International Journal of Information Security manuscript No.
(will be inserted by the editor)

Hristo Koshutanski · Aliaksandr Lazouski · Fabio Martinelli · Paolo Mori

Enhancing Grid security by fine-grained behavioral control and negotiation-based authorization

Abstract Nowadays Grid has become a leading technology in distributed computing. Grid poses a seamless sharing of heterogeneous computational resources belonging to different domains and conducts efficient collaborations between Grid users. The core Grid functionality defines computational services which allocate computational resources and execute applications submitted by Grid users. The vast models of collaborations and openness of Grid system require a secure, scalable, flexible and expressive authorization model to protect these computational services and Grid resources. Most existing authorization models for Grid have granularity to manage access to service invocations while behavioral monitoring of applications executed by these services remains a responsibility of a resource provider. The resource provider executes an application under a local account, and acknowledges all permissions granted to this account to the application. Such approach poses serious security threats to breach system functionality since applications submitted by users could be malicious.

We propose a flexible and expressive policy-driven credential-based authorization system to protect Grid computational services against a malicious behavior of applications submitted for execution. We split an authorization process in two levels: a coarse-grained level that manages access to a computational service; and a fine-grained level that monitors the behavior of applications executed by the computational service. Our framework guarantees that users authorized on a coarse-grained level behave as expected on the fine-grained level. Credentials obtained on the coarse-grained level reflect on fine-grained access decisions. The framework defines trust negotiations on coarse-grained level to overcome scalability problem, and preserves privacy of credentials and security policies of, both, Grid users and providers. Our authorization system was implemented to control access to the Globus Computational GRAM service. A comprehensive performance evaluation shows the practical scope of the proposed system.

Keywords Grid security · Authorization · Access control · Trust negotiation · Behavioral control

Hristo Koshutanski was supported by the Marie Curie Intra-European fellowship 038978-iAccess within the 6th European Community Framework Programme. Aliaksandr Lazouski, Fabio Martinelli and Paolo Mori were partially supported by the EU project FP6-033817 GridTrust (Trust and Security for Next Generation Grids)

H. Koshutanski
Computer Science Department, University of Malaga
Campus de Teatinos, 29071 Málaga, Spain
E-mail: hristo@lcc.uma.es

A. Lazouski
Dipartimento di Informatica, Universita di Pisa
Largo Bruno Pontecorvo, 3, 56127 Pisa, Italy
E-mail: lazouski@di.unipi.it

F. Martinelli
Istituto di Informatica e Telematica, Consiglio Nazionale delle Ricerche
Via Giuseppe Moruzzi, 1, 56124 Pisa, Italy
E-mail: fabio.martinelli@iit.cnr.it

P. Mori
Istituto di Informatica e Telematica, Consiglio Nazionale delle Ricerche
Via Giuseppe Moruzzi, 1, 56124 Pisa, Italy
E-mail: paolo.mori@iit.cnr.it

1 Introduction

Grid technology concerns about sharing, managing, and integrating resources and services within a heterogenous, dynamic, and distributed computational environment. A group of entities deploys Grid technology to share their resources across organizational boundaries and to solve some particular tasks. Grid technology has been successfully exploited for execution of computational or data intensive applications with very large scale simulations e.g., earthquake simulation [47], large scientific data sets [9], or climate study [40,36].

One of the fundamental Grid functionality is the ability of clients to execute their applications on remote computational resources allocated by Grid [15]. This ability

gives high-value to distributed computing. Often entities dedicated to providing computational resources are different and independent from entities exploiting them. Grid, by its nature, is a large-scale open system which includes enormous number of entities. These entities may belong to different administrative domains with potentially unknown (trust) relationships among them.

In this paper, we focus on security issues in Grid. The Grid Security Infrastructure (GSI) [17] provides a set of mechanisms and tools to enhance security in the Grid. GSI has been implemented in the Globus toolkit [14], the most used middleware to set up Grids. By default, authorization in Globus is realized for service-level invocation and implies identity-based access control using Gridmap file. Gridmap faces scalability problem in Grid formed by a large number of unknown users. Several credential-based authorization models have been proposed [21, 50, 49, 33, 12, 4]. They follow the scenario where administrators of Grid domains issue credentials to users that specify permissions users can hold on Grid resources. Authorization system checks validity of presented credentials to grant or deny an access.

For open systems like Grid, an assumption on pre-existing relationships between entities does not hold, and any user should be treated as distrusted and potentially malicious. Existing authorization solutions for Grid have only granularity to manage access to service invocations (coarse-grained access control), while service execution and behavioral monitoring (fine-grained access control) remain under the responsibility of a resource provider. Service instance is assigned to the local account and has all permissions granted to this account. Such approach compromises the principle of least privilege [41]. Moreover, executing remote and potentially malicious applications on a local Grid platform of a resource provider poses serious security threats to breach system functionality. With dynamic coalitions and resource-consuming requests, existing approaches are neither flexible nor secure and expressive enough for coherent access control on both coarse- and fine-grained levels. Grid entities should be able to define and enforce a desired level of protection on their resources and establish trust relationships dynamically from scratch.

To overcome the above identified problems, we propose a flexible and expressive authorization system allowing coarse-grained authorization to access Grid computational services, and fine-grained monitoring of applications executions submitted by remote Grid users. The coarse-grained authorization implies that a remote Grid user has sufficient access rights on computational services intended to use, while the fine-grained monitoring ensures that a user's application explores resources only under the user's access rights granted on the coarse-grained level and in accordance to application behavior allowed by a fine-grained security policy. Integrating two levels of control in a single framework, we ensure that users authorized on the coarse-grained level exploit computational resources and behave as expected on the fine-grained level. This protects Grid node from unauthorized resource usage and malicious behavior of applications submitted by remote Grid users.

We implemented the trust negotiation scheme [26] for the coarse-grained authorization[1] to mitigate the scalability problem in Grid, and to enhance the expressiveness of the coarse-grained level. Trust negotiation is a policy-based technique that provides entities with the right to protect their own credentials and to negotiate with other entities access to those credentials. Trust negotiation allows two network entities to establish requirements to access a resource by mutually requesting each other sensitive credentials until a sufficient trust is established. The authorization decision is more efficient comparing to simple "yes/no" and based on entity's credentials rather than on their identity. Moreover, trust negotiation is an efficient technique for peer's privacy preservation and dynamic policy evaluation. We implemented trust negotiation only on the coarse-grained level due to its computational cost.

Fine-grained authorization is implemented through a reference monitor that enforces a behavioral control on the execution of remote users' application. A fine-grained behavioral policy determines the allowed sequence of actions which can be executed on underlying resources. Credentials obtained during the coarse-grained authorization are taken into account for access decisions at the fine-grained level. The novelty of exploiting credentials of coarse-grained in the fine-grained level allows us to enforce a coherent credential-based access control from service instance creation till its elimination. We use POLPA policy language [35] to define behavioral policies and to integrate credential-based decisions into the behavioral control process.

The proposed authorization system was successfully implemented and deployed in the Globus toolkit. The system governs access to the Grid Resource Allocation and Management (GRAM) service [13] and its underlying computational resources. GRAM is a fundamental Globus service enabling remote clients to securely instantiate, manage, and monitor computational tasks (jobs) on remote resources [53]. The authorization system controls the execution of java-based applications submitted by remote Grid users. We did a detailed performance evaluation of the system, and the obtained results were positive to show system practical aspects.

Below we summarize key system security properties:

Coarse-grained level. Credential-based authorization enhanced with trust negotiation. Trust negotiation preserves security threat of unauthorized disclosure of sensitive credentials and security policies. It preserves peer's privacy and reveals minimum of peer's information needed to succeed coarse-grained authorization. Trust negotiation is a suitable methodology when

[1] Available at http://www.interactiveaccess.org

establishing access rights with unknown and potentially malicious users. Trust negotiation establishes access rights dynamically from scratch. Integration of X.509 [57] identity and attribute certificates was implemented in our model.

Fine-grained level. Efficient and flexible behavioral control enhanced with credential-based access decisions on remote Grid users' applications execution. Fine-grained access control protects a Grid node from threats of a malicious behavior of applications submitted for execution. Fine-grained access control guarantees that computational resources are exploited properly. Enhancing expressiveness of fine-grained level, we introduced the use of (temporal) behavioral credentials reflecting access decisions over a long-term resource usage.

Two-levels integration. Our model integrates two layers of security, coarse- and fine-grained access control, through the usage of same user's attributes. From the security point of view, the integration of the two levels guarantees that a user will exploit fine-grained recourses in compliance with access rights granted on the coarse-grained level. Such approach satisfies the principle of least privilege and assumes fully credential-based access control on both levels. To the best of our knowledge, none of existing authorization systems for Grid could enforce both levels of access control.

The paper is organized as following. Section 2 describes related work in the field. Section 3 defines the high-level system architecture and its functionality. Section 4 reviews the fine-grained behavioral control model and its policy language. Section 5 outlines the credential-based access control model underlying both the coarse- and fine-grained levels. Section 6 presents a negotiation scheme underlying the authorization service, and its integration in Globus. A Grid usage scenario and an example of integrated security policies are shown in Section 7. Section 8 describes details on system integration into the Globus toolkit. Section 9 provides analytical details on the framework performance evaluation. Section 10 discusses security aspects of system design and its architectural components. Conclusions and future work are drawn in section 11.

2 Related Work

This Section presents current research efforts devoted to improving Grid authorization model, and how our system model enhances them. The default authorization framework in Grid, and particularly in Globus, implies identity-based authorization. Grid users are identified and simply mapped to local accounts on resource provider hosts. Such authorization system suffers from a lack of policy expressiveness, as well as, it is unable to scale and manage a large user base. As a result, it compromises the principle of least privileges. Recent studies on Grid security [10,27,33,12,4,38] tend to move from identity-based to credential-based authorization to overcome the scalability problems.

