Cougaar Agent Communities

Ronald D. Snyder
Mobile Intelligence Corporation
ron@mobile-intelligence.com

Dr. Douglas C. MacKenzie
Mobile Intelligence Corporation
doug@mobile-intelligence.com

Abstract

The ability to organize agents into abstract groups provides a powerful tool for agent organization and communication in large multi-agent systems. In this paper we outline the fundamental characteristics required of a robust and scalable agent grouping mechanism. These characteristics are then discussed in the context of the Cougaar community infrastructure. This implementation of agent communities illustrates the use of distributed state management, distributed event propagation and abstract messaging in a high-performance agent architecture designed for robustness and scalability.

1. Communities of Agents

As software agent systems grow in size and complexity it becomes very helpful to inject structure into the agent society by grouping agents into communities. A community of agents is a powerful concept that supports information hiding, aggregation, and hierarchical decomposition. Specifically, we expect a community construct to support:

- distributed, peer-to-peer implementation to avoid single points of failure and support scalability,
- flexible semantics for domain modeling,
- abstract messaging allowing messages to be targeted to all members of a community or to a subset based on attributes,
- dynamic creation of communities, addition/removal of agents, and attribute modification,
- support for mobile agents,
- asynchronous notification of state changes to members and other interested agents,
- security infrastructure supporting authentication and authorization,
- robust operation in high latency, loosely connected topologies where hosts come and go, temporary network partitions are routine, and agents move, die, are resurrected, and temporarily duplicated.

These characteristics constrain designs and implementations of a community infrastructure. The need to avoid a single point of failure and scalability limits both suggest a distributed implementation of a community infrastructure, where agents possess enough information locally to maintain critical operations during periods when portions of the system are unavailable as a result of both planned and unplanned disconnects. The realities of dynamic network connectivity forces a community infrastructure to handle a variety of connectivity issues including network partitioning. When network partitioning occurs it is possible that multiple instances of the same community can be created. When the network is healed, the infrastructure must handle reintegration and merging of these divergent snapshots of a community into a single, coherent image. In a real-world environment, where the system is just trying to do the best it can with the available resources, the only guarantees that a community infrastructure can make are that it will always provide the best information that is locally available, and that it will eventually recover from all perturbations.

These characteristics of a community infrastructure impose responsibilities on community clients. Clients must:

- tolerate stale data, and
- gracefully incorporate newer data as it becomes available

The combination of an agent community infrastructure and suitable clients will provide a powerful mechanism for managing the growth, change, and dynamics of large-scale agent systems.

The remainder of this document is organized as follows:

- Section 2 provides a top-level overview of the Cougaar multi-agent system,
- Section 3 describes the agent community infrastructure available in Cougaar,
- Sections 4 and 5 discuss the benefits of group abstractions for messaging and organization modeling,
- Section 6 describes a distributed event capability for agent synchronization and change notifications,
- Sections 7 and 8 discuss robustness and scalability considerations for an agent group capability.

2. Cougaar overview

Cougaar is a Java-based architecture for the construction of large-scale dist ributed agent-based applications. It is an Open Source product of a multi-year DARPA research project exploring large-scale, scalable, heterogeneous, distributed and survivable Multi-Agent Systems. Current work is focused on supporting high-quality robustness, security and scalability mechanisms in the infrastructure in the context of a large military logistics system. This section provides a brief overview of the Cougaar architecture

This work is supported by DARPA through Department of the Interior contract #NBCHC10004. Approved for Public Release, Distribution Unlimited.

summarized from the Cougaar Architecture Document [1]. Detailed information can be found in the Cougaar Developers Guide [2].

The *Agent* is the principal element in the Cougaar architecture. An agent typically models a particular organization, business process or algorithm. An agent consists of two primary components, a *blackboard* and *plugins*. The plugins are software components that provide behaviors and business logic to the agent.

