Discovery of Infrastructure in Multi-Agent Systems

Brent K Langley, Massimo Paolucci, Katia Sycara
The Robotics Institute,
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA - USA -
blangley,paolucci,katia@cs.cmu.edu

ABSTRACT

The role of the Multi-Agent System (MAS) Infrastructure is to support agents while they create more complex virtual communities. As agents dynamically appear and disappear from the environment (or move from one host or network to another) they may find that their infrastructure services have also dynamically become transient, creating potential disruptions to the activity of the MAS. Therefore agents should be provided with tools that allow them to dynamically reconfigure and adjust their operation, based on these changes. We describe the protocols, of a multicast-based discovery mechanism, that allow agents to rapidly track the changes in the infrastructure of their local networks, in order to maintain a consistently high level of interoperation with other agents. In addition, we show how the discovery protocols can be extended over the Internet through a Peer-to-Peer connectivity model based on the Gnutella architecture to create Agent-to-Agent^{TM}(A2A^{TM}) communities. The combination of local and global discovery allows agents to maintain survivable connectivity to both local infrastructure and to services dispersed throughout the Internet. The use of this service allows the agent systems to ultimately participate in more globally diverse and complex agents communities.

1. INTRODUCTION

The promise of Multiagent technology is for the automatic creation of virtual communities of heterogeneous agents that dynamically collaborate, compete, form teams or coalitions, and enter into auctions and negotiate on prices for services. Yet, all of this does not happen by magic. It is not enough to implement a set of agents, start them, and wait for them to spring into action and create a community. Rather, agents need a set of *infrastructure* services to support their interactions, as well as knowledge and conventions that allow them to understand and communicate with each other. This infrastructure provides white-page services that allow agents to locate each other; and yellow-page services (or middle agents [1, 12]) that identify which agents provide what services. Examples of middle agents include the OAA Facilitator [2], the RETSINA Matchmaker [6] and the Infosleuth Broker [3].

In order to take advantage of the infrastructure, agents must, first and foremost, identify the MAS infrastructure services that are currently available to them. In this paper, we show how agents find infrastructure, such as yellow and white page services in the RETSINA Multi-Agent System [8], using simple discovery protocols. In our schema, agents entering the system will transmit requests for desired infrastructure services that might be available in their local area network (LAN) or across the global network (WAN). Each service that matches the request will reply to make itself known, it is then up to the requesting agent to select with which services to register its own capabilities and location.

One aspect that is often unaddressed in MAS, is that infrastructure services, as well as agents, can appear and disappear dynamically. We suggest that as infrastructure services enter the MAS, that they utilize the discovery mechanisms to announce themselves so that the agents in the MAS can contact them. One immediate gain of this process is that agents do not need the infrastructure to be rigidly in place when they enter the system; rather, they can monitor the discovery messages that are transmitted to learn and adapt to their changing infrastructure landscape.

We see discovery as a procedure that enhances or replaces the hardcoding of infrastructure references in configuration files and agent code. While static configuration information may allow agents to connect to their infrastructure their use is problematic. First, it requires high-maintenance when the infrastructure is modified; in addition it proves to be very brittle, restricting the agents' ability to automatically recover from even partial failures of infrastructure components. Finally, the use of hardcoded references often mandates a specific order of system component initiation, because the existence of the MAS infrastructure often becomes a precondition of an agent's ability to function. Agents that use discovery are not hindered by these problems: they always know what alternative infrastructure components are currently available in the system, so they can switch at any time. In addition, if the desired type of infrastructure component is not currently available in the local network, the agent can look for a substitute on the global network or just wait until it is notified, through the discovery mechanism, that the desired service has become available on the local or wide area network.

Current protocols that implement discovery are often hindered by local network boundaries and Internet routing infrastructure. This is a serious limitation for MAS technologies that aspire to operate across the Internet. In this paper we show how agent discovery can utilize the fabric of a peer-to-peer (P2P) network, and augment the P2P protocol to guide and direct peer connectivity to form Agent-to-Agent (A2A) communities. Furthermore, we show how we've enhanceed this A2A architecture to maximize the probability for appropriate connectivity and to minimize extraneous traffic.

This paper is organized as follows: in section 2 we discuss the multicast-based discovery infrastructure used in the LAN; in section 3 we introduce the A2A paradigm, and show how it has been implemented and extends the discovery mechanism to operate over wide area networks, such as the Internet; /comment, utilizing a peer to peer architecture and, in section 4 we provide a qualitative evaluation of the different discovery protocols we have presented. Finally in section 5 we discuss some of the work we intend to do in the future to unify and expand the discovery mechanisms we have described, and then we conclude.

