Taking Back Cyberspace

Jeffrey M. Bradshaw, Institute for Human and Machine Cognition
Giacomo Cabri, University of Modena and Reggio Emilia
Rebecca Montanari, University of Bologna

Traditional systems usually have dealt narrowly with such concerns, if at all, by implementing controls relating to security or safety. Given their increased degree of potential autonomy, agent systems raise the bar on the range of trustworthiness issues that must be addressed. Virtually any technical or social aspect of agent behavior that relates to other agents and the environment can be of potential interest.

Certain roles in society carry with them such a high level of fiduciary responsibility that the appropriate organizations carefully monitor and control them through licensing, credentials, and periodic formal review. To the degree that our personal welfare depends on such people, we count trustworthiness the most important prerequisite of our willingness to commit aspects of our lives and resources into their hands. For similar reasons, people expect a high level of trustworthiness before delegating complex and important tasks to software agents.

Researchers must find a way to implement trustworthiness if people are to stop worrying and learn how to love agents.

IMPORTANCE OF AGENT TRUSTWORTHINESS

Despite agent researchers' ever-increasing energy and enthusiasm, trust issues remain a major reason for the agonizingly slow uptake of new research results in fielded systems. With rare exception, today's agents have not been deployed in critical, long-lived, secure, or high-risk tasks. Nor do they undertake missions that require widespread agent migration or collaboration among large numbers of agents interacting in complex, unpredictable ways.

Ideally, agents would be true citizens of the wired world, equipped with stamped passports and Berlitz traveler's guides that let them hail, meet, and greet agents of any sort while traversing the Internet's open landscape. If not able to team up on a given project, they would at least be able to ask intelligibly for directions. These kinds of agents, alas, exist today only in our imaginations (and, of course, in the vision sections of our research proposals).

Because they can operate independently without constant human supervision, agents can perform tasks that would be impractical or impossible using traditional software applications. On the other hand, if unchecked and bestowed upon poorly designed, buggy, or malicious agents, this autonomy could cause severe damage.

Ever more powerful intelligent agents will differ increasingly from traditional software, thus we must take into account the technical and social aspects of trustworthiness if people are to accept the agents we design and build.

Key to implementing any trustworthy system is an infrastructure-based mechanism to monitor and govern selected aspects of system behavior to maintain conformance to some set of desirable constraints or objectives. Such mechanisms, while independent of the application components themselves, must be under control of some responsible party. Moreover, they must be capable of dynamically adjusting their governing strategies to deal with changing contexts and priorities.

AGENTS AND TRADITIONAL SECURITY ISSUES

Virtually all aspects of traditional security concerns are relevant to agent systems. However, the additional complexity and autonomy of software agents poses increased security risks.

We need techniques and tools to assure that agents will always operate within the bounds of any behavioral constraints currently in force while remaining responsive to human control. For example, hosts should be protected from fraudulent agents trying to gain unrestricted access to private-node information or to perform denial-of-service attacks through node resource overuse. Further, agents should be protected so that malicious information received from other agents cannot alter their programmed behavior.

As with any complex system, security concerns must inform every phase of agent application development, from requirements engineering to design, implementation, testing, and deployment. Considering that developers widely use high-level methodologies to support agent requirements analysis and design activities, we also

need high-level approaches to model and express agent security-related features such as privacy, integrity, and access control.

Policies can control agent execution at high levels of abstraction. Given the complexity of the organizations to which agents belong and the wide range of actions and abstraction levels that must be considered in controlling agent behavior, policy representations and analysis and inference techniques tend to be more sophisticated and dynamic than in traditional systems.

Although policy representations differ widely in their expressivity, efficiency, and the comprehensiveness of their associated management tools and enforcement mechanisms, they tend to deal with two fundamental policy statement types: those relating to authorization, which determines what an agent is or is not *permitted* to do; and those relating to obligation, which determines what an agent is or is not *required* to do.

To facilitate dynamic modification of security requirements without affecting the agent code, security solutions must also clearly separate specifications and their implementation from agent code. Reflection and aspect-oriented programming offer two possibilities for achieving this needed degree of separation.

