

Task Coordination Paradigms for Information Agents

Marian Nodine, Damith Chandrasekara and Amy Unruh

MCC, 3500 W. Balcones Ctr. Dr., Austin, TX 78759

nodine@alum.mit.edu

Abstract. In agent systems, different (autonomous) agents collaborate to execute complex tasks. Each agent provides a set of useful capabilities, and the agent system combines these capabilities as needed to perform complex tasks, based on the requests input into the system. Agent communication languages (ACLs) allow agents to communicate with each other about how to partition these tasks, and to specify the responsibilities of the individual agents that are invoked. Current ACLs make certain assumptions about the agent system, such as the stability of the agents, the lifetime of the tasks and the intelligence of the agents in the system, etc. These assumptions are not always applicable in information-centric applications, since such agent systems contain unreliable agents, very long running tasks, agents with widely varying levels of sophistication, etc. Furthermore, not all agents may be able to support intelligent planning to work around these issues, and precanned interactions used in more component-based systems do not work well. Thus, it becomes important that proper support for task coordination be available to these agent systems. In this paper we explore issues related to coordinating large, complex, long running tasks in agent systems. We divide these issues into the following categories: tasks, roles, and conversations. We then discuss how these issues impose requirements on ACL, and propose changes to support these requirements.

1 Introduction

Agent systems [2] are groups of agents that work together as a single system to integrate their functionality. They consist of a group or groups of agents that interoperate, cooperating to execute large complex tasks. In these agent systems each agent has the capability to perform a particular set of tasks or subtasks. To execute a larger, more complex task, an agent system composes a solution to the task from the capabilities provided by the agents in the system. Naturally, a key piece in this picture is the need for the agent system to be able to coordinate the execution of these complex tasks via the messages between agents.

Proposed agent communication language (ACL) standards include FIPA [4] and various flavors of KQML [3, 6, 7]. These proposals are oriented towards speech act theory. Speech acts are utterances that perform some action or request some specification. An ACL message is a representation of a speech act, and thus provides guidelines as to the interpretation of its contents. This facilitates openness by providing a structure on which patterns of discourse can be defined. These ACLs, while well-suited to knowledge-based applications, make certain assumptions about the system, such as:

1. The agents are stable, and therefore the set of available agents is stable, relative to the tasks that are input into the agent system.
2. Related to the above, the lifespan of the tasks are assumed to be much smaller than the lifespans of the agents that execute the tasks.
3. The agents have a certain level of intelligence, that allows them to reason to some degree about the other agents.
4. Tasks are service-oriented, so communication occurs in a request-response manner.

In this paper, we explore issues and requirements that information-centric agents place on an ACL. These agents focus their efforts on collecting, processing, analyzing, and monitoring large amounts of information, and presenting the information at appropriate levels and in appropriate ways to users interested in that information. This particular class of agent systems is interesting because of its eclecticism - cooperating information agents may span a wide range of capabilities, from simple file-accessing agents to agents implementing mathematically-intensive analytical tasks to intelligent agents with sophisticated reasoning engines. We note, however, that the issues we discuss are not necessarily confined to information-centric agent systems. Information-centric agents have several issues that clash with the above assumptions, including:

1. Long-running tasks may easily outlive the agents they run on.
2. Many of the agents do not incorporate any kind of reasoning mechanism.
3. Agents may be mobile, going to the information as opposed to pulling the information to themselves.
4. Communication may be sporadic and task-related, and may not follow a request-response paradigm.

The issues we address here have emerged from our extensive experiences with the InfoSleuth™ [1, 8, 10] system, an information-centric agent system that has been under development for over five years and in use for the last four years. Recently, we have begun to develop long-running tasks in the areas of competitive intelligence and genomic research. These tasks have placed unforeseen strains on both our agents and our ACL. Our issues both synergize with and extend some of the issues we had with shorter-lived, information-centric applications as discussed in [9].

We continue this paper with a discussion about terminology related to tasks and conversations within an agent system. This terminology serves as a framework for the discussions in the rest of this paper. We continue by discussing some language requirements for supporting abstract tasks, tasks, roles and conversations within agent systems. Following this, we note some impedance mismatch issues that come up in any eclectic agent system, and their impact on the ACL and on the agents themselves.