2. LOCAL DISCOVERY

Agents in the RETSINA system use Simple Service Discovery Protocol[5] SSDP, to find the infrastructure services for their environment. SSDP was created in conjunction with the Universal Plug-and-Play[11] (UPnP) initiative driven by numerous vendors including Microsoft Corp. It was developed as a lightweight mechanism to discover service providers in an ad-hoc and dynamic environment. Two modes of this protocol's operation support the notification (and alerting) of a service's availability, and the search for services.

SSDP specifies three types of messages to be multicast. *Alive* messages are sent out to communicate that a new service has come online, *Byebye* messages are sent to communicate that the service is leaving the system, and *Search requests* are multicast to look for services that are currently available. Responses are only sent to Search requests, and they are transmitted via unicast. All SSDP messages are composed with UDP (User Datagram Protocol) [10] packets which is more lightweight protocol than TCP; but because UDP implements fewer checks and balances, it carries no guarantee of delivery. The resulting discovery process is *asynchronous*, where messages are publicly accessible, and responses (if any) may be singular, multiple, or repetitive; and possibly time-delayed.

Upon entering the RETSINA MAS, every agent and infrastructure service multicasts an Alive message and a sequence of search requests for services they need. The Alive message, shown in figure 1, is a communication to all the other agents, of the existence of the new agent. The message specifies the type of message sent: using the NOTIFY header and the ssdp:Alive indicator; the type of agent that sent out the message (in this case a retsina:Agent); where the agent can be found (<kriton.cimds.ri.cmu.edu:5555> which is its unicast address and port); and finally, the unique name of the agent: "SECstock". Alive messages are cached in agents by the appropriate service's client module, as information for future points of contact for locating other agents. This information is only cached for a limited duration, since agents or the interconnecting network may fail. A value of 15 minutes is used in the example below (900 seconds.) At 75% of this lease interval, if the agent is still functioning it will automatically send a fresh Alive packet to extend their lease cache entry.

```
NOTIFY * HTTP/1.1
Host: 128.2.222.18:32786
NTS: ssdp:alive
NT: retsina:Agent
USN: uuid:SECstock
AL: <kriton.cimds.ri.cmu.edu:5555>
Cache-Control: max-age = 900
```

Figure 1: An Alive message from agent SECstock

In addition to Alive and Byebye messages, an agent multicasts one or more Search request messages to look for services that it needs. (Service requests can specify a specific service that the agent requires or can indicate that all available service providers should respond.) An example of an infrastructure service request is shown in figure 2. The agent with a unique name of SECstock, is seeking a service of type `retsina:AgentNameServer`, and has provided its "return address" `Host: 128.2.179.228:1082` to indicate where the response should be sent. This message is ignored by all agents and services whose type doesn't match the requested type (in this example, anything that is not an ANS.)

All ANSs in the MAS recognize that their service is being sought and decide whether to respond to the request. To avoid problems that might occur if numerous services simultaneously attempt to unicast responses to the same agent and briefly flood the network, responders will stagger their replies by choosing a random delay time within the boundaries of a number of seconds as indicated by the MX parameter specified in the search request message. Similar search messages are transmitted to look for Matchmakers, and for other infrastructure services like Loggers, MAS visualization tools, and Multi-Agent Launchers [8].

```
M-SEARCH * HTTP/1.1
S: uuid:SECstock
Host: 128.2.179.228:1082
Man: ssdp:discover
ST: retsina:AgentNameServer
MX: 2
```

Figure 2: A Search Request from agent SECstock

When an agent or infrastructure service receives a multicast message of interest, it will reply directly to the sender via a unicast response. As opposed to a multicast message that is transmitted to all the agents, a unicast message is a direct point-to-point communication. For example, several ANS servers may reply to an agent request by each providing their own network address so they may be contacted. No other system on the network will see the individual responses to the original requester. But they all had an opportunity to see the request.

Finally, Byebye messages are multicast when the agent leaves the system. Every agent that had previously learned location information about this service, can remove the reference from its cache.

Discovery is also used as an event-based trigger to facilitate higher level coordination and community interactions within the MAS infrastructure. For example an ANS multicasts an Alive message to announce its presence in the system, followed by a search request to locate other ANSs. Existing ANS servers learn of the new ANS from its Alive message, and the new ANS learns of the existing ANS servers from their responses to its query. Once they know about each other, they will exchange information and registration requests, increasing the reliability and efficiency of the overall MAS by ensuring redundancy of agent registration information, and by providing load balancing for each other. The transmission of Byebye messages from ANS servers who leave the system, will preempt the future occurance of broken connections and socket timeouts during the systems intercommunications.

