Towards a Semantic Framework for Secure Agents

Extended Abstract

Patrick Lincoln   Carolyn Talcott
SRI International

March 16, 2003

1 Introduction

The agents paradigm is becoming increasely important for gathering and processing information, as well as many other tasks [5, 8, 6, 9]. Many factors contribute to the difficulty of understanding such systems and developing policies and mechanisms for achieving security goals. These include

- mobility of computation, agents, and devices
- agent autonomy
- heterogeneous communication media with wired and wireless connections and dynamic (possibly virtural) network topology
- multipolar security domains and stakeholders with diverse goals and concerns
- federations, collaboration, sharing of information across domain

We are developing a formal semantic framework and tool support for specifying, analysing and reasoning about secure agents and secure agent architectures. Within this framework we will be able to define executable models of agent systems, hostile environments, and mechanisms for control, detection, and protection. These models will serve as a basis for search and model-checking analyses, definition of abstraction mappings, and rigorous proofs of general properties. Using the framework we want to be able to represent and reason about:

- multiple views of a system and their relationships
- information—production, agregation, derivation, (mis)use, sharing, and flow
- high-level security goals, security policies and enforcement mechanisms, and their relationships across both concerns and domains

We have in mind not only the standard security concerns of authentication, authorization, integrity, and confidentiality considered in this more complex setting, but also concerns such as privacy, stealth, and information flow and agregation.

The secure agent framework should facilitate identification of vulnerabilities and assumptions at different levels of abstraction of system design. Furthermore the framework will support modeling concepts and mechanisms from existing work in areas such as intrusion detection and containment, fault tolerance, trust management, privacy, and multiple domain security formalisms in order to study the interactions of system requirements along different dimensions.

Web services such as the java-based web services (`http://java.sun.com/webservices`), can be considered as agents. There is a large body of work on languages, architectures and and frameworks for web agents and services to draw upon for ideas and for testing the adequacy of our developing semantic framework. Here are just a few examples. The Web Service Description Langauage (WSDL) (`http://www.w3.org/TR/2003/WD-wsdl12-20030124/`) enables one to separate the description of the abstract functionality offered by a service from concrete details of a service description such as "how" and "where" that functionality is offered. A semantic web service technology is being developed in the DARPA DAML program (`http://www.daml.org/services`). DAML-S supplies Web service providers with a core set of markup language constructs for describing the properties and capabilities of their Web services in order to support automation of tasks such as Web service discovery, execution, interoperation, composition and execution monitoring. The Open Agent Architecture developed at SRI `http://www.ai.sri.com/oaa/`) is a framework for integrating a community of heterogeneous software agents in a distributed environment, with services that facilitate cooperation and flexible interactions among agents.

2 A secure agent framework – first ideas

We present, informally, the current state of our design. Our approach builds on previous and ongoing work modeling object-based open distributed systems [14, 13], mobile languages [12], and network protocols and attacks (see [4, 10] for summaries). The formal tool support will be based on rewriting logic and the Maude environment [2, 3] and integrate / develop other tools as appropriate.

An agent system consists of nodes (hosts), agents, communication media (networks) and messages. Each node encapsulates a set of resources and provides services to access and control these resources. Nodes exist in a communication environment which could be constrained, for example behind a firewall. For a mobile node, the services available to and their quality may depend on its location and environment. Agents execute on nodes, use resources, move through the communication media, and generate and transmit information. Agents can be limited not only as to what services they may access but also how much and how often.

2.1 Services

A service may encapsulate or observe state, control use of a consumable resource, manage metadata, etc.

Execution environments

An execution enviroment encapsulates the execution of an agent, providing and controlling access to CPU and memory as well as to other node resources. For example Unix processes and Java virtual machines are execution environments. A trustworthy execution environment can be delegated responsibility to enforce some of the nodes resource allocation and service access policies, including access to itself (by an arriving agent). A defective or compromised environment could provide defective services, damage executing agents, or disseminate information extracted from agents using its execution service.

