Access and Usage Control in Grid Systems

16

Maurizio Colombo, Aliaksandr Lazouski,
Fabio Martinelli, and Paolo Mori

Contents

This chapter describes some approaches that have been proposed for access and usage control in grid systems. The first part of the chapter addresses the security challenges in grid systems and describes the standard security infrastructure provided by the Globus Toolkit, the most used middleware to establish grids. Since the standard Globus authorization system provides very basic mechanisms that do not completely fulfill the requirements of this environment, a short overview of well-known access control frameworks that have been integrated in Globus is also given: Community Authorization Service (CAS), PERMIS, Akenti, Shibboleth, Virtual Organization Membership Service (VOMS), Cardea, and PRIMA. Then, the chapter describes the usage control model UCON, a novel model for authorization, along with an implementation of UCON in grid systems. The last part of the chapter describes the authorization model for grid computational services designed by the GridTrust project. This authorization model is also based on UCON.

16.1 Background to the Grid

The grid is a distributed computing environment where each participant allows others to exploit his local resources [16.1, 2]. This environment is based on the concept of a virtual organization (VO). A VO is a set of individuals and/or institutions, e.g., companies, universities, research centers, and industries, sharing their resources. A grid user exploits this environment by composing the available grid resources in the most proper way to solve his problem. These resources are heterogeneous, and could be computational, storage, software repositories, and so on. The Open Grid Forum community formulated capabilities and requirements for grids by presenting

Peter Stavroulakis, Mark Stamp (Eds.), *Handbook of Information and Communication Security*

the Open Grid Service Architecture (OGSA) [16.3]. OGSA implies a service-oriented architecture where access to the underlying computational resource is done through a service interface.

This chapter refers to the Globus Toolkit [16.4–6] as the most used middleware to set up grids and that is compliant with the OGSA standard. However, alternative grid environments are available, such as Gridbus [16.7], Legion [16.8], WebOS [16.9], and Unicore [16.10].

Security is a very important problem in the grid because of the collaborative nature of this environment. In the grid, VO participants belong to distinct administrative domains that adopt different security mechanisms and apply distinct security policies. Moreover, VO participants are possibly unknown, and no trust relationships may exist a priori among them. VOs are dynamic, because new participants can join the VO, and some participants can leave it during the life of the VO. Another security-relevant feature of the grid environment is that accesses to grid services could be long-lived, i.e., they could last hours or even days. In this case, the access right that has been granted to a user at a given time on the basis of a set of conditions could authorize an access that lasts even when these conditions do not hold anymore. Hence, the grid environment features are different from those of a common distributed environment, and the security requirements of the grid environment require the adoption of a complex security model. Grid security requirements are detailed in [16.11–13], and they include authentication, delegation, authorization, privacy, message confidentiality and integrity, trust, and policy and access control enforcement.

16.2 Standard Globus Security Support

The Globus Toolkit is a collection of software components that can be used to set up grids. In particular, the grid security infrastructure is the set of these components concerning security issues. From version 3 on, Globus components exploit the Web service technology (Web service components). Hence, we refer to the security components as security services. In particular these services are:

Authentication The standard authentication service of Globus exploits a public key infrastructure, where X.509 end entity certificates (EECs) are used to identify entities in the grid, such as users and services. These certificates provide a unique user identifier, called the distinguished name, and a public key to each entity. The user's private key, instead, is stored in the user's machine. The X.509 EECs adopted by Globus are consistent with the relevant standards. The X.509 EECs can be issued by standard certification authorities implemented with third-party software.

Authorization The standard authorization system of Globus is based on a simple access control list, the *gridmap* file, which pairs a local account with the distinguished name of each user that is allowed to access the resource. In this case, the security policy that is enforced is only the one defined by the privileges paired with the local account by the operating system running on the underlying resource. The Globus Toolkit also provides some other simple authorization mechanisms, which can be configured through the security descriptors of the Web service deployment descriptor files. These alternative mechanisms are *self, identity, host, userName, and SAMLCallout*. As an example, the *host* authorization restricts the access only to users that submit their requests from a given host. Moreover, Globus allows the integration of third-party authorization services, exploiting the *SAMLCallout* authorization mechanism, which is based on the SAML Authorization Decision protocol [16.14]. In this case, the *SAMLCallout* authorization mechanism and the security descriptors are configured to refer to the external authorization service. The authorization service can run on the same machine of the Globus container, or even on a remote machine. This mechanism has been exploited to integrate in Globus the well-known authorization systems described in the rest of this chapter.

Delegation Reduces the number of times the user must enter his passphrase for the execution of his request. If a grid computation involves several grid resources (each requiring mutual authentication) requesting services on behalf of a user, the need to reenter the user's passphrase can be avoided by creating a proxy certificate. A proxy consists of a new certificate and a private key. The new certificate contains the owner's identity, modified slightly to indicate that it is a proxy, and is signed by the owner, rather than a certification authority. The certificate also includes a time after which the

proxy should no longer be accepted. The interface to this service is based on WS-Trust [16.15] specifications.

16.3 Access Control for the Grid

The standard authorization systems provided by the Globus Toolkit are very coarse grained and static, and they do not address the real requirement of the grid. As a matter of fact, the gridmap authorization system grants or denies access to a grid service simply by taking into account the distinguished name of the grid user that requested the access. Instead, it would be useful to consider other factors to define an access right. Moreover, once the access to the resource has been granted, no more controls are executed. As an example, in the case of computational resources, once the right to execute an application has been granted to the grid user, the application is started and no further controls are executed on the actions that the application performs on the grid resource. Furthermore, especially in the case of long-lived accesses, it could be useful to periodically check whether the access right still holds. Even if the access right held when the access was requested, during the access time some factors that influence the access right could have been changed. Then, the ongoing access could be interrupted. Since the Globus Toolkit allows the adoption of an external authorization system, many solutions have been proposed by the grid community to improve the authorization system, and this section describes the main ones.

16.3.1 Community Authorization Service

The Community Authorization Service (CAS) [16.16, 17] is a VO-wide authorization service that has been developed by the Globus team. The main aim of the CAS is to simplify the management of user authorization in the grid, i.e., to relieve the grid resource providers from the burdens of updating their environments to enforce the VO authorization policies. The CAS is a grid service that manages a database of VO policies, i.e., the policies that determine what each grid user is allowed to do as a VO member with the grid resources. In particular, the VO policies stored by the CAS consist of:

- The VO's access policies about the resources: these policies determine which rights are granted to which users.
- The CAS's own access control policies, which determine who can delegate rights or maintain groups within the VO.
- The list of the VO members.

