SAND99-1500C

DRAFT

Agent Communications using Distributed Metaobjects

Steven Y. Goldsmith
Shannon V. Spires
Advanced Information Systems Laboratory
MS 0455
Sandia National Laboratories
Albuquerque, NM 87185
505-845-8926
sygolds@sandia.gov, svspire@sandia.gov

Abstract

There are currently two proposed standards for agent communication languages, namely, KQML (Finin, Lobrou, and Mayfield 1994) and the FIPA ACL. Neither standard has yet achieved primacy, and neither has been evaluated extensively in an open environment such as the Internet. It seems prudent therefore to design a general-purpose agent communications facilitiy for new agent architectures that is flexible yet provides an architecture that accepts many different specializations. In this paper we exhibit the salient features of an agent communications architecture based on distributed metaobjects. This architecture captures design commitments at a metaobject level, leaving the base-level design and implementation up to the agent developer. The scope of the metamodel is broad enough to accomodate many different communication protocols, interaction protocols, and knowledge sharing regimes through extensions to the metaobject framework. We conclude that with a powerful distributed object substrate that supports metaobject communications, a general framework can be developed that will effectively enable different approaches to agent communications in the same agent system. We have implemented a KQML-based communications protocol and have several special-purpose interaction protocols under development.

Keywords: agent communication language, multiagent system, metaclass, metaobject protocol, distributed objects

1 Introduction

Communication among autonomous asynchronous agents is an essential function in network-based multiagent systems. There are currently two proposed standards for agent communication languages, namely, KQML (Finin, Lobrou, and Mayfield 1994) and the FIPA ACL. Until a standard emerges, an agent designer must accomodate this uncertainty in agent designs. Our design philosophy is to develop a general object-centered framework that enables programming of multiple protocols for communication and interaction. Figure 1 shows the general architecture for agent communication, discussed in detail in subsequent sections. The components are: (1) the send-object protocol that provides a standard interface for remote communication of objects; (2) a message object protocol that interprets the structure of the message object, enabling multiple communication protocols (e.g. KQML, ACL); (3) a metamodel that manages the update of remote agent models and the local agent's model; and (4) the model of local agent and models of remote agents. The framework includes an infrastructure for agent modeling because communication among two agents requires both a common message format and a shared ontology. Since agents may be in different states, communications is mediated through the receiver's model to ensure common semantics. The agent's self-model contains the deliberative mechanisms and knowledge bases that are exclusive to itself. The self-model has control over the operations of the remote agent models

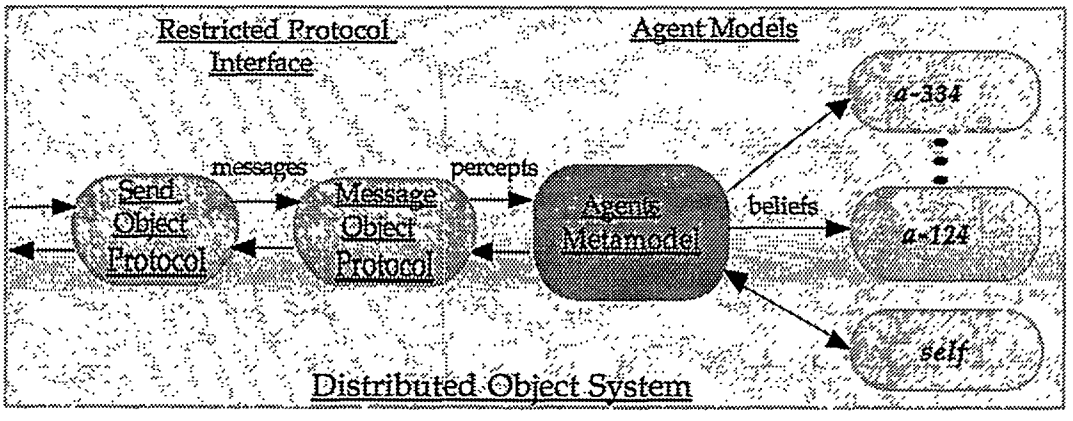

Figure 1. Distributed Object Agent Communications Architecture

through the metamodel. We assume that the agents communicate both the structure and the state of their models to one another for the purpose of collaboration.

The entire architecture is based on the object framework concept The classes and methods comprising the architecture are designed to be specialized with subclasses and methods that implement the agent designer's favorite communication, interaction, reasoning and representation mechanisms. Our objective is to provide both a research tool for evaluating new regimes and a practical system capable of operating in heterogenous environments such as the Internet.