Cougaar provides two separate mechanisms for component-to-component communication. Intra-agent communication is performed via a blackboard that is shared by all agent plugins. The blackboard supports predicate-based publish-subscribe semantics. Plugins publish objects to the blackboard to make them visible to other local components. Plugins may also have any number of subscriptions that enable them to receive objects published by other components. Inter-agent communication is performed by a Message Transport Subsystem (MTS) that sends messages between agents using a reliable point-to-point messaging protocol.

A Cougaar *Society* is a collection of agents that interact to collectively solve a problem or class of problems. A Cougaar *Community* is a notional concept referring to a group of agents with some common purpose or organizational relationship. Within a Cougaar Society, agents execute on a *node,* which is a single Java Virtual Machine that can contain multiple agents. All agents on the same node share the same processor, memory pool, disk, and communication channels. The allocation of agents to nodes is not necessarily domain related, but rather based on a distribution of agents to available resources.

3. Cougaar community infrastructure

A primary goal in the design of Cougaar is to support the development of very large distributed systems consisting of hundreds or thousands of agents. In a system such as Cougaar which is intended to provide near unlimited scalability it is useful to organize agents into peer groups that can efficiently perform localized functions. Cougaar communities provide a way of organizing societies into distinct groups of agents. A community represents a collection of agents and possibly other communities. A common use of communities is to define a destination for broadcasting messages to multiple agents. To support this use, communities and their member entities may also be associated with attributes that can be used to perform queries to select a specific member or subgroup.

3.1. Community composition

A Cougaar community consists of one or more Entity objects as depicted in the class diagram in Figure 1. An Entity (community member) may be an agent or another community. Each Entity is constructed from 2 basic components, an identifier and optional attributes consisting of JNDI-based [3] name-value-pairs that can be used to define characteristics such as "Role". These attributes provide the foundation for a flexible query and abstract addressing mechanism that is tightly integrated with the Cougaar blackboard and inter-agent messaging infrastructure.

A number of predefined communities will typically be defined for a society. These communities are specified in static configuration files that define an agent's bootstrap communities. The use of predefined communities is transparent to agents as the community infrastructure automatically creates the required communities and adds agents as they are started. Ad-hoc communities are dynamically created at runtime. The creation of an ad-hoc community requires an explicit action by an agent.

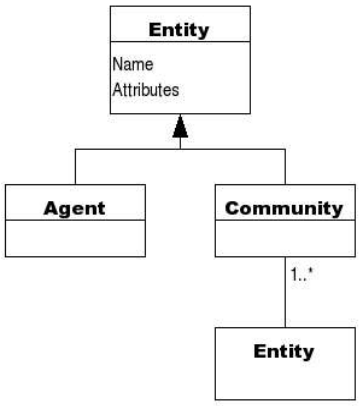

Figure 1. Community elements

3.2. Community management

A single manager agent is designated for each community. When ad-hoc communities are dynamically created, the agent creating the community automatically assumes the manager role in addition to other functions it is already performing. For predefined communities a manager may either be explicitly identified in the community configuration or may be arbitrarily selected from the community membership. The community manager is responsible for:

- *State management* - maintains a master copy of community state.
- *Access control* - controls community membership and attribute modifications.
- *State/event dissemination* - sends asynchronous state updates to members and other interested agents.

A community that does not exercise access control is said to be an *open community*. In an open community virtually any agent is able to join or otherwise modify community state without restriction. Closed communities, on the other hand, exert varying levels of access control. A typical level of control involves the use of an access control list to limit membership to a predefined set of agents. Other types of control could be used to restrict the ability of agents to modify Entity attributes. The Cougaar community infrastructure supports the use of pluggable modules to provide customized access control on a community-by-community basis. When a community is created the creating agent is able specify an access controller that is used to perform request authorization.

3.3. Community discovery

A key challenge in large distributed systems involves the development of a robust and scalable mechanism for locating entities. Cougaar provides a distributed *white pages service* that is used to map agent names to addresses, similar in function to the host name to IP address resolution performed by DNS. When communities are created the community name is bound to the address of its community manager in the white pages, along with a type designation (e.g., "community"). If a community name is known by an agent, the white pages provide an efficient way to determine whether the community exists and to retrieve the name/address of the associated community manager. The white pages are used solely for resolving names to agent addresses, no other state information is maintained.

