On MAS Scalability

Niek Wijngaards, Maarten van Steen, Frances Brazier
Department of Mathematics and Computer Science
Vrije Universiteit Amsterdam, The Netherlands
Email: {niek,steen,frances}@cs.vu.nl

Abstract

In open dynamic multi-agent environments the number of agents can vary significantly within very short periods of time. Very few (if any) current multi-agent systems have, however, been designed to cope with large-scale distributed applications. Scalability requires increasing numbers of new agents and resources to have no noticeable effect on performance nor to increase administrative complexity. In this paper a number of implications for techniques and management are discussed. Current research on agent middleware is briefly described.

1 Introduction

Agents, from an AI perspective, are autonomous, pro-active, reactive, and social entities [16]. They need to be able to communicate with other agents, and interact with the rest of the world (data repositories, objects in a virtual environment, etc.). They may be mobile, thus requiring a notion of location. They may or may not have a name or traceable owner. Agents, from a Computer Systems perspective, are (multi-threaded) processes (or sets of processes) that may migrate from one machine to another.

Ideally, multi-agent systems are hghly dynamic open systems, with an ever-changing population of agents: new agents emerge (or are created), existing agents die, move, learn/forget etc. The dynamics of such systems are hard to predict. The number of agents in large scale distributed applications such as e-business applications (virtual shopping malls and auctions), Internet-wide data warehouses, and navigation systems, can vary considerably over time. The systems need to be able to scale (in terms of the number of agents and available resources) almost immediately without noticeable loss of performance, or considerable increase in administrative complexity [21].

This problem of scalabilty is not an AI problem in itself. It is a problem with which the distributed computing community is still wrestling. A solution requires collaboration between these two disciplines. This paper addresses some aspects of this problem (security, for example, is not addressed). Section 2 provides an overview of the current status of multi-agent frameworks with respect to scalability. Section 3 discusses scaling techniques and agent management. Section 4 discusses different services with which agents can be located. Section 5 outlines our current work on an agent operating system.

2 Existing multi-agent frameworks

Scalability is an important, yet under-researched, aspect of agent platforms. This is partly due to the current status of agent technology. A large volume of research focuses on the development of intelligent agents that are able to communicate with other agents, interact with an external world, be autonomous, and react to their environment. A number of multi-agent frameworks/environments have been developed to construct multi-agent systems, but not for systems with (very) large numbers of agents.

One aspect of current research on multi-agent systems is that a large system is deemed to consist of hundreds of agents, maybe a thousand, but not millions. The claim that Auctionbot is scalable, for example, is supported by an experiment with only 90 agents [39]. Larger numbers of agents require scalable development frameworks and support environments.

The term “scalability” is not always used to refer to architecture, services and performance. In some cases it is used to refer to scalable functionality. For example, the SAIRE approach [23], claims to be scalable because it supports heterogeneous agents. Shopbot [9] claims to be scalable because its agents can adapt to understand new websites. In both cases, the term is *extensibile functionality* would seem to be more appropriate.

Researchers and developers of multi-agent frameworks are beginning to realise that scalability is an issue. A number of multi-agent frameworks (e.g. DECAF [17], InfoSleuth [22], April [20], AgentTcl [14], JAFMAS [7], Plangent [24] DESIRE [4]) do not seem to address the problem of scalability at all.

Other multi-agent frameworks rely on another framework to solve the problem of scalability. For example, scalablity in the CoABS (DARPA Control of Agent Based Systems) approach [32] assumes adequate support from computational grids in providing a plug-in backplane for agents [11].

In other multi-agent frameworks, aspects of scalability are specifically addressed. In ZEUS [38] scalability is defined to be the growth rate of the maximum communication load (as a function of the number of agents). Their conclusions are that the maximum communication load grows at worst linearly with the number of agents. This addresses a loss of performance problem, and is a step towards developing scalable multi-agent frameworks. In OAA (Open Agent Archiecture) [19] matchmaking agents are described which can handle larger number of agents. The RETSINA MAS infrastructure [31] is designed to support multi-agent systems that run on a number of LANs and to avoid single-point of failures (e.g., in agent name services).

Turner and Jennings [34] propose to (automatically) change the organization of agents in the multi-agent system to handle an increase in the population of a multi-agent system. For example, more middle agents or matchmakers are introduced to reduce overhead. Their approach is a possible step towards addressing administrative problems related to scalability.

None of the aformentioned approaches addresses minimizing the loss of performance as well as minimizing administrative overhead.