2 Distributed Objects

Our approach to the design and implementation of network agents relies heavily on a comprehensive distributed object subsystem implemented in the Common LISP Object System(CLOS). Agent designs involve compositions of objects and metaobjects, many of which are intrinsically capable of distribution in a network environment. Communicating among agents that are described as compositional objects has a natural interpretation; it is an instance of message passing among objects and as such has a well understood syntax and semantics. A distributed object is an object that has a commonly-known identity

DISCLAIMER

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

and is represented by some form of surrogate object in multiple address spaces around the network. Distributed object surrogates are of three primary types: proxies, copies, and replicants. (There can also be a fourth, hybrid type which combines features of the main three.)

Proxies are pure surrogates. A proxy object "stands in" for a real object that is located elsewhere. The proxy accepts messages destined for its "real" object, delegates them to the real object for processing, receives the result of the message, and passes the result along to the original message sender. Proxies are very handy for projecting an object's capabilities from its current location to other places on a network. They are immune to update issues, since any change to a real object will be immediately reflected in the responses of all of its proxies. Proxies are the primary object distribution mechanism of CORBA [ref http://www.omg.org]. The downside of proxies, of course, is that every message sent to a proxy invokes a network transaction.

Copies are just that; an object is copied and sent from one network location to another. Pure copies keep no information about their "source" object (and vice-versa) so they cannot be updated if the source object changes. But of course, if the data and functionality contained in the copy is needed frequently at another location, this may be an acceptable price to pay to avoid the network overhead of a proxy.

Replicants are copies that keep track of their source (and/or vice-versa) such that they can be updated if their source object changes. Replicants thus provide the best features of both proxies and copies: information currency with low network overhead, as long as accesses are more frequent than updates and we are willing to pay the price of more bookkeeping.

Hybrid objects can exhibit proxy, copy, or replicant behavior on a slot-by-slot basis. Hybrids are probably the most useful form of object distribution in general because the distribution mechanism choice can be made at a fine level of granularity.

In our discussion of copies and replicants above, we omitted one nasty detail: objects in a modern inheritance-based dynamic OO system [in which class and method meta data exist at runtime] never exist alone. Objects themselves are but the tips of two massive icebergs: an inheritance graph and a containment graph. In order to truly copy an object from Point A to Point B on a network, we must also copy its inheritance graph—its class, and its class's superclasses, and methods thereof—and we must also somehow distribute any objects it references or contains. In an OO system like C++ where classes are not first-class objects, this can only be done if the requisite classes and methods already exist on the destination machine. But in an OO system like CLOS where classes and methods are first-class objects, we can treat the classes and methods themselves as merely more objects to be copied and copy them on-demand, using the same mechanism we use to copy pure instances. It is the classes and methods that we refer to with the term *metaobjects*.

Distribution by proxy is popular in the distributed object community because it is immune to update problems and it does not require that classes or methods be present at the target node; it is fundamentally based on delegating messages to a remote "real" object. But as we've already noted, the performance penalty for such delegation can be large and sometimes must be avoided. Therefore distribution by transporting whole copies of objects is essential, especially when moving an agent on a network or sharing ontologies among fixed agents. But copying objects also requires copying class lattices (distributing class lattices by proxying them usually won't work) and methods. And even if the objects we move are pure copies (no updating expected), we must usually transport their class lattices and methods as *replicants*, not pure copies, because if a class definition or method changes, the changes must be promulgated. This is why most distributed object systems either make no attempt to copy or replicate objects or do so in only a limited fashion. Solving the replicant problem in general is quite difficult, especially in static OO languages. It gets even worse: in CLOS, classes themselves are instances of metaclass objects. If any transported class is an instance of a special metaclass, the metaclass must be transported also. Fortunately, the replication problem is soluble in CLOS because of

its extensive introspective capabilities and its metaobject protocol.

The actual movement of a CLOS object takes place in two stages: serialization and materialization. To serialize an object means to flatten it into a sequence of bytes that can be used to reconstruct the object at another place. In CLOS, the essential information that must be serialized is the object's class name and its slot contents. Serializing an object is relatively straightforward, provided we are careful to maintain referential integrity among slot contents, and to recursively serialize any other objects that may be referenced in its slots. Once a sequence of bytes is produced, it is transmitted over the network to the receiver.

At the receiver, materialization begins. The receiver looks at the class name of the incoming object and checks to see if that class is present locally. If not, it asks the sender to serialize and transmit the class metaobject. (When the class metaobject is materialized at the receiver, the receiver will check to see that all its superclasses and metaclasses are also present and may recursively request their transmission as well.) If the class is already present at the receiver, the receiver may check its timestamp, hashcode, or some other version-maintenance identifier to ensure that it has the latest version. If not, it may request that the sender transmit the latest version of the class. Methods and generic functions are also transmitted or updated along with the class metaobjects that specialize them. Finally, once the receiver is satisfied that the object's requisite infrastructure is present, it simply allocates space for an object of the appropriate class and fills in its slots with the original serialized data.