The community and white pages services are optimized for operations in which agents know the name of a target community. In the typical case where an agent simply wants to join a community for which it knows the community name, an efficient white pages lookup is performed to get the agent address bound to the community name. This address is then used for a join request that is sent directly to the manager agent.

However, in a situation where an agent wants to join or otherwise interact with a community based on that community's current membership or attributes, community discovery will involve a white pages search and multiple "get community descriptor" requests, some of which may be performed on remote agents.

4. Abstract communication

Developers of agent-based applications should not need to be concerned with the communications required to support their application. While they may be aware that publishing a task by one agent initiates communication with another, they should not be required to explicitly define the mechanics of this interaction. Cougaar has two features that address this issue: Relays and Attribute-Based Addresses (ABA). Relays provide a general mechanism for the blackboard objects of one agent to have manifestations on the blackboard of other agents. An ABA allows messages to be sent to agents based on attributes of the agents rather than explicit addresses. These features are independent but often used together.

A Relay is essentially an object wrapper that can be published to an agent's local blackboard and then automatically forwarded to the blackboard of multiple remote agents. While the publisher of a Relay may explicitly identify the recipients, the Relay is often created with an ABA that targets all members of a named community. When a community-based Relay is published, each agent that is a member of the specified community will receive a copy of the Relay payload on its local blackboard. Furthermore, each agent that later joins or leaves the community will have a copy of the payload added/removed from its blackboard until such time as the publisher removes the Relay source.

The ability of an agent to communicate with other agents without knowing their identity is a powerful concept that provides many benefits in large multi-agent systems with mobile agents. In a paper outlining reliability requirements for agent systems [4], the authors describe a *mobile process group* as a feature that is fundamental to the development of reliable systems containing mobile agents. A mobile process group is defined as an entity to which an application process refers without knowing the number and location of the members which form it. Mobile agents are capable of autonomously migrating through a network. A key aspect of the mobile process group concept is that the groups are dynamic; agents are able to join and leave groups based on application defined criteria. The combination of the Cougaar Relay, ABA and community infrastructure provides an abstract messaging capability similar to that described for mobile process groups. In Cougaar, agents are able to send messages to other agents based on their community membership or specific community-based attributes.

Although blackboard Relays are often used to multicast an object to all members of a community, the ABA that is used to specify the destination community can be constrained such that a Relay is only sent to community members containing specific attributes. As an example, consider an application where an administrative community is been defined for a group of agents. This community contains one or more management agents that are responsible for monitoring the health of other agents in the community and restarting any agents that become unresponsive. These monitoring agents are identified by the attribute "Role=HealthMonitor". If an agent is unable to contact a member of this community (Agent-A) it could send a health alert message to a health monitor agent in this community. However, a problem arises in that the agent does not know the identity of these monitoring agents and is unable to send the message using direct addressing. In

this case, the use of a community-based ABA/Relay provides a good solution. The agent would simply need to wrap the health alert in an Relay with an ABA and publish it to its local blackboard. The ABA would contain an abstract address that targets "*all agents with the attribute Role=HealthMonitor in a community with the attribute Type=Robustness and containing Agent-A as a member*".

5. Role-based organizational modeling

In addition to support for abstract communication, Cougaar communities and attribute based addressing provides the capability to construct large multi-agent societies using organizational concepts such as groups and roles. AALAADIN [5] is an organizational meta-model for structuring multi-agent systems using three basic constructs - groups, agents and roles. The model is depicted in Figure 2. The use of organization constructs enables the construction of large-scale loosely coupled systems. In this model, no constraints are placed on the internal architecture of agents. An agent is only specified as a communicating entity which plays roles within groups. A group then is an aggregation of agents and a role is an abstract representation of an agent function, service, or identification within a group. A single agent can handle multiple roles, and each role performed by an agent is local to a group.