Keahey and Welch [21,50] propose an approach that adapts an existing authorization system, called Akenti [51], to the Grid environment. Similarly, Stell et al. [49] integrate a role based access control (RBAC) infrastructure, called PERMIS [8], with Globus. The main functionality behind PERMIS and Akenti is that the information needed for an access decision is stored and conveyed in certificates, which are widely dispersed over a network (e.g., LDAP directories, Web servers, etc). The authorization engine has to gather and verify all user's related certificates and evaluate them against the access policy in order to take a decision. X.509 identity and attribute certificates are used in PERMIS to attest and convey users credentials. Relevant access control frameworks such as VOMS, PRIMA [33], XPOLA [12], GridShib [4], Cardea [29], and CAS [16,38] are also designed to embody credential-based authorization into the Globus toolkit.

However, the access control frameworks mentioned above manage coarse-grained access to a service invocation and suggest one-step (single "sign-on") authorization process. Most of them assume that the requester has some preliminary knowledge about authorization tokens required to access the resource. These tokens later are pushed or pulled to an authorization engine which replies usually with boolean grant or deny. This approach is not expressive and flexible enough to work in an open and dynamic environment like Grid.

Trust negotiation [54,55] is a promising technique that was proposed for authorization and access control in open environments where any entity could be a malicious. Trust negotiation allows to authorize peers which do not have complete knowledge about each other, belong to different administrative domains, and may have never interacted before. Trust negotiation implies credential-based authorization and bilateral exchange of credentials. Definitely, the next step in evolution of authorization frameworks for Grid is the incorporation of a trust negotiation capability.

Recently, only few trust negotiation schemas have been successfully developed for the Globus toolkit. Traust authorization service [27] incorporates into GridFTP service a trust negotiation functionality. The approach employs trust negotiation in Grid using existing negotiation mechanisms and protocols such as TrustBuilder [56] and Trust-X [7] systems. The Traust service allows to obtain the credentials needed to access data provided by the GridFTP server at runtime without previously assigned account. Thus, a user opens a new connection to the GridFTP server and starts the specific command "TRAUST". The response of the GridFTP server is forwarded to the external Traust server. The Traust server acts as a broker that negotiates with the user. If trust negotiations between the Traust server and user succeed,

the Traust server issues credentials needed to log into the GridFTP server. These credentials are mapped to the appropriate GridFTP account and the user may access the data.

Instead of using centralized trust negotiation service, the model proposed in [10] deploys PeerTrust [37] negotiation capabilities into the Grid service functionality. A policy decision point (PDP) of Globus authorization service is extended by an intercepter of service requests. The intercepter grants access to a service if a negotiation process has been successfully completed. The negotiation module, implemented as a service port type, negotiates with the requester. Negotiations are asynchronous and implemented though WS-Notification mechanisms. The implementation changes the service WSDL file and exploits state-full resources defined by WS-Resource Framework[2].

The authorization framework presented in this paper also enforces trust negotiation on coarse-grained level. Opposing to approaches in [27, 10], we integrated in Globus Toolkit our authorization service with trust negotiation capabilities exploiting the SAML authorization callout mechanism. Thus, the negotiation-based authorization service could be internal (i.e. on the same node) or external. This makes our approach more flexible as no modifications are required in the code of a service and its deployment on the coarse-grained level. The integration is done only by a proper specification of a service descriptor. Moreover, after a detailed analysis of our prototype performance the results were as promising as those reported in Traust. We recall this information in Section 9.

Several works express and enforce security policies on coarse-grained level assuming different policy complexity. CAS, Akenti, XPOLA and PRIMA already provide more expressive authorization model in comparison with Globus gridmap. They are plugged into Globus Toolkit and achieve the policy granularity by specifying access rules on service instance, as well as on the operation to be invoked and even operands of the operation. Access permissions to the underlying resources used by Grid services such as files and databases are also governed by those authorization systems. Our approach introduces flexible and expressive model to specify complex security policies. As example, a security policy (from Grid usage scenario presented in Section 7) manages access to the service and computational libraries aligned with this service.

Once access to a Grid service is granted (GRAM service in our case), the service execution and access control to underlying resources is enforced by the hosting environment. In the current Globus implementation, a service instance is assigned to a local account of a host's operating system and may perform any operation allowed by that account. From the security point of view, this access control model violates the least privilege principle and is not expressive enough to define a fine-grained access control over Grid resources. By the way, some attempts were done to security on fine-grained level. For instance, PRIMA proposes to use dynamic user accounts created on demand and to grant access permissions on short-live basis. The approach given in [30] enforces RBAC in the Globus and proposes to bind the appropriate authorization service to the resources in MDS information service. The EU DataGrid Security architecture [1] introduces several enforcement mechanisms like GridACL, LCAS and Java authorization manager. The approach in [22] enhances fine-grained monitoring of jobs executed by Globus GRAM service. The authorization model controls the resource requirements stated at the RSL of the job request and management requests coming from the user (e.g., suspend, stop, etc.).

Despite of this approaches, fine-grained access control over Grid services is not scrutinized sufficiently. As a matter of fact, a proper fine-grained access control is a vital requirement for Grid computational services (e.g. GRAM) where users could submit a potentially dangerous applications for execution. If there is no control over applications execution, nothing could prevent a user to submit malicious code and breach system functionality. To the best of our knowledge, fine-grained monitoring of applications execution was not presented so far in Grid.

The fine-grained level security support of our framework could be related to a kind of Intrusion Detection Systems (IDS), or Application Based Anomaly Detection systems. The Application Based IDS, a particular case of Host Based ones, are software components that monitor the actions executed by the applications they are paired with. Some of them simply use log files to detect the actions executed by the application, while others are interfaced with the application to be monitored through a specific component. To detect whether a particular application behavior is an attack, Anomaly Detection IDSs exploit a model of the usual behavior of the specific application to be monitored, which has been defined in advance, and determine if the current behavior complies with this model. Some systems, beside detecting the malicious behavior, are also able to confine the application, by avoiding the execution of forbidden actions. Some Application Based Anomaly Detection IDSs choose the system calls as security relevant actions performed by the applications; some examples can be found in [39, 19, 45].

Despite of some similarities, Application Based Anomaly Detection IDSs also present several differences with respect to the approach proposed in this paper. Firstly, our policy model expressiveness was enhanced by using composition operators allowing to define very complex sequence of actions. Secondly, fine-grained access control in our model takes into account several factors in the decision process, such as user's credentials obtained during coarse-grained authorization. Moreover, in case of IDSs, the applications to be monitored are typically

[2] http://www.globus.org/wsrf

the operating system daemons, and the policies are customized for each specific application, i.e. distinct applications are monitored with distinct policies. In our approach, instead, a single (behavioral) policy is paired with the computational node, and expresses how applications can exploit the resources provided by the node (look at the example in Section 7 where the behavioral policy defines constraints for using the computational libraries provided by the Grid node). Finally, to the best of our knowledge, it is the first time when a fine-grained monitoring system has been adopted in Grid computing to protect computational resources from the threat represented by the applications executed on behalf of remote Grid users.

Summarizing, our approach extends existing authorization solutions in Grid by encompassing coarse- and fine-grained levels into a coherent authorization framework and continues the work in [24,23]. We define trust negotiation on the coarse-grained level to authorize the invocation of a service instance, and to grant access for intended to use computational resources. Credentials obtained during the coarse-grained authorization are taken into account for access decisions at the fine-grained level. Fine-grained authorization is implemented through a reference monitor that enforces a behavioral control on executed user's applications. Our framework guarantees that users authorized on a coarse-grained level behave as expected on the fine-grained level.

3 System Architecture and Functionality

In this section we review system high-level architecture and its message flow. The important aspect here is to identify the main components necessary to enforce coarse-grained and fine-grained access control.

Figure 1 shows the overall architecture. It consists of two blocks representing a Grid user and a Grid resource provider belonging to different security domains. We note that in our notation each Grid user domain could act as Grid resource provider domain but in different sessions, and vice versa. We functionally distinguish their roles with respect to a given application execution request.

Grid resource provider holds a Globus container. The Globus container is the component that manages the infrastructure services (including the authorization service) and the set of Grid's services that are provided to remote Grid users. Hence, the Globus container receives the service request from the remote Grid user, performs the configured security checks, and starts the requested service. We start by symbolically dividing the Grid resource authorization facilities in two operational parts: coarse-grained access control on a service level (light-gray color area in figure 1); and fine-grained behavioral control on the usage of computational resources.

The coarse-grained level controls invocations of Grid services with service-level granularity. This level is evidenced in most of the access control systems for Grid, as we have seen in the previous section. We consider access control requirements on that level (matured enough in policy technologies and specifications) to be similar to those of existing trust management and negotiation models [31,43]. These requirements include: attribute-based access decisions, attribute-based delegation, policy constrains such as separation of duties or mutually exclusive attributes, etc. We extended the standard Grid authorization model with automated trust negotiation capabilities to satisfy these requirements, and free the assumption on preexisting trust relationships between entities.

Thus, the first component we introduce is the negotiation-based Authorization Service. It operates as a PDP on the coarse-grained level. The Authorization Service is invoked by the Globus Container once the request for a service has been received (messages 1 and 2). The authorization service implements the access control model outlined in Sections 5 and 6 and includes two subcomponents:

(i) Trust Negotiation Engine – responsible for carrying out the negotiation protocol described in Section 6. The protocol runs over a protected channel (message 3) and utilizes credential negotiation in order to enforce (on the fly) user's and resource provider's mutually satisfiable requirements.
(ii) Credential Manager – complements the functionalities of the trust negotiation engine. It provides interfaces for: certificate validation and verification, certificate transformation to logic predicates suitable for policy evaluation, user profile maintenance (PKI and logic level), allocation of requested credentials by an opponent. It conforms to X.509 certificate framework.

