Location Transparent Routing in Mobile Agent Systems Merging Name Lookups with Routing

Werner Van Belle (werner.van.belle@vub.ac.be)[1]
Karsten Verelst (kaverels@vub.ac.be)[2]
Theo D'Hondt (tjdhondt@vub.ac.be)

Programming Technology Lab (PROG)
Department Informatics (DINF)
Vrije Universiteit Brussel (VUB)
1050 - Brussels, Belgium

Abstract

Telecommunication systems these days are moving from static wide area component structures towards highly dynamic mobile infrastructures. This shift requires new algorithms to interconnect these mobile entities/components and route messages between them. In this paper we describe a naming and routing algorithm which can be used in fine-grained mobile component systems. As a case we use a homogenous environment of mobile multi-agent systems, which executes agents as they pass by.

Keywords: *mobile multi-agent systems, distributed systems, migration, routing, naming*

1. Introduction

A mobile multi-agent is an active autonomous software component that is able to communicate with other agents; the term mobile refers to the fact that an agent can migrate to other agent systems, thereby carrying its program code and data along. Regarding the expression *mobile multi-agent system* there is some confusion. A multi-agent system in AI denotes a software system that simulates the behavior of large groups of interacting agents, without focusing on the distribution aspect of these systems. In the world of distributed computing, a mobile multi-agent system is a *distributed* environment which multi-agents can be written in. (we will use the second definition). For example, there are a number of mobile multi-agent systems developed at various research labs: the *Aglet Workbench* by IBM[3], *Mole*[1] developed at the distributed systems group at Stuttgart, *AgentTCL*[2] and others provide an architecture in which mobile agents can exist.

Mobile multi-agent systems can be implemented as a library in an existing language [1], [3], [4] or as a special state of the art language that has high level primitives to write agents with [2]. Nevertheless, whether one conceives an agent system as a library or as a language, the main goal is the same. A mobile multi-agent system should offer as much flexibility as possible to the agent programmer while hiding the underlying network architecture as much as possible. This is a tendency we also encounter in distributed architectures like CORBA [5], Java RMI [6] and DCOM. They offer a means to transfer data and bring services online without too many complications. However, mobile multi-agent systems take it one step further. On top of providing these basic services, they offer high-level primitives and structures which can be used to write distributed programs without even knowing that one is writing programs for the network.

Multi-agent systems try to make the underlying network architecture as transparent as possible to the agent programmer. This includes shielding the agent programmer from worrying wether a sent message will arrive at its location. Nor should (s)he mind whether the data (s)he sends is serialized correctly. A particular issue

---

[1] Werner Van Belle is working at a project for the Belgium Institute for Science and Technology (IWT) and is currently researching software engineering in mobile multi-agent systems.

[2] Karsten Verelst is aspirant at the Belgium Fund for Scientific Investigation (NFWO) and is currently researching security in mobile multi-agent systems.

in *mobile* multi-agents is that keeping track of an agent poses a burden on the agent system: the agent system should make the location of an agent transparent to the agent programmer. This means that wherever an agent is running, wherever an agent *migrates* to, it should always stay connected to its communication partners, as it should stay connected to the resources it was using.

2. The Name of an Agent

The setting which we envision for mobile multi agents is one where there are a lot of agents, communicating intensely with each other. Programs are fine-grained agent clusters and agents migrate frequently. Agents interact with each other in a non-predictable way. (Agent A doesn't know it will be contacted by agent B, nor does it know which agents have its addres). Second, the network is a WAN, it is owned by different companies and we can never oversee the whole structure. As such the (network)distance between agents varies greatly and delay times can change without warning. To simplify the concepts in this paper, we take it for granted that every node in the network runs the same agent system so that we have a homogeneous environment to talk about.