The local policies of the grid resource providers, instead, are stored locally on the grid nodes. Hence, the CAS can be considered as a trusted intermediary between the VO users and the grid resources. To transmit the VO policies to the grid resources where they should be enforced, the CAS issues to grid users credentials embedding CAS policy assertions that specify the users' rights to the grid resources. CAS assertions can be expressed in an arbitrary policy language. The grid user contacts the CAS to obtain a proper assertion to request a service on a given resource, and the credentials returned by the CAS server will be presented by the grid user to the service he wants to exploit. This requires that resource providers participating in a VO with the CAS deploy a CAS-enabled service, i.e., services that are able to understand and enforce the policies in the CAS assertions.

The CAS system works as follows:

1. The user authenticates himself to the CAS server, using his own proxy credential. The CAS server establishes the user's identity and the rights in this VO using its local database.
2. The CAS server issues a signed policy assertion containing the user's identity and rights in the VO. On the user side, the CAS client generates a new proxy certificate for the user that embeds the CAS policy assertion, as a noncritical X.509 extension. This proxy is called a *restricted proxy* because it grants only a restricted set of rights to the user, i.e., the rights that are described in the CAS assertion it embeds.
3. The user exploits the proxy certificate with the embedded CAS assertion to authenticate on the grid resource. The CAS-enabled service authenticates the user using the normal authentication system. Then it parses the CAS policy assertion, and takes several steps to enforce both VO and local policies:
    - Verifies the validity of the CAS credential (signature, time period, etc.)
    - Enforces the site's policies regarding the VO, using the VO identity instead of the user one

- Enforces the VO's policies regarding the user, as expressed in the signed policy assertion in the CAS credential
- Optionally, enforces any additional policies concerning the user (e.g., the user could be in the blacklist of the site).

Hence, the set of rights that are granted to the user is the intersection of the rights granted by the resource provider to the VO and the rights granted by the VO to the user, taking into account also specific restrictions applied by the resource provider to the user.

Once the access has been authorized, the grid user is then mapped on the local account paired with the CAS. Hence, in the grid resource gridmap file there is only one entry that pairs the CAS distinguished name with the local account used to execute the jobs on behalf of the grid users. This simplifies the work of the local grid node administrator because he has to add one local account only for each CAS, instead of one local account for each grid user.

16.3.2 PERMIS

PERMIS is a policy-based authorization system proposed by Chadwick et al. [16.18–20] which implements the role-based access control (RBAC) paradigm. RBAC is an alternative approach to access control lists. Instead of assigning certain permissions to a specific user directly, roles are created for various responsibilities and access permissions are assigned to specific roles possessed by the user. The assignment of permissions is fine-grained in comparison with access control lists, and users get the permissions to perform particular operations through their assigned role. PERMIS is based on a distributed architecture that includes the following entities:

- Sources of authority, which are responsible for composing the rules for decision making and credential validation services (CVSs)
- Attribute authorities, which issue the attributes which determine the role of users
- Users, who are the principals who perform operations on the resources
- Applications, which provide the users with the interfaces to access the protected resource.

Obviously, users and resources can belong to distinct domains. A policy file written in XML contains the full definition of roles in regard to protected resources and permissions related to a specific role. The PERMIS toolkit provides a friendly graphical user interface for managing its policies. The policies may be digitally signed by their authors and stored in attribute certificates, to prevent them from being tampered with. PERMIS is based on the privilege management infrastructure that uses X.509 attribute certificates to store the user's roles. Every attribute certificate is signed by the trusted attribute authority that issued it, whereas the root of trust for the privilege management infrastructure target resource is called the source of authority. All the attribute certificates can be stored in one or more Lightweight Directory Access Protocol (LDAP) directories, thus making them widely available.

PERMIS also provides the delegation issuing service, which allows users to delegate (a subset of) their privileges to other users in their domain by giving them a role in this domain, according to the site's delegation policy. Since PERMIS is tightly integrated with the Globus Toolkit, input information for access decision consists of the user's distinguished name and resource and action request. For authorization decision making, PERMIS provides a modular policy decision point (PDP) and a CVS (or policy information point (PIP) according to the Globus model). PERMIS implements the hierarchical RBAC model, which means that user roles (attributes) with superior roles inherit the permissions of the subordinate ones. The PERMIS policy comprises two parts, a role assignment policy that states who is trusted to assign certain attributes to users, and a target access policy that defines which attributes are required to access which resources and under what conditions. The CVS evaluates all credentials received against the role assignment policy, rejects untrusted ones, and forwards all validated attributes to the policy enforcement point (PEP). The PEP in turn passes these to the PERMIS PDP, along with the user's access request, and any environmental parameters. The PDP obtains an access control decision based on the target access policy, and sends its granted or denied response back to the PEP. Hence, to gain access to a protected target resource, a user has to present his credentials and the PERMIS decision engine (CVS and PDP) validates them according to the policy to make an authorization decision. The current version of the PERMIS authorization service supports SAML authorization callout and provides Java application programming

interfaces for accessing the PIP and the PDP. Technical specifications and implementation issues can be found in [16.21].

16.3.3 Akenti

The paramount idea of Akenti, proposed by Thompson et al. [16.22–24], is to provide a usable authorization system for an environment consisting of highly distributed resources shared among several stakeholders. By exploiting fine-grained authorization for job execution and management in the grid, Akenti provides a restricted access to resources using an access control policy which does not require a central administrative authority to be expressed and to be enforced.

In this model, control is not centralized. There are several stakeholders (parties with authority to grant access to the resource), each of which brings its own set of concerns in resource management. The access control policy for a resource is represented as a set of (possibly) distributed X.509 certificates digitally signed by different stakeholders from unrelated domains. These certificates are independently created by authorized stakeholders and can be stored remotely or on a known secure host (probably the resource gateway machine). They are usually self-signed and express what attributes a user must have to get specific rights to a resource, who is trusted to make such use-condition statements, and who can certify the user's attributes. The Akenti policy is written in XML and there exist three possible types of signed certificates: policy certificates, use-condition certificates, and attribute certificates. Use-condition certificates contain the constraints that control access to a resource and specify who can confirm the required user's attributes and thus who may sign attribute certificates. Attribute certificates assign attributes to users that are needed to satisfy the usage constraints. Complete policies on the specification and language used to express them can be found on the Akenti Web site [16.22]. When an authorization decision is required, the resource gatekeeper asks a trusted Akenti server what access the user has to the resource. Then the Akenti policy engine gathers all the relevant certificates for the user and for the resource from the local file system, LDAP servers, and Web servers, verifies and validates them, and responds with the user's rights in respect to the requested resource. Akenti assumes a secure SSL/TLS connection between peers and the resource through the resource gateway which provides authentication using an X.509 identity certificate. The authorization algorithm in the grid using Akenti is very similar to PERMIS and has the following stages:

1. A resource provider authenticates a user and validates his identity as well as possibly some additional attributes.
2. The resource provider receives and parses the user's request.
3. The resource provider forwards the user's identity, attributes, and requests to a trusted Akenti server to authorize the user (i.e., whether the request should be granted or denied).
4. Finally, the Akenti server returns a decision to the resource provider that enforces it.