The credential manager keeps an internal database for credential transformations. The database also includes what logic data structures are used for identity and attribute certificates, how X.500 names are mapped to internal (logic) identifiers, and what internal identifiers trusted CAs and SOAs have.

In order to achieve interoperability of negotiations with a Grid resource provider a Grid user should have an instance of the authorization service including a trust negotiation engine and a credential manager on its side.

When a trust negotiation completes, the authorization service sends an access decision response back to the Globus container (message 4). Globus container includes itself a policy enforcement point (PEP) on the coarse-grained level which enforces access decisions (if positive) by creating a service instance (message 5). Here, we focus on the Globus Resource Allocation and Management service (GRAM). GRAM is the service that allows the execution of applications on the local resources on behalf of remote Grid users. The fine-grained level of our

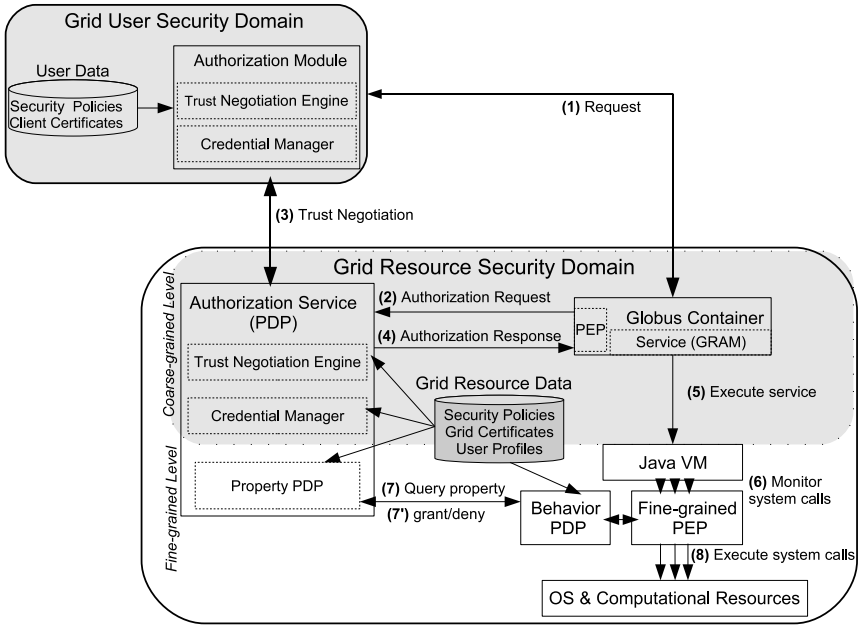

Fig. 1 High-level system architecture

authorization system monitors Java applications initiated by the GRAM service. However, the authorization request/response interactions are defined for any Globus container-related services so one can apply the system architecture to authorize and monitor executions of other (java-based) Grid services.

The most critical aspect on the fine-grained level is to provide efficient components controlling access to underlying computational resources. We have defined a Fine-grained PEP (a component integrated into the Java VM) that intercepts system calls performed by an application (message 6). The Fine-grained PEP enforces the decisions taken by a Fine-grained PDP. Sections 4 and 5 overview the security models considered for the fine-grained control. The Fine-grained PDP consists of a Behavioral PDP and a Property PDP. The Behavioral PDP is invoked by the Fine-grained PEP every time a system call is performed. The behavioral policy is defined by the computational resource owner to describe acceptable sequence of actions performed on computational resources by an application. When some of the resources have specific requirements or specific properties to be met regarding users' credentials (e.g., when opening a file), then the behavioral policy defines that as an external property evaluation. These specific properties are predefined for a given Grid domain and are encoded as logic rules. These rules define properties based on user's credentials gathered during trust negotiation on the coarse-grained level. The rules are stated in a property policy. Thus, the Behavioral PDP interacts with the Property PDP only on those application steps in accordance with the behavioral policy when evaluation of an external property is required (message 7 and 7'). If a system call is authorized the Behavioral PDP informs the Java VM (thought the Fine-grained PEP) to execute the call (message 8).

The Property PDP has the responsibility to evaluate user specific properties against user's active credentials submitted to the system. The Property PDP operates as part of the Grid authorization service and integrates credential-based information on coarse-grained level with fine-grained behavioral requirements and their evaluation. Since we aim at protecting computational resources from third-party applications executed on behalf of (remote) Grid users, so it is not enough that the behavioral policy defines allowed system traces based on application-only information such as system calls performed, but also on user-specific information, called user properties, relevant to the behavioral decision process and based on user high-level credential definition. In this way we also strengthen the coherence of the authorization decisions on both levels.

A Grid Resource Data component maintains a repository of security information relevant to the system. This includes: coarse-grained security polices; fine-grained behavioral and property policies; Grid certificates for mutual trust negotiations; and users' profiles of active credentials.

4 Behavioral Access Control Model

The model described in this section defines the fine-grained level of our system architecture, because it is aimed at improving the security of Grid computational services. Since a Grid computational service executes applications submitted by remote (and possibly malicious)

Grid users on the local resource, our authorization system monitors the execution of such applications and enforces a fine-grained and history-based security policy. The monitoring is fine-grained because instead of considering the execution of an application as a single atomic action, we split down the monitoring in basic actions performed by an application during its execution. In particular, since we are interested in the interactions with the underlying resource, the actions we monitor are the system calls that applications invoke on the operating system level. Hence, the sequence of actions performed by an application during its execution defines the behavior of the application itself. This sequence is not deterministic, because it may depend on various factors, such as specific input values.

The model is history-based because the actions that an application is allowed to perform at a given point of its execution depend on its past behavior, i.e. on the sequence of actions previously executed by the application itself. Hence, a given action a could be allowed only if some other actions have (not) been already executed.

We introduce a *behavioural policy* in order to define the set of actions and the execution patterns that applications must conform to. The behavioral policy is defined by the resource provider, i.e. by the Grid node administrator, to protect the resources shared on the Grid computational node, such as data, file system, network or software libraries. The policy is trusted, because it is defined and enforced by the same authority that owns the resources to be protected. The policy defines the behavioral patterns that should be followed to interact with these resources. The behavioral policy enforcement ensures that each application follows these behavioral patterns. A new instance of the same policy is created to monitor each application submitted to the Grid node. However, the policy specification does not depend on the specific application to be monitored, but on the resources to be protected. The policy is updated when the resources are changed or other administrative actions are demanded (e.g. to make a more restrictive behavioral pattern exploiting resources). In particular, a behavioral policy consists of several rules, each defining the admitted behavior to access a distinct resources. Rules could be independent one from each other, or there could be interactions among some of them. For example, let us suppose that the Grid computational node provides two conflicting software libraries, A and B, that export different implementations of the same functions, and that an application can use functions of A or functions of B, but it cannot use functions of A and B together. In this case, the policy could state that in principle applications can access any function of the two libraries, but when an application uses a function from A, then it can use functions from A only while access to B is forbidden, and vice versa.

To express a behavioural policy we adopt an operational policy language, because it is close to user's expertise, and we use a process description language because we deal with a sequence of actions. The name of the proposed language is POLPA, POlicy Language based on Process Algebra. The following grammar shows the operators of POLPA:

$$P ::= \bot \parallel \top \parallel \alpha(\mathbf{x}).P \parallel p(\mathbf{x}).P \parallel \mathbf{x} := \mathbf{e}.P \parallel P_1 or P_2 \parallel P_1 par_{\alpha_1,..,\alpha_n} P_2 \parallel \{P\} \parallel Z$$

where P is a policy, $\alpha(\mathbf{x})$ is a security-relevant action, $p(\mathbf{x})$ is a predicate, $\mathbf{x}$ are action parameters and/or variables and Z is a constant process definition $Z \doteq P$.

The informal semantics is the following:

- $\bot$ is the *deny-All* operator;
- $\top$ is the *allow-All* operator;
- $\alpha(\mathbf{x}).P$ is the *sequential operator*, and represents the possibility of performing an action $\alpha(\mathbf{x})$ and then behave as P;
- $p(\mathbf{x}).P$ behaves as P in the case the predicate $p(\mathbf{x})$ is true;
- $\mathbf{x} := \mathbf{e}.P$ assigns to variables $\mathbf{x}$ the values of the expressions $\mathbf{e}$ and then behaves as P
- $P_1 or P_2$ is the *alternative operator*, and represents the non deterministic choice between P_1 and P_2;
- $P_1 par_{\alpha_1,...,\alpha_n} P_2$ is the *synchronous parallel operator*. It expresses that both P_1 and P_2 policies must be simultaneously satisfied. This is used when the two policies deal with actions in $(\alpha_1, \ldots, \alpha_n)$;
- $\{P\}$ is the *atomic evaluation*, and represents the fact that P is evaluated in an atomic manner. P here is assumed only to have one action, predicates and assignments;
- Z is the constant process. We assume that there is a specification for the process $Z \doteq P$ and Z behaves as P.

As an example, if the policy includes the sequence $a.b.c$, where a, b and c are security relevant actions, the application can execute the action b only if a has been already executed, and it can execute c only if a and b have been executed.

