An Algorithmic Theory of Mobile Agents

Evangelos Kranakis[1] and Danny Krizanc[2]

[1] School of Computer Science, Carleton University, Ottawa, ON, Canada.
[2] Department of Mathematics and Computer Science, Wesleyan University, Middletown, Connecticut 06459, USA.

Abstract. Mobile agents are an extension of multiagent systems in which the agents are provided with the ability to move from node to node in a distributed system. While it has been shown that mobility can be used to provide simple, efficient, fault-tolerant solutions to a number of problems in distributed computing, mobile agents have yet to become common in mainstream applications. One of the reasons for this may be the lack of an algorithmic theory which would provide a framework in which different approaches can be analyzed and the limits of mobile agent computing explored. In this paper we attempt to provide such an algorithmic theory.

1 Introduction

The concept of an agent working on behalf of another entity is a simple yet powerful abstraction that has been found useful in many areas of computing. In certain applications adding the capability of movement to an agent can lead to further simplifications and efficiencies. Consider, for example, the following scenarios:

- *Network Maintenance.* In a heterogeneous network it is necessary to regularly provide nodes with software updates, check for security vulnerabilities, etc. A simple approach to this would be to have an agent (or team of agents) regularly visit the nodes to determine what maintenance is required and to perform it.
- *Electronic Commerce.* In some situations the success of a given transaction requires the near simultaneous success of multiple transactions. For example, when preparing for a trip one may be purchasing airline tickets, making hotel reservations and scheduling meetings. A mobile agent can move between applications making sure that all transactions are ready before committing to any.
- *Intelligent Search.* When searching for information across multiple sources it is often the case that queries must be adapted depending on the answers received. An agent with the ability to filter information locally and adapt its behavior while moving between sources is potentially more efficient than one that always has to return to the user for guidance.

- *Robotic Exploration.* In a potentially dangerous environment it makes sense for robots to be the first to explore a region. A simple and potentially cheap solution is to have a team of small communicating robots (agents) cooperatively explore rather than one expensive human-controlled robot.

In this paper we concentrate on modeling agents as developed in distributed systems research (see Chapter 13 of [9]) though much of what we discuss could be applied in other domains such as artificial intelligence (e.g., intelligent multiagent systems [20]), robotics (e.g., autonomous mobile robots [13]), computational economics (e.g., agent-based economic modelling [17]) and networking (e.g., active networks [16]). We first informally describe what we mean by a mobile agent system and discuss the potential advantages and disadvantages of such systems. We then develop a framework for an algorithmic theory of mobile agents. Finally, we show how the theory can be applied to analyze the problem of achieving agent rendezvous in a network.

1.1 What is a Mobile Agent?

We imagine a mobile agent to be a software entity endowed with the following properties:

- *Autonomy.* As is the case for real world agents such as travel agents, software agents should work with some degree of independence from their creator. They should be able to make at least some decisions without the need to consult a central authority.
- *Mobility.* In the case of mobile agents we insist that they have the ability to move from node to node in a distributed system. When such an agent moves it is assumed that it encapsulates some or all of its state to move with it.

Beyond the above, a number of researchers include the following attributes in their definition of a mobile agent:

- *Interactivity.* Obviously a agent must be able to interact with its environment, to make queries of nodes it visits, to report its findings, etc. But in many (possibly most) applications we imagine that more than one agent is present and the agents themselves are able to interact. Again in most instances this is likely to be cooperative behavior but competitive behavior is also possible. The exact form of this interaction depends upon the system but usually involves some sort of communication either by means of message passing or shared memory.
- *Intelligence.* The usefulness of an agent increases significantly with its ability to adapt to new situations, to learn from previous experience and to model correctly the intentions of the user who created it as well as those of the agents it encounters.

It is our goal to develop a flexible framework in which systems exhibiting any subset of the above properties can be analyzed.

