On Agent Domains, Agent Names and Proxy Agents[1]

Tim Finin and Anupama Potluri
Computer Science and Electrical Engineering
University of Maryland Baltimore County
Baltimore MD

Chelliah Thirunavukkarasu[2]
Enterprise Integration Technology
Palo Alto, CA

Don McKay and Robin McEntire
Valley Forge Engineering Center
Loral Government Systems Group
Paoli PA

Abstract

We consider the problem of how agents should be named and what kind of software infrastructure is necessary in order to locate an agent given only its name. We assume an agent environment which (1) is dynamic with agents being created and destroyed frequently; (2) undergoes re-organizations with agent groups and sub-groups forming and disbanding; and (3) supports agent communication by any of several transport mechanisms such as TCP/IP, email, http and distributed object systems. This leads us to propose the establishment of *agent domains* which are organized into an *agent domain hierarchy.* Agent name resolution can be done by *agent name servers,* analogous to Internet *domain name servers*. One of the additional benefits from this approach is that it easily supports the definition of *proxy agents*. We sketch how this proposal would impact the KQML agent communication language and protocol and describe an ongoing implementation of a generic *KQML Agent Name Server* and its integration into the KATS framework.

[1] This work was supported in part by the DoD, the Air Force Office of Scientific Research under contract F49620-92-J-0174, and by the Advanced Research Projects Agency monitored under USAF contracts F30602-93-C-0177 and F30602-93-C-0028 by Rome Laboratory.

[2] This work was done while the author was at UMBC.

Introduction

Agents need to talk to other agents. If you are an agent A and there is another specific agent B that you want to send a message to, how do you manage it? Well, clearly there is a need for some kind of referential expression that A can use for B and which can be given to the underlying machinery which will convey the message to B. One solution is to use an expression that locates the agent with respect to the message transport system. Examples of such "transport address names" would be a structure which contains an IP address and a port number, or a URL, or an email address for the TCP/IP, http and SMTP protocols. This is a common practice in many of our primitive agent systems today.

Another approach allows agents to use one of more symbolic names and to provide some kind of mechanism by which names can be registered and associated with their appropriate "transport address name". This approach is only slightly more sophisticated than the first. The name registration can be done in any of several ways, such as hand coding the associations into all of the agents, or broadcasting the associations over the transport mechanism or assuming the use of "communication facilitator" type agents.

The KQML language and protocol includes special commands (the register and unregister performatives) by which agents can announce the symbolic names by which they wish to be known. Special agents (commonly known as "communication facilitators") traffic in this knowledge and provide a name registration and resolution service. In the Loral/UMBC "KQML Agent Technology Software" (KATS) architecture, this name registration and resolution is handled automatically by a generic router sub-agent attached to each agent. From the agents perspective, all it has to do is to specify the set of symbolic names it wished to be known by. The router sub-agent automatically contacts the local "facilitator agent" [3] to register the agent by its symbolic names.

For example, suppose the agent named A wants to send a query to agent B. It passes a KQML form like

```
(ask-one :from A :to B :content ...)
```

to its router sub-agent (call it r(A)). This router is responsible, among other things, for resolving the agent name B into an address that can be given to the transport layer for delivery. In KATS, the router checks it's cache to see if it knows how to deliver a message to an agent named "B". If it does, it ships the message out. If not, it sends a KQML query to the local agent name server, asking for the address of an agent named "B". Upon receiving the information, it adds it to its cache and sends off the message.

There are additional wrinkles, of course, such as how to determine when a cache entry is stale and needs to be flushed, but this describes the current arrangement.