As usual for (process) description languages, derived operators may be defined. For instance, $P_1 par P_2$ is the *parallel operator*, and represents the interleaved execution of P_1 and P_2. It is used when the policies P_1 and P_2 deal with disjoint actions. The policy sequence operator $P_1; P_2$ may be implemented using the policy languages operators (and control variables) (e.g., see [18]). It allows to put two processes behavior in sequence. By using the constant definition, the sequence and the parallel operators, the iteration and replication operators, `i(`P`)` and `r(`P`)` resp., can be derived. Informally, `i(`P`)` behaves as the iteration of P zero or more times, while `r(`P`)` is the parallel composition of the same process an unbounded number of times.

The previous grammar shows that our policy can also include properties that have to be verified before the application is allowed to invoke a given system call on the

resource. These properties are represented by predicates that precede the systems calls they refer to. As an example, $a(\mathbf{x}).[p(\mathbf{x}), q(\mathbf{x})].b(\mathbf{x})$ means that the properties $p(\mathbf{x})$ and $q(\mathbf{x})$ must be verified before performing the action $b(\mathbf{x})$, but it is not required that are verified before the execution of the action $a(\mathbf{x})$. These properties could involve the evaluation of conditions of various nature. For instance, they can force the value of some system call parameter. In the following rule:

$$[(x_1 == \text{READ})].\text{open}(x_0,x_1,x_2,x_3)$$

the predicate that precedes the `open` system call states that the second parameter of this action, i.e. the open mode, must be equal to the constant READ. This allows the application to open files in read mode.

Predicates could also include properties that concern the evaluation of factors that does not involve only the system calls and their parameters. The exploitation of external factors provides a flexible way to evaluate distinct kind of conditions from the ones provided by the behavioural policy, i.e. to integrate and exploit other policies with the behavioral one. A very simple example could be a property that evaluates whether the current time is within a given time interval.

The definition of an integrated framework for the specification and analysis for security and trust in complex and dynamic scenarios was introduced in [34]. Particularly in our model, the behavioural policy could include properties that involve the evaluation of a set of credentials submitted by a user during trust negotiations on the coarse-grained level. This novelty was done in order to integrate fine-grained and coarse-grained levels and enforce a coherent credential-based access control from a service instance invocation till its termination. A new predicate, **property(**$User, \beta$**)** is embedded in our policy to state that a property β regarding user distinguished name $User$ is to be verified by an external evaluation. The credential evaluation is done by a logic engine and in accordance to a user property policy introduced in the next section. Let us consider the following example.

$$[\text{property(User, non\_profit)}\ ].\text{open}(x_0,x_1,x_2,x_3)$$

It allows an application to open files only if the property `non_profit` is validated according to an access policy. We note that the predicate denotes a template for a property query and $User$ is instantiated with the distinguished name of the user (an application running on behalf of), as shown in the usage scenario in Section 7. When an external property has to be evaluated, the behavioural policy delegates the decision to an evaluation module (in our case, the Property PDP). Once a decision is taken the monitoring is resumed and exploited by the behavioural policy.

The details concerning the implementation of the engine that evaluates the behavioral policy, i.e. the Behavioral PDP, are described in [35], while the integration of this engine in the proposed architecture is described in section 8.

5 Credential-based Access Control

In this section we examine the syntax and semantics of the access control model underlying both the coarse-grained access decision and the fine-grained property evaluation process. Access policies are written as normal logic programs [3]. A logic program is a set of rules of the form:

$$A \leftarrow B_1, \ldots, B_n,\ not\, C_1, \ldots,\ not\, C_m \tag{1}$$

A is called the head of the rule, each B_i is called a positive literal and each $not\, C_j$ is a negative literal, whereas the conjunction of B_i and $not\, C_j$ is called the body of the rule. If the body is empty the rule is called a fact.

In the model we also have constraints that are rules with an empty head.

$$\leftarrow B_1, \ldots, B_n,\ not\, C_1, \ldots,\ not\, C_m \tag{2}$$

A constraint (2) is used to rule out from the set of acceptable models situations in which all B_i are true and all C_j are false.

Below we list the core predicates defined for the logical model representation at coarse- and fine-grained level.

- `grant`($User$, Service : s, Action : p, Resource : r) a predicate denoting that a $User$ is granted to invoke an action p on a Grid service s which exploits an underlying resource r.
- `property`($User$, `Property`: p) a predicate denoting that a $User$ has a property p.
- `cred`($User$, Attr : a, Issuer : i) a predicate denoting a credential token of a $User$ having attribute a issued by i.
- `behav_cred`($User$, Status : s) a predicate denoting that a $User$ has a behavioral status s.

The access control model presented in this section abstracts from specific policy syntax so one can adopt different syntax for information representation such as resource representation, credential information, behavioral information, user properties etc. In the model we have predefined sets of identifiers for user properties (`Property`), attributes (`Attr`), credential issuers (`Issuer`), behavioral status (`Status`), resources (`Resource`) and actions (`Action`). Since Grid is an open system with a priori unknown clients we do not have a predefined set of client identities in the model. We denote that with a variable $User$ specifying untyped entity value in the respective fields of the predicates above.

5.1 Coarse-grained Access: Feedback on Missing Credentials

The intuition behind the coarse-grained access control model is to provide clients with a feedback on missing

`AccessDecisionWithFeedback`(r, $\mathcal{P}_\mathcal{A}$, $\mathcal{P}_\mathcal{D}$, $\mathcal{C}_\mathcal{A}$, $\mathcal{C}_\mathcal{N}$)

1: **if** r is a consequence of $\mathcal{P}_\mathcal{A}$ and $\mathcal{C}_\mathcal{A}$ **then** `grant`
2: **else**
3: **compute** a set of disclosable credentials $\mathcal{C}_\mathcal{D}$ entailed by $\mathcal{P}_\mathcal{D}$ and $\mathcal{C}_\mathcal{A}$. Remove from $\mathcal{C}_\mathcal{D}$ all presented and declined credentials, i.e. $\mathcal{C}_\mathcal{D} = \mathcal{C}_\mathcal{D} \setminus (\mathcal{C}_\mathcal{N} \cup \mathcal{C}_\mathcal{A})$,
4: **compute** a set of missing credentials $\mathcal{C}_\mathcal{M}$ such that
(i) $\mathcal{C}_\mathcal{M} \subseteq \mathcal{C}_\mathcal{D}$,
(ii) $\mathcal{P}_\mathcal{A}$ together with $\mathcal{C}_\mathcal{A}$ and $\mathcal{C}_\mathcal{M}$ grant r,
(iii) $\mathcal{C}_\mathcal{A}$ and $\mathcal{C}_\mathcal{M}$ preserve $\mathcal{P}_\mathcal{A}$ consistent.
5: **if** no set found **then** `deny` **else** ***ask***$(\mathcal{C}_\mathcal{M})$.

Fig. 2 Coarse-grained access decision with feedback on missing credentials

credentials in cases of not enough access rights. Also the work by [20,6] identifies the need of a feedback on access control requirements for open systems. The underlying access control model [25] is data-driven by two logic reasoning services: *deduction* and *abduction* [46]. We use deduction logic reasoning when taking decisions on whether a client has enough access rights to get a service, and abduction on logic programs as a core reasoning when computing a feedback on missing credentials. This section illustrates the essence of the interactive access control process as a core element for negotiation. We refer the reader to [25] for details on the two reasoning services and their deployment in an interactive access control model.

Each Grid security domain has a security policy for access control $\mathcal{P}_\mathcal{A}$ and a security policy for disclosure control $\mathcal{P}_\mathcal{D}$. $\mathcal{P}_\mathcal{A}$ protects Grid's resources by stipulating what credentials a requester must satisfy to be authorized for a particular resource while, in contrast, $\mathcal{P}_\mathcal{D}$ defines which credentials among those occurring in $\mathcal{P}_\mathcal{A}$ are disclosable so, if needed, can be demanded from a client.

Each user has a profile of active credentials $\mathcal{C}_\mathcal{A}$ available to a Grid security domain during a negotiation process. The server keeps user's set of active credentials for the duration of the user application request and its execution in Grid. The credential profile is also accessible by the fine-grained monitoring level regarding user property decisions.

A Grid security domain also keeps a set of declined credentials $\mathcal{C}_\mathcal{N}$ that keeps track of what credentials a client has declined to provide within a negotiation session. Declined credentials are internal to the negotiation model and are kept only for the duration of a current authorization process. They are not used at fine-grained level. The purpose of the declined credentials is to avoid loops in a negotiation process and to guarantee successful interactions in presence of alternative solutions.

Figure 2 shows the core access decision algorithm with feedback on missing credentials. Input to the decision process is the service request, access policy, disclosure policy, user's set of active and declined credentials. First step checks if the user has enough access rights to access the resource according to its active credentials and Grid's access policy. If the check succeeds the function returns grant.

In case of not enough access rights, the algorithm performs two steps to compute a feedback on missing credentials. It first computes a set of disclosable credentials inferred form user's active credentials and the Grid's disclosure policy, and from the resulting set, it removes the already presented and declined credentials. Second, the algorithm uses an abduction reasoning to compute a set of missing credentials, out of the disclosable ones, that is sufficient to unlock the requested resource. The abduction reasoning guarantees that if a solution exists then it is consistent with the access policy and user's active credentials. If a solution set is found it is returned back, else a denial message is returned instead.

The coarse-grained access control process is implemented by using the DLV system [28] as a back-end engine for the deductive and abductive computations, while the fine-grained level is implemented by using only the deductive computation.

5.2 Fine-grained Access: User Property Evaluation

Fine-grained property evaluation supports the application behavioral control process in inferring user properties based on user's profile of active credentials. The definition of user properties is in accordance with the requirements defined by the behavioral control process. In addition, the inference of user properties may be dependent on user behavioral status reflecting a long term resource usage, where user behavioral status is encoded as behavioral credentials.