The above is the standard "pull" mechanism for demanding an object's infrastructure when the object is pushed. Objects that are replicated, not merely copied, can also be updated on a "push" basis by the sender when necessary.

Proxies are still very useful in many cases and can be implemented in CLOS much more dynamically than in CORBA: no a priori knowledge of allowed messages is needed. Any message sent to a proxy that the proxy does not immediately understand can be automatically delegated to the proxy's "real" counterpart by overriding the CLOS no-applicable-method mechanism. New messages can thus be created on-the-fly for real objects and any proxies to those real objects can immediately take advantage of them.

We have demonstrated that there is no inherent barrier to providing copies, replicants, and proxies as distribution mechanisms for objects and metaobjects. Nevertheless, the reader will have noted we have said nothing yet about the security implications of such wide-open distribution. Even though our basic mechanism is quite general, it is usually necessary to impose some limitations on its power because of security considerations.

Our distributed object substrate provides a general purpose communications mechanism capable of implementing many different agent communication systems, including KQML. However, most standard distributed object systems are not powerful enough to implement the features needed to provide security, shared knowledge/ontologies and agent modeling.

3 Autonomy, Integrity and Communication

Autonomy is a cornerstone in the modern specification of intelligent agents. Roughly speaking, autonomy implies an agent acts without the direct intervention of humans or others, and have some king of control over their actions and internal state (Castelfranchi 1995). We define autonomy in an agent by the following statements:

1. An agent is a locus of unique identity
2. An agent is a locus of self-control

An autonomous agent will be self-determined with respect to its beliefs, goals and actions. It will be known to other agents by a unique name that identifies it as an independent entity within the agent community. In a multiagent collaborative system, agents rely on the autonomy of one another to make certain inferences about the motives,beliefs, goals and actions of other agents. Casterfranchi (1995) identifies two distinct classes of autonomy: stimulus autonomy and executive autonomy. A message from another agent qualifies as a stimulus to the receiver. An agent may choose to response to a stimulus or not, depending on the current state of the agent's deliberations. Executive autonomy requires that an agent cannot be directly motivated with the goals of another agent unless the agent decides that the goals are congruent with its own. Under no circumstances should an agent attempt to satisfy a goal object obtained directly from another agent without first evaluating and criticizing the goal within it own deliberation mechanism.

Implementing stimulus autonomy and executive autonomy requires designing a safe communications protocol that maintains the integrity of the agent while allowing effective communication. We propose that the functional property of *agent integrity* is a necessary element for agent autonomy. Integrity is an operational concept that seeks to protect the agent's internal structures from direct manipulation by another agent, including human actors. An agent cannot be self-determined or self-controlled unless it is impossible for others to directly influence its beliefs and actions unbeknownst to the agent. Distributed object protocols introduce vulnerabilities that undermine agent integrity. Like the Nefarious Neurosurgeon of Dennett (1984) who introduces electrodes into the brain of the victim Jones and controls his every thought, an agent that can dispatch an arbitrary method invocation to the address space of another agent is capable of direct intervention in the agent's activities. Agents operating within a multiagent system that does not restrict the remote method invocation process cannot believe in a distinct locus of identity and control for one another, since impersonation by a nefarious agent of a zombie surrogate is a possibility. Integrity mechanisms force RMI to implement a restricted protocol that cannot address arbitrary objects and methods within an agent program.

Simple impersonation through active attacks on the communications links is prevented by cryptographic authentication and encryption protocols applied at the transport layer.

4 Object Communication

A careful look at the life cycle of a single agent-to-agent message, i.e. the simplest an instance of agent communication, reveals that messaging involves the most fundamental actions of an agent. Messaging is a deliberate, motivated action, designed to achieve a specific goal. In the speech act interpretation of a message, the agent desires to entrain a specific mental state in the receiver . Figure 2 represents the sequence of events leading to transmission.