The Problem

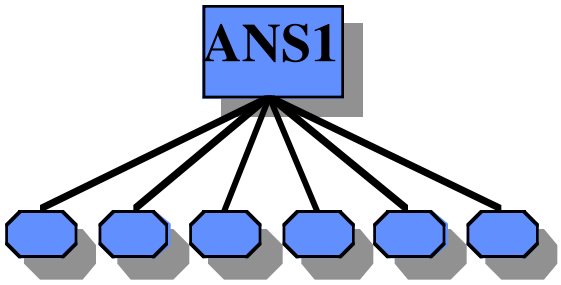
Although this approach works quite well as far as it goes, it just does not go very far. The problem is that it only supports communication between two agents if they both register with a common agent name server. There are several possible solutions. All agents could

[3] How does it find it? Well, our current implementation assumes an environment variable which points to it. An alternative convention are possible, such ...

use a single master name server possibly located deep under Cheyenne Mountain. Another approach is to have the name servers share their registration databases. Still a third, and more general, technique involves having the name servers use a distributed protocol to seek out the contact information on non-local agents. We next describe our protocol for such a distributed agent name resolution scheme.

Distributed agent-name resolution

We propose to organize agents into "agent domains" in much the same way that the Internet is organized into "host domains". An "agent domain" can be thought of as a collection of agents that are associated with a particular set of facilitator-class agents. In particular, every agent domain must have an "agent domain name server" (or "agent name server" or ANS for short) running. There may be other facilitator-class agents, such as brokers, associated with the agent-domain.



agents registered with agent name server ANS1

Figure 1 -- a set of agents are associated with an agent name server by sending it a KQML "register" performative.

Agent domains will be organized into a hierarchy. Agents will register with an ANS, as shown in figure one. An ANS, being an agent itself, will register with a "parent ANS", resulting in a hierarchy, as shown in Figure Two. Each agent will have one or more local names. An agent can also be referred to by its "domain qualified name". For example, consider the agent-domain hierarchy in Figure Three.

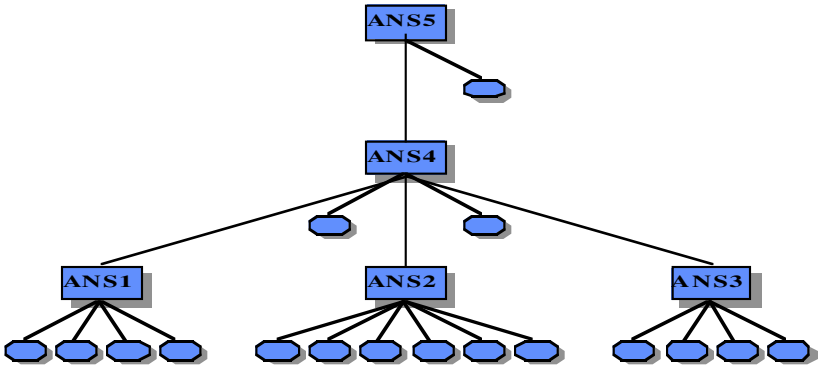


Figure 2 -- Agent name servers are organized into a hierarchy through the registration process.

One possibility we might consider is just to "piggy-back" on the existing Internet host structure. For example, why not refer to the agent "colossus" running on the machine

"cujo.cs.umbc.edu" as "colossus@cujo.cs.umbc.edu" and assume a standard port for KQML speaking agents. This idea is attractive in that it makes efficient use of a well thought out and implemented architecture. However, there are several problems which argue against this. The primary difficulty is that we do not want to tie KQML and agent communication in general to a single transport mechanism. Current research groups are using a variety of mechanisms to carry KQML messages -- TCP/IP, SMTP, CORBA objects, and HTTP. We would like to continue to keep KQML flexible in this regard. A consequence of this is that we need a general mechanism for naming agents that is independent of the transport mechanism.

What should agent names look like?

We propose a naming scheme similar to the one used for hosts on the Internet. Every agent will have one or more local names optionally followed by a domain qualifier. A local name can be any non-zero length sequence of characters chosen from the character set

{a-z,A-Z,0-1,-,_,.,+,#}

A domain qualifier begins with the character "." and consists of one or more agent domain names separated with a "." character. Thus a fully qualified agent name has the structure:

<local name> . <domain1>.<domain2>.<domain3>...<domainN>.

The following would all be valid names for an agent with the local name "colossus" registered in the "umbc.edu." agent domain (and assuming that it is in turn registered in the "edu." domain which is in the top-level "." domain.)[4]

colossus
colossus.umbc
colossus.umbc.edu.

