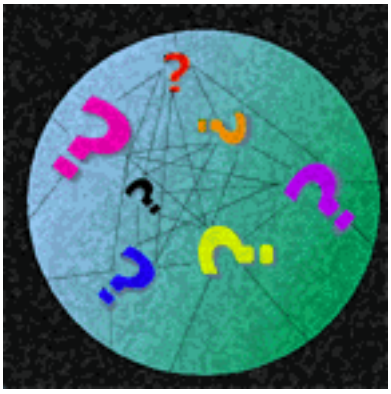
Protecting the Integrity of Agents: An Exploration into Letting Agents Loose in an Unpredictable World

by Michael J. Grimley and Brian D. Monroe

Introduction



After years of research, the definition of a software agent is still vague. For our purposes, we use a definition from Jennings and Wooldridge [6]. They state that a **software agent** ``... is capable of flexible autonomous action in order to meet its design objectives." By flexible, it is meant that the agent must be responsive, proactive, and social. Their behavior is based upon predefined rules and actions provided by their designer. However, this technology is different from other forms of artificial intelligence because of the agent's ability to be social, meaning that it can interact with other agents, including humans. In addition, **proactivity**, the exhibition of goal directed behavior, implies more functionality than traditional applications. These agents are also capable of existing in a community and working cooperatively to attain a common goal. Furthermore, some agents are capable of leaving their local host and visiting foreign servers that may contain agent communities of their own.

The potential uses for agents are numerous. Agents are currently being used for workflow management, network management, air-traffic control, information retrieval management, electronic commerce, education, personal digital assistants, e-mail, digital libraries, command and control, smart databases, and other services. This technology has the potential to revolutionize the way we support people in performing tasks. Agents allow a user to have an adaptable and customizable assistant available at all times. However, as with many new and exciting technologies, there is a downside.

Problems may occur within the communities mentioned above. The agents within the community may have conflicting beliefs, goals, commitments, or ways of carrying out tasks. Furthermore, there is the issue of security within the community. These conflicts and security problems can lead to hazardous situations that can compromise an agent's integrity, and lead to potential failure of systems.

The issue of agent integrity has heretofore been ignored in the realm of agent literature. With the potential of malicious hosts and inaccurate information, along with the many unsolved problems arising from agent interaction, the agent's integrity is in jeopardy. The safety issues pertaining to agents must be

explored further before the technology as a whole can move forward. These issues include:

- Foreign hosts altering code; manipulating results; obtaining `private' information.
- A user's agent proceeding with faulty information.
- Other agents interfering with the completion of a user's agent's goals.

Strategies must be developed to protect the integrity of agents. With the use of this new technology in safety-critical applications, better ways of engineering agent-based systems must be devised. In this paper, we discuss these issues and the ramifications of letting agents loose in an unpredictable world; solutions are also proposed.

Dangers of Malicious Hosts

If an agent spends all of its time in a closed system, no matter how widely distributed, there is no need for concern about the damage that it might incur at the hands of a malicious host. Even if it visits numerous hosts that are not under the control of its owner, if these are limited to those that have proven themselves trustworthy, there is still little concern for the types of dangers described herein. However, in two of the most promising applications for mobile agents (e-commerce and information gathering), their power lies in their ability to visit numerous servers where the host may not be completely trusted.

Under these conditions, once an agent leaves the safety of its home, its owner has no direct control over what happens to it and no real control over where it goes. It often survives solely by `depending upon the kindness of strangers'.

> ``The host can observe every step the agent takes, read every bit of code, data and state, and even manipulate the way the agent works, as it, among other things, interprets the agent code" [4].

This problem is magnified when the agent is constructed from standard templates, which are becoming popular [4]. This gives the potential perpetrator a head start in determining what the agent is doing and how it is doing it, thereby allowing the host to more easily extract, tamper with, or modify the visiting agent's code or data. The danger is also increased when a particular agent visits the same hosts repeatedly, giving a malicious host many opportunities to `get to know' the agent and its behaviors.