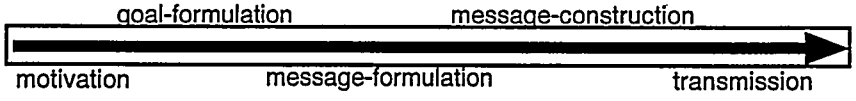

Figure 2. Events Leading to Transmission

<u>motivation</u>
The motivation for transmission is generally derived from some higher goal of the agent. Fundamentally, the agent must inform another agent or obtain information from another agent, obviously in a social setting with other known agents.

goal-formulation
The agent creates a goal object that encapsulates the details of the communication act. Satisfaction of the goal is complete when the object has been successfully transmitted.

message-formulation
The actual message is formulated with a sender, receiver, and content object. Depending on the communication protocol employed (e.g. KQML), additional information may be added. The exact formulation is compatible with the communication protocol employed by the receiver object.

message construction
A specific class of message object is constructed for transmission as copied distributed object. The copied object will be transmitted directly to the receiver.

transmission
The message object is transmitted to the receiver.

The Send-Object protocol (Figure 1) implements transmission of a message object. Each agent is registered in the network with a well-known proxy object. An agent holds the proxy to another agent in the agent model (discussed below).

Send-object(agent-proxy, message-object)

The Send-Object method (Figure 1) implements transmission of a message object. Each agent is registered in the network with a well-known proxy object. An agent holds the proxy to another agent in the agent model (discussed below). The send-object method is invoked in the target agent's environment through remote delegation via the proxy. The invocation is restricted to a specific namespace in the target agent that contains the agent proxy and proxy class, the send-object metaobject, the classes of possible message objects, and filtering functions to evaluate the message and its content. The distributed object system checks the serialized message for references to other namespaces and rejects the message it contains other references. Thus the send-object protocol is a virtual chokepoint for messages, preventing direct invocation of methods on objects outside the restricted namespace. We call this element the Restricted Protocol Interface.

Receiving an object from another agent is also a deliberate act on the part of the receiver. It requires the necessary motivation and goal creation to create the context for evaluating the communicated object. In general, an agent must associate the communicated beliefs with persistent goals to determine their salience and to formulate the proper actions in response.

motivation
The motivation for reception is derived from a normative persistent goal provided by the framework that creates within the agent the desire to receive information from other agents.

goal-formulation
The agent creates a goal object that determines which agents will be considered for interaction. The goal is mutable, and agents may be removed from consideration for a variety of reasons, including security, chronic poor performance on collaborative tasks, and prioritization under severe resource constraints.

message-expectation

Certain messages or classes of messages may be expected, perhaps in response to a previous transmission in the context of a conversation. The framework enables a message object to directly invoke a specific achievement goal in the self-model that has been deferred pending more information. An expectation mechanism within the Message-Object element (Figure 1) can directly determine the context for message processing through a reference to the context goal. This provides a mechanism for implementation continuous conversations between agents.

message deconstruction
Each message must be deconstructed according to its class. For example, a KQML message will be reduced to its component fields and the salient objects extracted by the Object-Message protocol. The components representing the percepts are then passed to the metamodel for processing.

reception
The new beliefs are presented to the self-model and updates the remote agent model.

elaboration
A deliberation mechanism within the receiving agent is activated to determine the ramifications of the new beliefs with respect to the agent's goals.

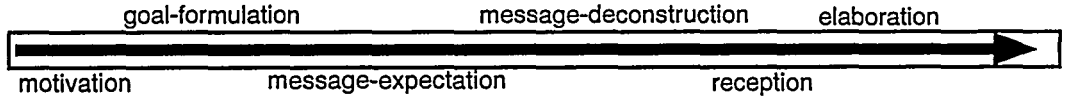

Figure 3. Events Leading to Reception

The architecture provides the source of motivation for social interaction among agents. The framework provides classes and and method metaobjects that enable the construction of sending and listening goals.

5 Agent Models

Agents have local beliefs about other agents and the world. In order to distinguish its local beliefs from those of other agents, each agent has a distinct model of itself and distinct models of other agents. The object constant self denotes the local agent and constants of the form a-1, a-2 , and a-100 denote the other agents in the environment. Models of other agents allow the local agent to reason about the beliefs, goals, and actions of others. The Agents Metamodel (Figure 1) manages the update of an agent's models from communicated information. The communications protocol passes message objects to the Agent Metamodel (Fig 1) for elaboration and interpretation. The metamodel makes certain inferences about the beliefs of the local agent and other agents based on communicated messages. First, the receiving agent must be able to recognize the sender agent as the true source of a message. Each agent in the system has a unique and verifiable identity. Cryptographic authentication of each message by digital signature enables the receiver to attribute the message to the identified sender with certainty. Although the exact operation of the metamodel depends on the particular representation of belief, the following logical model based on deductive belief (Konolige 1984) illustrates the point. The predicate message(y,x,z) denotes a message with content object x sent from agent y signed with digital signature z. The metamodel computes the signature using the digital signature function, reified as a trinary relation dsa(x,y,v), where x is the message, y is the agent id (used to obtain the public key) and z is the computed digital signature. Note that this digital signature scheme is distinct from the authentication protocols used at the transport level. Agents require a different signature scheme to authenticate their identity to one another at the knowledge level. Certain collaborative activities may require more

specialized signature schemes still. Protocols for encryption and authentication at the link level may be constrained by the network and transport layer underlying the communications system.