Furthermore, we propose a correspondence between the names of agent domains and agent domain servers. Thus in the above example, agent *colossus* is registered with the ANS with local name *umbc* which is registered with the ANS with local name *edu* which is registered with the global ANS. Thus, the fully qualified name of an agent could be defined by its local name followed by a "." followed by the fully qualified name of its official agent name server.

There are obvious alternatives to the syntax we are proposing which would model agent names after email addresses (e.g., colossus@umbc.edu) or URLs (e.g., kqml://umbc.edu/colossus). There are several arguments against using either of these existing formats. One argument that applies to both is that we would like to avoid confusion about what a particular address means, e.g., is it the name of an agent, a reference to a document, or a reference to a mailbox. One might think that such a confusion could be a feature rather than a bug, since each of these might be a very reasonable way to think of and interact with an agent. However, there is clarity to be gained by separating the concept of a abstract reference to an agent that is independent of communication channel and a reference to an agent that implies a means of communication. The email style address has an advantage of using a special character (the @) to separate the "local name" from the "host name". When standards for SMTP were being developed, this

[4] An alternate is possible, such as one based on URLs. For example, we might choose to represent an agent using "*agent://<domain>.<domain>...<domain>/<local name>*" as in the example "agent://umbc.edu.kqml/colossus".

was quite useful since it provided a mechanism to support gateways between email systems that used very different protocols[5].

How agent names are resolved

The process of resolving a name is similar to the one used for the Internet DNS. One difference is that agent with a given name can have many addresses -- one for each transport mechanism that it can use. Thus, the agent_address is a function from agent names and agent transport types to transport addresses. We assume that an agent can be referred to using its fully qualified name[6] or any non-ambiguous abbreviation.

Suppose agent A_1 wants to resolve the fully qualified name N_2 into an address for transport type T_2. The process starts when A_1 asks its agent name server.[7] The query is passed up the hierarchy of agent name servers as long as the address is not known and N_2, is not recognized as being the name of some descendant. If an agent name server gets the query and knows the address, the process stops and a response is sent to A_1. If the root of the agent domain hierarchy is reached and the address was not found, the process fails and an appropriate error message is sent to A_1. If an agent name server recognizes that N_2 is the name of some descendant, it is passed down to the appropriate immediate child agent name server. This process continues until we find an agent name server that knows the address or we recognize that we can go no further. In this latter case, the process fails and an appropriate error message is sent to A_1.

Resolving partially qualified agent names follows a very similar process. There are a number of details that must be decided on in standardizing this name resolution protocol -- i.e., whether answers are sent directly back to the agent initiating the query or passed back through the hierarchy and cached along the way. These details should only effect the performance of the name resolution process.

Taking names seriously

.
Agents should take names seriously. What we mean by this is that application agents should always refer to other agents by their names, and not the underlying transport addresses, if known.[8] Agents should leave the resolution of these names into transport addresses up to specialized agents (e.g., agent name servers) and sub-agents (e.g., routers). Adapting this convention will directly support the concept of a *proxy agent* , the use of logical agent services, and other important notions. We will discuss the concept of a proxy agent in more detail and sketch how it can be easily implemented by adopting a few simple conventions for agent name servers.

---

[5] In the 1980's it was very common to see complicated email addresses that might involve several intervening gateways, such as foo@quux%bar@umbc.edu. For the most part, the necessity for these has gone away.

[6] i.e., its local name followed by a dot followed by the fully qualified name of its agent name server.

[7] How does an agent know the address of its agent name server? Since the process has to ground out somewhere, we assume that an agent knows at least the address of its agent name server upon creation or can access it via some standard environment variable, as is done in the KATS framework.

[8] Of course, some agents, such as agent name servers, will know transport addresses and some agent components (such as a router sub-agent) will have to know and use transport addresses. Moreover, we would expect a such sub-agents to remember the transport address for a given agent name and to use it for an extended session.