Some researchers in the field have gone so far as to say that agents should not visit hosts that have not proven themselves completely trustworthy [11]. This `guilty until proven innocent' mindset would certainly limit the number of hosts, and thereby the amount of information, a particular agent would have access to. This is a high price to pay if we wish to take full advantage of mobile agent technology. An alternate, and we believe preferable, method would be to visit hosts that have not yet been proven trustworthy, but with the understanding that the agent is vulnerable to attack.

Before agent developers can devise ways to prevent attacks, they must determine what types of attacks agents are susceptible to. We will now describe several types of attacks that agents should be prepared to handle. Although many of these issues have been discussed separately by various individuals, the only previous attempt to give an overview of all of the potential attacks is an excellent paper by Fritz Hohl [4] which discusses, in much greater detail, several of the issues stated here. To assist in the understanding of the issues involved we will use the typical example of a shopping agent that travels the World Wide Web, visiting numerous online booksellers searching for the best price on a given selection. It retains only the lowest price found and the site on which it was located.

Invasion of Privacy

Because the host has full access to the agent's memory space, it is capable of reading the data that resides there. This data could include secret keys used for encryption purposes, electronic money, or other data to which the agent does not intend others to have access. This type of attack is impossible to detect because neither the state nor the code of the agent is altered. The implications of the first two situations are obvious, but there are other types of data that an untrustworthy host might find useful.

In our shopping agent example, a host could determine what the current low price is, and then make sure its bid is just slightly lower, even if its regular price is significantly lower [4]. It could also read the agent's itinerary, and make sure that its bid will beat all of the future bids as well.

Data or State Manipulation

The host may not be satisfied with just reading an agent's data, it may wish to alter it as well. If it is able to read a value in memory, it can just as easily change that value. It can search out and modify data received from other hosts, as well as alter data the agent will use in the future. In our agent example, it could change the requested number of books to purchase to `2', thereby ensuring that its price (for one) will be the lowest.

The host may also alter the agent's beliefs or commitments in order to have it favor itself over other sources. An even more efficient technique would be for the host to place itself in the lowest bidder slot and then just send the agent home, so it wouldn't even have to worry about future sources.

If the visiting agent is carrying electronic money, the host could trick the agent into spending it there. The host could also trick the agent into thinking it was `home' and thereby have it release all the money or information it has.

Code or Control Flow Manipulation

Given enough time, a malicious host could also examine the agent's code in order to alter it to suit its needs or desires. This could involve implanting a virus before sending it on its way to the next

destination or simply altering it in ways to benefit the manipulating host.

In a related attack, if the host can determine the control flow of the agent, it can ensure that whenever relevant decisions are made, they are made in its favor. In regards to our shopping agent, that would mean always evaluating its price as being the lowest. This is also another way the host can `fool' the agent into thinking it were home. If there is a control structure that delivers money or information based on the determination that the agent is home, a `true' evaluation will cause the agent to execute that task.

Purposeful Misinterpretation of Agent Code

Because most mobile agents are coded using interpretive languages, such as Java and TCL, an agent is at the mercy of the interpreter residing on the foreign host. If someone has decided to develop an interpreter that executes instructions in a manner inconsistent with the standard interpreter, it would be very difficult for an agent to detect this.

> ``There appears to be no reliable way to authenticate an interpreter.... Bugs may survive lengthy testing even if they were not designed to be hard to find" [13].

Although this may not cause any damage to an agent, it cannot be assured that an agent is executing as intended.

Cloning

One of the more straightforward attacks is to clone the visiting agent. Because the agent is not delayed or tampered with, this is another attack that may be difficult, if not impossible, to detect. After creating an exact copy of the agent, the host can then spend as much time as it needs to evaluate its code and behavior. After completing this evaluation, it can then wait for the next visit of the agent and then perform any of the attacks listed above or simply send out the clone in its place.

Denial / Delay of Service or Misdirection

In two other related attacks, the host may not release the agent at all or, if it does, it sends it to another host which is not on the agent's itinerary. When done in conjunction with some of the other attacks, it gives the host more time to evaluate the agent. Even if the host delays the exit of the agent for just a short time, it might be enough time to get the information it needs.

