Hierarchical and Declarative Security for Grid Applications

Isabelle Attali, Denis Caromel, and Arnaud Contes

INRIA Sophia Antipolis, CNRS - I3S - Univ. Nice Sophia Antipolis,
BP 93, 06902 Sophia Antipolis Cedex - France
First.Last@inria.fr

Abstract. Grid applications must be able to cope with large variations in deployment: from intra-domain to multiple domains, going over private, to virtually-private, to public networks. As a consequence, the security should not be tied up in the application code, but rather easily configurable in a flexible, and abstract manner. Moreover, any large scale Grid application using hundreds or thousands of nodes will have to cope with migration of computations, for the sake of load balancing, change in resource availability, or just node failures.
To cope with those issues, this article proposes a high-level and declarative security framework for object-oriented Grid applications. In a rather abstract manner, it allows to define a hierarchical policy based on various entities (domain, host, JVM, activity, communication, ...) in a way that is compatible with a given deployment. The framework also accounts for open and collaborative applications, multiple principles with dynamic negotiation of security attributes and mobility of computations. This application-level security relies on a Public Key infrastructure (PKI).

1 Introduction

This paper aims at proposing an infrastructure where Grid security is expressed outside the application code, outside the firewall of security domains, and in both cases in a high-level and flexible language. The security model discussed in this paper is based on *ProActive* [1, 2], a Java library for developing concurrent, distributed and mobile applications. Moreover, as the costs of *Authentication, Integrity, Confidentiality* (A,I,C) are far from negligible, these characteristics have to be set according to the security requirements and to a given deployment (both hosts and networks).

Overall, we propose a framework allowing:

- declarative security attributes (A,I,C), outside any source code and away from any API;
- policies defined hierarchically at the level of administrative domain,
- dynamic security policy, taking into account the nature (private or public) of the underlying networks;
- dynamically negotiated policies (for multi-principals applications),

– policies for remote creation and migration of activities.

Our security framework focuses on authentication of users and processes; it supports user-to-process, process-to-user, process-to-process authentication. We provide authentication solutions that allow a user, user's processes, and the resources used by those processes to verify each other's identity and to communicate securely. We assume that processes are running on a trusted environment, our framework is not intended to protect objects from malicious JVM but applications from each other and network attacks.

Section 2 introduces and discusses related work. Section 3 provides the requested background on the research *ProActive* platform. Section 4 introduces the security model.

2 Related Work

There is a large body of previous work on security in distributed systems. Legion [3] is the closest system to ours. It encapsulates functionalities as objects and provide mechanisms for their location, migration, etc. Objects in these systems interacts via remote method invocations and the main security objectives are authenticating the communications parties, protecting traffic, enforcing access control, delegating rights and enforcing site-specific security concerns. Unlike our system, Legion does not support a hierarchical management of security policies.

The Globus system [4] is a computational grid providing pervasive, dependable, and consistent access to high-performance computational resources, despite geographical distribution of both resources and users. In [5], authors propose a strategy for addressing security within the Globus' Open Grid Services Architecture (OGSA). It defines a comprehensive Grid security architecture that supports, integrates and unifies popular security models, mechanisms, protocols, platforms and technologies in a way that enables a variety of systems to interoperate securely. The use of a user proxy addresses single sign-on and avoids the need to communicate user credentials. The architecture also supports an inter-domain and intra-domain security interoperability. The notion of virtual organization is defined as a set of individuals and/or institutions sharing resources and services under a set of rules and policies governing the extent and conditions for that sharing.

The .NET [6] framework provides security features allowing protection of a host against malicious code. Security system is based on user- and code-identity using public/private keys, hash functions, digital signatures and security policies. There are four policy levels Enterprise, Machine, User and Application Domain.

However, all these security architectures are not aware of object migration. Mobile security concepts are widely studied in agent systems. There are numerous agent systems like MAP [7], Ajanta [8], Aglets [9], Mole [10]. In an agent system, a user creates an agent which is a mobile object and migrates its agent to hosts where the agent can perform some tasks. Security in such systems involves the protection of a host from the agents that reach it and run on it, protection of an agent from other agents, from its host, from the outside environment. The security model considers the issue of authentication and authorization to protect the hosts from the agent and the agents from others. Security is based on public key authentication.

Nevertheless, agent systems' security model considers the existence of a fixed base. An agent starts from the base, goes to remote sites and comes back to its base. In our model, applications are fully mobile.