1.2 Why Mobile Agents?

The applications one has in mind for mobile agents can generally be solved by traditional distributed computing approaches so why use them? While not a panacea, it can be argued that they offer a number of advantages over the standard solutions including:

- *Efficiency.* Assuming that the agents are sufficiently compact (program plus state) they offer potential savings in both latency and network bandwidth. In a situation where n sites must be visited in sequential order, where for instance the output from one site is used as part of the input to the next, a mobile agent can perform the task by moving along an n edge cycle incurring the cost of n communication steps whereas the communication pattern of a centrally located agent would be a star with $2n$ communication steps (to and from each site) required. In a situation where parallel access to the sites is possible then a team of mobile agents (in this case sometimes referred to as *clones*) can visit all of the sites faster than a single stationary agent.
- *Fault-tolerance.* In situations where a user has limited or even intermittent connectivity to the network (e.g., mobile devices) a mobile agent may overcome this deficit by acting on behalf of the user during blackout periods and returning useful information when connectivity returns. In situations where nodes may go down on a regular basis with limited notice a mobile agent can potentially move to another node and continue operating.
- *Flexibility.* It is generally easy to add features to agents that allow them to adapt their behavior to new conditions. More sophisticated agents may be designed to incorporate learning from past experience.
- *Ease of Use.* In many situations it is natural both from the perspective of the user and the programmer to imagine that they are dealing with autonomous agents. In some instances this improves the user's experience of the application. In other instances an agent-based paradigm makes the system's design and implementation easier to perform. While these benefits are hard to quantify they can be as important as the above.

In order to evaluate (at least theoretically) the above claims, especially those concerning efficiency and fault-tolerance, a model for analyzing the behavior of mobile agent algorithms is needed.

1.3 Whither Mobile Agents?

While many have touted the advantages of mobile agents and numerous mobile agent systems have been developed [3, 10, 12, 18, 19], they have yet to become a ubiquitous element of distributed systems. The reasons cited for this are many. They are said to lack a so-called "killer app" or to suffer from the "who goes first" phenomenon. A common objection to introducing mobile agents is security. There are many who believe that opening up your network to the "invasion" of potentially harmful mobile agents is not worth the advantages they may provide.

On the other hand, much work has been done to insure that mobile agents can be implemented in a secure fashion [15].

But perhaps the most common objection to mobile agent systems is that anything that can be achieved by mobile agents can just as easily be achieved using traditional means such as static agents. The counter argument to this is that no one has ever claimed that mobile agents were required to solve any particular problem only that they would provide a potentially more efficient and fault-tolerant solution to many common problems. But still some object that the resulting mobile agents will be too complex to realize the possible savings. While there has been some experimental work that attempts to verify that such savings exist [3–5], it is our contention that at least in part what is needed is an algorithmic model in which one can prove that a mobile agent solution achieves given complexity bounds and can thus provably provide the claimed efficiencies.

2 An Algorithmic Model for Mobile Agents

Work on the design and analysis of algorithms proceeds within the confines of a given algorithmic model. For example a popular model in which sequential algorithms are analyzed is the standard RAM model of computing. For parallel algorithm analysis a number of models, such as the PRAM, are used. For each paradigm an appropriate algorithmic model is developed. In general, a model is an abstraction which attempts to capture the most important aspects of a computational process. It consists of a description of allowable operations (or transitions) that can be performed by the process. For example the RAM model allows for read/write operations, arithmetic operations, etc. Once this is established one generally defines a set of measurable resources of interest, e.g., time (number of primitive operations), space (number of registers or potential states), etc. At this point one is ready to analyze algorithms for well-defined problems (often expressed as input-output conditions). Assuming the model captures the computation sufficiently accurately it can be used to:

– analyze the complexity (the amount of a given resource used) of different algorithms for a problem in order to determine which is most efficient, and
– determine lower bounds on the complexity of any algorithm for a given problem or relate the complexities of different problems.

In order to model mobile agents we must model both the agents themselves and the networks that host them. Rather than describe one model we present a framework for a class of related models for both hosts and agents among which one may choose a model to be used depending upon the application one has mind.

2.1 Mobile agents

We are interested in modeling a set of software entities that act more or less autonomously from their originator and have the ability to move from node to node in a distributed network maintaining some sort of state with the nodes of the network providing some amount of (possibly longterm) storage and computational support. Either explicitly or implicitly such a mobile (software) agent is most often modeled using a finite automaton consisting of a set of states and a transition function. The transition function takes as input the agent's current state as well as possibly the state of the node it resides in and outputs a new agent state, possible modifications to the current node's state and a possible move to another node. In some instances we consider probabilistic automata which have available a source of randomness that is used as part of their input. Such agents are referred to as *randomized* agents.

An important property to consider is whether or not the agents are distinguishable, i.e., if they have distinct labels or identities. Agents without identities are referred to as *anonymous* agents. Anonymous agents are limited to running precisely the same program, i.e., they are identical finite automata. As the identity is assumed to be part of the starting state of the automaton, agents with identities have the potential to run different programs.