2 Terminology

We will begin with a set of definitions for terms about tasks in agent systems.

2.1 Tasks

In an agent system, a task is executed by multiple cooperating agents. The types of tasks that an agent system can execute are called *abstract tasks*, and are composed of a set of *abstract subtasks* that communicate among themselves using an *extended conversation*. The abstract subtasks form a hierarchy within the abstract task.

An abstract task is instantiated into a *task* by specifying some input from a user or external process. This instantiation can be either *fixed* at the time the input is received, or *emergent*, becoming defined from the context as the task execution unfolds in the agent system. Emergence is affected by the tasks themselves and the nature of the agents that run them; for the purposes of this paper we will largely ignore how the tasks themselves are generated, including whether or not the task is emergent.

2.2 Roles and Conversations

During the execution of a task, *roles* within the task are also instantiated by taking information derived from the specific user input and using this information as input to a relevant abstract subtask. Different roles may be initiated and terminated as a task progresses; thus both tasks and roles occupy specific spans of *time*. Furthermore, both the task and its different roles potentially access various subsets of both the request and the environment the task is running in; this is the role's *space*. The task's space-time is naturally a superset of the space-time of all of its roles.

Within the agent system, different agents have different capabilities, and every role in the system (hopefully) corresponds to one or more agents' capabilities. An agent is assigned to a role by some (other) agent when there is a need for the role to execute, one of the agents' capabilities matches that role, and there is no other agent currently executing that role in the given space-time. This assignment may be done in a variety of ways. Once a role is assigned to some agent, we call the instantiation of that role on that agent an *activity*.

The roles in an agent system need to communicate with each other to coordinate the execution of a task. The set of all the communications regarding the task is called its *extended conversation*. The extended conversation is broken up (conceptually) into a set of interrelated pairwise conversations between roles, which we call *localized conversations*. For any localized conversation, there is an *initiating role* and a *responding role*. The initiating role is the one that specified the particular (sub-)request that caused the instantiation of the responding role. A localized conversation is initiated at some point in the space-time of the initiating role, and can either be terminated explicitly or end implicitly when one of its roles terminates.

2.3 Activities

In an agent system where all of the agents are stably online for the execution of all of the tasks input by the users, each role is matched up to some agent in the system, and the localized conversations can be handled as structured exchanges of messages between the agents. However, there are agent systems where tasks are long-lived, often existing beyond the lifetimes of the agents that may be implementing the different roles. This

may be because the agents are mobile (occasionally going offline to move), or they are unreliable (occasionally exiting precipitously), or they are inherently short-lived (due to the architecture of the agent system). In this case, a role is likely to execute on different agents in different areas of space-time, as a set of activities. If the role has multiple activities running at a given time, they must agree on how to partition the space of the role for the time they are running concurrently. If an activity is assigned to an agent that is online, then the activity is online and its role is online; otherwise, if the agent is offline, the activity is offline. If a role has only offline activities, then the role itself is offline. If a role is assigned to no agent at all, and the current time is within the time of the role, then the role is unassigned. Thus, a role can be in one of three states at any point in its life: *unassigned*, *offline*, or *online*.

2.4 Example

Suppose we have an agent system that supports an abstract subscription task that operates as follows: a user connects to the system via some portal agent, and specifies some subscription. The portal agent forwards the subscription to an agent capable of doing subscriptions. The class of subscription agents in the agent system executes the subscription by periodically querying for the subscription information, and forwarding any changes that have occurred since the last poll. This means that periodically it looks for some information fusion agent that can merge information from different resources, and poses a query to it. The information fusion agent operates by decomposing the query, locating resource agents that can access databases for the query fragments, and forwarding the query fragments to those resource agents. It then merges the information and returns a response to the subscription agent. The subscription agent then computes the difference from the last query and forwards the results to the portal, which transmits them to the user. This is the abstract task.

Supposing we have a user Fred, who is interested in subscribing to information about companies that are developing recycling technologies. Since this is a subscription task that can be handled by the system. Fred's request instantiates a task based on the abstract subscription task. Each subtask in the abstract task gets mapped to one or more roles in the actual task, where each role is responsible for some piece of the input request (based on the nature of the subtask it is instantiated from).

