Scalable and Secure Global Name Services for Mobile Agents

Volker Roth

Fraunhofer Institut für Graphische Datenverarbeitung
Rundeturmstraße 6, 64283 Darmstadt, Germany
vroth@igd.fhg.de

Abstract. In this article we investigate secure global name services which are scalable to the Internet and account for particularities of mobile agents (frequent changes in locations). Name services are required for instance for message routing between mobile agents and for tracking agents. In this article we propose first protocols targeted at providing such name services. We focus in particular on security issues.

Keywords: mobile agents, name server, agent name.

1 Introduction

Early research on mobile agent systems concentrated on how to migrate mobile agents. Manifold agent systems exist today and there is a notable shift towards research in how to best support *transparent communication* between mobile agents [18, 15, 9, 10]. *Transparent* means that agents need not be aware of the actual location of agents with which they wish to communicate. Two problems must be solved in order to achieve transparent communication:

- The peer agent must be located, for instance by means of a *name service* that maps an agent's location invariant name onto its current whereabouts.
- Messages must be routed to the peer. This can become difficult since mobile agents might "run away" from messages. Guaranteed delivery is addressed for instance by Murphy, Picco, and Moreau [10, 9].

In this article we address the problem of how to locate agents. Our goal is to provide adequate global name services for mobile agents, which scale to the Internet and account for the particularities of mobile agents (frequent changes in locations). Establishing name services for mobile agents poses a number of difficulties:

- Name service updates and lookups must be fast. Since mobile agents can migrate at any time a huge rate of updates must be expected.

– The load must be distributed between a sufficient number of name servers. A suitable unambiguous mapping between agents and name servers must be established.
– The number of name servers must be gradually scalable to increasing demand.
– Name services bring security problems that must be addressed properly. Yet heavyweight cryptographic protocols (e. g. three way authentication [6]) most probably counter the demand for fast lookups and updates.

In this article we sketch an approach to solving these problems. We aim at creating a protocol which is suitable to support public global name services for mobile agents in Internet scale. *Public* means that lookups on agent locations are not restricted in principle, yet we would like to account for privacy issues arising for agent owners in establishing a global service for agent tracking.

The remainder of this article is organized as follows: Section 2 illustrates and discusses four basic models of agent tracking. We identify one model as suited for our purpose. In Section 3 we present basic attacks on name services for mobile agents. We refer to these attacks in Sections 4 and 5, in which we present first an ad hoc approach to securing name services and then an improved approach. The second approach yields a number of features that we expect from a secure and global public name service for mobile agents. In Section 7 we conclude by pointing out remaining risks and areas for further research.

2 Tracking Agent Locations

Several models of tracking agents are conceivable. Aridor and Oshima [1] already gave an initial discussion of agent name services and suggested three methods of locating agents: *brute force, logging,* and *redirection*. Milojičić et al. distinguish four models [8] which incorporate those of Oshima and Aridor: *updating at home node, registering, searching,* and *forwarding*. In this section we elaborate on these four models and we adopt the terminology given below:

	Milojičić et al.	Oshima et al.
direct response	updating	logging
buffered response	registering	
searching	searching	brute force
forward references	forwarding	redirection

The simplest approach is to have agents update their locations whenever they hop from one host to the next (see Figure 1). We refer to this method as *direct response*. A grave disadvantage of this method is that the home base of an

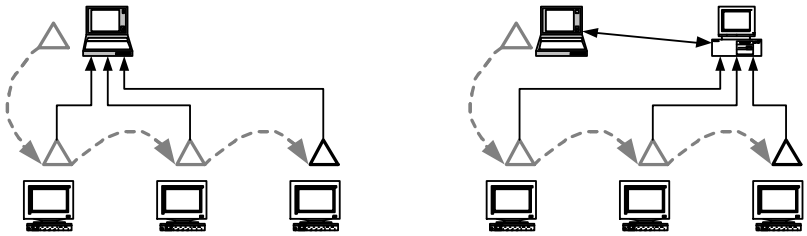

Fig. 1. Direct response (left) means that an agent updates its home base on its new location whenever it hops from one host to the next. The buffered response (right) is the typical name server model. A dedicated server which is permanently online is updated on the new locations of agents whenever the location changes.

agent needs to be permanently online. This is not desireable if e. g., the home base is a mobile device.