3 Background on Distributed Mobile Objects

ProActive is a Java library for distributed and mobile, and Grid computing, implemented on top of RMI [11] as the transport layer. *ProActive* features remote active objects, asynchronous two-way communications with transparent futures, high-level synchronization mechanisms, and migration of active objects. As *ProActive* is built on top of standard Java APIs, it does not require any modification to the standard Java execution environment.

3.1 Base model

A distributed or concurrent application built using *ProActive* is composed of a number of medium-grained entities called *active objects*. Each active object has one distinguished element, the *root*, which is the only entry point to the active object; all other objects inside the active object are called *passive objects* and cannot be referenced directly from objects which are outside this active object (see Figure 1); the absence of sharing is important with respect to security.

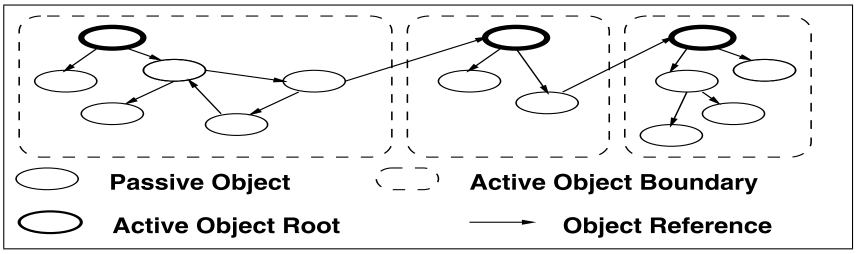

Fig. 1. A typical object graph with active objects

3.2 Deployment descriptors: Nodes and Virtual Nodes

The security architecture presented in this article relies on two related abstractions for deploying Grid applications: *Node* and *Virtual Node*. We provide here a very brief introduction to those concepts, see [2] for further details.
A *Node* gathers several active objects in a logical entity. It provides an abstraction for the physical location of a set of activities. At any time, a JVM hosts one

or several nodes. Active objects are bound to a node at creation or after migration. In order to have a flexible deployment (eliminating from the source code machine names, creation protocols, registry and lookup protocols), the system relies on *Virtual Nodes* (VNs). A VN is identified as a name (a simple string), used in a program source, defined and configured in an XML descriptor file. The correspondence between Virtual Nodes and Nodes, the mapping, is created by a deployment descriptor.

4 Declarative Grid security

Grid applications usually involve hundreds of objects on worlwide grids. Grid's resources are dynamically acquired and often differ at each execution. Grid security needs to be defined at a high level of abstraction to address theses issues. One must be able to express policies in a rather declarative manner. The general syntax to provide security rules is the following:

```
Entity[Subject] -> Entity [Subject]
                              : Interaction # [SecurityAttributes]
```

Being in a PKI infrastructure, the subject is a certificate, or credential. Other "elements" (Domain, Virtual Node, Object) are rather specific to Grid applications and, in some cases, to the object-oriented framework. An "entity" is an element on which one can define a security policy. "Interaction" is a list of actions that will be impacted by the rule. Finally, security attributes specify how, if authorized, those interactions have to be achieved.

In order to provide a flavor of the system, we consider the following example.

```
Domain[inria.fr] -> Domain[ll.cnrs.fr] : Q,P # [+A,+I,?C]
```

The rule specifies that between the domain *inria.fr* (identified by a specific certificate) and the parallel machine *ll.cnrs.fr*, all communications (reQuests, and rePlies) are authorized, they are done with *authentication* and *integrity*, *confidentiality* being accepted but not required.

Figure 2 provides the general syntax of policy descriptors. The keywords *A*ccept and *D*eny allow to define policies in explicit acceptance or denial form. For instance, to operate in mode "*everything that is not authorized is precluded*", potentially to a specific domain, a policy should include a rule:

```
Deny: Domain[*] -> Domain[inria.fr] : *
```

The remainder of this section defines and details the different elements. We first identify the participating entities (Hosts, Domains, etc.), the security subjects (e.g. users), security objects (e.g. objects, resources, communication, ...), and of course the security relationships (i.e. attributes) between them.

4.1 Hierarchical Security Entities

Grid programming is about deploying processes (activities) on various machines. In the end, the security policy that must be ensured for those processes depends upon many factors: first of all, the application policy that is needed, but also the machine locations, the security policies of their administrative domain, and the network being used to reach those machines.