Behavioral credentials are temporal (one-time-use) credentials that indicate user behavioral status over a service usage. These credentials grant some additional rights to a user and are based on user's past behavior. The Grid resource owner defines the acceptable behavior of applications and, as such, defines the possible values of behavioral credentials to be used at the fine-grained level. Behavioral credentials are internally generated by the Behavioral PDP in accordance with pre-defined rules encoded as part of the behavioral specification. Behavioral credentials are neither managed by third party providers, nor they are defined by users. They do not reflect on the coarse-grained authorization specification, but give meaning only to the fine-grained access control process. Users have indirect impact on behavioral credentials, by the behavior of applications they execute via the GRAM service.

We denote $\mathcal{P}_F$ to be a user property policy defining the acceptable properties considered by the behavioral model, and $\mathcal{C}_\mathcal{B}$ to be a set of behavioral credentials. Both the policy and the properties are defined by the Grid node administrator and exploited on the same node. The credentials granting the properties to the Grid users are local as well, because are generated by the Behavioral

PDP and exploited by the Property PDP on the same Grid node. Since both the policy and credentials are defined and exploited locally, we consider them as trusted.

To query for a property the Behavior PDP specifies $\langle q, \mathcal{C}_\mathcal{B} \rangle$ tuple as an input to the Property PDP, where $q = \texttt{property}(user, property)$ is a query to the policy $\mathcal{P}_F$. The Property PDP first allocates user's profile of active credentials $\mathcal{C}_\mathcal{A}$ by the *user* identifier and then evaluates, if $\mathcal{P}_F$ together with $\mathcal{C}_\mathcal{A}$ and $\mathcal{C}_\mathcal{B}$ entail the requested user property q. The Property PDP returns grant if q is derived else deny message is returned.

6 Negotiation-based Authorization

In this section we outline the negotiation schema [26] underlying the authorization service, and discuss its implementation aspects and integration in the Globus toolkit.

6.1 Negotiation schema overview

We first define the three security policies that clients (Grid users) and servers (a provider of Grid computational service and resources) have:

- $\mathcal{P}_{\mathcal{AR}}$ a policy for protecting opponent's *own* resources based on *foreign* credentials
- $\mathcal{P}_{\mathcal{AC}}$ a policy for protecting opponent's *own* credentials based on *foreign* credentials
- $\mathcal{P}_\mathcal{D}$ a policy for disclosure the need of (missing) *foreign* credentials

Figure 3 shows the negotiation protocol. The protocol runs on both client and server side. The meaning of $\mathcal{C}_\mathcal{A}$, $\mathcal{C}_\mathcal{N}$and $\mathcal{C}_\mathcal{M}$ is read as the set of presented foreign credentials, the set of declined foreign credentials and the set of missing foreign credentials, respectively. We also denote with $\mathcal{O}$ a set of own credentials with respect to a negotiation opponent. We also defined the notion of suspended credential requests to handle the fact that during a negotiation process entities may start to request each other credentials that are already in a negotiation. The set $\mathcal{O}_{neg}$ keeps track of the opponent's own credentials that have been requested and which are still in negotiations. Hence, if a request for a credential already in a negotiation the protocol suspends the request until the respective negotiation thread is finished. When the original thread returns an access decision the protocol resumes all threads awaiting on the requested credential with the decision.

A negotiation process has the following main steps:

1. A client, Alice, sends a service request r and (optionally) a set of credentials $\mathcal{C}_p$ to a server, Bob.
2. Bob's negotiation dispatcher receives the requests, checks if it is a service request and runs the negotiation protocol in a new thread with new negotiation session.
3. When the protocol is run, it updates opponent's set of active credentials with the newly presented ones and checks if the request is already being in a negotiation (steps 1 and 2).
4. If Alice's request is not to be suspended then Bob looks at r and if it is a request for a service he calls for an access decision with his *policy for access to resources* $\mathcal{P}_{\mathcal{AR}}$, his policy for disclosure of foreign credentials $\mathcal{P}_\mathcal{D}$, the set of Alice's active $\mathcal{C}_\mathcal{A}$ and declined $\mathcal{C}_\mathcal{N}$ credentials (step 9).
5. If r is a request for a credential then Bob calls for an access decision with his *policy for access to own credentials* $\mathcal{P}_{\mathcal{AC}}$, his policy for disclosure of foreign credentials $\mathcal{P}_\mathcal{D}$ and Alice's active $\mathcal{C}_\mathcal{A}$ and declined $\mathcal{C}_\mathcal{N}$ credentials (step 11).
6. In the case of computed missing credentials $\mathcal{C}_\mathcal{M}$ (steps 12 and 13) Bob transforms $\mathcal{C}_\mathcal{M}$ into single requests for credentials and awaits until receives all responses (steps 1–5 of `AskCredentials` function). At this point Bob acts as a client, requesting Alice the set of missing credentials. Alice runs the same protocol with swapped roles.
7. When Bob receives all responses, he restarts the loop and consults for a new access decision.
8. When a final decision of grant or deny is taken, the respective response is returned back to Alice.

6.2 Negotiation schema implementation

We adopted a thread-based negotiation of missing credentials by transforming the need of missing credentials into a sequence of single requests each asking for a foreign credential from the missing set. Each request for a credential spurs a new negotiation thread that negotiates access to this credential.

One of the technical issues in the protocol is in the way the server requests missing credentials back to the client. We use the keyword `parfor` for representing that the body of the loop is run each time in a parallel thread. Thus, each missing credential is requested independently from the requests of the others. At that point of the protocol, it is important that each of the finished threads updates presented and declined sets of credentials properly without interfering with other threads. We note that each credential request marks (updates) the requested foreign credential as declined after a session time expires.

The thread based implementation with shared $\mathcal{C}_\mathcal{A}$ and $\mathcal{C}_\mathcal{N}$ is necessary to allow for a polynomial execution time of the trust negotiation protocol with respect to the number of queries to the abduction algorithm. Indeed, without a shared memory of received credentials it is possible to structure policies in a way that a credential would be asked many times. In this way, the protocol queries for credentials are bounded by the number of credentials occurring in the policy $\mathcal{P}_{\mathcal{AC}}$.

Declining a credential in a negotiation process is when an entity is asked for it and the same entity replies to the

Negotiation session: $\mathcal{C}_\mathcal{A}$, $\mathcal{C}_\mathcal{N}$ and $\mathcal{O}_{neg}$. Initialization: $\mathcal{C}_\mathcal{A}$= $\mathcal{C}_\mathcal{N}$= $\mathcal{O}_{neg}$= $\emptyset$.
`NegotiationDispatcher`{
`OnReceiveRequest` $\langle r, \mathcal{C}_p \rangle$ do
1: **if** `isService`(r) **then**
2: `reply` $resp$ = `NegotiationProtocol`(r, $\mathcal{C}_p$); // in a new session thread.
3: **else**
4: `reply` $resp$ = `NegotiationProtocol`(r, $\mathcal{C}_p$); // in a new thread under the original session.
`OnSendRequest` $\langle r, \mathcal{O}_p \rangle$ do
1: $result$ = `invoke NegotiationProtocol`(r, $\mathcal{O}_p$)`@Opponent`; // in a new session thread.
}
`NegotiationProtocol`(r, $\mathcal{C}_p$){
1: $\mathcal{C}_\mathcal{A} = \mathcal{C}_\mathcal{A} \cup \mathcal{C}_p$;
2: **if** $r \in \mathcal{O}_{neg}$ **then**
3: suspend and await for the $result$ on r's negotiation;
4: return $result$ when resumed;
5: **else**
6: $\mathcal{O}_{neg} = \mathcal{O}_{neg} \cup \{r\}$;
7: **repeat**
8: **if** `isService(`r`)` **then**
9: $result$ = `AccessDecisionWithFeedback`(r, $\mathcal{P}_{\mathcal{AR}}$, $\mathcal{P}_\mathcal{D}$, $\mathcal{C}_\mathcal{A}$, $\mathcal{C}_\mathcal{N}$);
10: **else**
11: $result$ = `AccessDecisionWithFeedback`(r, $\mathcal{P}_{\mathcal{AC}}$, $\mathcal{P}_\mathcal{D}$, $\mathcal{C}_\mathcal{A}$, $\mathcal{C}_\mathcal{N}$);
12: **if** $result$ == `ask(`$\mathcal{C}_\mathcal{M}$`)` **then**
13: `AskCredentials(`$\mathcal{C}_\mathcal{M}$`)`;
14: **until** $result$ == `grant` **or** $result$ == `deny`;
15: $\mathcal{O}_{neg} = \mathcal{O}_{neg} \setminus \{r\}$;
16: resume all processes awaiting on r with the $result$ of the negotiation;
17: return $result$;
18: **end if**
}
`AskCredentials(`$\mathcal{C}_\mathcal{M}$`)`{
1: **parfor** each $c \in \mathcal{C}_\mathcal{M}$ **do**
2: $response$ = `invoke` i`AccessNegotiation`(c, $\emptyset$)`@Opponent`;
3: **if** $response$ == `grant` **then** $\mathcal{C}_\mathcal{A} = \mathcal{C}_\mathcal{A} \cup \{c\}$ **else** $\mathcal{C}_\mathcal{N} = \mathcal{C}_\mathcal{N} \cup \{c\}$;
4: **end parfor**
5: **while** $\mathcal{C}_\mathcal{M} \not\subseteq (\mathcal{C}_\mathcal{A} \cup \mathcal{C}_\mathcal{N})$ **do** `wait`();
}

Fig. 3 Negotiation protocol and dispatcher

request with answer deny. When an entity is asked for a credential and there is a counter request for additional credentials then the thread started the original request awaits for the reply and treats the requested credential as not yet released.