The *buffered response* illustrated in Figure 1 is the prototypical name server model. Agents update a name server on their new location whenever the location changes. The name server is permanently online. If the owner of an agent wishes to know the current location of one of his agents he just queries the name server. This model is used for instance in the *Mobile Object Workbench* [3].

Searching is illustrated in Figure 2. The general idea is that agent servers provide a simple name service which takes as its input the name of an agent and outputs *Yes* if the agent with the given name is currently hosted by that server, and *No* otherwise. Clients query these name services according to a search strategy in order to locate an agent. For instance Chen et al. [4] discuss a search heuristic based on the assumption that the execution times of agents on a server have a binomial distribution. For illustrative purposes we sketch a simpler probabilistic search strategy based on linear extrapolation:

> Let A be an agent and let $H_1, \ldots, H_n$ be its known route. Let δ_t be the mean time an agent stays on a host of the route. Let t_0 denote the point in time at which A was dispatched to its first host. At time $t > t_0$ the owner of A wants to know its current location. Let $i = \min(n, \lfloor (t - t_0)/\delta_t \rfloor)$. The owner queries the name services of H_j, $j = i \pm 1, 2, \ldots$ for $1 \leq j \leq n$.

Heuristic search is not suitable for free-roaming agents since the set of hosts and the order in which they are visited by the agent are in principle not known in advance.

Forward references point from one hop of an agent to its next hop. Forward references are subject to expiration. After a fixed amount of time the forward

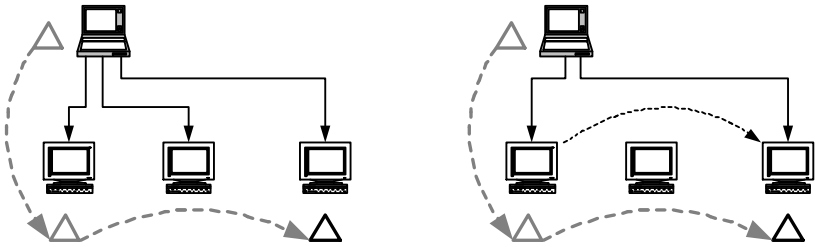

Fig. 2. Searching (left) is based on a simple name service per agent server. Clients request confirmation of the location of an agent based on a predictor of the agent's location. Forward references (right) follow the path of an agent. Each reference points to the host to which an agent migrated from the host holding the reference.

reference is deleted and the chain of references pointing to the agent's location is broken. It is conceivable that references are renewed in certain cases e. g., when the references are actively used. Forward references have two principle disadvantages: first they can grow long, and second the search algorithm has to catch up with the agent. Though the chain of references can be reduced in length e. g., by successively eliminating the middle host in a chain of three hosts.

Strategies based on forward references and dynamic forshortening of reference chains are frequently proposed, e.g. by Wojciechowski, Sewell, and Moreau [18, 9]; they are used in *Mole* for the purpose of orphan detection [2].

The four models differ in terms of which parties are active when agents shall be located, and which host acts as the principal name server. The differences between the four discussed models are summarized below:

Model	Active	Name server
Direct response	agent	agent origin
Buffered response	agent, requester	dedicated server
Heuristic search	requester	none
Forward references	requester	agent origin

Both direct response and heuristic search are unsuited as a general name service scheme because of their limitations noted above. Forward references are somewhat unreliable. Obviously, the chain cannot be re-established when a server in the chain crashes. The time to live of the chain of references is bounded by the least recent expiration time of any of its elements. Keeping the chain together by means of re-activation signals causes constant network traffic which is undesireable. A viable solution is the progressive shortening of the chain. Though this still produces undesired network traffic.