The knowledge the agent has about the network it is on and about the other agents can make a difference in the solvability and efficiency of many problems. For example, knowledge of the size of the network or its topology or the number of and identities of the other agents may be used as part of the agent's program. If available to the agents, this information is assumed to be part of its starting state. (One could imagine situations where the information is made available by the nodes of the network and not necessarily encoded in the agent.)

An important consideration for the case of teams of agents is how they interact. For example, are agents able to detect the presence of other agents at a given node? Assuming that the agents are designed to interact, the method through which they communicate is an important aspect of any model. For example one might consider a case where agents have the ability to read the state of other agents residing at the same node. Or one might only allow communication via a shared memory space or via message passing. Other properties that may be considered include whether or not the agents have the ability to "clone" themselves, i.e., produce new copies of themselves with the same functionality and whether or not they have the ability to "merge" upon meeting (sometimes referred to as "sticky" agents).

2.2 Distributed networks

The model of a distributed network is essentially inherited directly from the algorithmic theory of distributed computing (see for example [14]). We model the network by a graph whose vertices comprise the computing nodes and edges correspond to communication links.

The nodes of the network may or may not have distinct identities. In an *anonymous* network the nodes have no identities. In particular this means that an agent can not distinguish two nodes except perhaps by their degree. The outgoing edges of a node are usually thought of as distinguishable but an important distinction is made between a globally consistent edge-labeling versus a locally independent edge-labeling. A simple example is the case of a ring where clockwise and counterclockwise edges are marked consistently around the ring in one case, and the edges are arbitrarily - say by an adversary - marked 1 and 2 in the other case. If the labeling satisfies certain coding properties it is called a *sense of direction* [7]. Sense of direction has turned out to greatly effect the solvability and efficiency of solution of a number of problems in distributed computing and has been shown to be important for the study of mobile agents as well.

Networks are also classified by how they deal with time. In a synchronous network there exists a global clock available to all nodes. This global clock is inherited by the agents. In particular it is usually assumed that in a single step an agent arrives at a node, performs some calculation, and exits the node and that all agents are performing these tasks "in sync". In an asynchronous network such a global clock is not available. The speed with which an agent computes or moves between nodes, while guaranteed to be finite, is not a priori determined.

We have to consider the resources provided by the nodes to the agents. All nodes are assumed to provide enough space to store the agent temporarily and computing power for it to perform its tasks. (The case of malicious nodes refusing agents or even worse destroying agents - so-called *blackholes* - is also sometimes considered.) It is also assumed that the nodes will transport the agents to other nodes upon request. Beyond these basic services one considers nodes that might provide some form of long-term storage, i.e., state that is left behind when the agent leaves. This long-term storage may or may not be shared among all agents using the services of the node. So for example this memory might be best thought of as a *whiteboard* on which an agent can leave messages for themselves or for other agents. A further service the node may provide to the agents is mechanism for sending and/or receiving messages via message passing.

Finally, when analyzing fault-tolerance one has to consider how a host network component might fail. Again, here we inherit the standard network fault models considered in distributed computing such as crash failures, omission failures, Byzantine failures, etc. One might also consider failures that do not effect the working of the network but only the agent subsystem, e.g., loss of shared data.

2.3 Resource measures

For a given choice of agent plus network model there are a number of resources of interest for which one can define a complexity measure. Of paramount concern are measures that reflect the time and bandwidth efficiency of a given algorithm. In the synchronous setting it is clear that to measure time one should use the assumed global clock. In an asynchronous setting things are not so clear though in most instances authors choose to evaluate what the worst case time would be assuming that time proceeded synchronously. The total bandwidth consumed by the agent depends upon its size as well as the number of moves it makes during an execution of its algorithm. Generally the size of an agent is identified with the number of bits required to encode its states, i.e, it is proportional to the log base two of the number of possible states. If the agent sends messages then the size and number of these messages must also count towards any measure of its bandwidth. Other complexity measurements of interest include the size of shared memory required at each node assuming the agents communicate via shared memory, the number of random bits used by a randomized agent and the number of and kind of faults an algorithm can successfully deal with.