As the task executes, each of the roles must be assigned as needed to capable agents. The matching process locates an agent that has the capability to execute a role, and starts up that role as an activity on that agent. This match may be time-dependent, in that the same role may be assigned to different agents at different times. At the current time T , the agents involved in the task are FredPortal, SubagentB, MQRelation1, CompanyResource and RecyclingResource. This scenario is summarized in Table 1.

3 Abstract Tasks (and Agent Capabilities)

Abstract tasks define the kind of work that a particular agent based system can do. An abstract task consists of a set of abstract subtasks and the localized conversation

Abstract Subtasks (subscription)	Roles (Fred's request)	Activities (Fred's request Time T)
Portal Subtask	Portal Connected to Fred	FredPortal initiating subscription
Subscription Subtask	Subscription to Companies developing recycling technology	SubagentB executing that subscription and sending out queries
Information Fusion Subtask	Query fusing company information with recycling technology information	MQRelationalI decomposing the current query into subqueries for company information and recycling technology information
Database Access Subtask	Query for company information	CompanyResource retrieving company information
	Query for recycling technology companies	RecyclingResource retrieving recycling technology information

Table 1. Example abstract subtasks, roles, activities

types that link pairs of abstract subtasks. Note that these abstract tasks are arranged hierarchically, with some abstract tasks initiating and governing other abstract tasks.

The nature of the abstract tasks that an agent system can do is directly related to the capabilities of the agents within the agent system, in that for each abstract subtask there must be some agent in the agent system that has the capability of executing that type of subtask. The abstract subtask hierarchy of an abstract task mimics the patterns of interaction among these capabilities, because the different capabilities require help from other capabilities to fulfill their tasks. For instance, in our example from the previous section, the capability for information fusion in this case requires access to other agents that have the capability of database access.

4 Tasks

Tasks, with their associated roles and localized conversations, are instantiated based on some input request to the agent system. Recall that the extended conversation related to a task is the collection of localized conversations among the different roles in the task. The extended conversations associated with the different tasks can either follow deterministic interaction patterns among the capabilities in the system, or can emerge as the execution of each instance of the abstract task unfolds. For example, in the restricted scenario in the introduction, the interaction pattern associated with the abstract task is deterministic, in that its basic structure is known before it is instantiated into any tasks. Some agent(s) with the portal capability subscribes to some agent(s) with the subscription capability; each such agent in turn periodically queries some agent(s) with the information fusion capability. Each agent with the information fusion capability queries one or more agents with the database access capability (specifically, the set of all the database access agents that, at the given time, can provide information related to the query). The results are percolated back in an equally deterministic manner.

Suppose, however, that some other agent enters the system with a subscription capability, but which itself induces a different interaction pattern. The selection of which subscription agent to use happens dynamically, during the execution of a task involving subscriptions, at the time a particular agent is selected to fill the subscription role. If many such options exist, and some are unknown, then at any given time it will be difficult to predict how the execution of the task will unfold among the agents themselves, and the extended conversation emerges over the lifetime of the task.

4.1 Task Identification and Specification

An ACL needs to facilitate the management of the task-related services within the agent system. In this section, we discuss some general requirements for ACL support for tasks. Table 2 summarizes these requirements.

Task-1: Each task in the system should have its unique identifier used in all messages pertaining to that task.

From the perspective of the roles within the task, the role may be responding to some localized conversation, and initiating one or more localized conversations. An ACL needs to support the ability to relate all conversations related to a specific role, especially if the task is monitored and/or controlled externally. To facilitate this, there should be some easy way of correlating all localized conversations with their roles; for instance, by providing a task identifier and associating it with all messages and roles related to the task.

Task-2: The ACL should support the ability in responses to explain what the response contains, where the response came from and how it was derived.

