state of its agents and the communications in a Virtual Blackboard. Thus, conversation state, even associated with the localized conversations, is always accessible and recoverable. This means that localized conversations that cover long periods of time, such as subscriptions, may be emulated even across the failure of the agent responding to the subscription. A second interesting aspect of policy C-2 is that it decouples the triggering of a task from the actual activation of that task, using a control agent to monitor the task and activate the subtasks as needed. However, this decoupling requires the presence in the agent system of a specialized set of control agents to set up the conversations and to guide the actual activation of the tasks via the localized conversations.

Because some conversation control policies allow for certain types of conversations to be suspended and later resumed by other agents, these policies may require additional messages in the agent communication language related to when

POLICY C-1
1. Extended conversation is instantiated by a particular request being received by an agent in the initiating role. 2. Extended conversation is continued by locating specific agents to fill ensuing roles using the specified location and message sending policy. 3. Unexpected termination of the localized conversation terminates the extended conversation.
POLICY C-2
1. Extended conversation is instantiated by a particular request being received by an agent in the initiating role. 2. Extended conversation is continued by forwarding the conversation to some control agent. The control agents will coordinate the remainder of the execution. 3. Extended conversation state is saved on the virtual blackboard. Each role is assigned a mailbox and a place to save its state. 4. "Localized" conversations may be suspended and resumed by control agents according to the attached policies. Control agents are also responsible for assigning agents to roles at any given time.
POLICY C-2 Subscription
To suspend a subscription: 1. Terminate the subscribe conversation. Save the last snapshot of the result in the extended conversation state on the virtual blackboard. *To resume a subscription:* 1. Start the same subscription conversation with the new agent filling the same role. 2. When the first result is returned, "diff" it with the last snapshot and return that as the first result.
...

Table 2: Example policy options for control of extended conversations.

and how to execute specific tasks, which we will call "directives". Useful directives that we note include *set up the task*, *suspend the conversation*, *resume the conversation*, *ping an agent* to see if it is alive, *ask an agent* what the status of the task is, and *terminate the task*. Note that these directives and the transmission of them is really orthogonal to the extended conversation specification.

6. RELATED WORK

Conversation policies have been developed in an ongoing manner by different researchers. The SICP and Agent Communication Language workshops have encouraged communication among researchers in conversation policies. Additional work on conversation policies has occurred during the standardization discussions for FIPA [5] and KQML [4]. Conversation specification, either for localized and extended conversations, has been proposed using various methodologies [9, 10, 15]. Of particular interest Barbuceanu's work [1].

Little work has been done in supporting conversations among agents in the robust manner described in this paper. Greaves et.al have worked on conversation specification and separation of conversational policies [6], an important need in this area. Parunak has discussed conversations in industry applications [15]. The JATLite system provides some robustness using persistence in message passing in its Agent Message Router [7]. Jennings et.al have dealt with some of the direc-tive-related issues in their ADEPT agent communication language [8]. However, their focus is more on negotiation and less on robustness.

Within InfoSleuth [2, 12], we have been examining agent communication language and conversational issues [13, 11]. In [14] we have addressed issues of conversation robustness for localized conversations. Some of this work has been implemented generically within the InfoSleuth agent shell.

7. CONCLUSIONS

In this paper, we have examined some conversational issues related to very long-running conversations that have the possibility of outliving the agents that participate in them. This work has been motivated by several InfoSleuth applications that are focused on continual operation, and which comprise both data-gathering components and heavyweight analysis components. Specifying the policies for maintaining and controlling the extended conversations that support this type of task proved to be a challenge that has not been fully addressed within the agent conversation community.

In this paper, we presented three policy-related issues associated with the specification and operation of extended conversations. The first is that extended conversation policies, emergent or not, must be specified in terms of roles that the agents in the conversation participate in. Thus, specific conversations occur between *roles*, rather than between *agents*. Different (but appropriate) agents may fill a given role in a given extended conversation at a given time. Secondly, given that extended conversations are specified among roles, there must be some mechanism for locating which agent is filling a given role at a given time, and/or passing messages to that agent. This type of location policy can either be specified as a property of the agent system itself, or as a property of the specific extended conversation. Thirdly, given that an agent-based system may wish to run a mix of subtasks with different computation and timing constraints, a conversation must be enacted on the basis of some control policy that controls when, where and how each of the subtasks executes, for optimal performance.

Related to this, we have discussed the implementation issues we have encountered when implementing extended conversations that follow a more persistent control policy. This new approach was radically different from the earlier inline computation approach. Its control policy is to post intermediate data and state information to a persistent virtual blackboard, and to cast subtasks to incorporate trigger conditions for when the subtask has work to do, and activation conditions and strategies for when the work actually commences. Transitioning between the two control strategies brought up some new and interesting implementation issues, including the need to deal with mismatches with the underlying localized conversations, the need to have discourse about the control of the extended conversation as well as about the nature of the task, and the need to carefully cast the tasks and conversations such that they can be suspended and resumed.

Acknowledgments

The authors would like to thank Brad Perry, specifically, for his contributions and insights into the problem space defined here. We would also like to thank the rest of the Info-Sleuth crew, headed by Marek Rusinkiewicz.

8. REFERENCES

[1] M. Barbuceanu. Coordinating agents by role based social constraints and conversation plans. In *Proc. National Conference on Artificial Intelligence*, pages 16–21, 1997.

[2] R. Bayardo et al. Semantic integration of information in open and dynamic environments. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 1997.

[3] J. M. Bradshaw. An introduction to software agents. In J. M. Bradshaw, editor, *Software Agents*. AAAI Press / MIT Press, 1997.

[4] T. Finin, Y. Labrou, and J. Mayfield. KQML as an agent communication language. In J. Bradshaw, editor, *Software Agents*, chapter 14. AAAI Press / MIT Press, 1997.

[5] FIPA. http://www.fipa.org.

[6] M. Greaves, H. Holmbeck, and J. Bradshaw. What is a conversation policy? In *Proceedings of the Workshop on Specification and Implementation of Conversation Policies*, 1999.

[7] JATLite introduction. http://java.stanford.edu/index2.htm.

[8] N. R. Jennings, P. Faratin, T. J. Norman, P. O'Brien, and B. Odgers. Autonomous agents for business process management. *International Journal of Applied Artificial Intelligence*, 14(2):145–189, 2000.

[9] K. Kuwabara. AgenTalk: Coordination protocol description for multi-agent systems. In *Proceedings of the International Conference on Multi-Agent Systems*, 1995.

[10] Y. Labrou. *Semantics for an Agent Communication Language*. PhD thesis, University of Maryland at Baltimore County, Sep 1996.

[11] M. Nodine and D. Chandrasekhara. Agent communication languages for information-centric agent communities. In *Proceedings of the Hawaii International Conference on System Sciences*, 1999.

[12] M. Nodine et al. Active information gathering in InfoSleuth. *International Journal of Cooperative Information Systems*, 9(1/2):3–28, 2000.

[13] M. Nodine and A. Unruh. Facilitating open communication in agent systems. In M. Singh, A. Rao, and M. Wooldridge, editors, *Intelligent Agents IV: Agent Theories, Architectures, and Languages*. Springer-Verlag, 1998.

[14] M. Nodine and A. Unruh. Constructing robust conversation policies in dynamic agent communities. In *Proceedings of the Workshop on Specifying and Implementing Conversation Policies*, 1999.

[15] H. V. D. Paranuk. Visualizing agent conversations: Using enhanced dooley graphs for agent design and analysis. In *Proceedings of the International Conference on Multi-Agent Systems*, pages 275–282, 1996.