Before we tackle the problem of location transparent routing we have to give our components a name, which is a standard concern in distributed systems.[3] For example, if one wants to refer to a webserver at a given place one gives the URL of that machine: everybody knows what is referred to when the complete name http://progwww.vub.ac.be:8080/cgi-bin/counter.perl is specified. If we look at CORBA, we see that every component in the network gets an IOR (*Interoperable Object Reference*) which can be obtained from an ORB (*Object Request Broker*). In RMI an object registers its name by a registry (*rmiregistry*). However, this kind of naming may no longer be valid in mobile multi agent systems because components are mobile. The question now is how do we give our agents a name.

Current naming schemes in mobile multi-agent systems are mainly based on internet naming, without taking into account the possible mobility of agents. If an agent migrates to another host its name changes to point out its new location. For example, if the agent progwww.vub.ac.be/hello moves to another agent system progpc15.vub.ac.be, its name will change to progpc15.vub.ac.be/hello. This changing of names upon migration is highly undesirable. In a mobile multi-agent system, every agent should have its own name, which (1) defines the agent in a unique fashion and (2) doesn't change whenever the agent migrates to another agent system. These naming restrictions make things for agents more complex than in current-day distributed systems.

In this paper we will assume that every agent has a name consisting of the name of the agent system (say, the computer) and a free form string which is used to refer to the agent. For example we have an agent system tecra.vub.ac.be which creates an agent ralf. This agent will be called tecra.vub.ac.be/ralf. To avoid name clashes we say that every agent system manages its own name space. So, machine tecra.vub.ac.be shouldn't spawn more than one tecra.vub.ac.be/ralf agent.[4]

3. Solutions

Given the fact that agent names do not change upon migration we still need a way to route messages between agents. Routing in distributed systems often consists of resolving a name to an address and let the underlying network protocol deliver messages to the right place. We will now have a look at possible solutions, which are used today.

3.1. Routing and Forwarding

A basic solution to this problem is the use of a forwarding system: whenever an agent sends a message to another agent it will always send the message to its home address from where on it will be forwarded to the actual location. Whenever an agent migrates to another place it will inform its home location of its new position.

This is a technique which is easily implemented in all high level conceived distributed systems/languages. For example, in Java RMI or CORBA we can transport a set of objects to another place and leave forwarding objects instead of the real objects.

Nevertheless, this solution has a major drawback, which is that we always take extra hops before delivering messages. If agents have moved far from their place this is a significant penalty. Likewise, if we move a cluster of interacting agents to the end of the world we will see a significant delay in their communication (every message has to be send to their homeplace and back to where they are).

3.2. Notification of Communication Partners

Thinking a bit further along this line reveals how we can not only notify the home address of an agent, but also all communication partners of this agent. This implies that the receiver of an agent has to keep track of all references

[3] A *name* refers to an entity, a component, and in our case an agent. A naming scheme defines which names are valid and how names partition the name space.

[4] Of course, if agents are also able to spawn other agents, it may be useful to change the naming system to something like 'creationmachine/ownermachine/<name>'

to itself and whenever the agent migrates it will send an update to everybody who had a reference to it [1].

The drawback of this approach is that it only works well for small scale systems, where there is an equal communication between partners. Whenever there are agents which serve lots of other agents the penalty of moving would be too high. For example, an agent is serving some kind of webpage and 3000 clients spread across the world have its address. At the moment this agent migrates it will have to notify 3000 partners of its new address. So, this solution has an administration overhead too large to be generally useful.

3.3. Name Serving and Routing

Another solution is using a nameserver which keeps track of all agents. If we want to contact an agent we ask the central name server to resolve the name, after which we will send the message to the given address. This solution basically resembles the current internet network infrastructure. On top of a delivery system (IP) we find a naming system to resolve names to IP addresses (DNS).

This is a good technique whenever we (1) do not require too many simultaneous name lookups and (2) names don't change all that often. Of course, in mobile multi-agent systems agents migrate frequently, which very often results in address-updates. So without adapting this technique we won't be able to send messages efficiently in a location transparent fashion.