16.3.4 Shibboleth

Shibboleth [16.25, 26], is an Internet2/MACE project implementing cross-domain single sign-on and attribute-based authorization for systems that require interinstitutional sharing of Web resources with preservation of end-user privacy. The main idea of Shibboleth is that instead of users having to log-in and be authorized at any restricted site, users authenticate only once at their local site, which then forwards the user's attributes to the restricted sites without revealing information about the user's identity.

The main components of the Shibboleth architecture are the Shibbolet handle service (SHS), which authenticates users in conjunction with a local authentication service and issues a handle token; the attribute service, which receives the handle token that a user exploited to request the access to the resource and returns the attributes of the user; the target resource, which includes Shibboleth-specific code to determine the user's home organization and, consequently, which Shibboleth attribute authority should be contacted for this user. A typical usage of Shibboleth is as follows:

1. The user authenticates to the SHS.
2. The SHS requests a local organizational authentication service by forwarding user-authentication information to confirm his identity.
3. The SHS generates a random handle and maps it to the user's identity. This temporal handle is registered at the Shibboleth attribute service.

4. The handle is returned to the user and the user is notified that he was successfully authenticated.
5. Then the user sends a request for a target resource with the previous handle.
6. The resource provider analyzes the handle to decide which Shibboleth attribute service may provide the required user attributes to make an authorization decision, and contacts it by forwarding the handle that identifies the user.
7. After validation checks on the handle have been done and the user's identity is known, the attribute service exploits the attribute release policy to determine whether the user's attributes can be sent to the resource provider.
8. The Shibboleth attribute authority casts the attributes in the form of SAML attribute assertions and returns these assertions to the target resource.
9. After receiving the attributes, the target resource provider performs an authorization decision exploiting the user's request, the user's attributes, and the resource access control policy.

Detailed specification of all Shibboleth's functional components, such as identity provider and service provider and the security protocol used based on SAML can be found in [16.25]. GridShib [16.27, 28] is a currently going research project that investigates and provides mechanisms for integrating Shibboleth into the Globus Toolkit. The focus of the GridShib project is to leverage the attribute management infrastructure of Shibboleth, by transporting Shibboleth attributes as SAML attribute assertions to any Globus Toolkit PDP.

16.3.5 Virtual Organization Membership Service

The Virtual Organization Membership Service (VOMS) [16.29, 30], is an authorization service for the grid that has been developed by the EU projects DataGrid and DataTAG. The VOMS has a hierarchical structure with groups and subgroups; a user in a VO is characterized by a set of attributes, 3-tuples of the form group, role, and capability. The combined values of all these 3-tuples form a unique attribute, the fully qualified attribute name, that is paired with the grid user. The VOMS is implemented as a push system, where the grid user first retrieves from and then sends to the grid service the credentials embedding the attributes he wants to exploit for the authorization process. The VOMS system consists of the following components:

- User server: This is a front end to a database where the information about the VO users is kept. It receives requests from the client and returns information about the user.
- User client: This contacts the server presenting the certificate of a user and obtains the list of groups, roles, and capabilities of that user.
- Administration client: This is used by the VO administrators to add users, create new groups, change roles, and so on.
- Administration server: This accepts the requests from the client and updates the database.

To retrieve the authorization information the VO grants him, the grid user exploits the VOMS user client that contacts the VOMS user server. The VOMS server returns a data structure, called VOMS *pseudo-certificate* or *attribute certificate*, embedding the user's roles, groups, and capabilities. The pseudo-certificate is signed by the VOMS user server and it has a limited time validity. If necessary, more than one VOMS user server can be contacted to retrieve a proper set of credentials for the grid user. To access a grid service, the user creates a proxy certificate containing the pseudo-certificates that he has previously collected from the VOMS servers.

To perform the authorization process, the grid node extracts the grid user's information from the user's proxy certificate and combines it with the local policy. Since when the VOMS is used the grid resource is accessed by exploiting the grid user's name, i.e., the distinguished name in the user's certificate, the user name should be added in the gridmap file of each grid resource and paired with a local account. To this aim, the grid resource provider periodically queries VOMS databases to generate a list of VO users and to update the gridmap file mapping them to local accounts.

16.3.6 Cardea

Cardea is a distributed authorization system developed as part of the NASA Information Power Grid [16.31]. One of the key features of Cardea is that it evaluates authorization requests according to a set of relevant characteristics of the grid resource

and of the grid user that requested the access, instead of considering the user's and resource's identities. Hence, the access control policies are defined in terms of relevant characteristics rather than in terms of identities. In this way, Cardea allows users to access grid resources if they do not have existing local accounts. Moreover, Cardea is a dynamic system, because the information required to perform the authorization process is collected during the process itself. Any characteristic of the grid user or of the grid resource, as well of the ones of the current environment, can be taken into account in the authorization process. These characteristics are represented through SAML assertions, which are exchanged through the various components of the architecture.

From the architectural point of view, the Cardea system consists of the following components: a SAML PDP, one or more attribute authorities, one or more PEPs, one or more references to an information service, an XACML context handler, one or more XACML policy administration points, and an XACML PDP. The main component of the system is the SAML PDP, which accepts authorization queries, performs the decision process, and returns the authorization decision. To exploit Cardea in the existing grid toolkits, proper connectors, e.g., an authorization handler in the case of the Globus Toolkit, generate the authorization query in the format accepted by the SAML PDP. The SAML PDP, depending on the request, determines the XACML PDP that will evaluate the request. The values of the attributes involved in the authorization request are retrieved by querying the appropriate attribute authorities. Finally, the PEP is the component that actually enforces the authorization decision, and could even reside in a remote grid node. Hence, the final authorization decision is transmitted by the SAML PDP to the appropriate PEP to be enforced.

The components of the Cardea system can be located on the same machine, and in this case their interactions are implemented through local communication paradigms, or they can be distributed across several machines, and in this case they act as Web services.

16.3.7 PRIMA

PRIMA (privilege management and authorization) [16.32] is focused on management and enforcement of fine-grained privileges. PRIMA enables the users of the system to manage access to their privileges directly without the need for administrative intervention. The model uses on-demand account leasing and implements expressive enforcement mechanisms built on existing low-overheard security primitives of the operating systems. PRIMA addresses the security requirements through a unique combination of three innovative approaches [16.32]:

- *Privileges*: unforgeable, self-contained, fine-grained, time-limited representations of access rights externalized from the underlying operating system. Privilege management is pushed down to the individuals in PRIMA.
- *Dynamic policies*: a request-specific access control policy formed from the combination of user-provided privileges with a resource's access control policy.
- *Dynamic execution environments*: a specifically provisioned native execution environment limiting the use of a resource to the rights conveyed by user-supplied privileges.