Research on specific services in multi-agent systems such as directory services also address scalability. The approach taken by Shehory [29] is an example in which agents locate agents based on each agent’s own caching lists of agents they know. The theoretical analysis is based on a population of size 10,000; no experiments have yet been conducted.

3 Managing scalability

Scalability problems generally manifest themselves as performance problems. This section describes well-known techniques that can tackle these problems. Administrative complexity in the form of agent management issues is also briefly discussed.

3.1 Scaling techniques

Three scaling techniques are discussed which may be used to minimise loss of performance: (1) hiding communication latencies, (2) distribution, and (3) replication.

Hiding communication latencies is applicable in the case of geographical scalability, that is, when an agent system needs to span a wide-area network. To avoid waiting for responses to requests that have been issued to remote agents or services the requesting agent is programmed to do other useful work. This approach does require that an agent can be interrupted when the expected response (if any) is to be delivered.

Distribution generally involves partitioning a (large) set of data into parts that can be handled by separate servers. A well-known example of distribution is the natural partitioning of the set of Web pages across the approximately 25 million Web servers that are currently connected through the Internet. Other examples of distribution include the vertical or horizontal partitioning of tables in distributed databases [25].

When considering large-scale networks like the Internet it becomes crucial to combine distribution with latency hiding. Unfortunately, this is not always possible, for example when an agent simply needs an immediate response.

A third, and widely applied technique is to place multiple copies of data sets across a network, also referred to as *replication*. The underlying idea is that by placing data close to where they are used, communication latency is no longer an issue, so that agent-perceived performance is high. Having multiple copies means that such performance is good for all agents, no matter where they are located.

Unfortunately, replication introduces a serious problem. Whenever a replica is updated, that replica becomes inconsistent with the other replicas. Matters become worse when multiple concurrent updates need to be carried out simultaneously, because all replicas have to be the same after all updates have been processed. Keeping replicas consistent introduces a consistency problem that can be solved only by means of global synchronization. However, global synchronization in a large-scale network is inherently nonscalable, as it requires communication between *all* parties that are to be synchronized.

The only solution to the consistency problem is to allow replicas to be somewhat out of synch with respect to updates. In other words, a weak consistency model is adopted. The form of, and to what extent weak consistency can be tolerated is highly application dependent. As a consequence, scalable multi-agent systems will need to support configurable and perhaps even adaptive replication strategies. No single strategy will show to be optimal under all conditions. Even for relatively simple systems such as the Web, differentiating strategies can make a lot of difference [26].

3.2 Agent Management

In a multi-agent system spread across a large-scale network with a vast number of agents possibly roaming from node to node, a massive management problem is caused solely by the scale of the system. Two related issues are discussed in this section: (1) the extent to which agents *know* each other (also referred to as referential coupling), and (2) orphaned agents, that is, agents that are no longer “in use” and cannot be traced back to an owner.

3.2.1 Referential Coupling

An important concept in agent-based distributed systems is the coupling between agents. Cabri et al. [5] make a distinction between temporal coupling and referential coupling (referred to as spatial coupling). Referential coupling is about whether agents explicitly refer to each other, or that they can otherwise remain anonymous. Temporal coupling deals with the issue whether two or more agents can communicate only if they are all up and running. When it comes to management issues, it is mainly referential coupling that is important.

In *referentially coupled systems*, agents explictly refer to each other. As a consequence, references need to be systemwide unique, but may also need to be true identifiers [37]. A true identifier is a reference that cannot be reused and is associated with exactly one agent. In addition, each agent can have at most one true identifier. Generating true identifiers is practically feasible. However, the real problem lies in the dereferencing process, that is, resolving a reference to the current address of the associated agent. If agents are not allowed to move, resolving a reference is relatively easy: its current address is encoded in the reference. For mobile agents, matters may become exceedingly difficult, as explained below.

In *referentially uncoupled systems*, agent communication is anonymous. Anonymity has the advantage that the system does not necessarily need to keep track of an agent’s current location. However, it does introduce the problem how communication and agent coordination should be realized. There are basically two approaches to support anonymous communication and coordination.

First, a publish/subscribe mechanism can be used, using what is known as subject-based addressing. In this approach, an agent is allowed to publish a message by attaching a subject to it. Agents that are interested in messages on a specific subject should subscribe to that subject. The underlying communication system ensures that published messages are delivered to their subscribers. The technique for this matching is either based on network-level multicasting (as in TIBCO/Rendezvous [33]), application-level multicasting [3], flooding [8, 18], or by means of a network of brokers (as in IBM MQSeries [15]). Obviously, each of these approaches has its own scalability problems.