Section 3.2 defined the notions of *Virtual Nodes*, and *Nodes.* Virtual Nodes are application abstractions, and nodes are only a run-time entity resulting from the deployment: a mapping of Virtual Nodes to processes and hosts. A first decisive feature allows to define application-level security on those application-level abstractions:

Definition 1. *Virtual Node Security*
Security policies can be defined at the level of Virtual Nodes. At execution, that security will be imposed on the Nodes resulting from the mapping of Virtual Nodes to JVMs, and Hosts.

As such, virtual nodes are the support for intrinsic application level security. If, at design time, it appears that a process always requires a specific level of security (e.g. authenticated and encrypted communications at all time), then that process should be attached to a virtual node on which those security features are imposed. It is the designer responsibility to structure his/her application or components into virtual node abstractions compatible with the required security. Whatever deployment occurs, those security features will be maintained. We expect this usage to be rather occasional, for instance in very sensitive applications where even an intranet deployment calls for encrypted communications.

The second decisive feature deals with a major Grid aspect: deployment-specific security. The issue is actually twofold:

1. allowing organizations (security domains) to specify general security policies,
2. allowing application security to be specifically adapted to a given deployment environment.

Domains are a standard way to structure (virtual) organizations involved in a Grid infrastructure; they are organized in a hierarchical manner. They are the logical concept allowing to express security policies in a hierarchical way.

Definition 2. *Declarative Domain Security*
Fine grain and declarative security policies can be defined at the level of Domains. *A Security Domain is a domain to which a certificate and a set of rules are associated.*

This principle allows to deal with the two issues mentioned above:
(1) the administrator of a domain can define specific policy rules that must be obeyed by the applications running within the domain. However, a general rule expressed inside a domain may prevent the deployment of a specific application. To solve this issue, a policy rule can allow a well-defined entity to weaken it. As

we are in a hierarchical organization, allowing an entity to weaken a rule means allowing all entities included to weaken the rule. The entity can be identified by its certificate;
(2) a Grid user can, at the time he runs an application, specify additional security based on the domains being deployed onto.

The Grid user can specify additional rules directly in his deployment descriptor for the domains he deploys onto. Note that those domains are actually dynamic as they can be obtained through external allocators, or even Web Services in an OGSA infrastructure [5]. Joker rules might be important in that case to cover all cases, and to provide a conservative security strategy for un-forecasted deployments.

Finally, as active objects are active and mobile entities, there is a need to specify security at the level of such entities.

Definition 3. *Active Object Security*
Security policies can be defined at the level of Active Object. Upon migration of an activity, the security policy attached to that object follows.

In open applications, e.g. several principals interacting in a collaborative Grid application, a JVM (a process) launched by a given principal can actually host an activity executing under another principal. The principle above allows to keep specific security privileges in such case. Moreover, it can also serve as a basis to offer, in a secure manner, hosting environments for mobile agents.

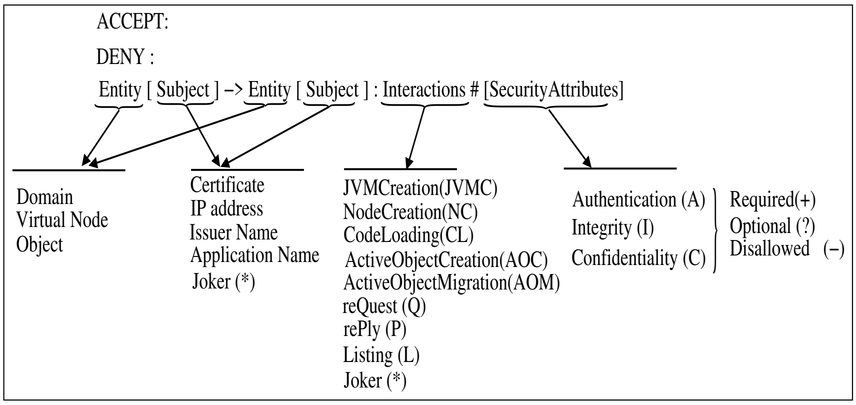

Fig. 2. Syntax and attributes for policy rules

4.2 Interactions, Security Attributes

Security policies are able to control all the *interactions* that can occur when deploying and executing a multi-principals Grid application. With this goal in mind, interactions span over the creation of processes (JVM in our case), to the monitoring of activities (ActiveObjects) within processes, including of course the communications. Here is a brief description of those interactions:

- JVMCreation (JVMC): creation of a new JVM process
- NodeCreation (NC): creation of a new Node within a JVM (as the result of Virtual Node mapping)
- CodeLoading (CL): loading of bytecode within a JVM
- ActiveObjectCreation (AOC): creation of a new activity (active object) within a Node
- ActiveObjectMigration (AOM): migration of an existing activity object to a Node
- Request (Q), Reply (P): communications, method calls and replies to method calls
- Listing (L): list the content of an entity; for Domain/Node provides the list of Node/Active Objects, for Active Object allows to monitor its activity.