Agents react to Alive messages from infrastructure components by registering with that new service if they deem it appropriate, or by ignoring the service all together. In the case of the Agent Name Service, an Agent will automatically send a registration request to the new ANS as it comes alive. This action by the Agent will help to make the Agent's name registration survive system outages, and automatically refresh systems in an environment that operate a single ANS server that crashes and then recovers, but has lost its registration database.

2.1 Alternative Discovery protocols

SSDP was created as a lightweight discovery protocol for the Universal Plug-n-Play (UPnP)[11] initiative, and defines a minimal protocol for multicast-based discovery. Because it is so minimal and lightweight, it was advantageous to utilize SSDP in the RETSINA architecture which supports agents on many platforms, including handheld computers. Nevertheless, we considered other discovery protocols such as SLP and Jini.

Service Location Protocol (SLP)[4] defines three types of systems that can participate in the discovery process, and the interactions between these entities. SLP deals with User Agents (UA), Service Agents (SA), and Directory Agents (DA). SA's have a service that they can provide (a producer.) UA's want to find a SA to perform some task (consumers.) DA's store SA advertisements as a "central clearing house" for finding services (yellow-pages.) The SLP protocol states that a UA should try to "discover" a DA using multicast when coming online. If found, all further discovery will take the form of a lookup process between the UA and the DA using directed unicast conversations. Likewise all SA's should have found the DA, and registered or advertised themselves with it. If a DA is not available, the UA can directly multicast a request for a type of service; and any SA's that receive the message and can provide the specified service should respond to the UA.

Java's Jini is similar, except that it requires the a-priori presence of a DA. The direct discovery of SAs by UAs is not allowed. Otherwise the protocols are similar. In fact, SSDP, SLP and Jini map the spectrum of permutations available for discovery. The minimal case is where there is no discovery. At the next level, SSDP allows any client (UA) to discover any service (SA) without the assistance of any other infrastructure component, such as a DA. Jini allows UA and SA entities to discover DA systems which provide a mediated lookup service. Finally, SLP implements the DA to provide a mediated lookup service, but a UA can still discover an SA directly, in the absence of a DA.

Compared with SLP, the RETSINA infrastructure already has the equivalent of a DA service in the form of the Agent Name Service (ANS) [7]. The use of SLP within RETSINA would therefore create redundant services and duplication of functionalities. A similar problem would be created with Jini, with the addition that Jini is typically (but not necessarily) a Java-specific solution, while the RETSINA MAS infrastructure supports heterogenous agents running on a number of platforms and implemented on a numer computer languages (e.g. Java, C, C++, Lisp, etc.)

2.2 Limits of Multicast

Multicast performs a controlled broadcast on the network that prevents traffic from being propagated onto segments of the network where the packets are not required or desired. In addition, multicast discovery packets are limited in their transmission by a Time-To-Live (TTL) value that indicates the number of routing devices that the packet should be allowed to propagate over. However, not all routers or firewalls are able (or administratively configured) to route multicast packets, which prevents discovery protocols from being used pervasively and globally.

Since the spread of multicast messages is controlled at the level of packet routing in the network, it is very sensitive to the topology of the overall network. Specifically, the use of multicast is the result of a difficult tradeoff between spreading the message to all the expected recipients and controlling the message so that it does not needlessly consume excessive bandwidth. The result is that, typically, Multicast cannot be reliably used outside the LAN of the sender. Our dilemma is that if we want to extend Multiagent systems across the whole Internet, the exclusive use of multicast for discovery provides a serious limitation.

3. GLOBAL DISCOVERY

Multi-Agent Systems exemplify the paradigm of an Internet of services in which sites (agents) interoperate and exchange services rather than just static pages. In order for MAS to expand to the whole Internet, the infrastructure of MAS should also be available across the Internet. We envision an Internet-wide MAS to be served by distributed white and yellow pages registries. While each registry may directly serve agents in a restricted area, they also form an integrated network of services that exchange information to support the communication of agents across the Internet. For example, if a white pages service, such as the RETSINA ANS, cannot resolve a request for an agent in its own LAN, it asks peer ANS servers across the Internet to resolve the reference. In turn, they forward the request to other ANSs that they know, until either the agent is found or the request has traveled too far and failed.

To extend the MAS infrastructure to Wide Area Networks, discovery should also be extended to the Internet so that infrastructure services can find each other and collaborate. In addition, agents should be allowed to look for infrastructure services across the entire Internet when they cannot find any service in their local network. In this section we describe the A2A discovery and global lookup infrastructure based on the Gnutella P2P network. A2A discovery and lookup allows agents and infrastructure components to dynamically expand their search across the network to find ANSs, Matchmakers, and other Middle Agents. A2A makes it possible for communities of agents to be automatically constructed, even though the group members are separated across wide expanses of the Internet.