Let n be the number of hops of the agent when it is located. The chain is of length n so each time the agent shall be located n queries are required to follow the chain to its end. If the requester starts new searches at the last known location of the agent and the agent did m hops since the last query then at least $m + 1$ queries are required to locate the agent. Each of the servers the agent comes by has to store a forward reference.

In the case of buffered response n updates of the agent's name server entry are required plus one query to the name server. Additional searches cost one name server lookup plus m name server updates if the agent made m hops in the mean time. Only one reference needs to be stored by the name server.

Let c be the number of requesters querying for an agent, and let a be the number of agents in the network. Below we give a rough estimate on the space and time requirements of both models for the first lookup and any following lookups. T_u and T_l denote the number of network operations required for updating a location and for performing a lookup respectively. S denotes the number of references that must be stored in the network. n and m_i should be taken as mean values.

Model	T_u	T_l	S	
Buffered response	$a \cdot n$	c	a	1^{st}
	$a \cdot m_i$	c	0	i^{th}
Forward references	0	$c \cdot n$	$a \cdot n$	1^{st}
	0	$c \cdot m_i$	$a \cdot m_i$	i^{th}

The overhead T_u and S is required whether queries are started or not. The advantage of forward references over buffered response is that network traffic is generated only if a query is started. However, if queries are frequent and multiple requesters query for an agent's location then buffered response outperforms forward references easily. Furthermore the buffered response approach allows to allocate overheads primarily at dedicated servers while the forward references approach requires agent servers also to carry the burden of managing and answering queries of requesters. In summary, buffered response seems to be the most promising approach for mobile agents.

3 Threats to Name Services

Name services are targets of potential attacks on the mobile agent infrastructure. DoS (Denial of Service) attacks on a name server's network connection are particularly disruptive because all agents served by it are affected simultaneously. We do not further address this type of attack because it is not particular for name services, and must be addressed on the level of network infrastructure.

Name	Role description
Alice	The owner of an agent.
Bob	The operator of an agent server.
Carol	She looks up agent locations and sends a message to Alice's agent.
Mallet	An adversary.
Nick	He provides a name service for Alice.

Table 1. This table lists the names of the actors and their roles in the attacks described in this article.

The primary threats we discuss are DoS attacks specific to name services and attacks on the privacy of mobile agent owners. Please note that name services for mobile agents in principle do not differ from ordinary name services. Therefor some attacks described below apply to name services in general. However, the requirements to be fulfilled by name services and the defensive strategies have to account for mobility. We use the names and roles listed in Table 1 in order to illustrate some potential attacks below:

1. Mallet registers a huge number of chosen agent names with Nick which all point to the agent server of Bob. Mallet uses names likely to be used also by Alice and her friends. Therefor Carol and her friends all over the world all innocently send messages to Bob's server which chokes under the sudden load.
2. Mallet asks Nick to make an entry with the name of Alice's agent. When Carol ask for the location of Alice's agent Nick points to Mallet's server. So Carol sends her message to Mallet who reads it.
3. When Alice tries to register her agent with Nick he tells her that the name is already allocated to Mallet's agent. Alice has to choose another Name.
4. Mallot finds out the name of Alice's agent. He constantly asks Nick for the new location of that agent and learns where it goes. From this information, Mallet infers Alice's preferences.
5. Mallet opens up a name service; over time he finds out lots of interesting things about Alice and her friends. He sells this information to others.

If Mallet runs a name server then it is possible for him to make Alice believe that her agent took a particular route that it did not. However, if Alice becomes suspicious then she can easily verify the correctness of Mallet's responses. Mallet will not want to run this risk unless it is worth it. Consequently Alice should not reveal which agent is hers.