Possible Solutions

Some researchers are of the opinion that ``it is impossible to prevent agent tampering unless trusted (and tamper-resistant) hardware is available" [2]. Several others have offered potential solutions to the attacks described above. They all entail varying levels of overhead so decisions must be made about the level of

security required versus the runtime efficiency desired.

Data Encryption

One of the more obvious solutions to the `invasion of privacy' and `data manipulation' attacks is to encrypt the agent's data. In order to do this, we must first distinguish between two types of data: that which will be used by the agent in the attainment of its goals and that which has been received by visited hosts which will not be accessed again until the agent has returned home.

In the latter situation, encryption is a simple and viable solution. The agent can encrypt the incoming data using a public key, and can deliver this data in its encrypted form to its home base, where the private key is safely kept.

If, on the other hand, the data must be accessed by the agent, then the agent must also carry the decryption key. Unfortunately, this key would be just as vulnerable as any other data or code. There has been some research done on `mobile cryptography' [12] that addresses this problem, but a workable solution has yet to be developed.

Code Encryption

Encryption of code involves the same problems as the encryption of data that will be used during execution. If the code has to be decrypted before execution, then the host will have access to the key as well as the decrypted code.

One novel, but as yet unimplemented, approach to this problem is called **blackbox security** [5]. Here, the encrypted agent is executed within its own execution layer, which accompanies it on its travels. In this manner, the foreign host never has direct access to the agent's code or data.

Time-Sensitive Agents

This approach takes advantage of the fact that it takes time for a malicious host to evaluate an executing agent. If the amount of time an agent executes on a foreign host is limited, the chance that it will be tampered with is minimized. Once the maximum amount of time an agent can safely execute on an untrusted host is determined, it can be programmed to either shut itself down or move on to the next location in its itinerary.

Code Obfuscation [4]

This technique takes its queues from `Software Engineering' and `Program Comprehension'. A great deal of research has been done to determine good programming practices; those things which make programs easier to read and understand. It is widely agreed that things such as meaningful variable names,

modular design, clear and concise algorithms translated into clear and concise code, and adequate documentation contribute greatly to the understanding of a program. Alternately, the lack of these things should make a program very difficult to comprehend, and therefore difficult to evaluate.

A relatively simple method to accomplish this is to redesign a program to make it less comprehensible: remove all modularity by inserting the code from each method (function) in place of each call to that method; create meaningless random variables and use them in phony control structures; and rename all identifiers with meaningless names (e.g., 'f2s', 'tzx', or simply 'x'). A more complex technique is called **variable recomposition** [4]. In this technique, variables are split up and then recombined into bitsets containing parts of several variables. Using this method a variable may be reconstructed by concatenating the values from different sections of the bitsets (e.g., instead of BookTitle, one might have V1(bf1[4]+bf2[7]+bf3[5])).

An alternate method is to create a program that does the translation automatically. Ideally, it would produce a different form of the code each time it is induced to allow for those occasions where an agent makes repeated visits to the same hosts.

A combination of code obfuscation and time-sensitive agents is probably the most promising approach at this time, if for no other reason than the fact that they are currently the most easily implemented.

Detecting Compromised Agents

If all else fails, a user should at least be able to determine whether his agent has been tampered with. One of the simplest and easiest to implement methods is using **detection objects** [7], dummy values in variables which would not be changed in the normal course of events. This method would not apply to the shopping agent of our example, but if we made one minor modification by having the agent store all of the prices of all of the booksellers it visited, we could insert a dummy company with a very low price and check to see if it was tampered with when the agent returns.

Another issue related to detecting tampering is what to do once it has been detected. If one can determine which was the offending host, one option is to `boycott' that location. Some people have proposed a central clearinghouse of information concerning unreputable hosts [12].

Phoning Home

Another technique that does not prevent tampering, but can lessen the damage when it occurs, is to have the agent `phone home' just before leaving each host. This could involve transferring any data it has acquired, thereby preventing its loss or its disclosure to future hosts. Otherwise, it is simply a method to let a user know that the agent is still functioning, and if not, where it ran into trouble.