Communication

Communication services provide access to the communication media using communication abstractions such as connections and messaging. The communication abstractions may be provided with different levels of QoS in dimensions such as reliability, fault tolerance, confidentiality, integrity, and authentication. Mobile agent transport is a special case. This service packages an agent description and transmits it to a remote node to be executed.

Resource management and other services

Resource management services include: scheduling, reservation and admission control; adding and removing services; and managing policies (resources include metadata). Other services that a node might provide are: access to and management of databases—directories, repositories, libraries; access to connected devices—printer,fax, audio, sensors, actuators; trust management and security services (TMSS); and node-level health maintenance—intrusion detection and defense.

System-wide services and service interactions

Nodes may cooperate to provide system-wide services, for example distributed directory services and higher-level abstractions, such as group communication services (multicast, peer-to-peer, grid) or transactional services.

Services may have interdependencies, for example resource management and communication services may make use of TMSS, communication and TMSS may use directory services, and resource management may modify policies used by TMSS. Understanding and controlling the interactions of services with one another and with security goals is one of the key objectives to be supported by the agent semantic framework.

2.2 Agents

What do agents look like? We separate an agents structure into base-level and metadata. The base-level includes an identifier, a behavior description (static state), and data (dynamic state, bindings for parameters, partial results). Metadata might include credentials for access to services, resource allowances for use of cpu, memory, renting storage space, creating new agents, or further travel. Various elements of an agents structure maybe signed and/or encrypted. The separation of base-level structure from metadata is a means of supporting separation of concerns in reasoning about an agents behavior and the effects of its environment.

An agent runs in some execution environment. Service calls are the only way an agent can interact with its host node. Agents interact with one another by message passing and by effects on node state through common service access. We model an agents behavior by an abstraction of its behavior state and rules generating the possible sequences of service calls it makes during its execution.

Given some abstraction of service calls into service types, type systems will be developed to type agents according to the abstracted set of call sequences (resource use). An agent could come equipped with a proof that it has a given type as part of its metadata. Such service types could be used in policy specifications. The semantic framework allows a clear semantics to be given to such policies.

The following are some interesting questions about agents.

- It may be desirable for an agent to travel incognito. What does this mean formally? How does it interact with need to provide credentials or to validate results of an agents activities?
- Can an agent decide dynamically to trust some new host?
- Can an agent learn of new hosts and travel there securely? For example by getting the address and public key from a trusted host?
- Agents may have policies, for example how to use credits or when to expose information. How are these best represented?
- How can an agent achieve stealth? Can we engineer virus technology to produce stealthy agents, for example to protect them from damage? Not to mention to achieve access that would otherwise be denied.

2.3 Views of a system

To organize our thoughts and help to manage the complexity of reasoning about agent systems, a system is specified by giving multiple views and relations between the views. As an example a specification could be based on three views: end-to-end, system-wide, and local behavior. In the end-to-end view we can reason about principals, resources (including data/information), goals, and relations such as *principal A has authority over resource B*. In the system-wide view we can reason also about nodes (and their resources), messages (communications), and the communications media (at some suitable level of abstraction). From the loca behavior view, we can reason additionally about the behavior of agents and of node-level services. A local behavior provides a executable specification that can be used to run test scenarios and well as a basis for analysis or implementation. A multi view specification must also provide justification the the views form a coherent whole. Justifications include mappings between views, and conditions under which a executable specification meets system-wide or end-2-end requirements (see [15] for a detailed example of thie approach).

2.4 Security regions and groups

We take a security region to be a collection of resources under a common authority. The perimeter of such regions can be physical or logical. Nodes and execution environments provide ‘physical’ perimeters. Nodes can be organized and/or encapsulated in larger physical regions (lan, enterprise network behind a firewall). A logical perimeter could be a VPN or a federation of selected resources from multiple nodes. Regions might expand or contract, merge or split. Physically mobile nodes move from one region to another just like mobile agents can move from one node to another.

We propose a notion of security group as the analog of security clearance. Principals (or their agents) can form security regions and groups in order to selectively share sensitive information.

There are a number of questions to be studied here. How are group policies defined? How is group membership controlled? What are the consequences of an agent belonging to multiple groups? Possibly some ideas from secure group communication services can be used to make progress here.