A second approach to support anonymous communication and coordination is to make use shared dataspaces that are based on generative communication [13] such as JavaSpaces [12]. These shared dataspaces implement an associative memory that can also be used for searching and matching agents, as discussed below. Building efficient implementations for local-area networks is already difficult; large-scale wide-area multi-agent systems make matters worse [28]. There are no obvious solutions.

3.2.2 Orphaned Agents

Returning to referentially coupled systems, there is another intricate management problem that needs to be addressed within a large-scale MAS. If agents explicitly refer to each other, it becomes relatively easy to impose a (possibly hierarchical) structure by which one agent is responsible for managing other agents, notably its siblings. Management in this context generally refers to life cycle management: creating and destroying agents. However, what happens to an agent that is no longer referenced by other agents?

In many ways, this problem is akin to garbage collection in distributed systems [1], a notoriously hard problem to solve when there are many (passive) objects floating around. Considering that agents act autonomously, the situation becomes somewhat different. For example, a (possibly temporarily) anonymous autonomous agent that is actively collecting information should most likely be allowed to survive. However, it is probably not acceptable to allow such an agent to live forever in a more or less anonymous way.

A solution to collecting "garbage" agents, that is, agents that are no longer useful, is to make use of leases. A lease is essentially a contract that allows an agent to continue its work (even anonymously) until the lease expires. At that point, the host on which an agent is executing has the right to exterminate the agent. A lease-based approach to garbage collection in large-scale distributed systems has shown to be practically feasible. Agents should be allowed to extend their lease. Note this approach may also work in referentially uncoupled systems.

4 Scalable services

Many scalability problems in large-scale distributed systems, including agent-based systems, are related to limited scalability of searching and matching facilities. Unfortunately, some of these problems are inherently nonscalable and can be tackled only by considering the application for which agents are developed. In the following, the types of problems involved are discussed.

4.1 Names, Identifiers, Attributes, and Addresses

Naming plays an important role in any distributed system. Names are used in many different ways, but their main purpose is to facilitate matching and communication. Four different types of names are distinguished:

Human-friendly name: This type of name is a character string to be used by end users for looking up objects and agents. A typical example of such names are URLs as used in the Web.

Identifier: An identifier is generally a name intended to be read by machines only. As discussed above, identifiers are often used as unique references to objects and agents.

Address: An address is a name that specifies exactly where and how an object or agent can be contacted. As such, it describes a location, but often implicitly also the protocol through which communication can take place. In the Domain Name System (DNS), a human-friendly name is translated to a network-level address associated with the Internet protocol (IP).

Attribute: An attribute is a descriptive name, associated with one or more values, and is used to describe a property of an object or agent. Attributes are mostly used to assist searching for an object using only partial information on the properties that the returned object should have.

Given these types of names, scalable naming in multi-agent systems is generally concerned with two issues. The first issue is how to efficiently resolve a human-friendly name or an identifier to an address. The second issueis how to accomplish efficient attribute-based searching or matching. Different scalability problems relate to each of these issues.

4.2 Scalability in Naming Services

A naming service such as DNS is used to resolve a human-friendly name to an address. DNS can scale to millions of names by physically distributing the name space across multiple servers, and applying a simple name resolution mechanism. As an example, consider resolving the name *www.cs.vu.nl*. This name is handled by at least three name servers. A DNS root name server is capable of resolving the name *nl*, for which it returns the address of the name server that can handle names in the *nl* domain. Using this address, and agent can request name resolution of *www.cs.vu*, which will return the address of the name server handling the *vu* domain. In turn, the *vu* name server can be asked to resolve *www.cs* for which it returns the address of the *cs* name server. The latter, finally, knows the address of the Web server named *www*.

True scalability of DNS comes from the fact that name-to-address mappings are extensively cached by all name servers. In other words, DNS internally makes extensive use of replication. This replication does not lead to inconsistencies as described above, for in most cases name-to-address mappings hardly ever change. In other words, updates are rare compared to lookups.

Unfortunately, when dealing with mobile agents, DNS cannot be used to locate an agent because name-to-address mappings are no longer stable. Instead, whenever an agent moves to another location, it update its address, thereby changing the name-to-address mapping. Other solutions are necessary.