Adopting reflection would require splitting the agent system into two levels: the *base level*, which contains the agents; and the *metalevel*, which would include the rights identified for such agents and the metaobjects that transparently monitor the agents and decide whether to grant or deny their requested actions.

Aspect-oriented programming suggests treating security policies as aspects to be woven into the agent code at appropriate join points. Unfortunately, how and when to perform this weaving remains an open aspect-oriented programming question, one that poses a problem analogous to that of base and metalevel association in reflective systems.

AGENT MOBILITY

Mobile agents are software components that can autonomously migrate across the network while acting on their users' behalf. A growing body of empirical evidence demonstrates the usefulness of this technology in particular application contexts.

Specifically, analysts frequently cite the benefits of flexibility in dealing with changing network conditions, disconnected operation, optimal bandwidth use, deployment of customized code on demand, and fault tolerance. For example, a mobile-agent-based application can deploy its components to the most currently attractive network locations and redeploy those components whenever network or other conditions change, leading to more efficient use of available resources and faster completion times.

Policies can control agent execution at high levels of abstraction.

Mobility programming

Mobility complicates agent design and development because developers must decide when and where to migrate agents. This can be difficult in current computing environments, which sometimes prevent making reasonable assumptions about network topology and node availability.

The complexity of computing optimal migration strategies seems to call for innovative planning and scheduling technologies, but it remains an open area with no adequate solutions.

Developers also need runtime environments that allow changing mobility decisions without affecting agent code—thus separating application logic from mobility. The programming model adopted in the Tacoma system, for example, distinguishes mobility and management aspects from application functionality per se. Unfortunately, Tacoma does not provide any high-level means to simplify programming of mobility strategies for unskilled developers.

Other approaches, such as the RAM system, advocate using reflection for dynamic mobility control of mobile object clusters. RAM associates each cluster at the base level with metaobjects that control the cluster migration policies and its binding to resources. The definition of hooks enables the jump from the base to the metalevel—hence, mobility adaptation. However, hook definition can occur only at compile time, thus precluding adaptation of mobility strategies to unanticipated circumstances.

Policy-based approaches to mobility have also been proposed. For example, frameworks such as Poema and Nomads use policies to govern agent systems' mobility behavior independently of the agent application code. Such an approach lets system administrators change policy specifications on the fly without affecting agent application implementation and without any need to program specific hooks in advance. The underlying policy infrastructure handles interpretation and enforcement of policies at runtime, transparently to application designers.

Security issues

Adding mobility to agents introduces novel security issues because it requires protecting agents' integrity and secrecy as they travel across multiple nodes and execute in various environments with different trust levels. Protecting agent integrity requires either prevention or a posteriori identification of a malicious execution that tampers with an agent's code or state.

To prevent attacks on agent integrity, some developers rely on special tamperproof hardware that avoids unauthorized modification of agent code or state by executing the agent in a physically sealed environment. Others attempt to make agent tampering more difficult by adopting algorithms that obfuscate the agent's code and data.

The approaches to agent-integrity tampering detection propose different forms of cryptographic encapsulation of the results of agent computation at each visited node to be used for subsequent verification.

Providing agent secrecy requires hiding the agent's code and state parts from the site responsible for its execution. However, preventing attacks on agent secrecy is difficult because a mobile agent that runs inside a foreign execution environment is completely at that system's mercy. No entirely foolproof general mechanisms exist to protect mobile agents from inspection.

AGENT INTERACTION

Agents introduce new dimensions to the long-standing distributed systems problem of managing complex interaction among loosely coupled components. Many of these new dimensions spring from the decidedly social nature of agent-agent and human-agent interaction. Developers typically model this interaction on observations of collaboration and competition in the real world.

Further, the diffusion of open systems in the Internet has led to more complex interoperability requirements for agent interactions. Independently developed agents must be able to rely not only on shared infrastructure, but also on shared background knowledge, syntax, semantics, and pragmatics for agent communication and collaboration.

In open systems, designers must always consider that agents representing the conflicting interests of their users will game the system to maximize their own selfish returns. Since open-agent systems developers will control the communication and coordination mechanisms but not the individual agents, they must carefully design these mechanisms so that no agent can exploit its position to unfair advantage.