The PRIMA authorization system can be divided into two parts [16.32]. The first part is the privilege management layer, which facilitates the delegation and selective use of privileges. The second part is the authorization and enforcement layer. The authorization and enforcement layers have two primary components. The first component is the PRIMA authorization module. The authorization module plays the role of the PEP. The second component is the PRIMA PDP, which, on the basis of policies made available to it, will respond to authorization requests from the PRIMA authorization module. These policies are created using the platform-independent language XACML. Two other components in the authorization and enforcement layer are the gatekeeper and the privilege revocator. The gatekeeper is a standard Globus Toolkit component for the management of access to Globus resources. It was augmented with a modular interface to communicate with the authorization components. The JobManager, also a standard component of the Globus Toolkit, has not been modified from the original Globus distribution. It is instantiated by the Globus gatekeeper after successful authorization. It starts and monitors the execution of a remote user's job. The privilege revocator monitors the lifetime of privileges that were used to configure execution environments. On privilege expiration, the privilege revocator removes access rights and deallocates the

execution environment automatically. No manual intervention from system administrators is required.

A typical access request in the PRIMA authorization system is as follows [16.32]. In step 1, the delegation of privileges and the provision of policies happens prior to a request is issued. In step 2, subjects select the subset of privilege attributes they hold for a specific (set of) grid request(s) and group these privileges with their short-lived proxy credential using a proxy creation tool. The resulting proxy credential is then used with standard Globus job submission tools to issue grid service requests (step 3). Upon receiving a subject's service request, the gatekeeper calls the PRIMA authorization module (step 4). The PRIMA authorization module extracts and verifies the privilege attributes presented to the gatekeeper by the subject. It then assembles all valid privileges into a dynamic policy. Dynamic policy denotes the combination of the user's privileges with the resource's security policy prior to the assessment of the user's request. To validate that the privileges were issued by an authoritative source, the authorization module queries the privilege management policy via the PRIMA PDP. The multiple interactions between the authorization module and the PDP are depicted in a simplified form as a single message exchange (step 5 and 6). Once the privilege's issuer authority has been established, the PRIMA authorization module formulates an XACML authorization request based on the user's service request and submits the request to the PDP. The PDP generates an authorization decision based on the static access control policy of this resource. The response will state a high-level permit or deny. In the case of a permit response, the authorization module interacts with native security mechanisms to allocate an execution environment (e.g., a UNIX user account with minimal access rights) and provides this environment with access rights based on the dynamic policy rules (step 7). Once the execution environment has been configured, the PRIMA authorization module returns the permit response together with a reference to the allocated execution environment (the user identifier) to the gatekeeper and exits (step 8). The following steps are unchanged from the standard Globus mechanisms. The Globus gatekeeper spawns a JobManager process in the provided execution environment (step 9). The JobManager instantiates and manages the requested service (step 10). In the case of a deny response, the authorization module returns an error code to the gatekeeper together with an informative string indicating the reason for the denied authorization. The gatekeeper in turn will record this error in its log, return an error code to the grid user (subject), and end the interaction. The privilege revocator watches over the validity period of dynamically allocated user accounts and all fine-grained access rights, revoking them when the associated privileges expire (step 11).

In summary, PRIMA mechanisms enable the use of fine-grained access rights, reduce administrative costs to resource providers, enable ad hoc and dynamic collaboration scenarios, and provide an improved security service to long-lived grid communities.

16.4 Usage Control Model

The UCON model is a new access control paradigm, proposed by Sandhu and Park [16.33, 34], that encompasses and extends several existing models (e.g., mandatory access control (MAC), discretionary access control (DAC), Bell–LaPadula, RBAC) [16.35, 36]. Its main novelty, in addition to the unifying view, is based on continuity of usage monitoring and mutability of attributes of subjects and objects. Whereas standard access control models are based on authorizations only, UCON extends them with another two factors that are evaluated to decide whether to grant the requested right: obligations and conditions. Moreover, this model introduces mutable attributes paired with subjects and objects and, consequently, introduces the continuity of policy enforcement. In the following we give a short description of the UCON core components: subjects, objects, attributes, authorizations, obligations, conditions, and rights.

16.4.1 Subjects and Objects

The subject is the entity that exercises rights, i.e., that executes access operations, on objects. An object, instead, is an entity that is accessed by subjects through access operations. As an example, a subject could be a user of an operating system, an object could be a file of this operating system, and the subject could access this file by performing a write or read operation. Both subjects and objects are paired with attributes.

16.4.2 Attributes

Attributes are paired with both subjects and objects and define the subject and the object instances. Attributes can be mutable and immutable. Immutable attributes typically describe features of subjects or objects that are rarely updated, and their update requires an administrative action. Mutable attributes, instead, are updated as consequence of the actions performed by the subject on the objects. The attributes are very important components of this model, because their values are exploited in the authorization process. An important subject attribute is identity. Identity is an immutable attribute, because it does not change as a consequence of the accesses that this subject performs. A mutable attribute paired with a subject could be the reputation of the subject, because it could change as a consequence of the accesses performed by the subject to objects. Attributes are also paired with objects. Examples of immutable attributes of an object depend on the resource itself. For a computational resource, possible attributes are the identifier of the resource and its physical features, such as the available memory space, the CPU speed, and the available disk space.

In the UCON model, mutable attributes can be updated before (*preUpdate*), during (*onUpdate*), or after (*postUpdate*) the action is performed. The onGoing update of attributes is meaningful only for long-lived actions, when onGoing authorizations or obligations are adopted. When defining the security policy for a resource, one has to chose the most proper attribute updating mode. As an example assume that the reading of a file requires a payment. When the application tries to open the file, the security policy could state that at first the subject balance attribute is checked, then the action is executed, and then the subject balance attribute is updated.

16.4.3 Rights

Rights are privileges that subjects can exercise on objects. Traditional access control systems view rights as static entities, for instance, represented by the access matrix. Instead, UCON determines the existence of a right dynamically, when the subject tries to access the object. Hence, if the same subject accesses the same object two times, the UCON model could grant him different access rights. In UCON, rights are the result of the usage decision process that takes into account all the other UCON components.

16.4.4 Authorizations

Authorizations are functional predicates that evaluate subject and object attributes and the requested right according to a set of authorization rules, to take the usage decision. The authorization process exploits both the attributes of the subject and the attributes of the object. As an example, an attribute of a file could be the price to open it, and an attribute of a user could be the prepaid credit. In this case, the authorization process checks whether the credit of the user is enough to perform the open action on the file. The evaluation of the authorization predicate can be performed before executing the action (*preAuthorization*), or while the application is performing the action (*onAuthorization*). With reference to the previous example, the preAuthorization is applied to check the credit of the subject before the file opening. onAuthorization can be exploited in the case of long-lived actions. As an example, the right to execute the application could be paired with the onAuthorization predicate that is satisfied only if the reputation attribute of the subject is above a given threshold. In this case, if during the execution of the application the value of the reputation attribute goes below the threshold, the subjects's right to continue the execution of the application is revoked.