Dealing With Unsure Information

Another issue that needs to be addressed is the fact that an agent can never be sure whether the information it has gathered is accurate. There are several reasons why an agent may hold incorrect information.

- Faulty reasoning on its part, possibly stemming from a misinterpretation of its environment.
- Faulty reasoning on the part of a host or other agent, leading to the agent receiving bad information.
- Out of date information.
- Intentional misinformation given to it by a host or other agent.

No matter what the cause, the result remains the same; the agent may hold incorrect beliefs, which may influence its decisions and actions. The agent may then make plans based upon these beliefs, thereby affecting a large portion of the community. This is why it is so difficult to correct the problem. With truth maintenance, several factors can be taken into account when evaluating the validity of data received.

What Is the Source?

If the source of the information is the environment in which the agent is running, delivered through the agent's own monitors, the agent itself is responsible for the accurate interpretation of the data. In this case, the faith a user has in the data should equal the faith he has that his agent is performing properly.

If the source is a database located on a foreign server, then the amount of trust a user has in the data can be determined by the reputation of the owner of that database, or the validity of information previously obtained from the source. Another consideration might be how often the database is updated. If a user is searching for population statistics the need for a regular update will not be as crucial as if he were receiving stock quotes.

If, on the other hand, the source is another agent, it can be trusted only as much as every place it has visited and every other agent it has come in contact with. In most cases, this information will not be available to the user, so decisions will have to be made on whether and how to use this data.

To Validate or Not To Validate

There may be times when it will be possible to validate data in some way, whether by querying another agent or database or by rechecking the same source. This will obviously generate a certain amount of overhead so it will have to be determined when the harm from using incorrect data outweighs the performance hit taken by attempting to validate that data.

As indicated above, the source of the information may be one consideration. It would also be helpful if the agent could carry with it an expected (or allowable) range of values and only consider validating

those that fall outside of that range.

Another consideration should be what use the data will be put to. If it is only going to be stored for future reference, then the need for the agent to validate it on the spot is probably negligible. However, if the data is going to be used to determine what the agent will do next, or to evaluate other data, then validation may be a viable option, after consideration of the other criteria.

Agent Coordination

According to the American Heritage Dictionary, **coordination** is defined as ``working together harmoniously or in common action and effort" [8]. Inter-agent coordination continues to be a major obstacle for agent-based systems. There have been many methods suggested to tackle this problem, but few have been successfully implemented. Coordination among agents is an essential attribute of all agent-based systems. ``Coordination is essential in enabling groups of agents to solve problems effectively. Without a clear theory of coordination, anarchy or deadlock can set in easily... " [9]. There are many problems associated with coordination that threaten an agent's integrity. These problems include conflicting beliefs and goals, unfulfilled commitments, and an inadequate knowledge base.

Conflicting Beliefs

An agent's beliefs are the information it has about its environment, internal state, and actions that it may perform. Unlike most programs (whose execution is determined by a predefined sequence of steps), an agent has the ability to reason about its beliefs and to decide what action to take next. However, this does not always work as planned.

The problem lies in the fact that an agent's beliefs about its environment may not be correct. Nick Jennings, a prominent agent researcher, stated, ``Beliefs are generally used to refer to the information that agents have about their environment. This information can be incorrect -- in just the same way that information we have about the environment (our beliefs) could be wrong" [6]. In human communities, conflicting beliefs may lead to conflicts or chaos. In agent communities, conflicting beliefs can cause conflicting tasks, which may lead to livelock. It leads to a lack of objectivity meaning that when two agents witness the same event, instead of perceiving the same thing to have occurred, they perceive two separate situations. These things may cause systems to fail thereby resulting in goals going unreached.