4.3 Scalability in Location Services

As an alternative to using a naming service, specialized location services have been constructed. A location service is tailored to maintain identifier-to-address mappings, and assumes that these mappings change regularly. There are various approaches to efficiently locating mobile agents [27], but only few can actually scale worldwide and can support arbitrarily migrating agents.

As it turns out, special attention needs to be paid to adding an efficient location service to a large-scale multi-agent system. In our own work on wide-area distributed systems, a distributed search tree has been constructed that can dynamically adapt itself to the migration pattern of an individual agent [35]. A coupling between human-friendly names and addresses of mobile agents is described in [2]. Any naïve solution to locating mobile agents in a large-scale multi-agent system that needs to support highly mobile agents can only fail. It is beyond the scope of this paper to go into further details, but the interested reader is referred to [27] for further information.

4.4 Scalability in Directory Services

Naming and location services are difficult to scale, but nevertheless solutions exist that can be used in large-scale multi-agent systems. Matters become harder in the case of attribute-based searching and matching. Scalable attribute-based searching and matching falls into the category of building scalable directory services [30]. The canonical example of a wide-area directory service is LDAP (Lightweight Directory Access Protocol). LDAP servers form a simplified implementation of X.500 directory services and currently deployed in modern distributed systems such as those based on Windows 2000 [6].

In its simplest form, an agent submits a query to a directory service in the form of a boolean expression in which each term is an *(attribute, value)* pair. The service returns a list of references to objects (or agents) that match the query. In other words, a returned object has its attribute values set according to the query as submitted by the agent. What makes a directory service so difficult to scale, is that to construct the list of matching objects, it is, in principle, necessary to search the *entire set* of objects that are registered by the service.

To circumvent such an exhaustive search, the only approach that can be followed is to build an index of mappings from attribute values to object references. Only in this way will it become possible to immediately identify the objects that match a query. Unfortunately, building and maintaining such an index on a worldwide scale is infeasible.

Solutions to general-purpose worldwide scalable directory services do not exist. At best, specialized directory services can be built that restrict the type of query that can be submitted, or limit the set of *(attribute, value)* pairs. An example of such a limited directory service is JavaSpaces, but as argued above, even there severe scalability problems exist.

Again, a difficult and challenging problem exists, for which presumably only application-specific solutions will work.

5 Agentscape

AgentScape is a focus of current research. It is a system that currently being designed as a worldwide distributed, scalable, secure, and extensible agent framework. It aims to provide support in two ways. First, support is provided on the level of a basic agent operating system. Second, support is provided by services, such as location and directory services, automated creation of agents, and management of agents, objects, locations and groups. AgentScape provides basic building blocks which can be extended and build upon by application developers.

5.1 AgentScape Operating System

AgentScape is a basic agent middleware system, intended to be usable for a wide range of multi-agent applications. As middleware, it offers primitives on the level of agents, shielding application developers from details at lower levels. In a sense, AgentScape is similar to UNIX. Within UNIX, everything is a file, on which operations are defined. Within AgentScape, two main concepts are distinguished: agents and objects. An agent is an active process, while an object is passive. Operations are defined on agents, akin to file operations in UNIX: move (`mv`), change owner (`chown`), change group (`chgrp`), change security modus (`chmod`), create, remove (`rm`), etc. Similar operations are defined on objects. Unique to AgentScape is the use of objects that are physically distributed across multiple machines, and that encapsulate their own distribution strategy. These objects are adopted from the Globe wide-area distributed system [36].

An important issue for AgentScape is that its model of agents and objects enable scalable solutions. In our approach, agents are expected to be mobile and can be implemented in different ways. This approach allows for implementing applications that require a high degree of interoperability across heterogeneous platforms. For a similar reason, our objects have self-managing capabilities. In contrast, most distributed-object models are based on remote objects in which the object state is not distributed, and is managed by the server the object is located [10]. Clients are only provided transparent access to an object through a proxy.

A default version of AgentScape will be present in every application using our middleware (akin to the presence of an operating system kernel). However, the middleware itself will be highly extensible to allow for application-specific solutions.

5.2 AgentScape Services

An agent operating system intended to be used in a worldwide setting needs services, for example, to enable retrieval of agents. Specific directory services are provided, with which agents, distributed (and possibly replicated) objects, and groups of agents or objects can be found. Another service is a multi-agent factory with makes automated agent creation and modification possible. Finally, management services are provided to reactively and pro-actively control agents, objects, locations, and groups in AgentScape. The challenge for these services is that they need to be scalable across a worldwide network and they can support vast numbers of agents and objects.