16.4.5 Conditions

Conditions are environmental or system-oriented decision factors, i.e., dynamic factors that do not depend upon subjects or objects. Conditions are evaluated at runtime, when the subject attempts to perform the access. The evaluation of a condition can be executed before (*preCondition*) or during (*onCondition*) the action. For instance, if the access to an object can be executed during daytime only, a preCondition that is satisfied only if the current time is between 8:00 am and 8:00 pm can be defined. Ongoing conditions can be used in the case of long-lived actions. As an example, if the previous access is a long-lived one, an onCondition that is satisfied only if the current time is between 8:00 am and 8:00 pm could

be paired with this access too. In this case, if the access started at 9:00 am and is still active at 8:00 pm, the onCondition revokes the subject's access right.

16.4.6 Obligations

Obligations are UCON decision factors that are used to verify whether some mandatory requirements have been satisfied before performing an action (*preObligation*), or whether these requirements are satisfied while the access is in progress (*onObligation*). preObligation can be viewed as a kind of history function to check whether certain activities have been fulfilled or not before granting a right. As an example, a policy could require that a user has to register or to accept a license agreement before accessing a service.

16.4.7 Continuous Usage Control

The mutability of subject and object attributes introduces the necessity to execute the usage decision process continuously in time. This is particularly important in the case of long-lived accesses, i.e., accesses that last hours or even days. As a matter of fact, during the access, the conditions and the attribute values that granted the access right to the subject before the access could have been changed in a way such that the access right does not hold anymore. In this case, the access is revoked.

16.5 Sandhu's Approach for Collaborative Computing Systems

The authors of UCON recognized the usefulness of their model also for collaborative computing systems, and hence also for grid systems, and reported initial work in this area [16.37, 38]. The UCON-based authorization framework was designed to protect a shared trusted store for source code management. The model authorizes a group of software developers to share and collaboratively develop application code at different locations.

The architecture of the authorization system proposed in [16.37, 38] for collaborative computing systems consists of user platforms, resource providers, and an attribute repository. The attribute repository is a centralized service that stores mutable subject and system attributes in a VO. Consistency among multiple copies of the same attribute is a problem to be tackled when adopting a distributed version of the repository for subject-mutable attributes. Object attributes are stored in a usage monitor on each resource provider's side. A usage session is initialized by a user (subject). The user submits an access request from its platform to a resource provider (step 1). Then, persistent subject attributes are pushed by the requesting subject to the PDP (step 2). After receiving the request, the PDP contacts the attribute repository and retrieves the mutable attributes of the requesting subject (steps 3 and 4) and the object attributes from the usage monitor (step 5). An interesting evaluation of the potential models (either push or pull) for the credential retrieval by the PDP is examined. The result is that for collaborative systems a hybrid mode must be considered, push for immutable and pull for mutable. This reflects the fact that the user may have interest in showing a good value for mutable attributes and the PDP should ensure it always has updated information. This update scenario is time-sensitive and can not be accepted for fine-grained real-time usage control. The attribute repository is trusted by all entities in a collaborative computing system.

The access control decision according to VO policies is issued by the PDP after all related information (subject, object, and system attributes) has been collected and all relevant policies have been evaluated. The decision is forwarded to the PEP and enforced in the execution environment of the resource provider (step 6). As the side effect of making a usage decision, attribute updates are performed by the PDP according to the corresponding security policy. New subject attribute values are sent back to the attribute repository (step 7), and the updated object attributes are sent to the usage monitor (step 8). The PDP always checks the attribute repository and the usage monitor for the latest attribute values when a new access request is generated. Any update of subject or object attributes and any change of system conditions triggers the reevaluation of the policy by the PDP according to the ongoing usage session and may result in revocation of the ongoing usage or updating of attributes if necessary. The approach supports decision continuity and attribute mutability of UCON within concurrent usage sessions.

A UCON protection system was implemented as a server-side reference monitor (both the PDP and the PEP were placed on the resource provider side). A reference monitor enforced the usage control policy written in XACML policy language. The XACML security policy is not expressive enough to define the original UCON model completely. It was noted in [16.37, 38] that XACML is only capable of specifying attribute requirements before usage and possible updates after the usage, but not during the usage. Also, the concept of obligation in XACML does not mean the same as that in the original UCON model.

16.6 GridTrust Approach for Computational Services

GridTrust is an EU-funded project aimed at developing the technology to manage trust and security in the next-generation grid. GridTrust identified the UCON model as a perfect candidate to enhance the security of grid systems owing to their peculiarities, and adapted the original UCON model to develop a full model for usage control of grid computational services. As a matter of fact, grid computational services execute unknown applications on behalf of potentially unknown grid users on the local computational resources. In this case, the subject that performs the accesses to the object is the application that is executed on the computational resource on behalf of the grid user. An initial attempt of providing continuous usage control for grid computational services was developed in [16.39], where the necessity of performing continuous and fine-grained authorization with behavioral policies was identified. Some results also appear in [16.40] that represent an attempt to exploit credential management to enforce behavioral policies. This approach is based on a policy specification language derived from a process description language, which is suitable to express policies that implement the original UCON model. Architecture for enforcing the usage control policies is also defined.

16.6.1 Policy Specification

The GridTrust [16.41] approach is based on a policy language that allows one to express usage control policies by describing the order in which the security-relevant actions can be performed, which authorizations, conditions, and obligations must be satisfied to allow a given action, which authorizations, conditions, and obligations must hold during the execution of actions, and which updates must be performed. The security language adopted is operational and it is based on process algebra (POLPA, policy language based on process algebra), which is suitable for representing a sequence of actions, potentially involving different entities. Hence, a policy results from the composition of security-relevant actions, predicates, and variable assignments through some composition operators; for further details on POLPA, see [16.42, 43].

The encoding of the UCON model follows an approach similar to the one described in [16.44], and it models the steps of the usage control process with the following set of actions:

- *tryaccess(s,o,r)*: performed by subject s when performing a new access request (s, o, r)
- *permitaccess(s,o,r)*: performed by the system when granting the access request (s, o, r)
- *denyaccess(s,o,r)*: performed by the system when rejecting the access request (s, o, r)
- *revokeaccess(s,o,r)*: performed by the system when revoking an ongoing access (s, o, r)
- *endaccess(s,o,r)*: performed by a subject s when ending an access (s, o, r)
- *update(a,v)*: performed by the system to update a subject or an object attribute a with the new value v.