For instance, a domain is able to specify that it accepts downloading of code from a given set of domains, provided the transfers are authenticated and guaranteed not to be tampered with. As a policy might leave open the integrity of communications, and also because not allowing confidentiality can be a domain (or even a country) policy, those 3 security attributes can be specified in 3 modes: Required (**+**), Optional (**?**), Disallowed (**-**)
For a given interaction, a tuple [+A,?I,-C] means that authentication is required, integrity is accepted but not required, and confidentiality is not allowed.

As a Grid operates in decentralized mode, without a central administrator controlling the correctness of all security policies, these policies must be *combined*, *checked*, and *negotiated* dynamically. The next two sections present that aspect.

4.3 Combining Policies

As the proposed infrastructure takes into account different actors of the Grid (domain administrator, Grid user), even for a single-principal single-domain application, there are potentially several security policies activated. This section deals with the combination of those policies to obtain the final security tuples of a single entity. An important principle being that a sub-domain cannot weaken a super-domain's rule.

During execution, each activity (Active Object) is always included in a *Node* (due to the Virtual Node mapping) and at least in one *Domain*, the one used to launch a JVM (D_0). Figure 3 hierarchically represents the security rules that can be activated at execution: from the top, hierarchical domains (D_n to D_0), the virtual node policy (VN), and the Active Object (AO) policy. Of course, such policies can be inconsistent, and there must be clear principles to combine the various sets of rules.

There are three main principles: (1) choosing the *most specific rules* within a given domain (as a single Grid actor is responsible for it), (2) an interaction is valid only if all levels accept it (absence of weakening of authorizations), (3) the security attributes retained are the most constrained based on a partial order (absence of weakening of security).

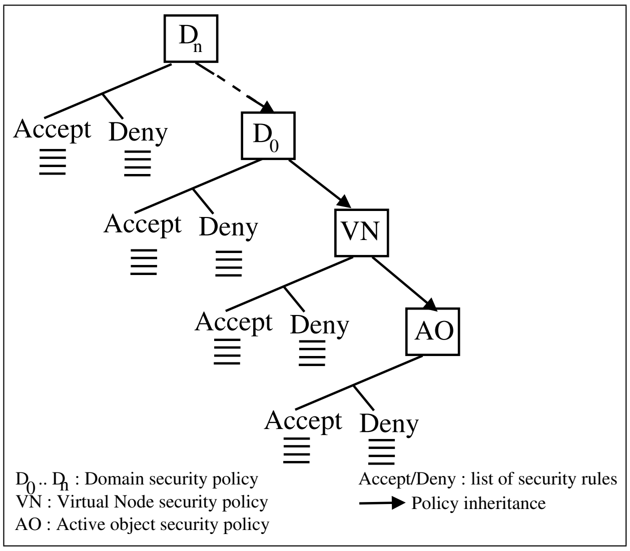

Fig. 3. Hierarchical security levels

The order on domains being inclusion (d1 $<$ d2 iff d1 included in d2), the *Most Specific Rules* are defined as the set of *minimal elements* in the partial order on rules defined as:
$D_1 -> D_2 \ \leq \ D_3 -> D_4 \ \Leftrightarrow D_1 \leq D_3$ and $D_2 \leq D_4$
It constitutes a *semi-lattice* with the top element being * –> * . As an example, if a security policy is written:

```
Domain[*] -> Domain[*] : Q,P : [+A,+I,+C]
Domain[CardPlus] -> Domain[CardPlus] : Q,P : [+A,?I,?C]
```

within the CardPlus domain, the second rule will be chosen (integrity and confidentiality will be optional). Of course, comparison of rules is only a partial order, and several incompatible most specific rules can exist within a single level (e.g. both ACCEPT and DENY most specific rules for the same interaction, or both +A and -A).

Between levels, an incompatibility can also occur, especially if a sub-level attempts to weaken the policy on a given interaction (e.g. a domain prohibits confidentiality [-C] while a sub-domain or the Virtual Node requires it [+C], a domain D_i prohibits loading of code while D_j (j≤i) authorizes it). In all incompatible cases, the interaction is not authorized and an error is reported.