4 Ad Hoc Name Server Protocols

In our discussion of tracking mechanisms for mobile agents we implicitly assumed that the principal name server responsible for an agent is known. However, knowing an agent's name server is one of the challenges in creating scalable name services for mobile agents. The easiest approach is to let Alice choose a name server for her agent x. We refer to this approach as approach 1. So whenever Alice publishes the name of her agent she actually publishes (name$_x$, "Nick"). The obvious problem is that Mallet can replace Nick's name with his own whenever he comes across a reference to Alice's agent. He can even spread fake references. As a matter of consequence, Alice needs to authenticate her choice of name server. So Alice does the following:

> Alice publishes the digitally signed authenticator s_x := Sig$_A$("Alice", name$_x$, "Nick") where A is Alice's private key. Alice puts s_x and name$_x$ into the static part of her agent x and signs the static part. Then Alice sends x on its way.
> When x comes by Bob's server Bob verifies Alice's signature on the static part of x and the first two elements in s_x. Then Bob asks Nick to record (s_x, "Bob"). Nick verifies that the entity known as "Alice" actually signed s_x. If the verification is successful then he enters the mapping.

This scheme thwarts a number of attacks Mallet could launch. Mallet cannot fake names (required in attacks 1 and 3). He can still trigger a DCA (Distributed Coordinated Attack) on Bob as described in attack 1 by collecting lots of long lived authenticators, and sending them to Nick with Bob's name as the target. "Long lived" denotes authenticators which are actively used and which remain valid for a sufficiently long period of time. This problem can be solved in a variety of ways. We sketch a simple solution below:

> Before Alice sends x on its way she chooses a random cookie r_i, and sends (s_x, "Alice", r_i) to Nick. Nick makes sure that s_x is actually signed by Alice and is not already registered, and stores this information. Then Alice prepares x as described above and sends (x, r_i) to Bob.
> Bob verifies x and s_x as in the previous protocol. Then he chooses a random cookie r_{i+1} and sends ((s_x, "Bob", r_{i+1}), r_i) to Nick. Nick verifies that s_x and r_i he got from Bob are bitwise identical to the values he got from Alice and updates the stored entry with Bob's name and the new cookie.
> When x says it wants to migrate, Bob sends (x, r_{i+1}) to the destination specified by x.

The cookies serve as proof of permission for updating the name server entries. One agent server passes the authority on to the next one. Once the next agent server updated the location of x even the previous name server cannot modify the name server entry anymore. If Mallet wants to launch a DCA then collecting authenticators is not enough. In order to convince Nick he also has to collect the cookies used for the most recent update of each authenticator. For the same reason attack 2 does not work anymore. Even if Mallet runs an agent server and x comes by it then Mallet cannot keep his name in Nick's reference and let go of x at the same time. As soon as x migrates to Bob he will send his own name to Nick. If Nick refuses to perform the update because Mallet gave a false cookie to Bob then Bob will send an error report to Alice.

The protocol is fairly efficient because Nick has to verify Alice's signature on s_x only the first time. Subsequent references to s_x can be verified by a fast binary comparison.

Although there are a number of drawbacks. First, the approach described above leaves Alice's identity pretty exposed. If Mallet runs the name service rather than Nick then he has no difficulties to link all agents to their respective owners. Mallet also learns the itineraries of the agents served by him; based on this information he is able to assemble highly detailed reports on the interests of the agent's owners.

Second, the use of s_x as the name of x is cumbersome. Using digital signatures mandates a PKI (Public Key Infrastructure) for managing the associations between identities and public keys. Therefor the de facto standards PKCS#1 (Public Key Cryptography Standard), PKCS#7, ITU-Recommendation X.509, and LDAP [14, 13, 6] must be supported by the actors in our drama. The length of s_x, if represented as PKCS#7 signed data, likely exceeds 200 bytes.

The question that must be raised is whether there is an alternative approach that is both more efficient and more secure with regard to privacy issues. This approach must also be scalable with gradually increasing demand. Below we outline an approach that is both simple and fulfills these requirements.