One issue that comes up immediately in an agent system where different agents can come and go is that the same sort of task may be executed at different times using different sets of agents, and potentially returning different results. For instance, if an agent with the information fusion capability executes the same query at two separate times, it may get different results because the underlying database resources have changed. This may in turn affect the remainder of the task. As a consequence, results may be incomplete or inconsistent with the user's expectations. Therefore, for any type of task in a dynamic system, it may be important that it have the capability to return results with an associated *pedigree* to aid the requestor in determining where the result came from. Additionally, the result possibly might contain an *explanation* for what was done and not done with respect to the task. Ideas related to this have been explored, for instance, in the DISCO [11] system. Responses also should be able to be annotated with support for using or viewing that information; for instance, it may recommend that large amounts of numerical data be displayed as a graph or scatter plot.

Thus we see four aspects of support, one for requesting the type of support, and three for support returned with a result: its pedigree, its explanation, and its viewing annotations. The user should be able to specify which of these are needed, and this information should be able to be propagated to all the roles that are cooperating to execute the task. Table 2 includes required message properties to implement these options.

Property	Value semantics	Requirement
:task-id	unique task identifier	Task-1
:request-annotations	pedigree, explanation and/or viewing-annotation	Task-2
:pedigree	list of resources	Task-2
:explanation	explanation	Task-2
:viewing-annotation	annotation information	Task-2

Table 2. Message properties related to task identification and specification.

4.2 Task Monitoring and Control

There is a certain amount of monitoring and control that must be done at the task level to ensure the overall correct operation of the task as well as recovery when some unexpected event related to the task occurs. Note that all messages pertaining to the task should contain the task identifier, as specified in Requirement **Task-1**.

Task monitoring can either be done internally to the roles in the task, or externally using a specialist monitoring agent. Task monitoring requires that the agent system be able to supply information about task status and about the agents that are currently executing the different activities current in the task. This leads to the following requirements, summarized in the speech acts of Table 3:

Task-3: All agents in the agent system should be able to respond to pinging for online status, both at the agent level and at the task level.

There are many unforeseen events that can occur during the execution of a task. In a dynamic agent system, with potentially unreliable agents, one type of event that needs to be watched for is the unexpected failure of an agent or a specific activity. Usually, such failures are not accompanied by neat notifications, but must instead be detected by either the roles that are communicating with the activity, or by the monitoring agent. Pinging allows a role to generically ask whether some agent or activity is still running and responding. The `ping` speech act required to implement this is shown in Table 3.

Task-4: The agent system should be able to respond to queries on task status.

While a task is executing, there are several different entities that may be monitoring the task status and progress, including the user, the roles in the task, some monitoring or control agent, etc. For example, some long-running consumer role may be monitoring the progress of another role that feeds information to it. When there is no information available from the role producing the information, the consumer waits for the producer to produce some. Alternatively, a control agent may want to monitor how fast the role is progressing and whether it is falling behind, and move the role to a different agent if it suspects that this would cause an improvement.

Task-5: The ACL should support the ability for tasks to control their roles and localized conversations, at the level of {initiate, start, stop, terminate} the role.

As with task monitoring, task control may either be done internally to the roles in the task, or externally using an explicit control agent. At a minimum, the ACL must support issues related to setting up, starting and running long-running roles, and dealing with unexpected events by terminating or adjusting roles to deal with unforeseen circumstances. Types of role control are summarized in Table 3.

Speech Act	Semantics	Requirement
ping	Tell me if you are really there	Task-3
status(query)	Return the status information as requested by the query	Task-4
error(explanation)	Indicates an error in the initiator that affects the role	Task-4
initiate(parameters)	Set the role up to be ready to run	Task-5
start	Begin executing the role	Task-5
stop	Stop executing the role	Task-5
terminate	Tear down the role so it is no longer ready to run	Task-5

Table 3. Speech acts related to task monitoring and execution.

4.3 Dealing with Instability at the Task Level

Instability issues arise when the task, and the roles that comprise the task, outlive or outgrow the agents that the roles are running on. Several requirements spring from the need to run these long-lived tasks, including:

Task-6: If the task has roles that can go offline, the task should provide each such role with persistent storage space for saving task status (we call this a locker), and a method for leaving messages for the role (we call this a mailbox).