3.4. Name Serving, Caching, Routing and Forwarding

With a little adaptation this technique becomes more useful. We can, for example, use a name server to resolve agent names. But afterwards cache the results so the next time we send a message we immediately know the place of the agent. This system of course requires a forwarding system on every host. If a message arrives for an agent which is not locally available we will route the message further using the same kind of name lookup. After doing this we send an invalid-cache message to the original sender. This is a system which we also find in Emerald. [9]

Although this system works fine and is scaleable we always have a set of misses when an agent has migrated. This is particularly painful if agents migrate frequently. For example, if an agent is communicating with let's say 10 partners and it migrates, there will be 10 misses for sure, which means, 10 messages send to the wrong location, 10 messages send to the right location and 10 name updates)

3.5. Name Serving, Routing and Tracking

Another adaptation of the nameserving and routing technique can be done if we know that all connections in the agent systems are typically connection oriented. In order to communicate with an agent we look up the name of the receiving agent, after which we create a connection to the agent. From then on we will be kept informed of the position of the agent. If the agent migrates, the system notifies us of the migration and we know its new location. This system is like the one that is proposed in [8] and [7] to support Mobile IP and its also the favorite in Mobile Telephony. The main problem with this system is the use of the nameserver which requires a lot of updates when agents migrate frequently and the fact that it is merely connection oriented than connectionless. In a wide area network we can't force all connections to be connection oriented.

Because all these current approaches do not fit in our problem, we will now introduce our solution.

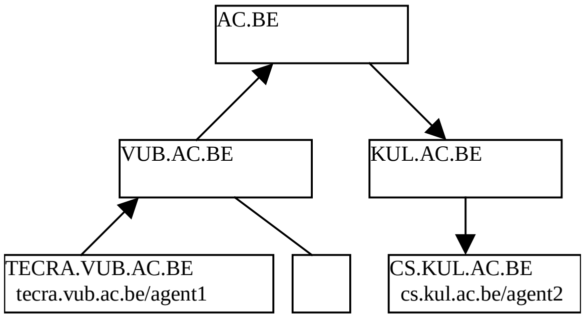
4. Merging routing and name serving

4.1 Sending Messages

The solution we propose is to make no distinction between the name of an agent and the address of an agent. Instead of resolving the name of an agent to find its place we immediately route messages to an agent based upon the receiver's name. This means of course that we need to change the infrastructure substantially. We no longer have a statically interconnected routing infrastructure and a different statically interconnected naming infrastructure, instead we have one hierarchical infrastructure in which we name and route messages.

Instead of sending a lookup request to a nameserver, we send the complete message to the nameserver, which will 'route' the messages further to the next nameserver.

For example: the figure below contains an hierarchical interconnection of nameservers/routers. If agent tecra.vub.ac.be/agent1 wants to send a message to an agent cs.kul.ac.be/agent1, he can pass the message to the local nameserver, in this case vub.ac.be. At the moment vub.ac.be receives a message it will send the message through to ac.be. Ac.be will see the message for cs.kul.ac.be and will pass it to the right downlink, in this case, kul.ac.be. From there on, it is delivered to cs.kul.ac.be/agent1 in exactly the same way.



More formally, the delivery of a message on every node is as follows:

1. Is the message for a name which is local ? Yes, deliver the name, No continue.
2. Is there a forward rule for this name ? Yes, forward the message to the given link, No continue
3. Is there a link with the same name suffix ? Yes, deliver the message to that link. No, continue
4. Deliver the name to the uplink.

4.2. Migrating Agents

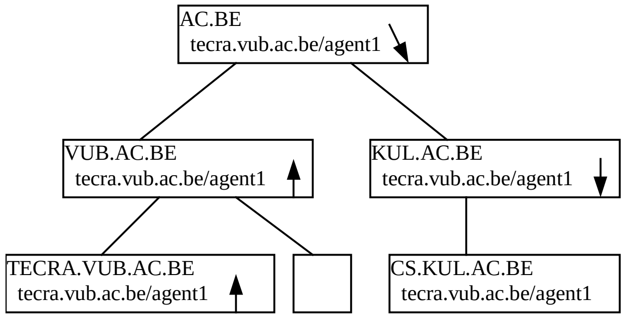
To migrate an agent in this setup, we do it in exactly the same way as if we would send a message, with the difference that every router/nameserver should *understand* the movement of an agent. This means that whenever an agent passes through a node, the node should update its routing-tables to point out the agent's new *direction*. Note that we do not point to the new location, but instead show the link to where the agent has gone to. This guarantees that there are no updates needed to this node, if the agent migrates further on.