6 Final Remarks

In this paper, we have addressed some important design issues for scalable multi-agent systems. To build worldwide distributed agent-based systems, we argue that the real challenges lie in solving the scalability problems mentioned in this paper. Unfortunately, there are no obvious solutions, so what is needed is middleware that can support a myriad of solutions, each probably tailored to specific application domains.

If these problems are not solved, the road to large-scale agent deployment will be exceedingly difficult to follow.

7 Acknowledgements

This work was supported in part by NLnet Foundation.

References

[1] S. Abdullahi and G. Ringwood. "Garbage Collecting the Internet: A Survey of Distributed Garbage Collection." *ACM Comput. Surv.*, 30(3):330–373, Sept. 1998.

[2] G. Ballintijn, P. Verkaik, E. Amade, M. van Steen, and A. S. Tanenbaum. "A Scalable Implementation for Human-Friendly URIs." Technical Report IR-466, Vrije Universiteit, Department of Mathematics and Computer Science, Oct. 1999.

[3] G. Banavar, T. Chandra, B. Mukherjee, J. Nagarajarao, R. Strom, and D. Sturman. "An Efficient Multicast Protocol for Content-based Publish-Subscribe Systems." In *Proc. 19th Int'l Conf. on Distributed Computing Systems*, Austin, TX, June 1999. IEEE.

[4] F. Brazier, B. D. Keplicz, N. Jennings, and J. Treur. "DESIRE: Modelling Multi-Agent Systems in a Compositional Formal Framework.." *International Journal of Cooperative Information Systems, special issue on Formal Methods in Cooperative Information Systems*, 6:67–94, 1997.

[5] G. Cabri, L. Leonardi, and F. Zambonelli. "Mobile-Agent Cooordination Models for Internet Applications." *Computer*, 33(2):82–89, Feb. 2000.

[6] D. Chappell. *Understanding Windows 2000 Distributed Services*. Microsoft Press, Redmond, WA, 2000.

[7] D. Chauhan. "Developing coherent multiagent systems using jafmas." In *Proc. International Conference on Multi Agent Systems, ICMAS98*, Cite des Sciences - La Villette, Paris, France, July 1998.

[8] A. Demers et al. "Epidemic Algorithms for Replicated Data Management." In *Proc. Sixth Symp. on Principles of Distributed Computing*, pp. 1–12, Vancouver, Aug. 1987. ACM. Also in Operating Systems Review, 22(1):8-32, Jan. 1988.

[9] R. B. Doorenbos, O. Etzioni, and D. S. Weld. "A Scalable Comparison-Shopping Agent for the World-Wide Web." In W. L. Johnson and B. Hayes-Roth, (eds.), *Proc. Proceedings of the First International Conference on Autonomous Agents (Agents'97)*, pp. 39–48, Marina del Rey, CA, USA, 1997. ACM Press.

[10] W. Emmerich. *Engineering Distributed Objects*. John Wiley, New York, 2000.

[11] I. Foster and C. Kesselman, (eds.). *Computational Grids: The Future of High Performance Distributed Computing*. Morgan Kaufman, San Mateo, CA., 1998.

[12] E. Freeman, S. Hupfer, and K. Arnold. *JavaSpaces, Principles, Patterns and Practice*. Addison-Wesley, Reading, MA., 1999.

[13] D. Gelernter. "Generative Communication in Linda." *ACM Trans. Prog. Lang. Syst.*, 7(1):80–112, 1985.

[14] R. Gray, D. Kotz, G. Cybenko, and D. Rus. "Agent Tcl." In W. Cockayne and M. Zyda, (eds.), *Proc. Mobile Agents: Explanations and Examples*. Manning Publishing, 1997.

[15] IBM Inc. *IBM MQSeries Publish/Subscribe User's Guide*, 7th edition, Nov. 2000.

[16] N. Jennings and M. Wooldridge. "Intelligent agents: theory and practice." *The Knowledge Engineering Review*, 10(2):115–152, 1995.

[17] J. G. Keith. "Towards a Distributed, Environment-Centered Agent Framework.

[18] M.-J. Lin and K. Marzullo. "Directional Gossip: Gossip in a Wide-Area Network." In *Third European Dependable Computing Conference*, volume 1667 of *Lect. Notes Comput. Sc.*, pp. 364–379. Springer-Verlag, Berlin, Sept. 1999.