3.1 The Gnutella P2P Network

Gnutella is both a file sharing mechanism and an asynchronous message passing system that allows its users to locate and share (typically MP3 music) files across the Internet. Each Gnutella application is both a "SERVer" and a "cliENT", so the protocol's creators coined a new term to describe a participant of the Gnutella network as a "servent". Other than the rapid and global deployment of this application, the exciting part of Gnutella is that it functions purely on Peer-to-Peer (P2P) connections between the servents without a centralized server.

Gnutella servents use their message passing system to perform two types of operations. First, they exchange messages to other servents that are available on the network so that they can maintain, or increase their level of connectivity to the overall gnutella network. Secondly, they exchange messages to search for specific files that might be available from other servents. This messaging system is primarily composed of binary packets of information, and text strings that represent search requests. File exchange is based on the HTTP protocol, and uses the same mechanisms as used in the retrieval of content from web servers.

Gnutella-Network discovery, used by traditional servent applications, is "boot-strapped" by connecting to a few well-known servents that the human operator running the servent expects to be there. Several servers have been set up on the Internet to cache connection information of available servent systems, so this intial phase is now typically automated. Agent applications can start with this model, and augment it with a-priori information from previous interactions, and by utilizing local discovery as described earlier in this paper.

As each servent starts, it will connect to a few remote systems. In order to discover other servents in the Gnutella network, it starts a PING / PONG process. PING messages are sent in hopes of receiving PONG messages that contain host, port, number of files, and kilobytes shared from other servents on the Gnutella network. As shown in Figure 3, each servent that receives a PING performs two operations: first it sends a PONG back along the same path from which the message came, so that eventually the PONG will reach the originating servent; second it forwards the PING to other servents with a reduced TTL. As soon as the TTL reduces to 0, the message is no longer forwarded and the search ceases to propagate. Because of the high degree of connectivity between servents on the Gnutella network, a PING may hit up to an exponential quantity of servents in its travels from servent to servent. As an example depicted in Figure 4, if all servents were maintaining 5 active connections, and had a TTL of 7, a PING message could reach as many as 109,225 other servents on the network.

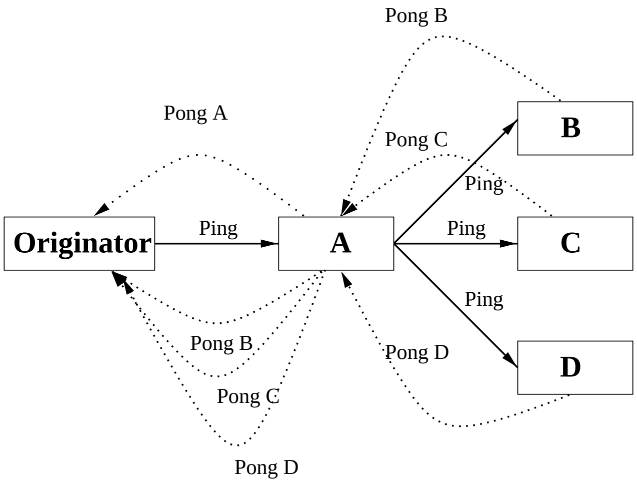

Figure 3: Propagation of Ping and Pong messages between message originators and other servents

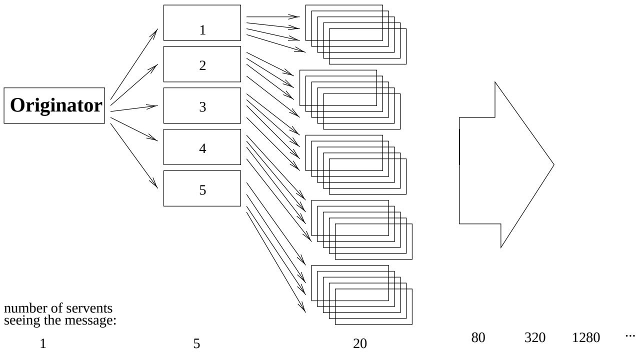

Figure 4: Fan-out of messages

Also, as time goes on, a servent will be pinged by other servents as they connect to the Gnutella network; and the servent will "overhear" PONG messages from even more servents that it is routing. Many of the PONG messages that a servent routes will be from systems that were not online when the servent first sent its own PING message. These additional PONG messages can be harvested, and used to curtail the over-utilization of PING messages. The messaging mechanism of Gnutella is based on the voluntary action of every servent to actually route the messages that they receive to other servents. The routes constructed will have more of a resemblance to graphs than to trees, so even if one servent does not route messages, there is a high likelihood that the original message will still be transmitted via an alternate route.