3 Sketch of a possible challenge problem

Figure 1. shows the architecture of a system for planning, scheduling, and monitoring activities to carry out some tasks. The activities take place external to the planning and scheduling subsystem. The box labelled security perimeter corresponds to a physical security region. Inside this box are several agents, each running in its own execution environment. The dotted ovals indicate external agents or principals (depending on the view) that interact with the system. The wrapper around the agents that communicate with external agents indicate protection mechanisms (node or execution environment services) that enforce system security policies. The dashed rectangles represent external activities, possibly at geographically widely distributed locations, being monitored by the system, and possibly also by external observers. Some of the activities are results of commands from the system, others (weather, commercial activity) represent the environment in which the scheduled activities take place. The 'lightening bolts' indicate some possible points of attack: impersonation, modification of requests, attempts to book all resources, queries that expose restricted access information. The dashed arrows indicate possible collusion by external observers.

Some specific security issues include:

- Is the data from datafeeds used for monitoring valid? What damage could bad data cause—aborted/revoked plans, physical damage, . . . ?
- Are negotiated permissions (access to external resources) secure? What happens if a shedule is based on false assumptions regarding such permissions?
- What about external information agregation or covert channels due to external resource providers. For example external agents could combine information about permissions given.
- To what extent are planning and resource allocation strategies known to external agents? Knowing (parts of) the planning/allocation strategy, can a strategy be derived that allows to manipulate the overall resource assigment to prevent some tasks from being successfully carried out?
- How can timing and choice of external resource distribution be determined by the resource broker/scheduler to delay release of time sensitive information? And how can such properties be formally specified and check?

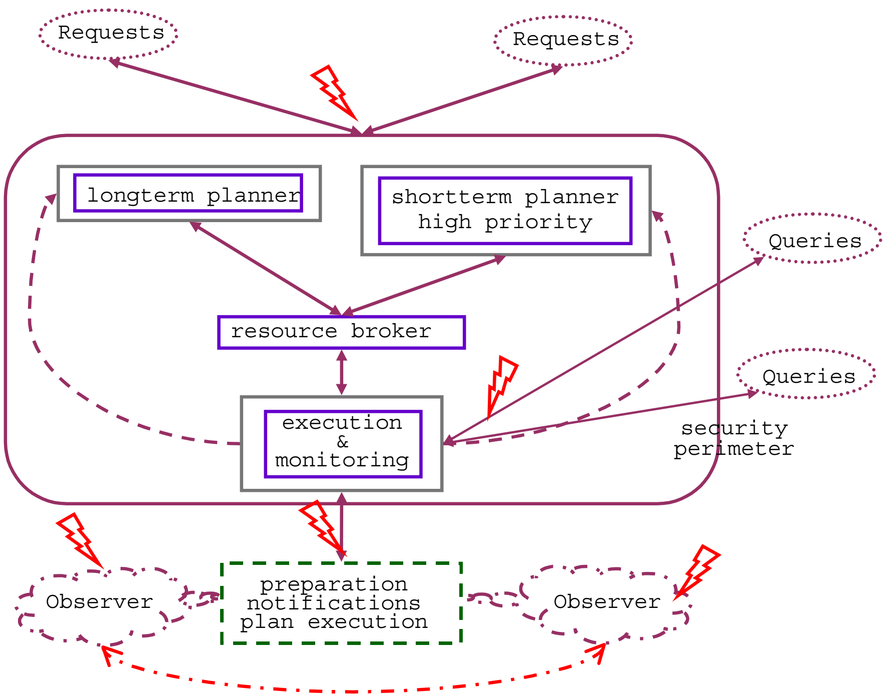

Figure 1: A plan-execute-monitor Architecture

4 Appendix: Infosphere Example

Consider an information management system such as that proposed for the Joint Battlespace Infospheres program [9, 1]. In a nutshell, an infosphere manages possibly distributed repositories of information objects, metadata schema and fuselets. Information objects have metadata describing the information content, according to a known schema. Clients interact with the system via publish, subscribe, and query interfaces. Fuselets [11] are local agents intended to carry out simple, special purpose transformations on published information objects. Fuselets can monitor for particular conditions and publish alerts, assemble and publish reports tailored to particular needs, etc. Published information may enter the system from remote sensors or observers, weather stations, external data repositories or data analysis systems, or entered by approved users.