5 Improvements

The difficulties arising in the protocol described above stem from the fact that Alice has to authorize her choice of name server. There is no way around this unless the name server responsible for an agent can be derived directly from that agent in an unambiguous way. Fortunately the latter is easily done as follows:

Let x be the agent of Alice and x' shall be the agent's *static* part. The static part of an agent comprises all its data, code, and configuration information that does not change during the agent's lifetime. It is a kind of *kernel* to which the

mutable parts of an agent must be bound. The kernel gives the agent its identity. In particular, we expect that the kernel also includes a digital signature of the static part, based on the private key of that agent's *owner*. This effectively renders the kernel unique.

Let H be a cryptographic hash function which is preimage and 2nd preimage resistant (see e.g. [7]). Collision resistance is not strictly necessary. In practice, the SHA (Secure Hash Algorithm) [5] is a suitable candidate. Let n be the length of the output of H in bits; for practical purposes let $n = 160$. We define $h_x = H(x')$ to be the name of x.

The name server of x is then identified by the first l bits of h_x with $0 < l < n$. Ideally the number of available name servers is a power of two, so 2^l are required. The value of l is adaptable to the global average number of agents in the Internet. The name servers should be spread over the globe. What we propose is in fact a global hashtable. We refer to this proposed approach as approach 2.

Consider $m = 2^l$ name servers. If the probability of a random break down of a name server is equally distributed and Alice makes a random pick among the name servers (approach 1) then her chances of picking one of k broken ones among m are k/m. If she takes approach 2 then her chances are also k/m given H has a reasonably equal distribution. If Alice knows that a particular name server is down and h_x is mapped to it then Alice can simply add a random nonce to x' and try again. The chances that she does not succeed after t tries is $(k/m)^t$. If half of the name servers are broken and Alice makes 4 tries then her chance of failing to generate an agent that is mapped to a working name server are 0.0625.

If Alice knows where x is about to go then Alice might want to choose a name server with good connectivity to the destinations of x in approach 1. In approach 2 the packets between Bob and Nick might circle the globe in the worst case. However, if x travels the globe and Alice makes any pick among the name servers then the effects are the same. If Alice still wants to pick one of a particular set of name servers in approach 2 then her chances of succeeding heavily depend on the actual number of name servers and the number of name servers she is willing to take. More precisely, her chances to pick one of k name servers among a total of m servers after t tries is $1 - ((m - k)/m)^t$. If 1% of the name servers are acceptable for Alice then her chances to pick one of those after 4 tries are less than 0.04, and less than 0.1 after 10 tries. With a probability of approximately 0.9 Alice will succeed after 230 tries. On a Pentium II 400MHz we measured less than 4 seconds for computing 230 SHA-1 hashes on about 10K of data.

The full name server protocol uses cookies as described above in order to prevent unauthorized updates of the name server entries. The protocol goes as follows:

> Alice prepares her agent x. She generates a random cookie r_i and puts it into the mutable part of x. Furthermore she generates a random string of n bits, and puts it into x'. Then Alice signs her agent and sends x to Bob.
>
> Bob verifies Alice's signature on x'. Then he generates a random cookie r_{i+1}, computes $h_x = H(x')$ and looks up Nick's address based on the first l bits of h_x. He sends (h_x, "Bob", r_{i+1}), r_i to Nick, and replaces r_i in x by r_{i+1}.
>
> Nick checks if h_x is already registered. If it isn't he simply enters (h_x, "Bob", r_{i+1}) in his database. If it is already registered then he makes sure that the received r_i equals the stored value before he updates his entry for h_x.

The chances of Mallet to guess the name of Alice's agent are negligible. Given a hash function output of $n = 160$ bits actually Mallet's chances to guess *any* agent name are negligible. If each currently living individual had 100 agents running permanently then Mallet's chances to accidently hit the name of any other agent is less than 2^{-120}.

For this reason Mallet cannot launch attack 3 unless he eavesdrops on Alice's network connection and registers h_x before Bob does. This is easily prevented. Neither can he launch attack 1 because his chances of guessing enough agent names a-priori are negligible.

Mallet cannot launch attack 2 unless he finds out the secret cookie used by Bob when he registers with Nick. This is also easily prevented.