The search mechanism of Gnutella uses the same message passing process utilized to PING other servents except that message routing occurs first, before a servent attempts to peform a more time-consuming local search. A QUERY message that is sent to the Gnutella network contains a number representing the minimum acceptable communications link speed for file downloads, and a string representing the content that is being sought. In typical Gnutella servents, the search string will be tokenized before a servent's local file system searches for filenames that match any of the string's keyword tokens. If a local file exists that matches one or more of the words in the query string, its information will be formed into a response to the QUERY packet. If more than one file matches a pattern, the servent can reply with multiple responses embedded in the same message. The QUERY-HIT message that is sent back contains information about the system's link speed and the name and size of each matching file. It also contains an integer index value to help map the request into the local file system's storage.

3.2 The A2A Network LookUp and Discovery

Gnutella provides a basic connectivity schema that allows its servents to discover other servents over wide area networks. By enabling Agents and infrastructure components of the RETSINA MAS to act as servents on the Gnutella network, we can take advantage of a fabric of wide-area connectivity that is already in existance and widely deployed. The result is that whenever an agent needs to locate an infrastructure component, such as an ANS server, it can send both a multicast discovery message to the local network, and a QUERY request to the Gnutella network. Since the speeds of the two processes are quite different (Gnutella being slower than multicast) the agent will connect first to the local service but still learn of other remote services as they become visible.

As QUERY requests fan-out over the Gnutella network, being sent from servent to servent (like any other MP3 request,) A2A servents providing RETSINA infrastructure functionality will recognize a request for a service that they provide, and reply with a QUERY-HIT message. However, these QUERY-HIT responses do not provide information about where to find a file, but the address where infrastructure component can be reached.

To create A2A servents, we augmented the standard Gnutella servents in several ways. A2A servents monitor their connections to develop confidence levels for remote servents, and use these values to manage their connectivity. They categorize remote servents based on numerous attributes, to increase connectivity to communities that are most beneficial. A2A servents quickly focus incoming queries to agent tasks that can respond, and bypass inappropriate requests. And finally, they implement the automatic answering of requests that can facilitate the congregation of agents into communities. In the following sections we explain further details about the confidence, categorization, search, and discovery mechanism within A2A servents.

3.2.1 *Confidence*

The A2A architecture implements a ranking operation for connections by using several counters to indicate the confidence that an agent has in the usefulness of a connection. The counters provide a metric to indicate a positive or negative confidence in the connection's worthiness to remain connected, or be reestablished at a future time. The counters track repetitive messages, consecutive messages of the same type, repeated queries, and malformed or unknown message types.

The ranking of servents is used by an agent to help decide which servents to remain connected to. A2A servents do not keep their connections open forever: a connection is open only for a set number of messages. As an agent interacts with other servents, it learns from their interactions and modifies its confidence level. The more information the agent gets from another servent the better it ranks it and the longer it keeps a connection open to it. On the other hand, if the information provided by the servent is of no interest, or the agent suspects that the other servent is being disruptive, it will use the decreasing confidence level close the connection prematurly. The ranking is also used to decide with which servents to connect next, so that the agent is more likely to open a connection to a servent with whom it has a high confidence. The result of the ranking is that the agent is more likely to be connected to other agents servents that provide useful information.

3.2.2 *Categorization*

Gnutella accumulates all connectivity candidates into one large pool. The RETSINA A2A software refines these selections and categorizes hosts into several prioritized groups.

Group	Description
PRIME	preferred, agent specified
ALT	high calculated confidence levels
LOCAL	nearby on the agent's current network
HOME	on agent's home network or system
OTHER	miscellaneous unclassified servents
CACHE	systems bootstrap PONG servers
BAD	servents or networks to be avoided
NEW	staging area for discovered servents
USED	servents active on current connections
TASK	focused on specific community interactions

Table 1: A2A Servent Categories

The agent tries to keep a set number of each category of servents connected. The total connectivity count must also fit within the minimum and maximum allowed connections settings. If there are insufficient "appropriate" connections in place to perform a lookup or other agent communications task, an ordered selection from these categories will be utilized to increase connectivity. If the total number of connections exceeds a maximum limit, the A2A Agent will go into a host-cache mode, where no new outgoing connections will be attempted, existing connections will eventually expire and be disconnected, and incoming connections will be limited to a very few interactions (PING/PONG and immediately disconnect.) The PONGs that the host-cache mode A2A Agent will respond with, are the PONG packets of the servents it is currently attached to. This way, if Agent A knew it needed to talk to Agent B, but Agent B could not support any additional connections, Agent A could be connected to Agent C who was already participating in a conversation with Agent B. Any QUERYs sent to Agent C would fan-out and reach Agent B. Scalability is enhanced while maintaining required connectivity and intercommunications capabilities.