The negotiation protocol has been implemented in Java. When a trust negotiation module is initially loaded it internally loads an application server and sets the dispatcher module resident in the memory awaiting on requests. When the server receives a request it automatically redirects the request to the dispatcher which in turn transforms it from raw data to a high-level representation.

The negotiation dispatcher is an essential component to the negotiation protocol. The dispatcher has a role of a *negotiation server* that manages entities requests and their negotiation sessions. Whenever a request arrives the dispatcher runs the negotiation for that request in a new thread that shares same session variables $\mathcal{C}_\mathcal{A}$, $\mathcal{C}_\mathcal{N}$ and $\mathcal{O}_{neg}$ with other threads running under the same negotiation session.

On each received request the dispatcher analyzes the session data from the request and its local database and acts as following. If no session data is specified in the request (and request for a service) then the dispatcher generates new session information ($\mathcal{C}_\mathcal{A}$=$\mathcal{C}_\mathcal{N}$= $\mathcal{O}_{neg}$=$\emptyset$) and runs the negotiation protocol with the new session information. If a session exists and the session data correctly maps to the corresponding one in dispatcher's local database then the dispatcher runs the negotiation protocol in a thread under the existing session. If the specified session does not match to any internal session then deny message is returned back.

Figure 4 shows the architecture and communications of the negotiation module. The architecture logically splits a negotiation process in two levels: negotiation requests and session management; and pure credential negotiation. The division is driven by the goal of making efficient and scalable negotiations for a multi-user environment such as Grid.

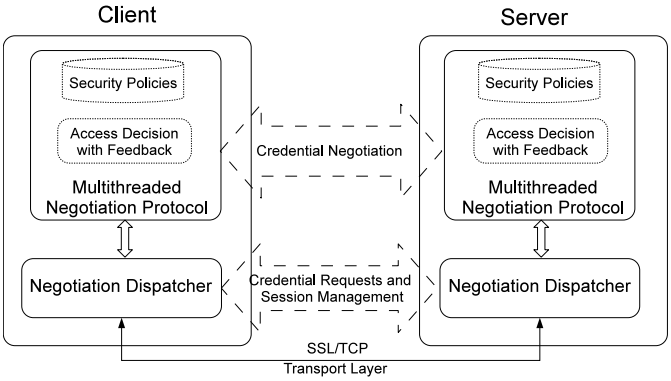

Fig. 4 Negotiation framework message interoperation

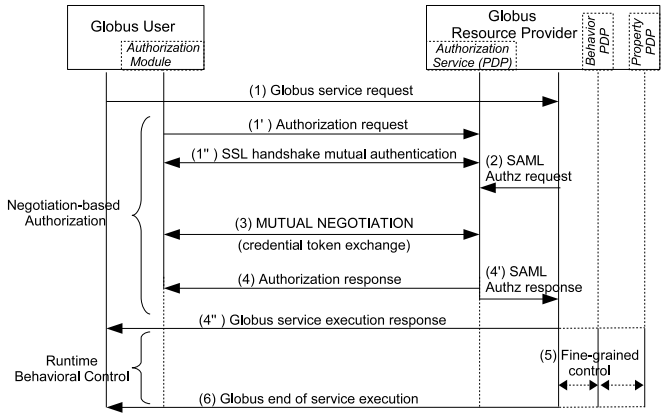

Fig. 5 Negotiation-based authorization and monitoring life cycle

6.3 Negotiation schema integration in Globus

We deployed the negotiation engine (with the credential manager module) in both Globus GRAM client (Grid user) and Globus container (Grid resource provider) to make the negotiation framework interoperable. We exploited standard Globus mechanisms for the seamless integration.

Figure 5 shows the negotiation-based authorization life cycle. It consists of the following steps:

1. A Globus client sends a Globus service request (1) handled by the Globus container. The request triggers an authorization request (1') initiated by the client side and directed to the Globus service provider's authorization service. Once step (1') is done, the service provider's authorization service establishes an SSL channel (mutual authentication) with the client's negotiation engine (1"). Important to note here is that step (1') essentially requests for access rights establishment on a specific computational resource that is to be exploited by the application requested in step (1). The user authorization module is configured to send in step (1') a complete request including the Globus service (GRAM in our case as the user submits a remote application) and the specific computational resource to be exploited.
2. The real trust negotiation starts when the Globus container has processed the application request (in step (1)) and sends a SAML authorization request to the authorization service, step (2). On its side, the authorization service compares the request to those requests received directly from Grid users and starts negotiating with the user that matches to the container's request.
   We note that step (2) is independent form the occurrence of step (1") but step (3) takes place only if steps (1'), (1") and (2) have taken place.
   In case of more than one computational libraries necessary for a client to obtain an authorization for, one can extend the authorization service to handle multiple negotiations by repeating steps (1')–(4) each for a single computational library.
3. The trust negotiation protocol runs over an SSL secure socket connection providing message confidentiality. The SSL connection with mutual authentication bootstraps a negotiation process with initial identity token exchange. Once the negotiation protocol is initialized the client and authorization server negotiate on the resource's requirements.
   The trust negotiation phase includes X.509 certificate tokens exchange, validation and transformation to logic predicates.
4. When the negotiation is over and access decision has been taken, step (4), the authorization service sends back to the Globus container a SAML response with the result of the authorization, step (4'). It is the starting point when the Globus container initiates the service execution, step (4"), and the behavioral application control, step (5).

7 A Grid Usage Scenario

This section presents an example of the security policies introduced so far and a Grid usage scenario.

Assume, a computing center decides to share part of its computational services on a Grid platform to acquire new users. The center offers a set of free java libraries that can be invoked by user applications for efficient computing of mathematical functions, (e.g. the Fast Fourier Transform, Matrix inversion, file format conversion, etc). According to an internal policy of the computing center, these libraries can be used by researchers or students of universities and by members of non profit organizations. At the same time, the computing center also offers a commercial version of the same libraries but used only by users that have paid a given fee, or that belong to a set of associations (e.g., IEEE members).

Moreover, the development staff of the computing center continuously work to improve the performances of the free libraries and the beta versions of the libraries

are tested directly by the computing center's users. Since the beta version of the libraries could include some errors that could, in principle, allow unfamiliar users intentionally or unintentionally to breach a system functionality, the computing center decides to allow only well-behaved users to access beta libraries. The behavior of a user is qualified as well-behaved if the user has exploited the computational services and its underlying resources for a given time interval without any security violations.

Figure 6 shows the coarse-grained security policies of the computing center and a user underpinning our negotiation scenario. The figure presents access, credential, and disclosure policies of the computing center, as well as, credential and disclosure policies of the user in the scenario.

We use a term starting with a capital letter (e.g., User, Rprovider, etc) to refer to a variable that represents any value in its field. The variable is referable from other predicates within a same rule. Terms starting with lower case letters represent constants in a policy (e.g., marioRossi, visaConfirmed, etc).

The coarse-grained access policy of the computing center governs access to a computational service and its underlying resources. Access permissions vary based on what underlying resources a user claims to utilize during his application execution. For example, the invocation of the free version of the java library is granted to senior researchers and PhD students at the University of Malaga, given that they submitted a credential attesting their positions. It also states that access to the beta version of the library is given to those who have access to the free version.

Students and researches at the University of Malaga officially enrolled to IEEE community may utilize facilities of the commercial version of the computational library without fees. Otherwise, to access the commercial version of the library a payment transaction is required. How the payment transaction goes in Grid is out of the scope of our model. We present an access policy governing a disclosure of information needed by the computing center to complete a payment transaction. For instance, a visa card credential could contain information embossed on the user's physical payment card. A ssn (social security number) credential could be asked additionally by the computing center to validate the user's account. In user's turn, the user could request a visa confirmed credential from the computing center to escape frauds and prove that the computing center is eligible to complete payment transactions.

Note, that on the coarse-grained level a user only claims which library his application needs for the execution. Even authorization is done for the claimed library, there is no way on the coarse-grained level to guarantee that the application will actually use this claimed library later on during its execution. However, these issues are managed by the fine-grained enforcement part. Fine-grained level assures that the application calls a library authorized to access on the coarse-grained level. Also, fine-grained level can grant some additional access rights to well-behaved applications. For instance, the coarse-grained access policy contains the same rules to access stable or beta version of the computational libraries. On this level of granularity these libraries are indistinguishable, and only a fine-grained PDP will decide whether to grant or deny access to a specific library during the execution of a user's application. Such integration of two levels of control is the main novelty of our approach.

The credential policy of the computing center allows access to computing center affiliation credential to users that have presented a credential attesting their employee status. Access to the visa confirmed credential of the computing center is not protected and given on request. Although this credential could be additionally protected we omit it simplifying the negotiation scenario.

The disclosure policy discloses the need of all credentials involved in the access policy. Since in the paper we do not focus on negotiation strategies and how to disclose sensitive credentials, we structure the policy as all credentials are disclosable at any request for negotiation.

Looking at the user side, the credential policy controls an access to the social security number and visa card credentials of the user. The policy states that the user allows access to his visa card credential only if a computing center proves his official agreement with Visa Inc. Similarly, the social security number credential is sent during negotiations if the computing center is allowed to collect such information by an appropriate government affiliation. In contrary, the credential stating user's employment record is disclosed freely on demand.

Figure 7 shows fine-grained security policies of the computing center. The behavioral policy defines a constraint on applications using the Grid computational libraries: if a given application has used a function from the free library (stable or beta version), then it cannot use any function from the commercial library, and vice-versa, even if the user that submitted the application holds the required attributes to exploit both libraries.

