Multi-User and Security Support for Multi-Agent Systems

Agostino Poggi, Giovanni Rimassa, Michele Tomaiuolo
Dipartimento di Ingegneria dell'Informazione – Università di Parma
Parma, Italy
{ poggi, rimassa, tomaiuol }@ce.unipr.it

Abstract

This paper discusses the requirements an agent system needs to be secure. In particular, the paper introduces a classification of modern distributed systems, and examines the delegation concept from a security point of view. After discussing the peculiar security and delegation issues present in distributed object systems, mobile agent systems and in multi agent systems, a case study is presented, describing the multi-user and security support that is being built into the JADE platform.

1 Introduction

The enormous availability and the low cost-per-user of distributed systems have brought almost every kind of business to try and exploit wide area digital networks for at least some purpose. This happened first with the Internet and it is happening again now with ubiquitous computing. Such a commercial exploitation of distributed systems puts several burdens on software applications, and security is by no means the most lightweight one.

As more and more services are being made available through the Internet or through wireless cellular networks, more and more data travel through public, multi-provider and multi-authority network infrastructures. All these data transfer operations need several security features, just like the familiar triple of confidentiality, integrity, and non repudiation properties.

In the past years, extensive research on cryptographic algorithms produced several fundamental results, and nowadays various public, trusted implementations of these algorithms are available. Still, given the difference between mathematics and reality, having strong cryptography is just a very first step towards having a secure computing infrastructure: security is a process, that entails policies and organizational structure [20]. So, the focus moved from algorithms and techniques to models, policies and infrastructures.

Above and beyond the need for security, the driving forces mentioned before are generally pushing towards more and more complex applications, where the security aspect needs to blend with many others, like interoperability, flexibility and adaptability.

In order to cope with those demanding requirements, the general principles of information hiding and loose coupling among software components are becoming more and more important. In particular, several researchers proposed the agent concept to describe the most modern, highly encapsulated software components. This concept has a number of facets, and different approaches tend to concentrate themselves on different facets.

This paper will analyze security issues in modern distributed systems, and will also discuss the concept of delegation, which plays a very important role in flexible software architectures, and becomes strongly related to security issues when software agents are used as system components.

To better structure the discussion and to highlight the link between a system's architecture and how it deals with security and delegation, distributed software systems will be classified into three different categories. While this classification is clearly neither exhaustive (that is, there are distributed systems that don't belong to any listed category) nor unique (that is, a given system can belong to more than a single category), it will provide some guidance in separating the various concerns. The main focus of the classification is to highlight differences among the categories, and to attach to each category its most typical security and delegation issues.

2 Architectural Styles for Modern Distributed Systems

The term *software architecture* denotes the large scale structure of a software system, a very useful concept to describe, understand and design complex software [9]. The software engineering community recognized that there are many more systems than architectures, and that most software systems tend to cluster around a small set of templates. These templates have been identified and documented as *architectural styles* in [19] or *architectural patterns* in [4] (albeit with some differences in the attitude between the two communities).

In particular, in [19] the authors attempt a classification of architectural styles, distinguishing six major classes:

1. *Data flow styles*, where the system structural information lies mainly in the motion of data among the various parts.

2. *Call-and-return styles*, where the system architecture is dominated by the order of computation, usually in a single thread of control.
3. *Interacting processes styles*, where the system architecture results mainly from the communication patterns among multiple concurrent computation flows.
4. *Data-centered repository styles*, where multiple computations revolve around a complex, centered data store.
5. *Data-sharing styles*, where data are directly shared among the different system parts.
6. *Hierarchical styles*, where the system is partitioned into subsystems with reduced coupling and limited interaction between them.

While the classification covers several architectures and is quite useful, it suffers from too much heterogeneity and fails to analyze in sufficient detail the most modern architectural styles for distributed systems, with which this paper is concerned. However, the classification criteria are general and extensible enough to accommodate newer and more precise classes.