Infospheres are intended to be used in military and intelligence efforts and thus security issues are important. What clients are allowed to interact, and what information can be published or retrieved by a given client is subject to access controls enforcing the infosphere's security policy.

In the following we give a flavor of how our secure agent framework and multiple view specifications might apply to specifying information flow properties of such systems.

Figure 2 illustrates an end-to-end view, focusing on interaction of clients mediated by an infosphere. When a client is authenticated and connected to the system it can be considered as an agent

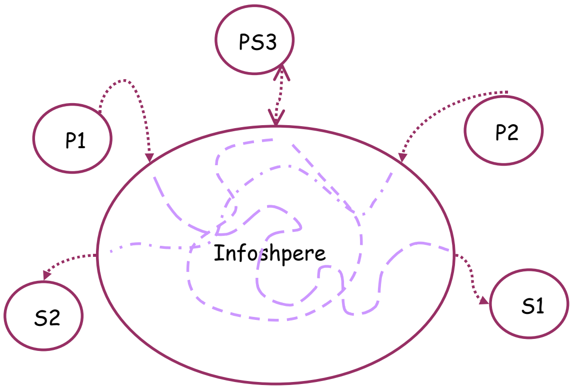

Figure 2: Infosphere end-to-end view

that is granted certain access privileges to the publish, subscribe, and query services. We model the system end-to-end view using interaction traces of publish, subscribe, query, notify, and retrieve events. Suppose the information objects are classified according to a set of security domains D. Events can be classified according to the information object communicated. (Probably only publish and retrieve are relevant.) We can give meaning to information flow requirements following Mantel [7]. For example, information must not flow from domain d to domain d' means that if we omit d events, then the resulting trace is also a possible behavior.

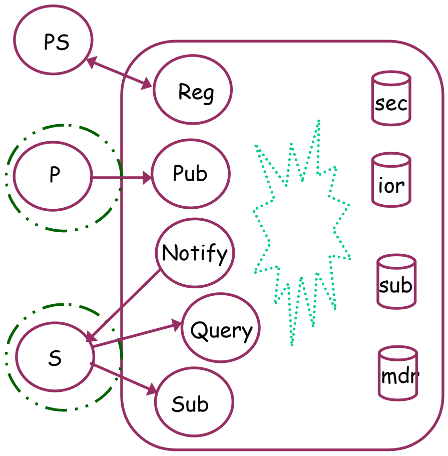

Figure 3: Infosphere system view

Figure 3 illustrates a system view – making explicit key architectural elements of an infosphere. System state is represented by data repositories: security policy (`sec`), information objects repository (`ior`), subscriptions (`sub`), and metadata schemas (`mdr`). There are the interaction services, and a registration service to admit an agent into a (virtual) execution environment. The auxiliary notification service monitors publication events and notifies clients when a publication matches a subscription. Descriptions of the JBI show clients interacting with an infosphere via the

Common API (represented by the services) across an authentication barrier. We model this barrier as an execution environment, for example that maintains connection state.

We require that the security policy (embodied in access control rules, trust management rules, etc.) ensures the end-to-end requirements. Further we require that grant client service requests are in accordance with the security policy. Satisfaction of these requirements implies that the infosphere meets the end-to-end information flow requirements.

Functional requirements can also be expressed. For example if there is a publication event that matches an active subscription then the subscriber will be notified (if access to the information is allowed). Also, responses to a query must in fact match the query specification.