In particular, the first rule set of the behavioral policy allows access to the free library, the second rule set allows to access the commercial version, while the third rule set allows to access the beta version. There is a variable OF, with initial value false, used to control when the free library is used so that preventing the use of the commercial one. In the same way, the variable OC prevents the use of the free and beta version libraries when the commercial library has been accessed. The forth rule of the behavioral policy allows to read data files and write (result) files in the "/tmp" directory.

The behavioral policy defines fine-grained access control on computational resources based on application behavior and user specific property. The policy defines that access to the libraries is given to users (referred to by their applications) satisfying specific (credential-based)

COMPUTING CENTER SECURITY POLICIES:
Access Policy:
`grant`(User, gramService, create, free_mathlib) ← `cred`(User, studentPhD, universityMalaga).
`grant`(User, gramService, create, free_mathlib) ← `cred`(User, researchSenior, universityMalaga).
`grant`(User, gramService, create, devel_mathlib) ← `grant`(User, gramService, create, free_mathlib).
`grant`(User, gramService, create, comm_mathlib) ← `grant`(User, gramService, create, free_mathlib), `cred`(User, ieeeEnrollment, ieeeInc).
`grant`(User, gramService, create, comm_mathlib) ← `cred`(User, visaCard, bankRoma), `cred`(User, ssn, governmentAuth).
Credential Policy:
`cred`(computerCenter, affiliation, governmentAuth) ← `cred`(User, employee, anEmployer).
`cred`(computerCenter, visaConfirmed, visaEurope) ← .
Disclosure Policy:
`cred`(User, studentPhD, universityMalaga) ← .
`cred`(User, researchSenior, universityMalaga) ← .
`cred`(User, ieeeEnrollment, ieeeInc) ← .
`cred`(User, visaCard, bankRoma) ← .
`cred`(User, ssn, governmentAuth) ← .
`cred`(User, employee, anEmployer) ← .

USER SECURITY POLICIES:
Credential Policy:
`cred`(marioRossi, visaCard, bankRoma) ← `cred`(Rprovider, visaConfirmed, visaEurope).
`cred`(marioRossi, ssn, governmentAuth) ← `cred`(Rprovider, affiliation, governmentAuth).
`cred`(marioRossi, employee, anEmployer) ← .
Disclosure Policy:
`cred`(Rprovider, visaConfirmed, visaEurope) ← .
`cred`(Rprovider, affiliation, governmentAuth) ← .

Fig. 6 Example of security policies for coarse-grained authorization

Behavioral Policy:

SF := {/usr/free/mathlib.jar}
SC := {/usr/comm/mathlib.jar}
SD := {/usr/devel/mathlib.jar}
OF := false
OC := false

[(OC == false),in(x_0,SF),**property(User, non_profit)**,eq(x_1, READ)].open(x_0,x_1,-,fdf)
OF:=true.
i([eq(x_3, fdf)].read(x_3, -, -, -)).
[eq(x_7, fdf)].close(x_7, -)

[(OF == false),in(x_9, SC),**property(User, commercial)**,eq(x_{10}, READ)].open(x_9,x_{10},-,fdc)
OC:=true.
i([eq(x_{12}, fdc)].read(x_{12}, -, -, -)).
[eq(x_{16}, fdc)].close(x_{16}, -)

[(OC == false),in(x_{18}, SD),**property(User, trusted)**,eq(x_{19}, READ)].open(x_{18},x_{19},-,fdd)
OF:= true.
i([eq(x_{22}, fdd)].read(x_{22}, -, -, -)).
[eq(x_{26}, fdd)].close(x_{26}, -)

[in(x_{28}, "/tmp/")].open(x_{28},x_{29},-,fdu)
i([eq(x_{31}, fdu)].read(x_{31}, -, -, -)
or
[eq(x_{32}, fdu)].write(x_{32}, -, -, -)).
[eq(x_{33}, fdu)].close(x_{33}, -)

Property Policy:
`property`(User, non_profit) ← `cred`(User, studentPhD, universityMalaga).
`property`(User, non_profit) ← `cred`(User, researchSenior, universityMalaga).
`property`(User, commercial) ← `property`(User, non_profit), `cred`(User, ieeeEnrollment, ieeeInc).
`property`(User, commercial) ← `cred`(User, visaCard, bankRoma).
`property`(User, trusted) ← `property`(User, non_profit), `behav_cred`(User, well_behaved).
`property`(User, trusted) ← `property`(User, commercial), `behav_cred`(User, well_behaved).

Fig. 7 Example of security policies for fine-grained monitoring

properties such as non_profit, commercial and specific temporal property such as trusted (by behavior of one's application) user. Before opening each of the three libraries, the behavioral policy consults the property policy for external validation if a user satisfies a given property. This validation is done based on the behavior of a user and credentials acquired during a trust negotiation process on the coarse-grained level.

The property policy defines that a non-profit user, according to the high-level credential requirements, is an entity having a credential attesting that it is either a PhD student or a senior researcher at the University of Malaga. A commercial user is defined as being a non-profit user and holding a credential for an IEEE membership. A commercial user is also an entity having a credential for a visa card. We note that the user property definitions on the fine-grained level are implicitly encoded on the coarse-grained level. The explicit encoding on the fine-grained level allows us to control if an application uses the computational resources according to its user's privileges. However, looking at the last definition, a trusted (by behavior) user is defined as any commercial or non-commercial user whose applications have not violated the behavioral policy for a pre-defined time slot, i.e., identified as well-behaved. Here, the behavioral credential for well-behaved user is in accordance to the behavioral logic, and generated by the Behavioral PDP (see section 5.2). The synergy of coarse- and fine-grained credential definitions implies a flexible control process on the fine-grained level.

One would ask if not keeping coarse-grained access and fine-grained property policies in a unified policy. In some scenarios this could be well the case, but however, in a general case, fine-grained property policy defines user specific properties based on a user profile of available credentials mixed with temporal behavioral credentials reflecting the user past behavior. The user property definitions and the behavioral credentials are only relevant to the fine-grained level and have no impact on the coarse-grained level.

In our scenario, the difference of coarse- and fine-grained security control is that on the coarse-grained level the security policies define access to the developer and free version of the computational libraries by the same set of required credentials (do not differentiate them), but the fine-grained security policies refine access to the developer library to those users qualified as trusted (by the behavior of one's application).

8 Prototype Integration in Globus

Our authorization framework was integrated into the Globus Toolkit version 4 (GT4). The default authorization system adopted by GT4 is the GridMapAuthzService. The GridMapAuthzService checks if the Distinguished Name (DN) of a user requesting a service is among a predefined list. However, additionally, Globus allows service providers to define alternative authorization systems that best suit their needs. The Open Grid Forum Authorization working group has defined a standard [52] for plugging in third-party authorization services, that requires the new authorization system to be run as a Globus service and the communication protocol between the authorization service and the Globus container must conform to the SAML protocol of requesting authorization assertion and responding to them (rf. Figure 1).

We integrated our coarse-grained authorization service exploiting this mechanism, called SAMLAuthzCallout, and we created an authorization chain that includes the GridMapAuthzService and our authorization service. The authorization service contains the negotiation engine. Respectively, a Grid user also maintains the authorization module and negotiation engine for mutual trust negotiations. The Globus user loads and initializes the authorization module on request. The negotiation process including network support and messages delivery between the trust negotiation nodes was implemented apart from the Globus transportation channels and uses SSL socket connection. Message exchanges in the protocol were custom-defined and optimized for efficient message delivery.

To implement the fine-grained controls, we integrated some components of our authorization system within the GRAM service. First we extended the standard Globus job description schema to define a new job type, the `java` one. In this way, to execute a Java application the Grid user specifies `java` as a job type in its job request. Second, we defined an alternative scheduling system that in case of a Java application executes it on our customized JVM, and we configured the Managed Job Service (MJS) component of the GRAM service to invoke this scheduler instead of the standard one. The standard error stream is used in order to return to Grid users error messages when an application has been stopped because of a security policy violation. The standard Globus mechanism to transfer files, i.e. GridFTP, is exploited to send to the remote Grid user the log file with the error description.

The Behavioral PDP interacts with the GRAM service in order to get the job request submitted by the remote Grid user. This interaction is simply implemented through an XML file that is generated by the GRAM service before invoking the security enhanced JVM. The file name is passed to the JVM as an input parameter and the JVM, in turn, passes it to the Behavioral PDP. The Behavioral PDP reads from this file the resource requirements in the job request which will be enforced during execution. The Behavioral PDP also imports from Globus the user DN extracted from the proxy certificate that has been submitted by the user.

The Behavioral PDP gets from the GRAM service the job request submitted by the remote Grid user. This interaction is simply implemented by the new scheduler

that invokes the security enhanced JVM passing the job request as an input parameter and the JVM, in turn, passes it to the Behavioral PDP. The Behavioral PDP reads from the job request the resource requirements (e.g. max execution time or max memory) that will be enforced during the application execution. The Behavioral PDP also imports from the job request the user DN.

The Behavioral PDP is activated by the JVM every time a Java application wants to perform an interaction with the underlying resource. This has been implemented by modifying the implementation of the Java core classes, i.e. the classes that manage the interactions with the underlying resource. In particular, these classes have been modified by substituting the invocations to the system libraries functions which perform the security relevant system calls with invocation to a proper set of wrapper functions we defined. The execution of external code through the Java Native Interface (JNI) [32] has been disabled, since it allows interactions with underlying resource through external libraries (e.g., non java-based ones). A wrapper function first activates the Behavioral PDP to perform the security checks before the execution of the security relevant system call and suspends itself (i.e. suspends the JVM). Then, after the Behavioral PDP reactivates the wrapper function, the last enforces the Behavioral PDP decision by either actually invoking the system library function or by interrupting the application execution. When the system library function has been executed the wrapper function activates the Behavioral PDP again to perform security checks after the system call execution. The interactions between the Behavioral PDP and the JVM are implemented by using semaphores and shared variables.