These actions refer to an access request (s, o, r), where s is a subject that wants to access an object o through an operation op that requires the right r. In particular, the operation op is a system call that the application executed on behalf of a remote grid user performs on the local resources provided by the grid computational service. By combining these actions, one can encode all the possible UCON models, also taking into account the mutability of attributes (immutable, preUpdate, onUpdate, postUpdate). An example of security policy that regulates the usage of server sockets in grid computational services is shown in Table 16.1.

The first four lines of the policy allow the application to execute a socket system call, i.e., the operation to open a new communication socket. *tryaccess(app_id,socket,socket(x_1, x_2, x_3,sd))* is the action that is issued when a socket system call has been invoked by the application, where *app_id* is the identifier of the application, *socket* is the object that is ac-

Table 16.1 Example of security policy for computational services

tryaccess(app_id, socket, socket(x_1, x_2, x_3, sd)).	Line 1
[(x_1 = AF_INET),(x_2 = STREAM), (x_3 = TCP)].	Line 2
permitaccess(app_id, socket, socket(x_1, x_2, x_3, sd)).	Line 3
endaccess(app_id, socket, socket(x_1, x_2, x_3, sd)).	Line 4
tryaccess(app_id, socket, listen(x_5, x_6, x_7, x_8)).	Line 5
[(x_5 = sd)].	Line 6
permitaccess(app_id, socket, listen(x_5, x_6, x_7, x_8)).	Line 7
endaccess(app_id, socket, listen(x_5, x_6, x_7, x_8)).	Line 8
tryaccess(app_id, socket, accept(x_9, x_{10}, x_{11}, x_{12})).	Line 9
[(x_9 = sd), (app_id.reputation ≥ T)].	Line 10
permitaccess(app_id, socket, accept(x_9, x_{10}, x_{11}, x_{12})).	Line 11
(([(app_id.reputation < T)].	Line 12
revokeaccess(app_id, socket, accept(x_9, x_{10}, x_{11}, x_{12})))	Line 13
or	Line 14
endaccess(app_id, socket, accept(x_9, x_{10}, x_{11}, x_{12}))	Line 15
);	Line 16
...	Line 17

cessed, and *socket(x_1, x_2, x_3,sd)* represents the socket system call with its parameters and results. The predicates in the second line represent a preAuthorization, because they are evaluated before granting the right to create the socket (line 3). These predicates involve the parameters of the socket system call and specify that only TCP sockets can be opened. Hence, if these predicates are satisfied, the *permitaccess()* action is issued in line 3 and the socket system call is executed. The fourth line of the policy concerns the *endaccess()* action, which is issued by the PEP when the socket system call is terminated, before continuing the execution of the application.

The ninth line concerns the execution of an accept system call, which is issued in case of server sockets and which waits for an incoming connection. Line 10 specifies preAuthorization predicates, which check that the socket descriptor *sd* is the one that has been returned by the previous socket system call, and that the reputation attribute of the subject that executes the application is equal to or greater than a given threshold T. This check is executed before permitting the execution of the system call, i.e., before the *permitaccess()* action in line 11. The accept system call ends when a remote client requests a connection with the local socket (line 15). Since we cannot predict when a remote host will connect with the local socket, we consider this system call a long-lived action. Hence, the policy includes an onAuthorization predicate paired with this system call (line 12). This predicate is evaluated during the execution of the accept system call, and the execution is interrupted if the value of the reputation attribute of the subject is lower than T. The interruption of the execution of the accept system call is implemented by the *revokeaccess()* action (line 13). The reputation of a subject is a mutable attribute, because it could be updated as a consequence of the accesses to the grid resources performed by the subject. Hence, the value of the reputation could change during the execution of the accept system call.

16.6.2 Architecture

The architecture to enforce the security policies previously defined exploits the *reference monitor* model, where the main components are the PEP and the PDP, as shown in Fig. 16.1 [16.43].

The PEP is integrated into the Globus architecture, and implements the *tryaccess(s,o,r)* and *endaccess(s,o,r)* operations. In particular, to protect grid computational services, the PEP is integrated with the Globus resource and allocation management service, which is the component of the Globus architecture that provides the environment to execute the applications on behalf of other grid users. Hence, the PEP monitors the accesses to local resources performed by these applications. The *tryaccess(s,o,r)* command is transmitted by the PEP to the PDP when the application tries to perform an access, whereas the *endaccess(s,o,r)* operation is sent when an access that has been previously granted is terminated.

The PDP is the component that performs the usage decision process. A new PDP instance is created for a specific job request and is in charge of monitoring the whole execution. At initialization time, the PDP gets the security policy from a repository, and it builds its internal data structures for the policy representation. The policy could be defined by the owner of the resource (local policy) by the VO (global policy) or by both. Here we suppose that the PDP reads the policy resulting from the merging of local and global policies, i.e., that the merging of the two policies and the resolution of possible conflicts has been executed in a previous step. After the initialization step, the PDP waits for messages from the

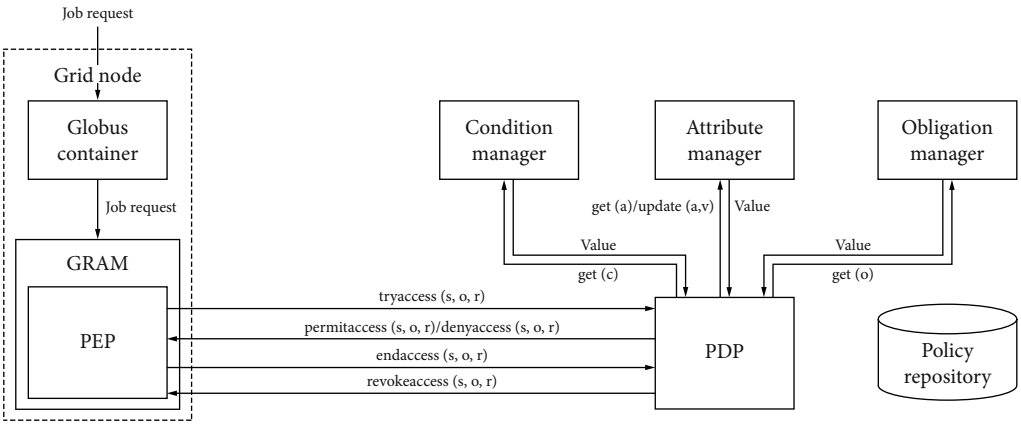

Fig. 16.1 Architecture of the policy enforcement system. *GRAM* Globus resource and allocation management, *PEP* policy enforcement point, *PDP* policy decision point

PEP, which invokes the PDP every time that the subject attempts to access a resource and every time that an access that was in progress terminates. However, the PDP instance is always active, because while an access is in progress it could invoke the PEP to stop it, depending on the security policy. This is the main novelty required by the UCON model with respect to prior access control work, where the PDP is usually only passive. As soon as a grid user performs an operation r that attempts to access a resource o that is monitored by the PEP, the PEP suspends the access and issues the *tryaccess(s,o,r)* action to the PDP. The PDP, according to the security policy, retrieves the subject and object attributes required for the usage decision process from the attribute manager. These attributes are exploited to evaluate the authorization predicates. If the policy includes the evaluation of some conditions, the PDP retrieves the value of these conditions from the condition manager. If all the decision factors are satisfied, the usage decision process grants the right r to the user. The preupdates of the attributes are executed by the attribute manager invoked by the PDP. Then, the PDP returns the *permitaccess(s,o,r)* command to the PEP, which, in turn, resumes the execution of the access (s, o, r) it suspended before.