If an activity has the potential to go offline, either expectedly or unexpectedly, then the task itself must provide information to enable the new activity to continue where the previous activity let off. One step in enabling a role to be taken offline is to provide it with a known location for saving any status information and intermediate state associated with the role. When a new agent picks up the role, it then knows exactly where to look for this information as it starts up. A second issue is that the role itself may be receiving messages while it is offline. These messages must be placed in some known mailbox as well, so that the role can process the messages that were received, in the order they were received. This ability to store up inputs and state is a necessary precondition for continuing the role seamlessly on the new agent.

Task-7: The ACL should support the ability to suspend (take offline) a role, and later continue it under specified conditions, perhaps on another agent.

Given that a role is suspendable and movable, and that the role itself is being controlled either by some control agent or by the role that initiated it, there must be some form of ACL support to enable the appropriate requests to be passed to the role itself. These messages need not contain information about the role, as that information can be retrieved from the role's locker and mailbox by the new activity, at its start-up time. New speech acts to support these features are shown in Table 4.

If a role that can be suspended/continued or moved, then the control structure for the task may need to take on the job of determining when this role should be brought online. Factors involved in determining this timing include whether there is unprocessed input to the role, whether there are available agents to execute the role, and whether the timing is correct with respect to other roles in the task or other tasks in the agent system. Note that the suspension request may go from the agent running the activity to the initiating role, informing that role of when its continuation is appropriate.

Task-8: The ACL should support the ability to reconfigure a role.

Speech Act	Semantics	Requirement
suspend(conditions)	Suspend executing the role, take it offline until the conditions are met.	Task-7
continue	Continue operation, placing role online on a given agent	Task-7
move(agent)	Move the role from one agent to another	Task-7
reconfigure(parms)	Reconfigure the role for slightly different operation	Task-8

Table 4. Speech acts related to controlling roles.

One additional tactic to deal with unreliable and bursty operation is to replicate and/or partition the role to run over several agents. Also, if a task is not operating as expected or needed, the user may adjust its operation to optimize it. In either case, the task itself may wish to reconfigure the role's input parameters. ACL support for this requirement is shown in Table 4.

5 Roles

Recall that a role is an abstract subtask that has been partially instantiated with input parameters based on the user request. A role itself is fully instantiated into one or more activities by assigning it to run on one or more agents. In the previous section, we discussed issues of tasks and their related roles, and what the task needs to control the specifics of its roles and their localized conversations. In this section we deal with issues specific to roles: what information is needed to initiate them, how they are instantiated into activities, what must be done to support persistence across activities.

5.1 Role Initiation

Role-1: The ACL should support the ability to specify how a role is to be executed and the results returned in its role-initiating messages.

In an agent system, when one role asks another to execute some abstract subtask related to its request, there may be additional nuances about how the role is to be executed and how the results are to be returned. These nuances must be specified in addition to the computational inputs to that role related to the user request. For example, for all aspects of agent interaction relating to data - telling or updating, querying, and responding, there are nuances about how the information is requested or presented that should not necessarily show up at the layer of the speech act. For instance, if you want to subscribe to a set of information and be notified of a notification may be sent periodically, or as soon as the information changes. The response may contain a fresh copy of all requested data, or only the changes. There are also issues related to the fact that resources may contain overlapping sets of data, also there may be a lot of resources available, containing more information than is really needed by the requester. In this situation, it is helpful to constrain the set of agents accessed, and sometimes to return a "best effort"

Property	Value semantics
:ask-policy	change in data, periodic, all the data, modifications only, etc
:query-effort	best effort, complete result
:query-context	context of query
:reply-out-of-band	ftp, http, etc
:locator	URL locator for the result

Table 5. Properties related to how to execute a role.

result as opposed to a complete result. Some of the message properties that implement these options for query-type roles are shown in Table 5.

These tags may vary depending on the nature of the role. For instance, very long-running roles need to specify how the task should execute, whether or not it can move or be suspended, etc. Short-lived roles may need no additional properties.

5.2 Assigning Agents to Roles

Role-2: An agent system must supply a capability to match agents to roles that the agents can execute.