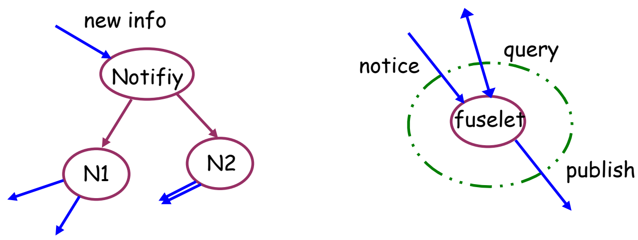

Figure 4: Infosphere behavior view

Figure 4 illustrates part of a behavioral view—the behavior of the notification service and that of a fuselet. We illustrate a notification behavior based on classifying subscriptions and delegating notification to different helpers depending on the subscription class. For example, one helper (`N1`) could handle group subscriptions by using multicast. We require that the group join for a particular group subscription is constrained to enforce the security policy. Another helper (`N2`) handles subscriptions to highly sensitive information and thus must make additional authentication and security precautions in making notifications.

A fuselet is illustrated running in its execution environment that monitors its behavior and controls its access (as a client) to information services. Specification of a fuselet describes its information transformation properties, including possible change of security domain. This can be ensured by analysis of fuselet itself, or enforced by its execution environment.

References

[1] Mercury capabilities guidelines, Jan. 2003. `http://www.if.afrl.af.mil/tech/programs/jbi/mercury.cfm`.

[2] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Marti-Oliet, J. Meseguer, and J. Quesada. Maude: Specification and programming in rewriting logic, 1999. `http://maude.csl.sri.com/manual`.

[3] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Marti-Oliet, J. Meseguer, and J. Quesada. Towards Maude 2.0. In *Third International Workshop on Rewriting Logic and Its Applications (WRLA'2000), Kanazawa, Japan, September 18 — 20, 2000*, volume 36 of *Electronic*

Notes in Theoretical Computer Science. Elsevier, 2000. `http://www.elsevier.nl/locate/entcs/volume36.html`.

[4] G. Denker, J. Meseguer, and C. L. Talcott. Formal specification and analysis of active networks and communication protocols: The Maude experience. In *DARPA Information Survivability Conference and Exposition (DISCEX 2000)*, volume 1, pages 251–265. IEEE, 2000.

[5] *Mobile Agents 2001*, volume 2240 of *Lecture Notes in Computer Science*. Springer Verlag, December 2001.

[6] *6th IEEE Mobile Agents Conference*, Lecture Notes in Computer Science. Spinger Verlag, 2002.

[7] H. Mantel. On the Composition of Secure Systems. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 88–101, Oakland, CA, USA, 2002. IEEE Computer Society.

[8] *3rd International Workshop on Mobile Agents for Telecommunication Applications (MATA 2001)*, volume 2164 of *Lecture Notes in Computer Science*. Springer, 2001.

[9] J. McCarthy, D. Frost, R. Katz, R. Sproull, C. Morefield, and V. Gawron. Building the joint battlespace infosphere volume 1. Technical Report SAB-TR-99-02, United States Air Force Scientific Advisory Board, 1999. `http://www.rl.af.mil/programs/jbi/docs.cfm`.

[10] J. Meseguer, P. C. Olveczky, M.-O. Stehr, and C. L. Talcott. Maude as a wide-spectrum framework for formal modeling and analysis of active networks. In *DARPA Active Networks Conference and Exposition (DANCE)*, pages 494–510. IEEE, May 2002.

[11] J. Milligan. Fuselet definition document, 2003. `http://www.if.afrl.af.mil/tech/programs/jbi/documents/fuselet_definitio%n.doc`.

[12] M.-O. Stehr and C. Talcott. PLAN in Maude: Specifying an active network programming language. In *Fourth International Workshop on Rewriting Logic and Its Applications (WRLA'2002), Pisa, Italy, September 19 — 21, 2002*, volume 71 of *Electronic Notes in Theoretical Computer Science*. Elsevier, 2002. `http://www.elsevier.nl/locate/entcs/volume71.html`.

[13] N. Venkatasubramanian. *Resource Management in Open Distributed Systems with Applications to Multimedia*. PhD thesis, University of Illinois, Urbana-Champaign, 1998.

[14] N. Venkatasubramanian, G. Agha, and C. L. Talcott. A formal model for reasoning about adaptive QoS-enabled middleware. In *Formal Methods for Increasing Software Productivity (FME2001)*, 2001.

[15] N. Venkatasubramanian and C. L. Talcott. A semantic framework for modeling and reasoning about reflective middleware, 2001.