3.2.3 Task-based A2A Search

The Task structure of A2A enhances the basic `QUERY` process provided by Gnutella. All servents in Gnutella are assumed to be working on the same type of process: file sharing. When answering a query, they provide information that facilitates the location and downloading of a file. MAS, however, can utilize this same network transport to locate service providers (with a response using a URL to describe how and where a service could be contacted), or to actually have work done (with the response being the results of the process performed.) An impediment to using the public Gnutella network is that the agent systems should be able to ignore non-Agent messages, or any query that relates to a task that they cannot perform. Agent communities can optimize their communications over the Gnutella network if they could rapidly and more precisely determine which of the many `QUERY` messages flowing through the network are appropriate to work with. The A2A architecture does this by explicitly categorizing conversations, and providing a short-cut to expedite the sifting of messages.

To become part of a community with similar interests, an agent will initiate a new "Task" to which it can pose Questions and receive Answers. Joining a Task requires that the Agent specify a textual description of the communitity. For example, a Task might be created to particpate in a group like "retsina:AgentNameServer", "retsina:matchmaker", "auction:automobile", "store:shoes", or "weather". This string will be encoded into a numeric value that is included with the outgoing question as the Gnutella "minimum connection speed" value. As questions arrive at an A2A servent, each of the agent's Tasks can quickly examine this value to see if the new Question's special code matches the topic that it services. If it doesn't match, the `QUERY` message might be from a normal Gnutella servent searching for "Britteny Spears" or another agent seeking some other information. This is a "quick-check", and is further validated by examining the actual query from the remote Agent.

As a question is posed to a Task object, the question's text will be automatically prefixed with the text that identifies the Task's actual type. For example, a query moving across the network, might appear as: "stock:quote:IBM" or "restina:AgentNameServer:lookup:brent". After passing the quick-check, described above, a receiving agent will verify that the query actually begins with the task's identification string (e.g. "stock:" or "retsina:AgentNameServer:"). Once it verifies that the incoming request was directed to this task, the task will process and respond to the query with an appropriate answer. Otherwise the query is ignored. To increase privacy and further reduce the probability of false hits being generated by non-agent servents, the questions and answers (QUERYs and QHITs) can have their payload automatically encrypted and decrypted.

3.3 A2A Discovery

Many questions will take the form of "who can do service X" or "where is Agent Y". This is especially true of "Look Up questions". These types of questions will have responses that are composed of host/port information so the originator of the query can then contact that system. Since the type of query is so common, the A2A Task object can support Agent-specified auto-answer questions. When any of the Questions arrive at the Agent, the underlying A2A architecture will see the common Question and immediately reply with the predefined Answer, without disturbing the Agent to process the request. One of these Auto-Answer Questions that is implemented when a new Task object is created is for discovery (e.g. retsina:AgentNameServer:discover). The answer to this question would be the hostname and port number where this service is listening for incoming Gnutella connections.

When a new Task begins operation, it will send one of these discovery questions to the currently established Gnutella P2P network. Any responses that come back are added to the Task's list of discovered hosts. This list is similar to the Prime, Alt, Local, and Home category lists mentioned previously. However, every time an Agent attempts to ask a Question to the network, the Task object will endeavor to ensure that a minimum number of Task-specialized connections are active before actually transmitting the question. The Task will periodically retransmit the discovery query in an attempt to maintain a preset number of task-specific hosts. Over time, connectivity of an A2A agent will automatically shift from the public Gnutella network, to communities of similarly focused agent populations.

3.4 Scaling

A distinction between SSDP (or multicast-based discovery protocols in general) and this Agent-to-Agent model, is that with SSDP, systems that *provide* a specific service respond to the multicast discovery messages. P2P and A2A blur the differentiation between systems that are exclusivly *client-like* or *server-like* in nature, and treats everyone as *peers*. The systems discoverd in any P2P network might be clients or servers of that service, or both. If required, an A2A Task can be futher augmented with an auto-answer indicating that it *provides* a specific service and clients of that type of service can ask a question explicitly seeking servers of that specific type. Since, typically, there will be many more clients in existence than there are services, and services may have to limit the number of concurrent connections; discovery of another system that is either a specific service provider, or (more likely) connected to a provider of that service, allows that agent to connect in a more opportunistic way. Scalable services that operate on the Internet will need to maintain strong connectivity to communities of clients, and not just to massive amounts of individual clients.

The use of a form of managed broadcast, by utilizing point-to-point fan out of messages to reach a multitude of systems, can help Agent infrastructure communications scale to larger levels of connectivity. Rather than maintaining individual connections to many different resources, an Agent can keep a few connections active into the P2P network fabric. A2A communications riding on top of this network can moderate the connections to increase the probability that a system of the desired type is within the TTL of any Agent's transmissions. Then, active communities of Agents working with similar interests will congregate together to allow the formation of robust intereconnectivity behaviors.