4.4 Dynamic Policy Negotiation

During execution, entities interact by pair with each other. Each entity, for each interaction (JVM creation, communication, migration, ...), will want to apply a security policy based on the resolution presented in the previous section.

Before starting an interaction, a *negotiation* occurs between the two entities involved. Figure 4 shows the result of such negotiation. For example, if for a given interaction, entity A's policy is [+A,?I,?C], and B's policy is [+A,?I,-C], the negotiated policy will be [A,?I,-C]. If, for a result rule, one of the communication attributes is optional, the attribute is not activated.

Besides the interactions not being accepted by an entity, two other cases lead to an error: when an attribute is required by one, and disallowed by the other. In such cases, the interaction is not authorized and an error is reported. If a valid security policy is found between two entities, the interaction can occur. In the case that the agreed security policy includes confidentiality, the two entities negotiate a session key.

		ENTITY A		
		Required (+)	Optional (?)	Disallowed (−)
ENTITY B	Required (+)	+	+	Error
	Optional (?)	+	?	−
	Disallowed (−)	Error	−	−

Fig. 4. Result of security negotiations

4.5 Migration and Negotiation

In large scale Grid applications, migration of activities is an important issue. The migration of Active Objects must not weaken the security policy being applied. When an active object migrates to a new location, three cases may happen :

- the object migrates to a node belonging to the same virtual node and included inside the same domain. In this case, all already negotiated sessions remain valid.
- the object migrates to a known node (created during the deployment step) but which belongs to another virtual node. In this case, all already negotiated sessions can be invalid. This kind of migration imposes re-establishing the object policy, and upon a change, re-negotiating with interacting entities.
- The object migrates to an unknown node (not known at the deployment step). In this case, the object migrates with a copy of the application security policy. When a secured interaction will take place, the security system retrieves not only the object's application policy but also policies rules attached to the node on which the object is to compute the policy.

5 Conclusion

We have proposed a decentralized, declarative security mechanism for distributed systems that features interoperability with local policies, dynamically negotiated security policies, and multi-users systems. The use of a logical representation of the application (Virtual Nodes) allows to have a security policy adaptable to the deployment, a crucial feature for Grid applications. The security mechanism allows to express all security-related configurations within domain policy files, and application deployment descriptor, outside the source code of the application. About security overhead, a full process (policy and session key negotiations) is achieved in 50 ms against 10 ms in unsecured mode.

As a very short term perspective, the implementation of group communication security is in the process of being finalized; a single session key being used for all group members. We do not advocate to have solved all Grid security problems. Rather, this work is an attempt to contribute to the construction of flexible solutions that are very much needed for Grid deployment.

References

1. Caromel, D., Klauser, W., Vayssière, J.: Towards Seamless Computing and Metacomputing in Java. Concurrency Practice and Experience **10** (1998) 1043–1061
2. Baude, F., Caromel, D., Mestre, L., Huet, F., Vayssière, J.: Interactive and descriptor-based deployment of object-oriented grid applications. In: Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing, Edinburgh, Scotland, IEEE Computer Society (2002) 93–102
3. Grimshaw, A., et al., W.W.: The Legion Vision of a World-wide Virtual Computer. Communications of the ACM **40** (1997)
4. Foster, I., Kesselman, C.: The Globus project: a status report. Future Generation Computer Systems **15** (1999) 607–621
5. Foster, I.T., Kesselman, C., Tsudik, G., Tuecke, S.: A Security Architecture for Computational Grids. In: ACM Conference on Computer and Communications Security. (1998) 83–92
6. Wesley, A., ed.: .NET Framework Security. Addison Wesley Professional (2002)
7. Puliafito, A., Tomarchio, O.: Security Mechanisms for the MAP Agent System (2000) In 8th Euromicro Workshop on Parallel and Distributed Processing (PDP2000), 2000.
8. Karnik, N.M., Tripathi, A.R.: Security in the Ajanta Mobile Agent System. Software, Practice and Experience **31** (2001) 301–329
9. Karjoth, G., Lange, D., Oshima, M.: A Security Model for Aglets. IEEE Internet Computing **1** (1997) 68–77
10. Baumann, J., Hohl, F., Rothermel, K.: Mole - Concepts of a Mobile Agent System. Technical Report TR-1997-15, University of Stuttgart, Institute of Parallel and Distributed High-Performance Systems, Distributed Systems (1997)
11. Sun Microsystems: Remote methode invocation http://java.sun.com/products/jdk/rmi (2000)