Even if Mallet runs a name service (attack 5) he cannot link the names of agents to their respective owners. He cannot even tell whether two agents have the same owner.

Alice can further improve the security by sending a relay agent to a random agent server which allows her agent to listen on a network port. She directs any lookups to this relay agent which in turn asks the name server and returns its answer to Alice directly. This is crucial particularly for the last hop of x because Alice should send (h_x, null, 0), r_i to the name server in order to indicate the termination of x. The proxy agent guarantees that this last call does not enable Mallet to infer the identity of Alice after the last hop of her agent.

6 Additional Related Work

In addition to the articles already cited in Sections 1 and 2, we would like to mention the *Globe* system [17]. The *Globe* system is a distributed directory designed to support billions of references to mobile objects. However, the authors acknowledge that their hierarchical approach is not scalable enough to fulfill this goal due to the enormous storage demands and relatively large number of requests that must be handled by higher-level directory nodes. In order to overcome these problems, they propose to use the first n bits of an object's globally unique handle as the identifier of directory *subnodes*, which share the load on their directoy level. This approach equals the one we chose in order to provide scalability.

The notion of *put-ports* and *get-ports* in *Amoeba* [16, pp. 607] can be regarded as an analogy to the *implicit* naming scheme we propose for agents in this article. An *Amoeba* server process registers with the *Amoeba* kernel using a (private) *get-port*, and the according *put-port* is computed by the kernel by means of a one-way function. Processes that wish to communicate with the server process address packets to the *put-port*. This prevents intruders from impersonating server processes. The server process can be regarded as a mobile agent, and the *put-port* as its name. In contrast to *Amoeba* we do not allow free choice of the private *get-port*, but compute the *put-port* directly from the agent's unique kernel, which is required to prevent one agent from impersonating another.

7 Conclusions

In this article we first described four known approaches to tracking agent locations. Each approach yields a principal name server and several advantages and weaknesses. We concentrated on the two most promising approaches for a more detailed discussion and finally decided that buffered response is our choice of name service.

We then sketched a number of possible attacks on the name service infrastructure. As a first solution we presented an ad hoc protocol for securing the name service. This protocol showed several weaknesses with regard to anonymity of agent owners. We refined the protocol and proposed a second approach to agent naming which yields better protection against attacks on the privacy of agent owners. Furthermore the second protocol is more efficient and easier to implement than the first protocol. Another property of the second protocol is that agents cannot claim a fake identity since the agent name is computed directly from the agent's kernel.

It should be noted that neither the first nor the second protocol offers protection against unauthorized name service lookups. This is not the idea of a public global name service. However, the second protocol anonymizes agent names. The identity of an agent's owner can be determined only by parties which get a copy of the agent itself (because the agent carries a signature of its owner). In order to authenticate the itinerary of one's agents alternative approaches can be used [11].

A second weakness is that agent servers can still update the name service entry with a name other than their own. In principle this is easily prevented by requiring agent servers to authenticate themselves. We refrain from this because identification yields a considerable overhead. Further research should be directed at this problem.

Yet another problem are DoS attacks on the name servers by registering huge numbers of fake agent names. Name servers should adopt a strategy of limiting the frequency of updates accepted from a single source. In order to prevent IP spoofing attacks, name servers should enforce peer identification if the rate of registrations rises beyond a certain threshold. This will slow down name services but it will also prevent a general unavailability of a name server.

Cookies should be encrypted during transport in order to prevent eavesdropping. However, since encrypted transport of agents is supported by more and more agent systems this does not likely causes additional overhead.

Agent servers that operate in disconnected networks or in LANs (Local Area Networks) that do not require a global name service for mobile agents can bind to local dedicated name servers. In some cases it might even be advantageous to set up proxy name servers that offer local name resolution, and fall back to the global naming infrastructure if no match is found.

8 Acknowledgements

We would like to thank the reviewers for their comments which were most helpful in improving the quality of this article. We also would like to thank the participants of the ECOOP 2000 MOS Workshop for the inspiring discussions and pointing out the *Globe* system.