For the purpose of this paper, three architectural styles will be introduced and described according to the guidelines of [19], listing the *components* and *connectors* that characterize each style, and focussing mainly on the *control/data interaction* criterion to highlight the difference between them, which is rather natural because all the three styles belong to the *interacting processes* family.

The first style, which is the only one explicitly cited in [19], is named *Distributed Objects*. Its components are called *managers* in [19], but are nothing but network-accessible objects; the objects are connected through *remote procedure call* (better, through *remote method invocation*). About the control/data interaction, the authors first state that the data-flow and the control-flow graphs are not isomorphic, then they say that no concept of direction is given for these two flows, thereby saying almost nothing at all. Clearly, this shows that their preliminary work fails to properly describe the *Distributed Objects* architectural style.

But the general classification framework is still valid, so this paper will characterize the control/data interaction for *Distributed Objects* as follows:

(1) *The data are encapsulated within objects, and can only be manipulated through the remote interfaces exposed by objects. Moreover, the connections through remote method invocations must comply with the constraints expressed by the interface exposed by the receiver object*.

The definition (1) is the extension to distributed systems of the *Design by Contract* principles [12]. The *Distributed Objects* architectural style can then be described as the style of distributed systems using objects as components, remote method invocations as connectors and *Design by Contract* to coordinate data and control. Popular examples are the CORBA [15] standard and the DCOM [6] infrastructure.

The second style can be named *Mobile Agents*. It has two kinds of components: the first one is made by active entities encapsulating code and state, that can be transferred across the network, named *mobile objects* or *mobile agents* in different cases. To the second kind belong more heavyweight components, hosting mobile agents and providing them with an execution environment. These are called *places*, *containers*, *processing environments* or in several other ways. The connectors available in this style are *life support*, connecting one mobile agent exactly with one place (its current location), *migration*, connecting two places and one agent, and *communication*, connecting two agents.

The *migration* connector allows an agent component to dynamically change its place, transferring its whole computation (i.e. code and data) somewhere else. This connector is one of the possible connectors based on *mobile code*, that can be derived from the taxonomy in [8]; even if a more general architectural style named *Mobile Code* could be defined, this paper will only consider the *Mobile Agents* case.

The *communication* connector is exactly the one the *Interacting Processes* family is based upon. If the *Mobile Agents* software system uses object oriented technology, the *communication* connector is typically method invocation (local or remote), and follows *Design by Contract*. The control/data interaction for the *Mobile Agents* architectural style can be defined as follows:

(2) *The data and control interact just like in the Distributed Objects case, but there is an additional migration connector that allows agent migration while retaining computation state*.

Example systems following this style are Mole [3] and Aglets [11].

The third architectural style can be named *Multi Agent Systems*. The only type of component present in this style is the *agent* (the notion of *environment* is sometimes explicitly modeled as a component, connected to all the agents), and the only connector is the *communication* connector (if the environment is represented as a component, then the two *sensing* and *acting* connectors can also be introduced).

The distinguishing feature of the agent component is its *autonomy*: an agent can initiate interactions without the need of an external stimulus. This basic autonomy feature may be held also by components of the *Mobile Agents* style: mobile agents that can initiate their own communication and migration interactions are autonomous in the simple sense of the definition above.

But, the major trait of the *Multi Agent Systems* style is the peculiar control/data interaction. The *communication*

connector allows agents to exchange messages belonging to an *Agent Communication Language* (*ACL* for short). Such languages are based on *speech act theory* ([2], [21]) and are generally equipped with some kind of logic framework that expresses their semantics. Using one of these ACLs to communicate among components strictly extends *Design by Contract*: beyond the request-for-service primitive (*imperative*), the query-for-information (*interrogative*) and the notification (*informative*) primitives are directly available.
Examples of this style are the FIPA standard [7] and the dMARS system [17].