Discovery Method	Sound	Complete	Adaptable	Speed of Discovery	Where to Use	Implementations
No Discovery	yes	yes	no	instantaneous	static infrastructure preplanned	OAA, Infosleuth UDDI, Jini
Multicast (LAN)	yes	high probability	yes	high	ad-hoc local workgroup	UPnP, Jini, Retsina
Peer-to-Peer (LAN & WAN)	yes	high probability	yes	low	ad-hoc global communities	Gnutella, Retsina

Table 2: The results derived from the query placed to the agent

Transmitting one request that reaches tens of thousands of systems can increase the reach of an Agent to global proportions. Tuning the selection of peers can dramatically improve the appropriateness of message transmission. Customizations via configuration parameters can allow limited capability Agents to choose whether to ignore (just relay), drop, or process specific types of Gnutella, or A2A messages. Fine-tuning parameters exist to control the initiation of PING messages, and to limit message frequency and size. "Harvesting" of information seen passing a system as it routes messages can proactivly discover information needed in advance of actually requiring a lookup message being sent. The added ability to harvest ALL Questions (regardless of there being a matching Task currently active at the Agent) allows an Agent to monitor and fully interact with all messages flowing over the Gnutella network. This feature could be used to monitor or categorize traffic loading, anticipate message storms, and automatically adjust connectivity to avoid denial of service attacks or system misuse. Private conversation could also be created by merely restricting the PING/PONG behavior from specific communications links and only allowing out-bound queries and their responses to pass.

4. QUALITATIVE EVALUATION

In this paper we have been advocating the need for a discovery mechanism as part of the MAS infrastructure. Here we make a qualitative evaluation of the various discovery mechanisms that we proposed in regards to: what is gained with their use; when they are adequate; and when, instead, they are inadequate. In the evaluation we will compare the base condition `no discovery` against the two discovery mechanisms described: `multicast` and `A2A`.

We will compare the three envrionments on 4 dimensions: soundness, completeness, adaptability, and speed of discovery. The summary of our analysis is displayed in table 2.

- **Soundness** measures whether the agent finds what it seeks. All three methods are sound. The base condition `no discovery` is trivially sound because the system administrator points the agents directly to the service that they need; the two discovery mechanisms only a allow the reply from infrastructure that matches a specified type.
- **Completeness** measures whether the agents will find all services that they are looking for. The base condition `no discovery` is trivially complete in the sense that the system administrator decides which agents should connect with which infrastructure; so there is no need for the agent to develop a picture of what services are available. The two discovery mechanisms are not complete. Multicast's UDP packest can be dropped or lost, and its discovery is typically limited to LAN segements. A2A has a broader discovery range, but does not guarantee a totally non-segemented network and so some systems may be unaccounted for.
- **Adaptability** measures whether the agents can adapt to changes in the infrastructure landscape. `No discovery` is not adaptable because the agents have hardcoded references to the infrastructure components, which is the problem we were forced to address when we looked at discovery. On the other hand, both `multicast` and `A2A` would detect changes and be able to recover from loss of connection from the infrastructure. (Again, multicast discovery would be restricted to a more localized operation.)
- **Speed of Discovery** measures the speed at which agents discover the infrastructure services they seek. In the `No discovery` condition the information is already provided, there is no point with measuring speed. `Multicast` is extremely fast and efficient; services are hit in a few milliseconds and any delay of their discovery process is due to the builtin packet-flood-prevention mechanisms of the protocol. `A2A` is much slower since requiring the message and response to be propagated across the network by the servents.

This analysis identifies some of the tradeoffs behind the different configurations. `No discovery` is useful only when the infrastructure is carefully pre-planned and never modified or perturbed in any way. In such cases, the System Administrator can confidently hardcode the references to infrastructure components and decide, in advance, where the agents are going to connect. The analysis also shows that multicast is very efficient and it can perform almost as efficiently as hardcoding, with the additional benefit of allowing automatic reconfiguration of the agents. The limit of multicast is scalability to the whole network; which is solved by the A2A discovery that has the potential to scales to the Internet. However, it suffers from increased discovery latency. Within the RETSINA project we implemented discovery in such a way that we use multicast as the primary discovery mechanism, and supplement it with A2A discovery for Wide Area implementations.

The last column of the table reports how other distributed systems are placed with respect to discovery. OAA and Infosleuth did not report of any discovery mechanism in their papers; UDDI[9] is supported by a directory service run by IBM and it does not report of any automatic discovery of lookup services. Jini supports discovery of lookup services (LUS), but it does not support discovery to interconnect distributed LUS. UPnP supports multicast to locate services within the local area network using the same SSDP protocol used by RETSINA; finally Gnutella based systems do not distinguish between local and wide area networks and support discovery only through the Gnutella `PING / PONG` protocol described above.