[19] D. Martin, A. Cheyer, and D. Moran. "The Open Agent Architecture: a framework for building distributed software systems." *Applied Artificial Intelligence*, 13(1/2):91–128, 1999.

[20] F. McCabe and K. Clark. “April: Agent Process Interaction Language.” In N. Jennings and M. Wooldridge, (eds.), *Proc. Intelligent Agents*, volume 890 of *Lecture Notes in Computer Science*. Springer-Verlag, 1995.

[21] B. Neuman. “Scale in Distributed Systems.” In T. Casavant and M. Singhal, (eds.), *Readings in Distributed Computing Systems*, pp. 463–489. IEEE Computer Society Press, Los Alamitos, CA., 1994.

[22] M. Nodine, B. Perry, and A. Unruh. “Experience with the InfoSleuth agent architecture.” In *Proc. Proceedings of the AAAI-98 Workshop on Software Tools for Developing Agents*, 1998.

[23] J. B. Odubiyi, D. J. Kocur, S. M. Weinstein, N. Wakim, S. Srivastava, C. Gokey, and J. Graham. “SAIRE–a scalable agent-based information retrieval engine.” In *Proc. Proceedings of the first international conference on Autonomous agents*, pp. 292–299, Marina del Rey, CA USA, Feb. 1997.

[24] A. Ohsuga, Y. Nagai, Y. Irie, M. Hattori, and S. Honiden. “Plangent: An Approach to Making Mobile Agents Intelligent.” *IEEE Internet Computing*, 1(4), July 1997.

[25] T. Özsu and P. Valduriez. *Principles of Distributed Database Systems*. Prentice Hall, Upper Saddle River, N.J., 2nd edition, 1999.

[26] G. Pierre, I. Kuz, M. van Steen, and A. Tanenbaum. “Differentiated Strategies for Replicating Web Documents.” *Comp. Comm.*, 24(2):232–240, Feb. 2001.

[27] E. Pitoura and G. Samaras. “Locating Objects in Mobile Computing.” *IEEE Trans. Know. Data Eng.*, 12, 2000. To appear.

[28] A. Rowstron. “Run-time Systems for Coordination.” In A. Omicini, F. Zambonelli, M. Klusch, and R. Tolksdorf, (eds.), *Coordination of Internet Agents: Models, Technologies and Applications*, pp. 78–96. Springer-Verlag, Berlin, Aug. 2001.

[29] O. Shehory. “A Scalable Agent Location Mechanism.” In *Proc. Lecture Notes in Artificial Intelligence, Intelligent Agents VI*, 1999.

[30] B. Sheresh and D. Sheresh. *Understanding Directory Services*. New Riders, Indianapolis, IN, 2000.

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

[32] C. Thompson, T. Bannon, P. Pazandak, and V. Vasudevan. “Agents for the Masses.” In *Proc. Agent-Based High Performance Computing - Problem Solving Applications and Practical Deployment at Autonomous Agents 1999*, Seattle, Washington, USA, May 1999.

[33] TIBCO Software Inc., Palo Alto, CA. *TIB/Rendezvous Concepts, Release 6.4*, Oct. 2000.

[34] P. J. Turner and N. R. Jennings. “Improving the Scalability of Multi-agent Systems.” In *Proc. Proc. 1st International Workshop on Infrastructure for Scalable Multi-Agent Systems*, 2000.

[35] M. van Steen, F. Hauck, P. Homburg, and A. Tanenbaum. “Locating Objects in Wide-Area Systems.” *IEEE Commun. Mag.*, 36(1):104–109, Jan. 1998.

[36] M. van Steen, P. Homburg, and A. Tanenbaum. “Globe: A Wide-Area Distributed System.” *IEEE Concurrency*, 7(1):70–78, Jan. 1999.

[37] R. Wieringa and W. de Jonge. “Object Identifiers, Keys, and Surrogates–Object Identifiers Revisited.” *Theory and Practice of Object Systems*, 1(2):101–114, 1995.

[38] P. D. Wilde. “Stability, Fairness and Scalability of Multi-Agent Systems.

[39] P. R. Wurman, M. P. Wellman, and W. E. Walsh. “The Michigan Internet AuctionBot: A configurable auction server for human and software agents.” In K. P. Sycara and M. Wooldridge, (eds.), *Proc. Proceedings of the 2nd International Conference on Autonomous Agents (Agents’98)*, pp. 301–308, New York, 9–13, 1998. ACM Press.