3 Security Threats and Countermeasures

Security is a relative concept: it makes no sense to claim a system is secure without explicitly stating what kinds of threats the system can successfully stand. Therefore, since security itself is more a process than a product, the first fundamental step is *threat modeling*. When trying to describe possible attacks against a given target, the system nature affects which kind of threats are to be feared the most. So, the previous classification based on architectural styles can help a lot in the threat modeling task.
If the target system follows the *Distributed Objects* style, there are several possible attack kinds, like e.g. unauthorized access, masquerading on denial of service. A system using the *Mobile Agents* style is vulnerable to many more threats, because it removes the trust relationship between the agent and its execution environment. A lot of research has been done to cope with these additional problems; two interesting examples are [16] and [18]. The *Multi Agent Systems* style removes the need for mobile-code induced security measures, but adds another layer of threats.
In a multi-authority setting where participants may have reasoning capabilities, it would be useful to have e.g. negotiation protocols where some pieces of information are guaranteed not to be inferred. So, there is a special security aspect involved in the knowledge-level interaction promoted by multi-agent systems.

4 Delegation and Security

From a general software engineering standpoint, the importance of designing systems made by several components, interacting through well defined interfaces is well known. So, it is often the case that a component is held responsible for a task, but it really relies on other helpers to carry on its duty.
In object-oriented programming, the term *delegation* indicates exactly the process above, particularly when the helper objects are unknown to clients. Along the lines of *actor languages* [1], the delegation mechanism has also proved to be a possible alternative to class-based inheritance: a more recent example of this is the Self language [22]. Nowadays, the most popular distributed objects frameworks allow both inheritance and delegation based implementation for a remote interface.
When delegation is seen as a replacement for inheritance, it doesn't affect security too much, but things are different for agent systems: an agent acts on behalf of someone, so the delegation action generally implies a transfer of capabilities and permissions. In [13], the authors propose an extension to the Java Security API to support secure delegation in mobile objects environments.
Adding ACL communication to autonomy further affects the delegation process: the act of delegating a task becomes loaded with social meaning. In [14], the authors analyze delegation and responsibility in multi-agent systems using a framework based on actions and states, eventually proposing a semantics for imperative speech acts rooted in deontic logic.

5 Case Study: Security and Multi User Support in JADE

As a concrete outcome of addressing the security and delegation issues in Multi-Agent Systems, this paper presents a case study about the design and implementation of multi-user and security support for a FIPA compliant platform for Multi-Agent Systems.
Drawing some conclusions from the considerations made in previous sections, two major requirements are:

- The security model should be tailored for the *Multi Agent Systems* architectural style specific features, without of course overlooking more fundamental issues, common to all three kinds of distributed systems.
- The security model should take into account delegation as a common interaction mechanism, enabling secure and flexible delegation between agents.

The remainder of this section details the security model implemented for the JADE agent platform.

5.1 The JADE agent platform

JADE [5] is a software framework to develop multi-agent systems in compliance with the FIPA specifications, available at [10]. It supports most of the infrastructure related FIPA specifications, like transport protocols, message encoding, and white and yellow pages agents. Moreover, it has various tools that ease agent debugging and management.
However, no form of security has been built into the JADE agent platform so far, and the system itself is a single-user system, where all the agents belong to a single authority and have equal rights and permissions. This means that it is not possible to use JADE in several real world application, such as electronic commerce.
To deal with these limitation, a multi-user support has been designed and implemented within an experimental version of JADE, leveraging the Java Security APIs to cast a security model on the agent platform. This model

takes into account some of the peculiarities of multi-agent systems as an architectural style, and has specific support for delegation.

5.2 A Security Model for JADE

Being JADE an agent system implemented using Java language, it's quite obvious to build a security model as close as possible to the existing one, trying to create extensions where necessary.

JADE security model should necessarily provide concepts representing principals, resources and access permissions, for managing authentication of users and agents, for granting necessary authorizations. Furthermore it should provide constructs to ease system administration, as hierarchies of principals and resources, groups and roles. The possibility to securely delegate tasks to other entities could be useful to complete certain tasks in a complex community of agents, where particular trust relations can rise among certain agents.

Finally, communications should remain reserved when this is required by applications critical about security concerns, as those managing financial transactions or implementing electronic commerce protocols.