Now the access is in progress and we have two possible behaviors. The first possibility is that the access operation r is entirely executed and it finishes normally. In this case, the PEP intercepts the end access event and forwards it to the PDP through the *endaccess(s,o,r)* action. The other possibility is that the policy includes an ongoing authorization predicate for the access that is now in progress. In this case, if the values of the attributes that are evaluated in the authorization predicate change when the access is in progress, i.e., before the *endaccess(s,o,r)* is received from the PEP, and the new result of the usage decision process does not allow the access anymore, then the PDP issues a *revokeaccess(s,o,r)* command to the PEP. The PEP, in turn, interrupts the access (s, o, r) to the resource. This requires that the Globus resource and allocation management service has been modified to provide a proper interface to accept the interruption command from the PEP and to implement it. After the termination or the revocation of the access, the PDP performs the postupdates of the attributes through the attribute manager. In the grid environment, where the attributes may be represented through credentials issued by many authorities, the update procedure could be very complex (see, e.g,. some discussions in [16.45]).

16.7 Conclusion

In this chapter, we gave a short overview of well-known access control frameworks that have been integrated in grid systems and particularly Globus: CAS, PERMIS, Akenti, Shibboleth, VOMS, Cardea, and PRIMA. Most of the existing models provide attribute-based access control and make an access decision once. After the access to the resource has been granted, no more controls are executed.

UCON is a new access control paradigm, proposed by Sandhu and Park [16.33, 34], and overcomes some limitations of existing authorization models. It introduces continuous usage control and attributes mutability. We described two models for authorization in the grid based on UCON. Actually, the models can be improved by presenting, for example, a unified policy model and more sophisticated mechanisms for continuous policy enforcement.

Acknowledgements This work was partially supported by the EU project FP6-033817 GridTrust (Trust and Security for Next Generation Grids), contract no. 033827.

References

16.1. I. Foster, C. Kesselman, J. Nick, S. Tuecke: The physiology of the grid: An open grid service architecture for distributed system integration. Globus Project (2002), http://www.globus.org/research/papers/ogsa.pdf
16.2. I. Foster, C. Kesselman, S. Tuecke: The anatomy of the grid: Enabling scalable virtual organizations, Int. J. Supercomput. Appl. **15**(3), 200–222 (2001)
16.3. Open grid forum: http://www.ogf.org/
16.4. The Globus Alliance: Welcome to globus, http://www.globus.org
16.5. I. Foster: Globus toolkit version 4: Software for service-oriented systems. In: *Proc. IFIP Int. Conference on Network and Parallel Computing*, LNCS, Vol. 3779, ed. by H. Jin, D.A. Reed, W. Jiang (Springer, 2005) pp. 2–13
16.6. I. Foster, C. Kesselman: The globus project: A status report, Proc. IPPS/SPDP '98 Heterogeneous Computing Workshop (1998) pp. 4–18
16.7. M. Baker, R. Buyya, D. Laforenza: Grids and grid technologies for wide-area distributed computing, Int. J. Softw. Pract. Exp. **32**(15), 1437–1466 (2002)
16.8. S.J. Chapin, D. Katramatos, J. Karpovich, A. Grimshaw: Resource management in Legion, Future Gener. Comput. Syst. **15**(5/6), 583–594 (1999)
16.9. A. Vahdat, T. Anderson, M. Dahlin, E. Belani, D. Culler, P. Eastham, C. Yoshikawa: WebOS: Operating system services for wide area applications, Proc. 7th Symp. on High Performance Distributed Computing (1998)
16.10. D. Erwin, D. Snelling: UNICORE: A Grid computing environment. In: *EuroPar'2001*, Lecture Notes in Computer Science, Vol. 2150, ed. by R. Sakellariou, J. Keane, J. Gurd, L. Freeman (Springer, 2001) pp. 825–838
16.11. I. Foster, C. Kesselman, G. Tsudik, S. Tuecke: A security architecture for computational grids, Proc. 5th ACM Conference on Computer and Communications Security Conference (1998) pp. 83–92
16.12. M. Humphrey, M. Thompson, K. Jackson: Security for grids, Proc. IEEE **93**(3), 644–652 (2005)
16.13. N. Nagaratnam, P. Janson, J. Dayka, A. Nadalin, F. Siebenlist, V. Welch, I. Foster, S. Tuecke: Security architecture for open grid services, Global Grid Forum Recommendation (2003)
16.14. V. Welch, F. Siebenlist, D. Chadwick, S. Meder, L. Pearlman: Use of SAML for OGSA authorization (2004), https://forge.gridforum.org/projects/ogsa-authz
16.15. IBM: Web service trust language (WS-Trust), http://specs.xmlsoap.org/ws/2005/02/trust/WS-Trust.pdf
16.16. I. Foster, C. Kesselman, L. Pearlman, S. Tuecke, V. Welch: A community authorization service for group collaboration, Proceedings of the 3rd IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY'02) (2002) pp. 50–59
16.17. L. Pearlman, C. Kesselman, V. Welch, I. Foster, S. Tuecke: The community authorization service: Status and future. Proceedings of Computing in High Energy and Nuclear Physics (CHEP03): ECONF C0303241, TUBT003 (2003)
16.18. D. Chadwick, A. Otenko: The PERMIS x.509 role based privilege management infrastructure, SACMAT '02: Proc. 7th ACM symposium on Access control models and technologies (ACM Press, New York 2002) pp. 135–140
16.19. D.W. Chadwick, G. Zhao, S. Otenko, R. Laborde, L. Su, T.A. Nguyen: PERMIS: a modular authorization infrastructure, Concurr. Comput. Pract. Exp. **20**(11), 1341–1357 (2008), Online, ISSN: 1532-0634
16.20. A.J. Stell, R.O. Sinnott, J.P. Watt: Comparison of advanced authorisation infrastructures for grid computing, Proc. High Performance Computing System and Applications 2005, HPCS (2005) pp. 195–201
16.21. Permis: http://sec.cs.kent.ac.uk/permis/index.shtml
16.22. Akenti: http://dsd.lbl.gov/security/Akenti/
16.23. M. Thompson, A. Essiari, K. Keahey, V. Welch, S. Lang, B. Liu: Fine-grained authorization for job and resource management using akenti and the globus toolkit, Proc. Computing in High Energy and Nuclear Physics (CHEP03) (2003)
16.24. M. Thompson, A. Essiari, S. Mudumbai: Certificate-based authorization policy in a PKI environment, ACM Trans. Inf. Syst. Secur. **6**(4), 566–588 (2003)
16.25. Shibboleth project: http://shibboleth.internet2.edu/
16.26. V. Welch, T. Barton, K. Keahey: Attributes, anonymity, and access: Shibboleth and globus integration to facilitate grid collaboration, Proc. 4th Annual PKI R&D Workshop Multiple Paths to Trust (2005)