Validation of the digital signature sanctions the belief by the local agent in the belief of the sender via the schema:

$$\mathbf{message}(\text{a-123}, x, z) \cap \mathbf{dsa}(x,\text{a-123},v) \cap \mathbf{eq}(z,v) \rightarrow \mathbf{Bel}(\textit{self}, \mathbf{Bel}(\text{a-123}, x))$$

Bel(*self* , **Bel**(a-123, x)) is asserted in the local (self) model of the agent, while the argument **Bel**(a-123, x) is asserted in the model of agent a-123. Alternatively, an invalid message is not believed by the local agent[1] :

$$\mathbf{message}(\text{a-123}, x, z) \cap \mathbf{dsa}(x,\text{a-123},v) \cap \neg\mathbf{eq}(z,v) \rightarrow \neg\mathbf{Bel}(\textit{self}, \mathbf{Bel}(\text{a-123}, x))$$

The conclusion **Bel**(c, x), where c is an arbitrary constant, is asserted in the model corresponding to the "unknown agent". This captures the notion "somebody believes x".

Control of an agent's models of other agents is mediated through the metamodel. The local agent may wish to check an agent's model for consistent beliefs. The metamodel provides a uniform protocol to the local agent for performing queries, proving assertions, and importing hypothetical beliefs from a model into its self-model.

Each model of a remote agent comprises a distinct namespace, a set of metaobjects (classes and methods) that implement the interface to the metamodel, and a separate thread to control execution of methods. At the framework level, instances and metaobjects transmitted by the actual remote agent are represented as simple beliefs of the form **Bel**(a, x), where a is the agent name and x is any object or metaobject. This captures the primitive notion that an agent believes in the existence of the referenced object or metaobject. Included are complex compositions of objects implementing part-whole relationships. Compositions are handled naturally by the underlying distributed object system by coercing the message content object and all its components into copied objects during materialization. The framework is easily specialized for a particular representation. Candidates include categorical taxonomies such as description logics (e.g. CLASSIC, LOOM, KL-ONE), KIF(Finin, Labrou, and Mayfield 1994), first-order logic and theorem provers, deductive data bases, BDI architectures, and so on. Custom representations rendered in the object language are also possible. These different representations may be active simultaneously in different agent models provided the necessary interface protocol to the metamodel exists.

5 Shared Ontologies

Direct communication of metaobjects between agents enables agents to share their models of one another and the environment. An agent decides which elements of its representation and in what representational scheme will be used by other agents to model its reasoning and behavior. Through an interaction protocol, agents can negotiate detailed descriptions of their shared models, enabling cooperation on joint tasks. The framework supports this in two ways. First, every model is ultimately rendered in CLOS through metaobjects and instances, providing a common programming language with which the agents remotely but safely program their corresponding models residing in other agents. This in effect creates an endosymbiont within the local agent representing a special projection of the remote agent without degrading the integrity of the local agent. Secondly, a model of another agent is a dynamic process under the control of the local agent. The local agent can use the model to predict the behavior of a remote agent, to the extent that model allows. This enables a powerful simulation mechanism within an agent that facilitates cooperative actions.

[1] The metamodel will attempt to validate the message for all agents in its knowledge base. If this fails, the message is invalid. If it succeeds, the valid agent id is substituted in the message.

6 Conclusions

We have described a general architecture that ensures agent integrity, supports agent modeling, and enables multiple representations and communications protocols to coexist in the same agent. The architecture is currently being implemented on a distributed system test bed comprising over 100 Windows NT and Linux platforms.

References

Castelfranchi,C. 1994. Guarantees for autonomy in cognitive agent architectures. In *Intelligent Agent I, ECAI Workshop on Agent Theories, Architectures and Languages*. Springer-Verlag.
Dennett, D. 1984. *Elbow Room*. MIT Press. Cambridge MA.

Finin, T., Labrou, Y. Mayfield, J. 1994. KQML as an agent communication language. Computer Science Department, University of Maryland Baltimore County.

Konolige, K. G. 1984. A deduction model of belief and its logics. Technical Note 326. Menlo Park, CA: SRI International, Artificial Intelligence Center.

Phillips, L.R., "CHI: A General Agent Communication Framework," Proc. of the Hawai'i International Conference on System Sciences, January, 1999

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under contract DE-AC04-94AL85000.