3 An Example: Randomized Rendezvous on the Ring

A natural problem to study for any multiagent mobile system is that of rendezvous. Given a particular agent model and network model a set of agents distributed arbitrarily over the nodes of the network are said to *rendezvous* if after running their programs after some finite time they all occupy the same node of the network at the same time. As is often the case, researchers are interested in examining cases that expose the limits of the problem being studied. For rendezvous the simplest interesting case is that of two agents attempting to rendezvous on a ring network. Of special interest is the highly symmetric case of anonymous agents on an anonymous network. In particular below we consider the standard model for an anonymous synchronous oriented ring [2] where

1. the nodes have no identities, i.e., the agents can not distinguish between the nodes,
2. the computation proceeds in synchronous steps,
3. the edges of each node are labeled `left` and `right` in a consistent fashion.

We model the agents as probabilistic finite automata $A =< S, \delta, s_0 >$ where S is the set of states of the automata including s_0 the initial state and the special state `halt`, and $\delta : S \times C \times P \rightarrow S \times M$ where $C = \{H, T\}$ represents a random coin flip, $P = \{\texttt{present}, \texttt{notpresent}\}$ represents a predicate indicating the presence of the other agent at a node, and $M = \{\texttt{left}, \texttt{right}\}$ represents the potential moves the agent may make. During each synchronous step, depending upon its current state, the answer to a query for the presence of the other agent, and the value of a independent random coin flip with probability of `heads` equal to .5, the agent uses δ in order to change its state and either move across the edge labeled `left` or `right`. We assume that the agent halts once it detects the presence of the other agent at a node.

The first question one may ask concerning this instance of rendezvous is whether or not it is solvable. It is fairly easy to see that if the two agents start at an odd distance apart on an even size ring they can never rendezvous in the above model as they are forced to move on each step and therefore will remain an odd distance apart forever. There are number of ways to fix this, the easiest perhaps being to add a third option to M of `stay`. For simplicity in the analysis below we will instead assume that they are an even distance apart on an even size ring.

For solvable instances of rendezvous one is interested in comparing the efficiency of different solutions. Much of the research focuses on the number of moves required to rendezvous or the expected number in the case of randomized agents (where the expectation is taken over the possible sequences of coin flips). In the synchronous setting the number of moves is equivalent to the time and is measured via the global clock. (In some situations, it makes a difference if the agents begin their rendezvous procedure at the same time or there is possible delay between start times. Here we will assume a synchronous start.) Also of interest is the size of the program required by the agents to solve the problem. This is referred to as the memory requirement of the agents and is considered to be proportional to the base two logarithm of the number of states required by the finite state machine encoding the agent. Ideally one would like to design an agent whose size is constant independent of the size of the ring and which performs rendezvous in expected linear time (as in the worst case the agents are linear distance apart initially). As we shall see, achieving both goals simultaneously is not possible in this case.

3.1 Random walk algorithm

Many authors have observed that rendezvous may be solved by anonymous agents on an anonymous network by having the agents perform a random walk. The expected time to rendezvous can be shown to be a (polynomial) function of the (size of the) network and is related to the cover time of the network. (See [11] for definitions relating to random walks. See [6] for an analysis of the meeting time for random walks.)

For example consider the following algorithm for rendezvous on the ring:

```
1. Repeat until other agent present:
2. If heads move right else move left
```

If we let E_d be the expected time for two agents starting at an (even) distance d on an a ring of (even) size n to rendezvous using the above algorithm it is easy to see that $E_0 = 0$, and $E_{n/2} = 1 + (1/2)E_{n/2} + (1/2)E_{n/2-2}$. The latter equation gives rise to the recurrence

$$E_{n/2} = 2 + E_{n/2-2}. \tag{1}$$

More generally, in executing the algorithm one of the following three cases may occur. The two mobile agents make a single step and either move in the same direction with probability 1/2, or in opposite direction either towards each other with probability 1/4 or away from each other with probability 1/4. From this we derive the identity

$$E_d = 1 + (1/2)E_d + (1/4)E_{d-2} + (1/4)E_{d+2}, \tag{2}$$

for $d = 2, 4, \ldots, n/2 - 2$. (Note that the case $d = n/2$ is special in that they are always at most distance $n/2$ apart.) Substituting $d + 2$ for d in Identity 2 and solving the resulting equation in terms of E_d we derive that for $d \geq 4$,

$$E_d = 2E_{d-2} - E_{d-4} - 4. \tag{3}$$

The initial condition $E_0 = 0$ and Identity 3 yield $E_4 = 2E_2 - 4$. More generally, we can prove the following identity for $2d \leq n/2$,