Controlling data sharing

Different proposals have been advanced to enable the data sharing that underlies agent coordination. These proposals include message passing, meeting-point abstractions, and event channels. Some of the most interesting data sharing approaches are based on *tuple spaces*. First proposed in Linda, but with historical affinities to earlier blackboard approaches, tuple spaces consist of shared data spaces that rely on pattern matching to allow access among uncoupled senders and receivers in open and dynamic environments.

Novel solutions are starting to emerge to improve the flexibility of data sharing control. Systems such as Tucson and MARS define specific context-dependent rules to adapt tuple space behavior to the application's or environment's requirements. These rules are defined independently of the agents themselves, allowing separate control of agent interaction.

The Internet has led to more complex agent interoperability requirements.

Role-based approaches represent another interesting direction in controlling agent interactions. Roles define common interactions between agents, such as the interactions between auctioneers and bidders in an auction. Because it promotes an organizational view of the system, this approach helps developers deal with roles separately from agents.

Four attributes—responsibilities, permissions, activities, and protocols—define another approach that exploits roles in the analysis phase. In Gaia, for example, the interaction model is a set of protocol definitions. Each protocol definition relates to a kind of interaction between roles and describes the dependencies and relationships between the different roles.

Conversation policy-based approaches also exploit recurring patterns among agents in various roles. Other approaches explicitly code the interaction issues into the roles that agents assume and leave the application logic in the agents. One such system, Brain, uses XML-based notation to define roles as sets of actions and events. Another, ROPE, considers cooperation processes first-class entities that define how the involved roles interact with each other.

Teamwork

In many situations, roles and interaction patterns are fixed in advance or can be easily discovered. In others, however, explicit reasoning about agent interaction at runtime is essential. For example, teamwork has become the most widely accepted metaphor for describing the nature of multiagent cooperation.

Most approaches depend on the key concept that shared knowledge, goals, and intentions function as the glue that binds team members together. A largely reusable explicit formal model of a team's shared intentions, general responsibilities, and commitments coherently manages these interactions and facilitates recovery when unanticipated problems arise. For example, in a general- teamwork model, if one team member fails and can no longer perform its role, each team member would be notified of the failure. This approach reduces the requirement for special-purpose exception handling mechanisms for each possible failure mode.

While early research on agent teamwork focused mainly on agent-agent interaction, developers are increasingly interested in various dimensions of human-agent interaction. Unlike autonomous systems designed primarily to take humans out of the loop, many new efforts specifically focus on supporting close and continuous human-agent interaction. Use of independent policy management mechanisms is becoming increasingly popular as a means of implementing reusable agent-agent teamwork models and enabling effective and natural human-agent interaction.

Security, mobility, and agent interaction can benefit from infrastructure-based governing mechanisms. Various solutions are becoming available to simplify agent analysis and control issues, independently from the agent application's core logic. Ideally, more uniform and interoperable approaches can eventually replace the plethora of current offerings adn enable many potential benefits:

- agents can interact more effectively;
- administrators can be assured that agent behavior conforms to desired constraints and objectives, even in the face of buggy or malicious code;
- users can be better aware of the agent's key state and intentions; and
- developers—their job simplified through independent management of security, mobility, and agent-interaction logic—can turn their focus to application logic.

Thus empowered, everyone can stop worrying about trustworthiness and start loving agents. ■

Jeffrey M. Bradshaw *is a research scientist at the Institute for Human and Machine Cognition, University of West Florida. Contact him at jbradshaw@ai.uwf.edu.*

Giacomo Cabri *is a research associate at the Department of Information Engineering, University of Modena and Reggio Emilia. Contact him at giacomo.cabri@unimo.it.*

Rebecca Montanari *is a research associate at the Department of Information and Systems Electronics, University of Bologna. Contact her at rmontanari@deis.unibo.it.*

Editor: Sumi Helal, Computer and Information Science and Engineering Dept., University of Florida, P.O. Box 116125, Gainesville, FL, 32611-6120, helal@cise.ufl.edu