The Java 2 platform adopts user configurable security policies to decide whether granting access permissions to executing code. These decisions follows a *code-centric* policy, that is they are taken considering from where the code comes from, whether it is signed or not and, in case of signed code, by whom. This code-centric control access adopted by Java is unusual; traditional security means, typically used by multi-user operating systems, are centered on users. It's quite obvious that a multi-agent system, and any other multi-user environment, hardly adapts itself to be administered with a code-centric security style; rather, it will need infrastructures and programming interfaces for authenticating users and assigning privileges.

In JADE security model a *principal* represents any entity whose identity can be authenticated. Principals are bound to single persons, departments, companies or any other organizational entity. Moreover, in JADE even single agents are bound to a principal, whose name is the same as the one assigned by the system to the agent; with respect to his own agents, a user constitutes a membership group, making thus possible to grant particular permissions to all agents launched by a single user.

Resources that JADE security model cares for include those already provided by Java model, including local file system elements, network sockets, environment variables, database connections. But there are also resources typical of multi-agent systems that have to be protected against unauthorized accesses. Among these agents themselves and agent execution environments.

A *permission* is an object which represents the capability to perform actions. In particular, JADE permissions, inherited from Java security model, represent system resources. Each permission has a name and most of them include a list of actions allowed on the object, too.

To take a decision while trying to access a resource, access control functions compare permission granted to the principal with permission required to execute the action; access is allowed if all required permissions are owned.

An useful tool to simplify security administration in an organization is delegation of administration rights, only with regard to definite subsets of whole organization domain. This is possible as the existence of resources organized in a hierarchy makes simple to assign to an administrator the responsibility to manage particular a group or set of resources. Moreover, the fact that it is not possible in any way to create objects or users with more access rights than their creator guarantees that further delegations will stay confined within the assigned domain.

5.3 Certification Authority

JADE security is in great part founded on the availability of a certain number of *certificates*, which can attest the identity of an entity or the permissions granted to it. To make verifying the authenticity of those certificates possible, it is necessary that those certificates are signed by a publicly recognized *authority*.

A JADE authority consists essentially of a couple of keys, one of which is kept private and the other is made known to all. This simple approach allows inserting of a JADE agent platform into an existing *Public Key Infrastructure* (PKI).

5.4 Authentication

The first step to assign access permissions to various agents executing on the system is to authenticate their users, and only then the agents themselves. In a JADE agent system a user authenticate herself providing a name and a password, and receiving back a certification of her identity along with granted permissions. Such permissions can be delegated to the agents launched on the system by the user, thus maintaining a precise control over what actions single agents are allowed to perform.

The creation of an agent container needs an authentication of the user responsible for it, too. This information will remain thereafter bound to the container, thus being able to protect even this object against unauthorized actions, as for example an attempt to close the container from a user or an agent lacking required permissions.

Agents need to be authenticated by the system when they require access to protected resources. Authentication involves the presentation of an opportune identification means, an *identity certificate*.

The elements of an identity certificate include the identity of the subject, the identity of the emitting entity, an identification of algorithms used to protect the certificate, its validity period. These elements can be used to establish

the certificate validity and the binding between certificate and agent.
Since principals are hierarchically structured, the identity of each agent includes that of its user and of all responsible entities in higher hierarchical levels, too.

5.5 Authorization

To assure necessary access protection, running code should be bound to an authenticated principal, so that permissions actually owned by that principal are controlled by the Java security support. In Java 2 security model a *protection domain* is simply a set of classes, such that their instances are granted the same privileges.
Domains are identified on the base of signatures on the byte code and the address from which it was downloaded.
However this form of authorization is not sufficient in an environment as JADE, where agents can belong to different users, and perhaps even to different companies.
Instead it's necessary to be able to decide about resource access on the basis of the authority responsible for a given execution, besides from source code; in other words a *subject* needs to be attached to the current access control context.
The subject in particular must be in possess of a certificate attesting the identity of the principal responsible for the running code and a certain number of certificates carrying permissions actually owned by the principal, typically an agent.
Besides from the identity certificate, each agent can dispose of different authorization certificates, too, which list particular permissions granted to the agent. The agent can obtain these certificates at creation time directly from its owner, or through a delegation from other agents living on the system. The system in any case checks that the passage of these permissions is legitimate; more precisely the system makes sure that the delegating entity really owns these permissions and has the right to further delegate them.