$$E_{2d} = dE_2 - 2d(d-1). \tag{4}$$

We prove by induction that there are sequences a_d, b_d such that

$$E_{2d} = a_d E_2 - 4b_d.$$

Indeed,

$$\begin{aligned} E_{2d} &= 2E_{2d-2} - E_{2d-4} - 4 \\ &= 2\left(a_{d-1}E_2 - 4b_{d-1}\right) - \left(a_{d-2}E_2 - 4b_{d-2}\right) - 4 \\ &= (2a_{d-1} - a_{d-2})E_2 - 4(2b_{d-1} - b_{d-2} + 1), \end{aligned}$$

which gives rise to the recurrences $a_d = 2a_{d-1} - a_{d-2}$ and $b_d = 2b_{d-1} - b_{d-2} + 1$ with initial conditions $a_0 = b_0 = 0, a_1 = 1, b_1 = 0$. Solving the recurrences we obtain easily that $a_d = d$ and $b_d = -\frac{1}{2}d + \frac{1}{2}d^2$, which proves Identity 4. To derive a formula for E_{2d}, it remains to compute E_2. Identity 4 yields the values

$$\begin{aligned} E_{n/2} &= \frac{n}{4}E_2 - 2\frac{n}{4}\left(\frac{n}{4} - 1\right) \\ E_{n/2-2} &= \left(\frac{n}{4} - 1\right)E_2 - 2\left(\frac{n}{4} - 1\right)\left(\frac{n}{4} - 2\right), \end{aligned}$$

which when substituted into Identity 1 shows that $E_2 = n - 2$. Finally, substituting this last value into Identity 4 we derive

$$E_{2d} = d(n - 2d). \tag{5}$$

Obviously the above algorithm translates into a finite automaton with a constant number of states and thus we have demonstrated:

Theorem 1. *Consider an n node ring. Two agents with $O(1)$ memory, starting at even distance $d \leq n/2$ can rendezvous in expected $\frac{d}{2}(n - d)$ steps.*

The agents in this algorithm are of optimal (to within a multiplicative constant) size but in the worst case $d = \Theta(n)$ and the expected number of steps is quadratic. One might ask if it is possible to achieve linear time.

3.2 Coin half tour algorithm

It is fairly easy to achieve a linear upper bound on the expected number of steps using the following algorithm referred to as the "coin half tour" algorithm by Alpern [1].

```
1. Repeat until other agent present:
2. If heads move right for $n/2$ steps else move left for $n/2$ steps
```

If we refer to each execution of step 2 as a phase and consider a phase to be a success if the two agents choose to travel in opposite directions and a failure otherwise then it is easy to see that (a) the expected number of failed phases before obtaining a success is one (b) the number steps in a failed steps is $n/2$ and (c) the expected number of steps in a successful phase is $\frac{n}{2}$. Therefore the expected number of steps until the agents rendezvous is n since they are guaranteed to rendezvous on a successful phase. Note that this is independent of their starting positions assuming $d > 0$. Further note that a finite automaton implementing the above algorithm requires $n/2 + O(1)$ states and thus we have shown:

Theorem 2. *Two agents with $O(\log n)$ memory, starting at even distance $d > 0$ on an even n node ring can rendezvous in expected n steps.*

The above algorithm is optimal (to within a multiplicative constant) in its running time but requires $O(\log n)$ bits of memory. Is it possible to achieve linear running time with less memory?

3.3 Approximate counting algorithm

By replacing the exact $n/2$ steps taken in step 2 of the coin half tour algorithm with an approximate expected $O(n)$ steps one can reduce the memory requirements for rendezvous in this instance. Consider the following algorithm for an agent with k bits of memory:

```
1. Repeat until other agent present :
2. (a) If heads set $dir$ = right else set $dir$ = left
   (b) Repeat until $2^k$ heads observed in a row: Move in direction $dir$
```

By defining a phase correctly and with some analysis it is possible to show that the phases have expected length $O(2^{2^k})$ and have constant probability of success and thus we can show (see [8]):

Theorem 3. *Two agents with k bits of memory, starting at even distance $d > 0$ on an even n node ring can rendezvous in expected $O\left(\left\lceil \frac{n}{2^{2^k}} \right\rceil^2 \cdot 2^{2^k}\right)$ steps.*

In particular, the above theorem implies that with $\log \log n$ bits of memory rendezvous can be achieved in linear time. It turns out that this is optimal as it can be shown that [8]:

Theorem 4. *Any algorithm that achieves two agent rendezvous in expected $\Theta(n)$ steps on an n node ring (satisfying the constraints of the model above) requires $\Omega(\log \log n)$ bits of memory.*