For correct matching of an agent to a role, some agent must consider the specifics of the role itself, and the capabilities of the agents that potentially may be assigned to the role, and filter through the possible agents to locate the best fit at the current time. This process is called *matching*. There are several methods by which matching can occur, summarized in Table 6. In one method, the role that needs to initiate the next role, and understands its specifics, can send requests to other agents that it knows about. Those agents in turn can try to match the role requests to their own capabilities. If they think there is a match, then they can negotiate for the role with the requester. The requester can then select among the possibilities to select the agent to match to the role.

Role requester	(Potential) responder	Third agent	Example
Requests	Matches, responds		Bid/propose system
Matches, requests	Responds		Gossip system
Requests	Responds	Matches	Facilitated system

Table 6. Matching possibilities.

The second method implies that each agent has a repository of agents that it knows about and their capabilities. In this case, the requesting agent compares its role request with the capabilities of the agents in its repository, and selects one that matches. The role requestor then initiates the role on that agent. This process may need to be repeated. Naturally, the success of this method depends on the ability of each agent to keep an up-to-date repository of the agents it may need. This may be done by soliciting *advertisements* from the other agents that it knows about.

The third method places that repository instead in a specialist agent, called a *facilitator*. The facilitator takes active steps to ensure that its repository is up-to-date, possibly relying on the agents to advertise their capabilities when they come online. The facilitator receives requests for agents with specific replies, matches specific agents to those requests, and does what is needed to ensure that the requesting agent and responding agent get connected with each other. The speech acts required to implement these three possibilities are included in Table 7.

Speech Act	Semantics	Requirement
request-match	Requests a match for a particular role.	Role-2
advertise	Asserts information about the agent's capabilities either for the first time, or when the agent has updated information about its capabilities.	Role-2 Role-4
change-status	Indicates that the agent has changed its status. Potential status values are {online, offline, out-of-service}.	Role-4
rfp(role)	Request for proposals for bids on a role.	Role-5
propose	Propose a match to my agent	Role-5
accept	Accept a role.	Role-5
refuse	Refuse a role.	Role-5

Table 7. Speech acts related to matching and negotiation.

Role-3: Advertisements and requests for roles should not be required to use a specific content language, but rather should use a language appropriate to the needs of the system. The agent system should provide an ontology that specifies the vocabulary for describing agent capabilities.

Many matching systems have been described that deal with matching at various levels. Many of these [5, 7] deal with matching at an interface level - whether the interface to the agent matches the "call" to the interface in the role. However, the interface is merely one aspect of the service, and sometimes more detail is needed about the semantics of the role being matched or the capability of the agent, or their methods for executing and returning results. The clearer the specification of an agent advertisement or a request for a match to a role, the better the matching system can be. This means that an agent must be able to describe its capabilities in more depth than just advertising its service interface. This in turn impacts the ontology of capabilities as well as the content language of the advertisement and request methods.

Role-4: An agent should be able update or delete its advertisement.

An agent offers up its services to an agent system by advertising itself. Later, the agent may wish to change its advertisement, or even advertise that it is going offline or shutting down. This enables other agents to keep their repositories up-to-date. The speech acts needed to implement this are shown in Table 7.

Role-5: An agent should be able to negotiate over or refuse to execute an activity.

Agents are autonomous, which means that an initiating agent cannot assume that it can just tell an agent to take on a role, but rather it must request that the agent take the role. The responding agent in turn must have the option of being able to interact with the requesting agent over the terms and conditions under which it will execute the role. Thus, the ACL must support the ability to negotiate about roles with agents that it has matched. The speech acts in the ACL needed to implement this are shown in Table 7.

Role-6: Roles should respond correctly to control directives from the task.

The task model implied by the agent communication implies that the role should be able to respond to appropriate directives at any time, including both before it receives any results, during the middle of the computation, and during the transmission of results. For many short-lived roles such as responding to short queries for information, the expected control directives may be very simple. However, other long-lived tasks may need to deal with more complex control directives.

Role-7: Roles that can be suspended and continued or moved must support the storing of state in the role's locker, the retrieval of information from the locker and mailbox, and the continuation of the role from where it left off.

This requirements indicate that agents with the capability to execute suspendable roles must be designed with the need for storing and recovering role state in mind. The task itself merely provides locker and mailbox space in a commonly-accessible location, but places no requirements on the agents themselves for how that space is used. The role itself must encapsulate information about how this space is used, what information is kept in it, and how to resume a task given that information. This frees the ACL from the need to define a common format for communicating role state. This is appropriate because roles may differ vastly in what state they need to save.