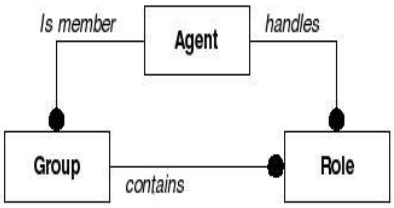

Figure 2. Role-based organization model

The AALAADIN constructs directly correspond to Cougaar's use of communities, agents and attributes. Using an organization model such as this, complex Cougaar societies can be defined using communities, agents, and roles (specified using entity attributes).

6. Distributed events

A sophisticated event framework is described by Baumann and Radouniklis in their agent group model [6]. They describe the use of event channels that can be used to disseminate change notifications and to also trigger arbitrary actions by group members. A similar capability can be achieved in Cougaar using communities and the inter-agent publish-subscribe semantics provided by Relays.

The community infrastructure provides fine grained asynchronous events to interested agents when changes occur in community state. Components wishing to be notified of community changes can simply register with their local community service to receive callbacks when changes in community state occur. Support for distributed change events is an important scalability feature, as it eliminates polling by agents for the purpose of detecting changes in community membership or attributes. However, implementing a distributed event mechanism presents scalability challenges of its own. The primary consideration involves minimizing network traffic without incurring an unacceptable level of notification latency. Figure 3 illustrates the approach used the Cougaar community infrastructure to disseminate change events to distributed agents. Scalability is addressed in two ways. First, the community manager exercises flow control in its dissemination of community updates. The rate at which status updates are transmitted is throttled to minimize network traffic during periods of high activity, and all changes generated between updates are aggregated into a single message. The other technique which is used to minimize messaging is to send updates to nodes rather than individual agents. The cache managers at the nodes are then responsible for identifying the changes since the last update and firing any required change events to local agents.

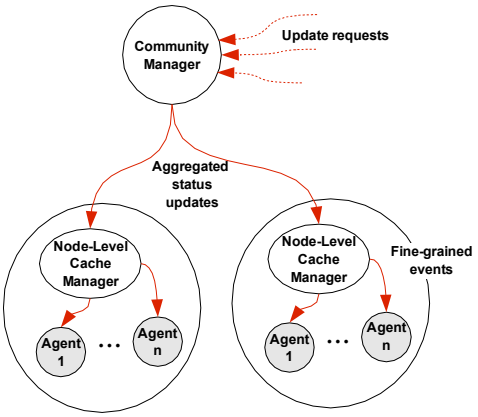

Figure 3. Distributed Events

In addition to the community-based change events, a generic event channel can be constructed using communities, blackboard relays and blackboard subscriptions. This technique involves the use of a

community to identify the agents that are interested in receiving a particular type of event. These agents will join this event channel community and create a local subscription to an event object which is an instance of a blackboard Relay. The generator of these events will locally publish the blackboard Relay which is then automatically transmitted to the blackboard of all agents in the event channel community. Any agent plugin which has subscribed to the event Relay will be asynchronously notified when the event is created or changed.

7. Fault tolerance

Cougaar is designed to be resilient to chaotic operating environments in which host computers and network resources are stressed and often fail for varying periods of time. The Cougaar philosophy is that the system must be able to maintain operations during these periods as best it can using available data. Once the failures have been corrected the system must then be capable of resolving any inconsistencies that may have been introduced.

7.1. Agent failures

Cougaar communities are tolerant to failures involving both member and manager agents. An agent failure is defined as the abnormal termination of an agent due to a failure in an underlying computing resource. In these situations it is assumed that the agent will eventually be restarted on the repaired resource or at some alternate location. Since community state data is cached at each node, surviving agents in the community are able to continue community-dependent processing even if a community manager is temporarily inaccessible.

The recovery of a regular (non-manager) agent is relatively straightforward. If a lost agent is restarted prior to the expiration of its membership lease, it simply needs to retrieve an updated community descriptor from its node-local cache or community manager. If, on the other hand, the membership lease has expired, the agent must re-join its community by submitting a request to the appropriate community manager. In both of these cases, it is important that each member agent retain some memory of the communities that they had joined prior to their unexpected termination. This is accomplished by publishing community membership state to its blackboard which is periodically persisted to non-volatile memory and subsequently restored during an agent restart.