Conflicting Tasks

Conflicting beliefs sometimes lead to agents performing tasks that are counterproductive. For example, consider an agent system whose responsibility is to monitor a hospital patient's vital signs and react appropriately to changes. If the patient's blood pressure rises, the blood pressure agent dispenses the appropriate medication to lower it to acceptable levels. The problem is that the blood pressure medication causes the patient's temperature to rise beyond accepted levels. This triggers the temperature

agent to dispense medication to bring the patient's temperature down. Unfortunately, this medication causes an abrupt rise in blood pressure, which causes the blood pressure agent to dispense more of the blood pressure medication. This cycle may continue until the system fails completely, resulting in harm to the patient. While this example is hypothetical, it is obvious that situations like this can occur in many systems. This situation is known as livelock.

Livelock

Livelock occurs when agents continuously act, but no progress is made toward the overall goal [10]. This situation is dangerous because in some systems it can go on for a long period before anyone notices. The agents performing the conflicting tasks and other agents within the system have no mechanism to detect that the problem has even occurred. On the surface, it seems as though all of the agents are progressing toward the goal, but in reality, they are in an endless loop. This is because of the problems associated with modal reasoning.

Unfulfilled Commitments

Unfulfilled commitments are another major obstacle for agent-based systems since they can result in entire systems coming to a halt. There are two types of commitments that can occur between agents, explicit and implicit. An explicit commitment is one in which an agent makes a contract with another agent. The contract states a job to be performed and any other requirements that must be met such as time or expected output. This type of commitment is made when an agent does not have the capability to complete part of its tasks so it finds another agent that does.

An implicit commitment is the expectation that an agent will perform its assigned role or service within the community without failure and whenever called upon. This type of commitment always exists and if it fails, the system may fail. There are many reasons why commitments go unfulfilled; in this section, we will highlight the main ones.

Inability to Reason

A major reason for commitments going unfulfilled is an agent's inability to effectively reason about other agent's roles or specific situations within the agent community. The two types of reasoning that have been a major thorn in the side of agent-based systems are common-sense and modal reasoning.

Common-sense Reasoning

Commonsense reasoning involves simple things such as time, space, and causality. If we were not able to reason about them and include them in the plans that we make or the tasks we perform, there would be chaos and confusion. This is a problem that plagues agents and the commitments that they make.

> ``In most applications, commonsense reasoning involving notions such as time, space,

> causality, etc. is handled in an ad hoc application specific manner. Indeed, this issue of commonsense reasoning has defied formal treatment in AI, and presents perhaps the major stumbling block to the development of reasonably smart agent systems" [9].

If agents do not have the ability to reason about things such as time and causality, how can they effectively keep the commitments that they make? If there happens to be a time requirement, can the agent make a reasonable estimation about the completion time? If not, there will be a great deal of missed deadlines and failures. This is especially dangerous in a safety-critical environment.

Modal Reasoning

Modal reasoning is necessary for agents to reason about their own and other agents' beliefs. Without this capability, the system is sure to fail. According to Divine Ndumu, an agent researcher from BT Laboratories:

> ``Current modal reasoning formalisms still face a number of problems; particularly the logical omniscience problem wherein an agent believes all the logical consequences of its beliefs, and the problem with opacity of beliefs where variable bindings in one belief statement cannot be assumed to hold in other belief statements" [9].

In the example of the patient monitoring agents mentioned above, the situation could have been avoided completely if the agents had the capability of modal reasoning. The temperature agent would have been able to reason about the blood pressure agent's beliefs and tasks and would have decided that the rise in temperature was due to the blood pressure medication that was administered. Livelock would have never occurred.

Relating Task Description to Agent Capabilities

Another reason for agents not meeting commitments may be their inability to relate a given task to their current capabilities. This could stem from a number of things, one being a defunct knowledge base, meaning that the agent's knowledge of a particular task is incomplete. For instance, ``if one agent is allocated the task `make a Christmas cake', that agent may be able to measure ingredients, mix them, use the oven, etc. but does not have a recipe for Christmas cake" [1]. Therefore, the agent will be unable to reason about how to complete the task at hand.