4 Conclusions

Distributed applications have relied heavily on the “client/server” paradigm, whereby a client is making requests from a user machine to a server which services the requests across the network. Although this model works well for certain applications it breaks down in highly distributed systems when network connections are poor, multiple clients and servers are involved, and the application requires a predictable response time. By using mobile agents, nodes can have the dual role of either client or server and the resulting networks scale better since the flow of control moves across the whole system. An algorithmic theory of mobile agents, as proposed in the present paper, helps not only to illuminate these advantages but also understand better the limitations of mobile agents by looking at, for example, memory/time trade-offs in randomized algorithms for the rendezvous problem. We believe this theory has the potential to expose both the effectiveness and the limits of using mobile agents for a host of other problems.

Acknowledgments

Research of the first author was supported in part by NSERC (Natural Sciences and Engineering Research Council of Canada) and MITACS (Mathematics of Information Technology and Complex Systems) grants. The authors would like to thank the participants of the Workshop on Mobile Computing held on Elba Island, May 2004. The ideas presented here were developed in cooperation with all those involved in the meeting.

References

1. S. Alpern, The Rendezvous Search Problem, *SIAM Journal of Control and Optimization*, 33, pp. 673-683, 1995.
2. H. Attiya, M. Snir and M. Warmuth, Computing on an anonymous ring, *Journal of the ACM*, 35, pp. 845-875, 1988.
3. J. Baumann, F. Hohl, K. Rothermel and M. Strasser, Mole: Concepts of a Mobile Agent System, *World Wide Web*, 1 (1998), pp. 123-137.
4. A. Carzaniga, G. Picco, and G. Vigna, Designing Distributed Applications with Mobile Code Paradigm, Proc. 19th Int. Conf. on Software Engineering, 1997, pp. 22-32.
5. T. Chia and S. Kannapan, Strategically Mobile Agents, Proc. of first Int. Workshop on Mobile Agents, 1997, pp. 149-161.
6. D. Coppersmith, P. Tetali and P. Winkler, Collisions among ramdom walks on a graph, *SIAM Journal of Discrete Mathematics,* 6 (1993), pp. 363-374.
7. P. Flocchini, B. Mans and N. Santoro, Sense of direction: definition, properties and classes, *Networks* 32 (1998), 29-53.
653-664, 2006.
8. E. Kranakis, D. Krizanc and P. Morin. Randomized Rendezvous on the Ring, in preparation.

9. D. Milojicic, F. Douglis and R. Wheeler (Editors), Mobility: Processes, Computers and Agents, *ACM Press*, 1999.
10. D. Milojicic, D. Chauhan, and W. LaForge, Mobile Objects and Agents (MOA), Proc. of 4th USENIX Conf. on Object-Oriented Technologies, 1998, pp. 1-14.
11. R. Motwani and P. Raghavan, *Randomized Algorithms*, Cambridge University Press, New York, 1995.
12. H. Peine and T. Stolpmann, The Architecture of the Ara Platform for Mobile Agents, Proc. of First Int. Workshop on Mobile Agents, 1997, pp. 50-61.
13. N. Roy and G. Dudek, Collaborative robot exploration and rendezvous: Algorithms, performance bounds and observations, *Autonomous Robots* 11 (2001), 117-136.
14. N. Santoro, Design and Analysis of Distributed Algorithms, John Wiley and Sons, 2007.
15. D. Singelee and B. Preneel, Secure E-commerce using Mobile Agents on Untrusted Hosts, Computer Security and Industrial Cryptography (COSIC) Internal Report, May 2004.
16. D. Tennenhouse, J. Smith, W. Sincoskie, D. Wetherall and G. Minden, A Survey of Active Network Research, *IEEE Communications Magazine*, 35 (1997), pp. 80-86.
17. L. Tesfatsion, Agent-Based Computational Economics: Growing Economies From the Bottom Up, *Artificial Life*, 8 (2002), pp. 55-82.
18. T. Walsh, N. Paciorek, and D. Wong, Security and Reliability in Concordia, Proc. of 31st Hawaii Int. Conf. on System Sciences, 1998, pp. 44-53.
19. J. E. White, Telescript Technology: Mobile Agents, in *Software Agents*, MIT Press, 1996.
20. M. Wooldridge, Intelligent Agents, in Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence, G. Weiss, ed., pp. 27–77, MIT Press, 1999.