The recovery of a community manager involves the reconstruction of the master copy of the community state. Since this information is published to the manager's blackboard, it will normally be recovered from the most recent persistence snapshot when the manager is restarted. However additional precautions must be made for the recovery of this data in the event of loss or corruption of the persistence data, otherwise this loss would constitute a single point of failure. In a situation where a restarted manager cannot recover a persisted community descriptor, the periodic cache updates sent to member agents will cease. A resulting cache update timeout will trigger the member agents to resubmit join or attribute modification requests to the manager enabling it to reconstruct the global community state.

7.2. Network partitioning

The availability of locally cached community data minimizes the effects of a partitioned network relative to existing communities. In these situations there is only a minor impact if the partitioning separates members from their manager or other members for an extended period. In these cases the membership leases will expire requiring the members to re-join the community when the partitioned segments are reconnected.

The most significant issue involving a partitioned network is encountered when the same community is created within multiple partitions. This type of partitioning can commonly occur when using predefined startup communities without designating a manager agent. During a cold start in which a multi-node community is bootstrapped, agents often won't find a current manager and will then assert that role resulting in the creation of multiple communities with the same identity. When the network is reconnected the presence of multiple communities with the same identity must be resolved. The resolution is performed by selecting a single manager and merging of all members into a combined community.

8. Scalability

The Cougaar architecture is designed from the bottom up to support large scale applications . A key element in achieving high level scalability is to limit the information that is passed between agents and to avoid the use of centralized services and data stores. The community infrastructure adheres to this design philosophy. Each agent caches community state locally, minimizing the need to perform remote operations, and updates from the community manager are batched to reduce network traffic. The primary scalability consideration in the use of communities involves the generation of a list of all communities and use of global searches based on community attributes. Generating a list of communities involves a sequential search of white pages bindings. Constraining this list based on membership or attributes will require a remote fetch for each community that is not cached locally. For instance, a search for all communities containing a specific attribute requires that a list of all communities be obtained from the white pages. Then, for each community that is not locally cached, a remote “get

community" operation must be performed to obtain a copy of its state. Since community information is cached the bulk of this penalty is incurred with the initial search.

This community infrastructure has been successfully utilized in a large-scale logistics application and has demonstrated support for communities in excess of 1000 agents deployed across approximately 100 host computers.

9. Summary

In this paper we've identified some key characteristics of an agent group construct for use in robust highly-scalable multi-agent systems. We reviewed the mechanisms employed in the Cougaar multi-agent system to provide support for a construct satisfying these characteristics. These mechanisms include a flexible community infrastructure, attribute-based addressing, and a distributed publish-subscribe protocol.

10. Acknowledgments

The work described here was sponsored by the DARPA UltraLog contract #NBCHC10004. These ideas represent contributions made by Qing Lin and many other individuals who have participated in the DARPA ALP and UltraLog programs.

11. References

[1] BBN Technologies, *Cougaar Architecture Document V11.0*, 8 March 2004, http://www.cougaar.org/

[2] BBN Technologies, *Cougaar Developers Guide V11.0*, 8 March 2004, http://www.cougaar.org/

[3] R. Lee and S. Seligman, *JNDI Tutorial and Reference*, Addison-Wesley, 2000

[4] F. M, de Assis Silva and R. J. de Araujo Macedo, "Reliability Requirements in Mobile Agent Systems", *Proceedings of the Second Workshop on Tests and Fault Tolerance*, Brazil, 2000

[5] J. Ferber and O. Gutknecht, "A Meta-Model for the Analysis and Design of Organizations in Multi-Agent Systems", *Proceedings of the 3rd International Conference on Multi Agent Systems*, Washington, 1998, p. 128

[6] J. Baumann and N. Radouniklis, "Agent Groups for Mobile Agent Systems", *Distributed Applications and Interoperable Systems*, London, 1997, pp. 74 – 85.