6 Localized Conversations

Localized conversations occur between roles in a task as a part of the extended conversation. A localized conversation consists of one or more activity-activity conversations, where a new activity-activity conversation is started when the role is first assigned to an agent, and every ensuing time that a role moves from one agent to another.

6.1 Conversation identification

Conv-1: The ACL should support properties for transmitting identifiers for both the localized and the activity-activity conversation.

These identifiers are useful both in sorting the messages by conversation, in determining where to send a response, and in monitoring the operation of the task itself. KQML ignores issues of conversation identification, relying instead on message chaining. This enforces a request-response mode on the conversational structure. Suppose however there are requirements for other sorts of modes of interaction, such as the ability to cancel a long-running request before a response has been received. With a message-chaining conversation model, this conversation structure is at best awkward. We conclude rather that each message in the conversation should be explicitly related to a conversation, as in FIPA. However, our delineation of roles and activities requires additional identifiers.

Property	Value semantics	Requirement
:local-conv-id	unique conversation identifier for localized conversation	Conv-1
:current-conv-id	unique conversation identifier for activity-activity conversation.	Conv-1
:sequence-number	identifier to place message in proper sequence in the localized conversation.	Conv-2
:sending-role	role that sent the message.	Conv-3
:sending-agent	agent that sent the message.	Conv-3
:receiving-role	role that the message is intended for.	Conv-3
:receiving-agent	agent that the message is intended for.	Conv-3

Table 8. Message properties related to localized conversations.

6.2 Dealing with Instability at the Localized Conversation Level

Conv-2: The ACL should support sequencing for messages within conversations, to ensure that messages are received in order even across agent activity transitions.

Conversations with multiple messages add complexity and versatility to communication between agents but may be affected by incorrect message ordering. This problem is important when dealing with multimedia and internet retrieval information since the order of processing of the information in the messages becomes very important. The ACL needs to provide information for assembling the messages into the proper sequence if required, as indicated in Table 8.

While this problem certainly became apparent with large, ordered data sets in some of the early applications of InfoSleuth, it is becoming much more critical when agents can partition roles into a set of concurrent activities. It also becomes an issue when a role is taken offline for some period of time, the new activity must resume from some point in time, and some of the communication may have been lost. Then this sequence becomes important in determining from where in the execution to continue the role.

Conv-3: The ACL should provide information about both the origin and destination roles and the origin and destination agents of the messages.

This requires the message properties to include both the sending and receiving roles and agents, as shown in Table 8. Because the roles that sit at the ends of a localized conversation may be picked up by different agents in turn, the messages themselves should contain information on which role sent them, and where the sending role expected them to be delivered. This enables the roles to monitor when and to where a related role has moved, as well as enabling the forwarding of messages after a role has moved.

Conv-4: Roles that can go offline must support the ability to receive messages while in the offline state, in the mailbox supplied by the task.

Messages pertaining to some localized conversation do not necessarily get sent between the same two agents for the lifetime of the localized conversation, as different agents can assume the same role during the role's lifetime. In fact, at any given time, one or both of the roles involved in the conversation may be offline. Table 9 summarized the various combinations of role states and the possible message-sending options. In addition to the use of a mailbox, the table also incorporates the alternatives of (1)

posting the message on a public blackboard to be picked up later by the role, and (2) starting a new agent to fill the role and receive the message.

Sending role	Receiving role	Localized conv. status	Message passing possibilities
online	online	online	Any
online	offline	partially offline	mailbox, blackboard
online	unassigned	partially offline	mailbox, blackboard, start an agent
offline or unassigned	online	offline	None
offline or unassigned	offline or unassigned	offline	None

Table 9. Message sending options.