Dependency between Tasks

When there is a dependency between the tasks of agents, it usually means that there is some kind of implicit commitment. For instance, if Agent 1 performs a specific task within a community, Agent 2 may need the output of that task in order to start its specified task. If Agent 1 does not produce the appropriate output, then Agent 2 does not even start its task. Using the Christmas cake example, certain tasks need to be completed before the ingredients for the cake can even be mixed. ``The ingredients for

the meal must be purchased before the cooking can be done" [1]. If Agent 1 is the purchasing agent and fails in that task, then Agent 2 never gets to mix the cake. This is a very dangerous situation, because if Agent 1 fails in its task completely, then the system as a whole fails. The next step cannot be taken and Agent 2 sits idle.

Lack of Global Perspective

A problem that could cause agent-based systems to cascade into failure is their lack of global perspective. Without a clear sense of the global situation, it is possible that agents could work against each other, as discussed previously. The ``right hand may not know what the left hand is doing." Jennings and Wooldridge state that ``an agent's actions are, by definition, determined by that agent's local state. However, since in almost any realistic agent system, complete global knowledge is not a possibility, this may mean that agents make globally sub-optimal decisions" [6]. Because of this, they also state that ``an agent based solution may not be appropriate for domains in which global constraints have to be maintained...or in domains in which deadlocks or livelocks must be avoided" [6]. The example of the patient monitoring agents could be prevented completely if the agents had some kind of global perspective or knowledge. This global knowledge could be kept locally within the agent or in some kind of overseeing agent that keeps track of what is going on within the system.

Knowledge Base Problems

Some of the problems previously mentioned would not be an issue if problems associated with an agent's knowledge base were resolved. These problems include incorrectness and incompleteness [1].

Incorrect Knowledge

Incorrectness refers to the state of affairs when the agent's knowledge is inaccurate. Therefore, the agent would reason about things in an incorrect way thereby possibly causing a hazard (or if this reasoning cascades throughout the entire community, a system failure). According to Ciara Byrne of the University of Aberdeen, context is also a very important factor in whether or not knowledge is incorrect. She states that ``the same piece of knowledge may be correct in some contexts and not in others" [1]. An agent's knowledge base may become incorrect as a result of some of the issues discussed above or if the agent is unable to apply the knowledge in the correct context or situation.

Incomplete Knowledge

Incompleteness occurs when some knowledge that is essential to an agent's reasoning ability is missing. This relates to the example of the Christmas cake agent discussed previously. The agent's task is to make a Christmas cake, but the knowledge the agent has about this task is not adequate enough to carry it to completion. This becomes a large problem for the system as it leads to things such as deadlock and livelock. If one agent's knowledge base is incomplete it can have a domino effect throughout the rest of

the community because that agent's actions may either hamper other agents from performing their service or cause the system to fail completely.

Proposed Solutions to Agent Interaction Problems

Social Structures

According to Mark d'Inverno of the University of Westminster, ``a social structure is a set of relations that hold between agents in a society...In order to cooperate effectively with its peers, an agent must represent any social structures in which it plays a part, and reason with these representations" [3]. If the agents in the patient monitoring example had some sort of structure set up which represented the relationships between other agents within the community and the services they provide, the problem of livelock may have been avoided. The temperature agent may have been able to reason about the fact that the medication the blood pressure agent had administered caused an abrupt rise in temperature.

Global Broadcast of Change

A solution for conflicting beliefs within an agent community might be the global broadcasting of change. For example, if an agent within a community changes one of its beliefs, it would broadcast this change to the entire community of agents so they would be able to update their knowledge base in accordance with this change. In most cases, this would alleviate the problem of inconsistency and conflict. Of course, this solution would only work if the agents within the community kept some kind of social structure. Whenever they receive a broadcast that a change has occurred, they could simply update the structure, thereby changing the way in which they represent and reason about the rest of the agent community.

Third Party Monitors

Another way to make sure that agents' interactions do not jeopardize their integrity or the integrity of an entire community is to implement an overseeing body that has a global perspective. The overseeing body would monitor agent interaction and beliefs. This type of overseer alleviates some of the potential for chaos and danger within an agent community.