For example, if we migrate tecra.vub.ac.be/agent1 to cs.kul.ac.be, we take the state of the agent [5] and send it as an agent message to the local node tecra.vub.ac.be. This node will send the agent to its uplink, vub.ac.be and update its routing table to let tecra.vub.ac.be/agent1 point towards the uplink. Vub.ac.be will send the agent to ac.be and will keep a rule for agent tecra.vub.ac.be/agent1 to its uplink. The node ac.be will send the agent to the right downlink, in this case kul.ac.be and make a rule to point out the agent's new location.

If an agent moves through a node to its original position, the associated rules will be deleted. To keep tables smaller we can use a system of wildcards to annotate groups and clusters of agents.

[5] Note that real migration is by the way still a problem in mobile multi agent systems. Most agent systems these days use a kind of looping model to support migration, which is not transparent enough.

If an agent migrates while messages are being sent to it, these messages will follow the agent on its path and will finally arrive at the right location.



More formally: Every node has a routing table which keeps track of migrated agents. The algorithm to deliver an agent is on every node as follows:

1. Is the agent for this machine. Yes, deliver the agent and update the local table. No, continue
2. Is the destination place name higher in the tree ? Yes, send the agent to the uplink and update the local table with a rule '<agent> -> uplink'. No, continue
3. Is there a downlink with a name that is a suffix for the given placename. Yes, send the agent to the given link and add a rule '<agent> -> downlink <x>'. No, continue
4. Send the agent back, destination place doesn't exist.

5. Discussion & Performance Estimation

This solution is one which is very modular, in the sense that every node in the network has a name in the hierarchy. All nodes have one uplink and a number of downlinks. All the downlinks have a name which is suffixed with the uplink's name. How the routing in the uplink or how the routing in the downlinks happen doesn't matter. As long as every 'logical' node behaves as it should, this solution will work.

Regarding performance, this solution is very acceptable because

1) we always have the shortest path in the tree towards a receiving agent. This is because all routing goes hierarchical and as such we never will go higher in the tree than needed. If for example agent tecra.vub.ac.be/agent1 has moved to cs.kul.ac.be we see that any message sent to that agent will immediately take the correct path. So, whenever moving a cluster of agents these agents will only see the delays of the network at their new location, completely independent of their home location.

2) We will never have cold starts. That is to say, whenever an agent starts a communication session, the first message will immediately take the shortest path towards its communication partner. This is a very strong point because, if we have a fine grained system in which agents migrate very often, we lose tons of bandwidth with cold starts.

3) Furthermore we don't have an administration overhead to update or query the nameserver because there is none, which is also a very strong point if we work in a fine grained mobile multi agent system.

6. Tying up Loose Ends

6.1. Graph vs Tree

Nevertheless there are still a few things which should be cleared up. The solution as proposed here is completely hierarchical. This is a technique which can give problems because sometimes people conceive a network in other ways.

For example if we have a typical bus structured network and we want to use this routing scheme we see that the extra hop to the nameserver and back to the bus will give a significant delay. A possible workaround for this problem is to create only one name for the entire bus.

Another example where we don't have an hierarchy is the one where a specific node can forward messages to two other nodes (say a meshed network). If one wants to build a system like this one has to fall back to some hybrid solution, which is easy to implement. The routing algorithm we propose is modular and as such we can view these double linked nodes as one link which somehow manages its own tables and takes care of synchronizing both subnodes.

Another point of interest is the fact that geographical constraints can help to build such a hierarchical structure. Nodes which are close to each other should have the same parent node, which is of course a triviality, but which also makes it much easier to create this structure without violating the hierarchy.