16.27. Gridshib project: http://grid.ncsa.uiuc.edu/GridShib
16.28. D. Chadwick, A. Novikov, A. Otenko: Gridshib and permis integration, http://www.terena.org/events/tnc2006/programme/presentations/show.php?pres_id=200
16.29. Datagrid security design: Deliverable 7.6 DataGrid Project (2003)
16.30. R. Alfieri, R. Cecchini, V. Ciaschini, L. dell Agnello, A. Frohner, A. Gianoli, K. Lorentey, F. Spataro: VOMS: An authorisation system for virtual organizations, Proc. 1st European Across Grid Conference (2003)
16.31. R. Lepro: Cardea: Dynamic access control in distributed systems, Tech. Rep. NAS Technical Report NAS-03-020, NASA Advanced Supercomputing (NAS) Division (2003)
16.32. M. Lorch, D.B. Adams, D. Kafura, M.S.R. Koneni, A. Rathi, S. Shah: The prima system for privilege management, authorization and enforcement in grid environments, GRID '03: Proc. 4th Int. Workshop on Grid Computing (IEEE Computer Society, Washington 2003) pp. 109–
16.33. R. Sandhu, J. Park: Usage control: A vision for next generation access control. In: *Workshop on Mathematical Methods, Models and Architectures for Computer Networks Security MMM03*, LNCS, Vol. 2776, ed. by V. Gorodetsky, L. Popyack, V. Skormin (Springer, 2003) pp. 17–31
16.34. R. Sandhu, J. Park: The UCON_*ABC* usage control model, ACM Trans. Inf. Syst. Secur. **7**(1), 128–174 (2004)
16.35. D. Bell, L. LaPadula: Secure computer systems: MITRE Report, MTR 2547, v2 (1973)
16.36. R. Sandhu, E. Coyne, H. Feinstein, C. Youman: Role-based access control models, IEEE Comput. **9**(2), 38–47 (1996)
16.37. X. Zhang, M. Nakae, M. Covington, R. Sandhu: A usage-based authorization framework for collaborative computing systems, Proc. 11th ACM Symposium on Access Control Models and Technologies (SACMAT'06) (ACM Press, 2006)
16.38. X. Zhang, M. Nakae, M.J. Covington, R. Sandhu: Toward a usage-based security framework for collaborative computing systems, ACM Trans. Inf. Syst. Secur. **11**(1), 1–36 (2008)
16.39. F. Martinelli, P. Mori, A. Vaccarelli: Towards continuous usage control on grid computational services, Proc. of Int. Conference on Autonomic and Autonomous Systems and International Conference on Networking and Services 2005 (IEEE Computer Society, 2005) p. 82
16.40. H. Koshutanski, F. Martinelli, P. Mori, A. Vaccarelli: Fine-grained and history-based access control with trust management for autonomic grid services, Proc. of Int. Conference on Autonomic and Autonomous Systems (2006)
16.41. GridTrust project: http://www.gridtrust.eu/
16.42. F. Martinelli, P. Mori, A. Vaccarelli: Fine grained access control for computational services. Tech. Rep. TR-06/2006, Istituto di Informatica e Telematica, Consiglio Nazionale delle Ricerche, Pisa (2006)
16.43. F. Martinelli, P. Mori: A model for usage control in grid systems, Proc. 1st Int. Workshop on Security, Trust and Privacy in Grid Systems (GRID-STP07) (2007)
16.44. X. Zhang, F. Parisi-Presicce, R. Sandhu, J. Park: Formal model and policy specification of usage control, ACM Trans. Inf. Syst. Secur. **8**(4), 351–387 (2005)
16.45. X. Zhang, M. Nakae, M. Covington, J.R. Sandhu: A usage-based authorization framework for collaborative computing systems, SACMAT (2006) pp. 180–189

The Authors

Maurizio Colombo received his bachelor degree from the University of Genoa in 2003 and his master degree in Internet technologies from the University of Pisa in 2005. Currently he collaborates with the Information Security Group of Istituto di Informatica e Telematica, Consiglio Nazionale delle Ricerche on the study and development of mechanisms for access and usage control in SOA environments. He has almost 2 years' experience in the Security Laboratory of British Telecom Research Centre (Ipswich, UK). He was involved in EU projects such as TrustCom, BeinGrid, and GridTrust.

Maurizio Colombo
Istituto di Informatica e Telematica
Consiglio Nazionale delle Ricerche
Via. G. Moruzzi 1, Pisa, Italy
maurizio.colombo@iit.cnr.it

Aliaksandr Lazouski received his MSc degree in electronics from the Belarusian State University in 2006. He is currently a PhD student in the Computer Science Department at the University of Pisa in collaboration with Istituto di Informatica e Telematica, Consiglio Nazionale delle Ricerche. His research interests include access control models, trust management, usage control, and digital rights management.

Aliaksandr Lazouski
Dipartimento di Informatica
Università di Pisa
Largo B. Pontecorvo 3, Pisa, Italy
lazouski@di.unipi.it

Fabio Martinelli (MSc 1994, PhD 1999) is a senior researcher at Istituto di Informatica e Telematica, Consiglio Nazionale delle Ricerche. He is a coauthor of more than 80 publications. His main research interests involve security and privacy in distributed and mobile systems and foundations of security and trust. He is the coinitiator of the International Workshop on Formal Aspects in Security and Trust (FAST). He serves as a scientific codirector of the International Research School on Foundations of Security Analysis and Design (FOSAD). He chairs the working group on security and trust management of the European Research Consortium in Informatics and Mathematics (ERCIM).

Fabio Martinelli
Istituto di Informatica e Telematica
Consiglio Nazionale delle Ricerche
Via. G. Moruzzi 1, Pisa, Italy
fabio.martinelli@iit.cnr.it

Paolo Mori (MSc 1998, PhD 2003) is a researcher at Istituto di Informatica e Telematica, Consiglio Nazionale delle Ricerche. He is an author or a coauthor of more than 20 publications. His main research interests involve high-performance computing and security in distributed and mobile systems. He is involved in several EU projects on information and communication security (S3MS, GridTRUST).

Paolo Mori
Istituto di Informatica e Telematica
Consiglio Nazionale delle Ricerche
Via. G. Moruzzi 1, Pisa, Italy
paolo.mori@iit.cnr.it