The Safety Agency

One type of overseeing body currently under development is known as the Safety Agency. The Safety Agency addresses the safety and security problems of agent communities and gives agent community administrators the ability to track their agents. Within the Safety Agency there exist agents to monitor the effectiveness, security, and safety of other agent communities and keep the agent community administrator up to date on the safety/security problems that arise. This solution does, of course, rely upon the fact that someone must be keeping track of not only the safety/security problems of the

monitored agent community but also the belief base of the Safety Agency itself. This person should be an agent community administrator. The reason why the administrator has to monitor the belief base of the Safety Agency is that it contains agents that are susceptible to all the problems previously stated.

Conclusion

The potential problems agents may encounter when interacting with an unpredictable world are many. The first step in dealing with these problems is realizing what they are and how they may affect an agent's integrity. With the threat of malicious hosts, unsure information, and the unsolved problems of agent interaction, agents must be prepared for the worst.

We have discussed several existing problems and possible solutions. Unfortunately, many of these solutions are not one hundred percent reliable. For example, many of the techniques that make it safer for agents to execute on a foreign host make it less safe for the host to allow the agents to execute. If the host is unable to detect the behavior or motives of an agent, there is no way for it to tell if it means to do it harm.

As for agent interaction, there have been few, if any, proven and clear solutions. If these problems remain unsolved, then the potential for the failure of agent-based systems is great. When it comes to agents, what you do not know can hurt you.

References

1

Byrne, C. and Edwards P. *Refinement in Agent Groups.* Technical Report, Department of Computer Science, King's College, University of Aberden, Aberdeen, Scotland, UK, 1996.

2

Chess D., Grosof B., Harrison C., Levine D. and Parris, C. *Itinerant Agents for Mobile Computing.* Technical Report RC 20010, IBM, March 1995.

3

d'Inverno M., Luck M. and Wooldridge, M. Cooperation Structures. In *Proceedings of the 5th International Joint Conference on Artificial Intelligence* (1997, Nagoya, Japan) pp. 600-605.

4

Hohl, F. *An Approach to Solve the Problem of Malicious Hosts.* Institute of Parallel and Distributed High-Performance Systems (IPVR), Fakultat Informatik Universitat Stuttgart Breitwiesenstr, March 1997.

5

Hohl, F. Protecting Mobile Agents with Blackbox Security. *Workshop on Mobile Agents and Security* (October 1997) University of Maryland.

6

Jennings, N.R. and Wooldridge, M. Applications of Intelligent Agents. In *N. Jennings and M.J. Wooldridge (eds.)*, *Agent Technology: Foundations, Applications, and Markets.* Springer-Verlag. pp. 3-28, 1998.

7

Meadows, C. *Detecting Attacks on Mobile Agents.* Foundations for Secure Mobile Code Workshop, Center for High Assurance Computing Systems, 1997.

8

Morris, W. (eds.). *The American Heritage Dictionary.* Houghton Mifflin Co., Boston, 1976.

9

Ndumu, D. and Nwana, H. Research and Development Challenges for Agent-Based Systems. In *IEEE/BCS Software Engineering Journal* (1996).

10

Nwana, H., Lee, L. and Jennings, N. Coordination in Software Agent Systems. *BT Technology Journal.* 14(4), 1996.

11

Ordille, J. *When Agents Roam, Who Can You Trust?.* Innovations for Lucent Technologies, Computing Science Research Center, Bell Labs, 1996.

12

Sander, T. and Tschudin, C. *Protecting Mobile Agents Against Malicious Hosts.* Lecture Notes in Computer Science: ``Mobile Agents and Security''. G. Vigna (Ed.) Springer Berlin Heidelberg (June 18, 1998).

13

Swarup, V., Farmer, W. and Guttman, J. *Security for Mobile Agents: Issues and Requirements.* Proceedings of the National Information Systems Security Conference (NISSC), October 1996.

Michael Grimley (MJGrimley@acm.org) and Brian Monroe (BMonroe@acm.org) are undergraduate students at the University of Massachusetts Dartmouth.