5. FUTURE WORK

5.1 Support for Constrained Platforms

Some Agent platforms with extremly limited resources (cell phones, embedded devices, and palm computers) might be unable to maintain 5 concurrent open socket connections. By allowing an outgoing connection of the P2P layer to be a localized (low TTL) multicast connection, these small devices would be capable of concurrently communicating to multiple local systems as if each system had a separate connection. Using this vehicle to connect to larger platforms, each with multiple open connections to other Gnutella servents or A2A Agents, the resource-constrained system could easily participate in the larger expanse of network connectivity.

5.2 Wide and Local Discovery Integration

Using the multicast connectivity described above, the normal discovery operation of Gnutella `PING / PONG` transactions, along with the higher level discovery of the A2A layer, would remove the need for a separate local discovery protocol. Tuning the multicast connection to be more discriminating of which packets it routes to this multicast connection could help to maintain the traffic to levels easily supported by agents and networks with lower bandwidth capabilities.

5.3 Infrastructure Integration

Beyond the use of basic infrastructure discovery and lookup services, new areas of research can be examined with the A2A enhanced converstations and interactions between Agents. Now an advertisement to a Matchmaker could be overheard, and waiting clients could immediately respond. Also lookups could solicit responses from new (or hybrid) systems that could provide augmented or enhanced services; and provide services that the Agent might not have known to ask for. Learning-systems that observe solutions protrayed by interactions between other Agents could form their own infobase of heuristics that have been seen to work by other systems.

6. CONCLUSIONS

There are numerous vendors and standards groups working on ad-hoc discovery recommendations, implementations, and standards. Peer-to-peer technologies have become the latest catch-phrase/buzzword craze to cross-over from technical publications to the popular press; and due to this increased attention, there are (again) many vendors and groups, including Microsoft and Sun, that are refining and integating P2P technology into their product offerings.

We are not advocating that SSDP or Gnutella are the end-all mechanisms to achieve the desired results described in this paper; and it is not our role to develop the ultimate products in these areas and to see that they are widely deployed. This will happen with or without us. The specific technologies chosen had features and capabilities that fit well with our existing architecture and research goals. What we feel to be important for our group, and all Agent Research, is that we understand the impact that these technologies can have on our infrastructure and how we can use them to expand the depth of our research. Our Agent-to-Agent research can provide valuable real-world feedback to the developers of these new and future protocol standards. We also feel that A2A technologies dramatically increase the ability of Intelligent Agent Systems to dynamically interact with each other and their "environment"; and they will produce new operational and behavioral models for agent interaction.

7. ACKNOWLEDGMENTS

This research has been sponsored in part by the Office of Naval Research Grant N-00014-96-16-1-1222 and by DARPA grant F-30602-98-2-0138.

8. REFERENCES

[1] K. Decker, K. Sycara, and M. Williamson. Middle-Agents for the Internet. In *Proceedings of IJCAI97*, Nagoya, Japan, 1997.

[2] D. Martin, A. Cheyer, and D. Moran. The open agent architecture: A framework for building distributed software systems. *Applied Artificial Intelligence*, 13(1-2):92–128, 1999.

[3] B. Perry, M. Taylor, and A. Unruh. Information aggregation and agent interaction patterns in infosleuth. In *Proceedings of CIA99*. ACM Press, 1999.

[4] SLPv2. Service Location Protocol. http://www.ietf.org/html.charters/srvloc-charter.html.

[5] SSDP. Simple Service Discovery Protocol. http://upnp.org/draft-cai-ssdp-v1-03.txt.

[6] K. Sycara, M. Klusch, S. Widoff, and L. Jianguo. Dynamic Service Matchmaking Among Agents in Open Information Environments. *ACM SIGMOD Record*, 28(1):47–53, 1999.

[7] K. Sycara, B. Langley, O. Juarez, and M. Paolucci. An Exploration in MAS Scalability. Manuscript submitted for publication., 2001.

[8] K. Sycara, M. Paolucci, M. van Velsen, and J. Giampapa. The RETSINA MAS Infrastructure. Technical Report CMU-RI-TR-01-05, The Robotics Institute, Carnegie Mellon University, 2001.

[9] UDDI. UDDI Technical White Paper. http://www.uddi.org/.

[10] UDP. User Datagram Protocol. http://www.freesoft.org/CIE/RFC/1122/72.htm.

[11] UPnP. Universal Plug and Play. http://upnp.org/.

[12] H.C. Wong and K. Sycara. A taxonomy of middle-agents for the internet. In *Proceedings of ICMAS2000*, Boston, MA, July 2000.