5.6 Delegation

The essence of a secure delegation is to be able to verify that an object, declaring to act on behalf of others, has really the necessary authorizations. Moreover, in the case of multi-agent systems, the delegation mechanism should be flexible enough to be used as the basis for a secure treatment of speech acts and interaction protocols, allowing e.g. to revoke a delegation or to forbid further delegation of a given task.
To grant this characteristics, the security model thought for JADE adopts a faceted approach, supporting different styles and protocols: it provides for example simple delegation (*representation* or *deputation*) and *cascade* or *chained* delegation, and provides means to revoke granted delegations. The model adopted for secure delegation in JADE derives from the one of [13], adapted to multi agent systems with a different implementation and a different permissions structure.
A series of agents can be involved in a certain conversation. For example, an agent *A*, utters an imperative speech act to another agent *B*, which agrees to carry out the task. B could complete the task by itself or could in turn request a sub-task to another agent, C. This forms in effect a delegation chain where A is the *initiator*, C is the *final target*, and B is an *intermediate*.
There are three approaches or modes that can be applied to such chains:

- *No delegation*. The intermediate only exercises its own rights for further requests.
- *Simple delegation*. Also known as representation, can be limited or unlimited.
- *Cascaded delegation*. Rights coming from initiator are combined with those from delegated.

JADE uses certificates for implementing secure delegation, too. A *delegation certificate* is a document which attests the necessary authorization to execute determined actions on behalf of others.
The agent which starts a delegation process sets the content of a certificate to govern the use of computation resources and security mechanisms by delegated agents. Then this certificate must be signed by an acknowledged authority that attests its legitimacy, checking in particular that the delegating entity is really in possess of the permissions it intends to delegate. In the certificate the delegating and delegated principals are reported. The latter can even identify a group of users or agents to which the delegation is conceded. Furthermore a list of permissions granted to delegated entities is present.

5.7 Security Policies

Usually services implement a security model based on access control, defining a set of protected resources together with the conditions under which identified principal can access these resources. The model defined for JADE follows this approach, and defines a security policy to specify which resources are accessible to which principals. This policy extends the default policy adopted in Java 2, and in effect the two policies together form in a logic sense a single access policy for the whole Java-based multi agent environment.

5.8 Secure Communication

JADE containers can be connected to the main platform through remote links, thus in general through geographic networks, too. Then, communications between agents situated in different locations are exposed to attacks by eavesdroppers who can obtain reserved information. Anyway it is obvious that a solution granting communications between remote JADE containers to remain reserved should be provided.
To assure protection of transmissions in JADE, the *Secure Socket Layer* (*SSL*) protocol is used. It's a general

purpose protocol for TCP/IP networks, which can provide authentication and encryption of TCP connections.

Using SSL as a base to protect JADE transmission is appealing for a number of reasons.

First of all SSL is emerging as a standard for secure Internet communications. Thanks to its popularity numerous implementations are available, some of which fully written in Java, too. Moreover, SSL plugs itself into the system at socket level, so it can be used under several application protocols such as HTTP, IIOP and JRMP (the RMI on the wire protocol). SSL also allows mutual authentication of both parties of a network connection. This feature allows a platform to protect itself from intrusions if some malicious host tries to mask itself as a host where a trusted JADE platform is running.

Since RMI is a distributed object broker above TCP/IP sockets, enabling secure RMI communications is simply a matter of replacing standard sockets with SSL sockets. The same approach could be followed for IIOP and HTTP, but the current implementation limits itself to RMI and therefore to intra-platform communication. This is due to the lack of FIPA specifications for a secure inter-platform protocol; this issue, together with others, is being addressed by FIPA in its ongoing security workplan.