References

1. Yariv Aridor and Mitsuru Oshima. Infrastructure for mobile agents: Requirements and design. In Rothermel and Hohl [12], pages 38–49.
2. Joachim Baumann and Kurt Rothermel. The Shadow Approach: An orphan detection protocol for mobile agents. In Rothermel and Hohl [12], pages 2–13.

3. Michael Bursell, Richard Hayton, Douglas Donaldson, and Andrew Herbert. A Mobile Object Workbench. In Rothermel and Hohl [12], pages 136–147.
4. Wen-Shyen E. Chen and Chun-Wu R. Leng. A novel mobile agent search algorithm. In Kurt Rothermel and Radu Popescu-Zeletin, editors, *Mobile Agents (MA '97)*, volume 1219 of *Lecture Notes in Computer Science*, pages 162–173. Springer Verlag, Berlin Heidelberg, 1997.
5. FIPS180–1. Secure Hash Standard. Federal Information Processing Standards Publication 180–1, U.S. Department of Commerce/National Bureau of Standards, National Technical Information Service, Springfield, Virginia, April 1995. supersedes FIPS 180:1993.
6. International Organization for Standardization, Geneva, Switzerland. *Information technology – Open Systems Interconnection – The Directory: Authentication Framework*, nov 1993. ISO/IEC 9594-8, equivalent to ITU-T Rec. X.509, 1993.
7. Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. Discrete Mathematics and its Applications. CRC Press, New York, 1996. ISBN 0-8493-8523-7.
8. D. Milojicic, W. LaForge, and D. Chauhan. Mobile Objects and Agents (MOA). In *Proc. of the Fourth USENIX Conference on Object-Oriented Technologies and Systems (COOTS '98)*, April 1998.
9. Luc Moreau. Distributed directory service and message routing for mobile agents. Technical Report ECSTR M99/3, Department of Electronics and Computer Science, University of Southampton, November 1999.
10. Amy L. Murphy and Gian Pietro Picco. Reliable communication for highly mobile agents. In *Proc. First International Symposium on Agent Systems and Applications, and Third International Symposium on Mobile Agents (ASA/MA '99)*, pages 141–150. IEEE Computer Society Press, 1999.
11. V. Roth. Mutual protection of co-operating agents. In *Secure Internet Programming*, volume 1603 of *Lecture Notes in Computer Science*. Springer-Verlag Inc., New York, NY, USA, 1999.
12. K. Rothermel and F. Hohl, editors. *Proceedings of the Second International Workshop on Mobile Agents (MA '98)*, volume 1477 of *Lecture Notes in Computer Science*. Springer Verlag, Berlin Heidelberg, September 1998.
13. RSA Laboratories. Cryptographic message syntax standard. Public Key–Cryptography Standards 7, RSA Laboratories, Redwood City, CA, USA, 1993. Available at URL: `ftp://ftp.rsa.com/pub/pkcs/`.
14. RSA Laboratories. RSA Encryption Standard. Public Key–Cryptography Standards 1, RSA Laboratories, Redwood City, CA, USA, 1993. Available at URL: `ftp://ftp.rsa.com/pub/pkcs/`.
15. Peter Sewell, Pawel Wojciechowski, and Benjamin Pierce. Location-independent communication for mobile agents: a two-level architecture. Technical Report 462, Computer Laboratory, University of Cambridge, April 1999.
16. Andrew S. Tanenbaum. *Modern Operating Systems*. Prentice Hall, Inc., 1992.
17. M. van Steen, F. J. Hauck, P. Homburg, and A. S. Tanenbaum. Locating Objects in Wide-Area Systems. *IEEE Communications Magazine*, pages 104–109, January 1998.
18. Pawel Wojciechowski and Peter Sewell. Nomadic Pict: Language and Infrastructure Design for Mobile Agents. In *Proc. First International Symposium on Agent Systems and Applications, and Third International Symposium on Mobile Agents (ASA/MA '99)*, volume 2, pages 821–826, October 1999.