7 Impedance Issues

Because of the eclectic nature of the agents systems we are considering, there is not always a clean correlation between the capabilities and reliability of the agents in the system and the needs of the tasks requested of it by the users. Three separate areas where an agent system may encounter such impedance mismatches are:

Capability mismatch: The capabilities of the available agents may not exactly cover the roles needed to execute the task or the advertisements may not provide enough information to determine whether or not the agent can execute the task. This is detected when some agent attempts to assign some role to some other agent, but fails to locate an agent that can fill the role. In this case, either the agent assigning the role must either select an agent to fill the role that is “close enough”, and provide some sort of glue functionality to compensate for the inadequacies, or fail to complete the task. If similar inadequacies occur across different tasks, this may indicate that either a new agent type needs to be inserted into the agent system or some existing agent needs to be enhanced.

Localized conversation mismatch: The localized conversations supported by the agents may not match the needs of the localized conversations between roles. This is detected when an agent looks inside its localized conversational model, and determines that its conversation support is inadequate. This can be, for example, as a consequence of trying to build more sophisticated behavior over less sophisticated behavior, such as implementing complex conversations over distributed object systems. In this case, the agents themselves may need to be designed to incorporate the intended conversational policy, and maintain conversational states and support mechanisms internally.

Stability mismatch: The stability of the individual agents may not match the stability needs of the roles that run on them. This happens when the application itself requires more stability than the agents that support it. This would occur, for example, if some task requires 24 hour a day, 7 day a week stability, and the computers the agents are running on are unreliable. In this case, also, the agent capabilities must include the ability to restart roles over agent failure by reassigning roles and by enabling task persistence through the saving and restoring of state.

8 Conclusions

In this paper we discussed task coordination for agent based systems. Task coordination requires support at two levels: one within the implementation of the agents themselves, and one within the ACL. In this paper, we present requirements that apply to agents and ACLs, based on our experiences in InfoSleuth running information-centric tasks.

We define a *task* as a collection of *roles* that can be instantiated on agents, and a set of conversations among those roles, used for coordination and information transfer. As tasks, and some of the roles that implement them, have the potential to outlive the agents running them, we define *activities* as the instantiations of a task's roles on specific agents at a given time. A particular role may execute as different activities on different agents at different times during its lifetime.

In this paper, we also discussed issues related to supporting the coordination of these large, complex, and possibly long running tasks. Current conversation support in ACLs is insufficient to support the communication necessary to execute these types of tasks. The key contribution of this paper is the suggestion of a general paradigm for decomposing and chaining together tasks, and a set of requirements for an ACL that support this paradigm.

References

1. R. Bayardo et al. Semantic integration of information in open and dynamic environments. In *Proc. ACM SIGMOD Int'l Conf. on Management of Data*. ACM Press, 1997.
2. Jeffrey M. Bradshaw. An introduction to software agents. In J. Bradshaw, editor, *Software Agents*, chapter 1. MIT Press, AAAI Press, 1997.
3. T. Finin and G. Wiederhold. An overview of KQML: A knowledge query and manipulation language, 1991. (Available through the Stanford University Computer Science Department.).
4. FIPA. <http://www.fipa.org>.
5. Object Management Group and X/Open. *The Common Object Request Broker: Architecture and Specification, Revision 1.1*. John Wiley and Sons, 1992.
6. Y. Labrou. *Semantics for an Agent Communication Language*. PhD thesis, University of Maryland at Baltimore County, September 1996.
7. Y. Labrou and T. Finin. A proposal for a new KQML specification, 1997. <http://www.cs.umbc.edu/kqml/kqmlspec.ps>.
8. M. Nodine and A. Unruh. Facilitating open communication in agent systems. In Munindar Singh, Anand Rao, and Michael Wooldridge, editors, *Intelligent Agents IV: Agent Theories, Architectures, and Languages*. Springer-Verlag, 1998.
9. Marian Nodine and Damith Chandrasekhara. Agent communication languages for information-centric agent communities. In *Proc. Hawaii Int'l Conf. on System Sciences*, 1999.
10. Marian Nodine et al. Active information gathering in InfoSleuth. *Int'l Journal of Cooperative Information Systems*, 9(1/2):3–28, 2000.
11. A. Tomasic, L. Raschid, and P. Valduriez. Scaling heterogeneous databases and the design of DISCO. In *Proc. Int'l Conf. of Distributed Computing Systems*, pages 449–457, 1996.