6 Conclusions and Future Work

This paper discussed the issues that need to be considered to provide a security support for a framework for distributed systems, with a focus on multi agent systems. The interesting connection between delegation and security model was analyzed, and a practical case study for the JADE agent platform was described. The prototype implementation is making its way to the mainstream JADE code base. Beyond improving the implementation to increase its effectiveness and usability, more work is needed on the security model, that does not completely address higher level resources such as ontologies or conversations. This work will likely involve JADE users and the FIPA community at large into a public discussion about security needs and solutions.

7 References

[1] G. Agha. *"Actors, A Model of Concurrent Computation in Distributed Systems"*. MIT Press, 1986.

[2] J.L. Austin. *"How to do things with words"*. Oxford University Press, 1962.

[3] J. Baumann, F. Hohl, K. Rothermel and M. Strasser. *"Mole – Concepts of a Mobile Agent System"*. World Wide Web, 1(3):123-137, 1998.

[4] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, S. Stal. *"Pattern Oriented Software Architecture: A System of Patterns"*. John Wiley & Sons, 1996.

[5] F. Bellifemine, A. Poggi, G. Rimassa. *"Developing Multi-Agent Systems with a FIPA-compliant Agent Framework"*. Software: Practice & Experience, 31:103-128, 2001.

[6] D. Box. *"Essential COM"*. Addison-Wesley, 1998.

[7] FIPA 2000 Specifications. Available at http://www.fipa.org.

[8] A. Fuggetta, G.P. Picco, G. Vigna. *"Understanding Code Mobility"*. IEEE Transactions on Software Engineering, 24(5): 342-360, 1998.

[9] D. Garlan, M. Shaw. *"An Introduction to Software Architecture"*. In Advances in Software Engineering & Knowledge Engineering, Vol. II, World Scientific Pub Co., 1993, pp. 1-39.

[10] The JADE Project Home Page. http://jade.cselt.it.

[11] D.B. Lange, M. Oshima. *"Programming and Deploying Java Mobile Agents with Aglets"*. Addison Wesley, 1998.

[12] B. Meyer. *"Object-Oriented Software Construction, 2nd Ed."*. Prentice Hall, 1997.

[13] N. Nagaratnam, D. Lea. *"Role-based Protection and Delegation for Mobile Object Environments"*. Proc. of COOTS '98, 1998 USENIX Conference on Object-Oriented Technology and Systems, Santa Fe, NM, 1998.

[14] T.J. Norman, C. Reed. *"Delegation and Responsibility"*. In Proc. of ATAL '00, 7th International Workshop on Agent Theories, Architectures and Languages, Boston, MA, 2000.

[15] Object Management Group. *"CORBA 2.4 specification"*. http://www.omg.org.

[16] H. Peine. *"Security Concepts and Implementation in the Ara Mobile Agent System"*. In Proc. of 7th IEEE Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, Stanford University, USA, 1998.

[17] A.S. Rao, M.P. Georgeff. *"BDI Agents: From Theory to Practice"*. Proc. of ICMAS '95, 1st International Conference on Multi-Agent Systems, San Francisco, CA, 1995; 312-319.

[18] V. Roth, M. Jalali. *"Concepts and Architecture of a Security-centric Mobile Agent Server"*. Proc. of ISADS '01, 5th International Symposium on Autonomous and Decentralized Systems, Dallas, TX, 2001.

[19] M. Shaw, P. Clements. *"A Field Guide to Boxology: Preliminary Classification of Architectural Styles for Software Systems"*. Proc. COMPSAC '97, 21st International Computer Software and Applications Conference, August 1997, pp. 6-13.

[20] B. Schneier. *"Secrets and Lies"*. J. Wiley and Sons, 2000.

[21] J.R. Searle. *"Speech acts: An essay in the philosophy of language"*. Cambridge University Press, 1969.

[22] R.B. Smith, D. Ungar. *"Programming as an Experience: The Inspiration for Self"*. Proc. ECOOP '95, Aarhus, Denmark, 1995.