6.2. Changing Hierarchies

A problem with our solution, if taken too literally, is that one is unable to cope with changing hierarchies. For example, if one creates a hierarchy like the one above, and connects the lower machine tecra.vub.ac.be to the node kul.ac.be, the name of the machine will no longer be tecra.vub.ac.be, but tecra.kul.ac.be, which is not something we would immediately want.

So, machines should also be regarded as mobile entities, which indicates that this kind of routing and naming should be somehow virtual. The hierarchical structure that is built should be based on the geographical constraints of the environment and not on the structure of existing routers.

7. Pushing Things into the Future

Where do we go from here ? If we take a look at current day distributed and communication systems we see a shift from statically interconnected components towards a highly dynamic interaction between mobile components. We see this in mobile telephony, portable hardware (portable computers, palmtops and others). If we look at software we see the same kind of evolution, think of mobile agents, Java applets, servlets and other forms of mobile code and mobile services. So maybe it's time to tackle the problem of location transparent routing at the roots. Instead of writing extravagant stubbing systems, maybe it is better to develop an interconnection of intelligent routers and switches to handle this problem.

We could for example, instead of using two separate infrastructures, IP and DNS, where IP is too static and DNS is too heavy to handle mobility, recreate a structure and merge these two kind of services.

If we look at the proposed solution, we see that the system is scaleable, modular and perfectly suited for location transparent routing when dealing with mobile entities.

Of course there are still problems left. The fact that the structure itself is static, but the agents can move is not affordable. Somehow, if we want this solution to work in real life applications we should be able to cope with changing hierarchies.

8. More Information

We have implemented this router architecture in a state-of-the-art mobile multi-agent system. The agent system is based on an educational language, called Pico. The system itself supports location transparent routing as discussed in this paper and strong migration (at any time can we move an agent, even if it's in the middle of the execution of code). More information can be found at http://pico.vub.ac.be/cborg.

Furthermore, we have done tests with reinforcement learning to make this algorithm adaptive to small network changes and do load balancing.

9. Conclusion

In this paper, after enumerating the techniques used today to solve location transparent routing in mobile multi agent systems, we created a location transparent router architecture by merging a hierarchical naming service with a message-queuing algorithm. Afterwards we gave a clear reason why the performance of this approach implies no overhead at all. The paper wraps up with two practical applications of this router architecture. The first application is in the field of mobile multi-agent systems themselves. By using this architecture we can easily introduce location transparency into a mobile multi-agent system: the sending of messages can happen in a location transparent way and it is moreover not very difficult to implement resource transparency. A second application domain is mobile computing. Mobile computing is getting more and more attention these days. And if we want to be able to cope with this we have to offer new network protocols, requiring for instance hardware routers that are able to cope with portables and with *migrating* IP-numbers.

10. Acknowledgements

We would like to thank Theo D'Hondt, Tom Tourwe, Johan Fabry, Roel Wuyts, Johan Brichau and others for proofreading this paper. Furthermore Robert Jansen of the computing centre deserves some special credit.

11. References

[1] K. Rothermel and R. Popescu-Zeletin, Eds. *Mobile Agents*, Lecture Notes in Computer Science Series, vol 1219, Springer 1997

[2] R.S. Gray, *Agent TCL: A Transportable Agent System*, Proc. CIKM, Wksp Intilligent Information Agents, J.Mayfield and T. Finnin, Eds. 1995

[3] http://www.trl.ibm.co.jp/aglets: *The Aglet Workbench*

[4] http://www.objectspace.com/voyager: Voyager

[5] http://www.omg.org/… The Corba homegrounds. Here you can also find information about the Mobile Agent Facility

[6] http://www.chatsubo.com/rmi: Java RMI Development Page

[7] *J. Solomon* Mobile Ipv4 Configuration Option for PPP, IPCP, The internet Society RFC2290, Feb '98

[8] *Charles E. Perkins*, IP Mobility Support Sun Microsystems, Oct '96

[9] *Norman C. Hutchinson:* Emerald: An Object-Based Language for Distributed Programming; Departement of computer science, university of Washington, Seattle, Jan '87