Proxy agents and their protocols

A *proxy agent* is an agent that handles all of the incoming and outgoing messages (perhaps with respect to a particular transport mechanism) for another agent. A simple proxy mechanism can be used to provide a number of services:

- *firewall gateways* -- agents which are behind a security firewall and use a proxy agent to communication to agents outside the firewall.
- *protocol gateways* -- An agent which is unable to send or receive messages via a particular transport mechanism (e.g., email) can still communicate with agents who only use that mechanism by having a proxy agent to mediate between two transport mechanisms.
- *message processing* -- The proxy can provide a processing service, such as logging incoming or outgoing messages, without altering the stream.
- *filtering and annotating* -- The proxy can alter the stream by filtering out certain incoming messages, blocking outgoing messages to particular destinations, annotating incoming messages, etc.
- *agent composition* -- A proxy agent facility allows one to develop a notion of "agent composition" similar to functional composition.

As an example, suppose we have two agents A and B, both of which use the agent name server F. A has proxy agent p(A) and B has proxy agent p(B). Suppose A wants to send a message to B. The following events take place:

1. A hands off the message to its router subagent r(A).
2. r(A) asks F for B's address.
3. f gives r(A) the address of p(A), A's proxy.
4. r(A) delivers the message to p(A) but the :TO field equals b.
5. p(A), knowing that it is a proxy for a (possibly among others) and noticing that it has received a message from a with the :TO field of B, understands that the message is not really intended for it, and asks its router r(p(A)) to deliver it to B.
6. r(p(A)) asks F for the B's address.
7. F gives r(p(A)) the address of p(B) -- B's proxy).
8. r(p(A)) delivers the message to p(B) with the :TO field equals B.
9. p(B), knowing that it is a proxy for B (possibly among others) and noticing that the :TO field is B, understands that the message is not really intended for it, and asks its router r(p(B)) to deliver it to B.
10. r(p(B)) asks F for the address of B.
11. F recognizes that p(B)) is B's proxy so it gives p(B) the real address of B.

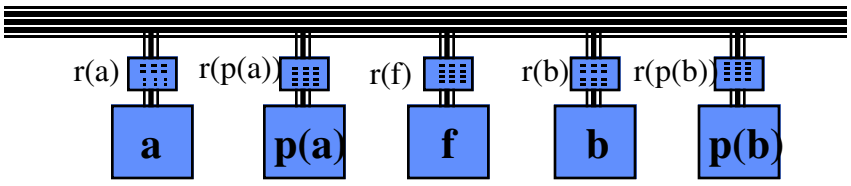


Figure 3 -- A conversation among five agents and their sub-agents.

This example demonstrates the use of proxy agents for both outgoing messages and messages. The proxy agents may do some additional processing of the messages they get, of course, like logging or traffic analysis, etc. The scenario above is the worst case in that it assumes all of the router subagent caches are empty. Subsequent communications would find the caches filled, so the agent name server would not have to be involved.

Implementing the concept of a proxy agent is rather trivial once we have agent name servers and agents who contact agents by name rather than by transport address. First, if an agent P is willing to serve as a proxy agent, it has to be able to provide some of the functionality that an agent name server does. Second, if A wishes to use P as a proxy for transport mechanism T, it must (1) get permission from ask P for this and (2) unregister with A's agent name server for transport T (if it was so registered). Third, P should register with A's agent name server *in A's name* for transport T. Good design dictates that all of the agents involved should also explicitly "know" that P is acting as A's proxy with respect to messages carried by transport mechanism T.[9]

Changes to KQML and standard utility agents

This naming scheme will not require any major changes to KQML such as the addition of new performatives or new parameters. It will have an impact on the form of the register performative and on the standard agent ontology and on the protocols used by standard utility agents such as an agent name server and a router. This, in turn, will effect the protocols that all agents who use these standard utility agents follow.

An agent name server will have to store more information about the agents that are registered with it and will have to handle some additional performatives. When an agent registers with an agent name server, it should provide a set of symbolic names it will respond to and a set of transport type/address pairs. Authentication information may be provided as described in (Thirunavukkarasu, Finin and Mayfield 95). A standard agent name server must handle requests to register and unregister from agents as well as various kinds of queries against its registration database.[10]

In reaching a consensus on the precise details of how to add these changes to KQML we will have to choose what aspects are expressed by adding to or modifying the basic components of KQML (i.e., performatives and parameters and their semantics) and which are expressed by extending the common "*agent ontology*" that is assumed by KQML.[11]

Conclusions

We have discussed the problem developing a global naming scheme for software agents and how such names can be resolved into usable addresses. We have assumed an agent environment which (1) is dynamic with agents being created and destroyed frequently; (2) undergoes re-organizations with agent groups and sub-groups forming and disbanding; and (3) supports agent communication by any of several transport mechanisms such as TCP/IP, email, http and distributed object systems. We proposed the use of *agent domains* which

---

[9] This will allow the agents to recognize, for example, that messages delivered to P's address but addressed to A were not sent, delivered or addressed in error.

[10] A careful specification of the requirements for an agent name server would include the performatives register, unregister, ask-one, ask-all, stream, subscribe and deny.

[11] This ontology is only now being articulated by defining it using the Ontolingua language.

are organized into an *agent domain hierarchy.* Agent name resolution will be done *agent name server* agents which use a distributed protocol similar to that used by Internet *domain name servers*. This approach supports the definition of *proxy agents* which have a variety of uses. We have briefly discussed how this proposal would impact the KQML agent communication language and protocol and describe an ongoing implementation of a generic *KQML Agent Name Server* and its integration into the KATS framework.

Acknowledgements

This work has been the result of very fruitful collaborations with a number of colleagues with whom we have worked on KQML and other aspects of the Knowledge Sharing Effort. We wish to specifically thank and acknowledge James Mayfield, Richard Fritzson, Charles Nicholas, and R. Scott Cost.

Bibliography

Paul Albitz & Cricket Liu, *DNS and BIND*, O'Reilly, 1992, ISBN:1-56592-010-4.

Tim Finin, Don McKay, Rich Fritzson, and Robin McEntire. KQML: an information and knowledge exchange protocol. In International Conference on Building and Sharing of Very Large-Scale Knowledge Bases, December 1993. A version of this paper will appear in Kazuhiro Fuchi and Toshio Yokoi (Ed.), "Knowledge Building and Knowledge Sharing", Ohmsha and IOS Press, 1994. An online copy can be obtained from "http://www.cs.umbc.edu/kqml/papers/kbks.ps".

Tim Finin, Don McKay, Rich Fritzson, and Robin McEntire. The KQML information and knowledge exchange protocol. In Third International Conference on Information and Knowledge Management, November 1994.

Rich Fritzson, Tim Finin, Don McKay and Robin McEntire. KQML - A Language and Protocol for Knowledge and Information Exchange, 13th International Distributed Artificial Intelligence Workshop, July 28-30, 1994. Seattle WA.

Tim Finin, Yannis Labrou, and James Mayfield, KQML as an agent communication language, invited chapter in Jeff Bradshaw (Ed.), ``Software Agents'', MIT Press, Cambridge, to appear, (1995).

Mark R. Horton, *What is a domain?,* available on-line as <http://www.dns.net/dnsrd/docs/domain.ps>.

Yannis Labrou and Tim Finin. A semantics approach for KQML—a general purpose communication language for software agents. In Third International Conference on Information and Knowledge Management, November 1994. Available on-line as http://www.cs.umbc.edu/kqml/papers/kqml-semantics.ps.

R. Patil, R. Fikes, P. Patel-Schneider, D. McKay, T. Finin, T. Gruber, and R. Neches. The DARPA knowledge sharing effort: Progress report. In B. Nebel, C. Rich, and W. Swartout, editors, Principles of Knowledge Representation and Reasoning: Proc.\ of the Third International Conference (KR'92), San Mateo, CA, November 1992. Morgan Kaufmann.

R. Neches, R. Fikes, T. Finin, T. Gruber, R. Patil, T. Senator, and W. Swartout. Enabling technology for knowledge sharing. AI Magazine, 12(3):36--56, Fall 1991.

James Mayfield, Yannis Labrou and Tim Finin. Evaluation of KQML as an Agent Communication Language, the IJCAI-95 Workshop on Agent Theories, Architectures, and Languages.