The Behavioral PDP invokes the Property PDP as an external library function. The interactions between the two components have a critical impact on the monitoring performance. Since the Property PDP has been implemented in Java but the Behavioral PDP in C we invoke the Property PDP without loading the JVM every time. We use the Java Native Interface (JNI) library that allows us to invoke Java methods from inside a C code. As a part of the functionality of this library, we have a special command that explicitly loads the JVM when needed for a Java class execution, and also commands to obtain links to a Java class and its methods (in our case the Property PDP main class). In this way, at initialization time, the Behavioral PDP loads the JVM, creates links to the the Property PDP methods, and keeps them in memory for all subsequent invocations. When the Java application, being monitored, terminates the Behavioral PDP releases the JVM and then terminates.

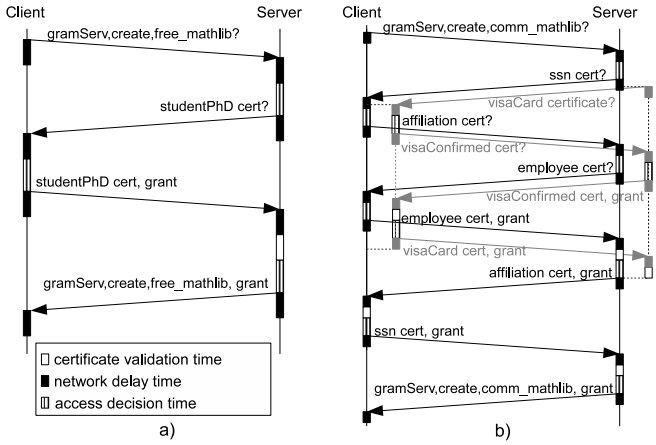

Fig. 8 Grid usage trust negotiation scenarios

9 Prototype Performance Evaluation

The section outlines the experiments we performed to evaluate the impact on the adoption of our framework. We divide the experimental results in two parts: coarse-grained and fine-grained time performance. We followed the Grid usage scenario. All experimental trials were executed on Pentium 4 with 2.8GHz and 1GB RAM running Linux.

9.1 Coarse-grained Performance

We tested the negotiations overhead on coarse-grained level based on the scenario presented in Section 7 and by evaluating it with: (i) increased number of presented credentials, (ii) increased number of negotiated (disclosable) credentials and (iii) simulated two extreme cases of negotiation strategies giving us the boundaries of overall possible negotiation timing.

We will first illustrate typical negotiation exchanges and their time consumption against network delay, certificate validation and access decision. Figure 8 shows two negotiation processes based on the presented Grid usage scenario.

We divide a trust negotiation overhead into (relatively independent) time cases:

- network delay for data transmission over SSL channel.
- certificate validation, verification and conversion to data structures suitable for logic access decision. Certificate tokens are encoded as X.509 certificates.
- logic access decision evaluation based on security policies.

Figure 8(a) presents a simple negotiation scenario with just few message exchanges. The client (Grid User) requests to obtain access rights to the free_mathlib library and the server (Computing Center) requires a PhD

student credential to grant access. The table below shows the negotiation time (in milliseconds) of the different cases.

network delay	83.2
certificate validation	14.0
access decision	101.5
overall time	287.0

The overall time is counted from the point when a client initiated a request till the time the server replied with a grant message. This time also includes the time for user profile management such as profile generation, update and deletion.

The logic access decision engine was called for access decision evaluation three times - once on the client side and twice on the server side. The submitted certificate was validated once on the server side only.

We count overall negotiation time of all steps done by both server and client sides. However, to calculate time of concurrent processes we projected all processes on one time line in order to accurately deal with overlapping time. For instance, for two concurrent processes we update the overall time by the difference from the earliest process started till the latest that ended.

Figure 8(b) shows a negotiation case with two concurrent negotiations (in concurrent threads), where one of them is drawn in gray. The table below summarizes the measured time performance (in milliseconds).

network delay	182.1
certificate validation	39.0
access decision	380.1
overall time	707.6

In this scenario, the logic access engine was called 10 times, 5 on each side, and the client validated 2 certificates while the server 3.

In either cases, the certificates validation and verification had less time consuming part than the network delay. The conclusion from the first part of the experimentation is that the overall negotiation time is more sensitive to network delay time than to certificate validation. The more interactions during a negotiation the more influence the network delay will have on the overall negotiation. Absolute value of network delay time depends also on the size of the data to be processed. There was no direct way to manage and reduce this value for the given scenario.

The access decision time had the most impact on the overall authorization process. Looking at Figure 2, the access decision depends on two main computations: deduction part (based on the access policy and client's set of credentials) and abduction part (based on the access policy, client's set of credentials and the set of disclosable credentials/hypotheses). We performed two sets of experiments to analyze the access decision behavior with respect to the two logic computations. We used the Grid usage scenario in Figure 8(b) and its security policies (rf. Figure 6) for the experiments.

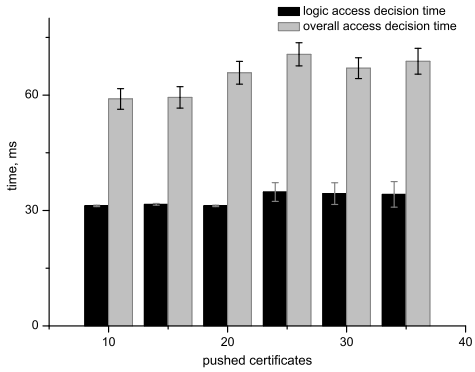

Fig. 9 Authorization service performance with increased number of pushed credentials

First set of experiments focused on measuring access decision time of deduction computation versus an increased number of pushed certificates to be verified, validated and transformed to logic facts. Technically, a client pushes *visaCard* and *ssn* certificates along with other artificially generated certificates to get access to the computational library. For this purpose we generated additional X.509 certificates that in combination with the two from the scenario formed the different tests. Figure 9 shows the access decision performance measured in milliseconds.

The server invoked the logic access engine only once on each trial and replied with grant decision. The average time for the logic access decision almost remained the same while the number of pushed certificates was increased from 10 to 35. The average time was calculated based on 10 repeating measurements on each trial. The measurement error is presented by a cursor arrow on the top of the columns.

The conclusion from the experimentation was that the logic access decision took approximately half of the overall time and did not change during the trials. While, the certificate validation and network delay time (included in the overall decision time) increased during the trials and equally influenced to the overall performance.

Second set of experiments was focused on the pull model where the server determines what credentials are required to establish trust. Here, the disclosure policy determines what credentials are disclosable (available) so that abduction reasoning finds those that are necessary to grant a service request.

We used the scenario of Figure 8(b) but this time we modified the disclosure policy (by adding new logic rules) on the server side so that the set of disclosable credentials feed to the abduction reasoning increased on each trial.

Figure 10 shows the set of measurements performed. On each trial the server computed *visaCard* and *ssn* credentials as missing credentials returned to the client. The

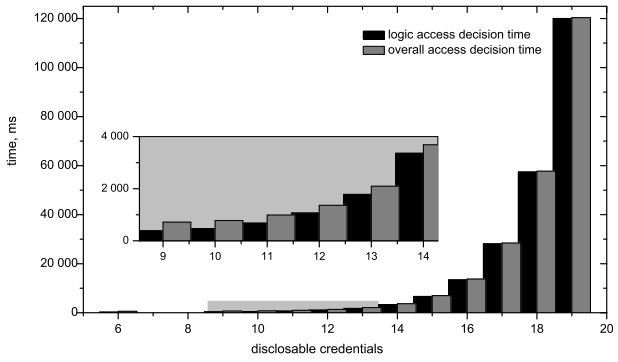

Fig. 10 Authorization service performance dependence on the disclosure policy

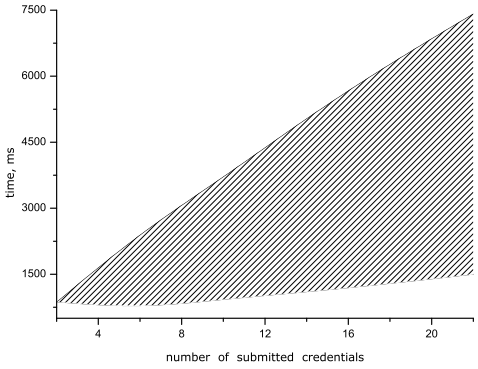

Fig. 11 Time bounds for trust negotiations

access decision time on every trial was very sensitive and varied considerably to the number of disclosable credentials (i.e., a number of logic rules in the server's disclosure policy). The performed tests measured the logic access decision time versus overall decision time.

Abduction reasoning time grew exponentially with increasing disclosable credentials from 9 to 19. With 19 hypotheses (disclosable credentials) the logic engine took approximately 2 minutes (99.7% of the overall time) to compute the missing credentials. Here, the impact of network delay and certificate validation onto the overall time became completely negligible. We expect at around 12 disclosable credentials as a reasonable threshold (approximately a second) for a decision. However, such a limitation also depends on possible negotiation strategies and negotiation session validity. We refer the reader to [11] for in deep analysis and results on abduction problems and complexity.

Next and last set of experiments we performed was on the time performance of the negotiation protocol with respect to possible negotiation strategies in terms of number of client-server interactions.

The sequence of credential exchange during a negotiation process is controlled by negotiation strategies [54, 55]. One can adopt variety of negotiation strategies and privacy settings [59, 42, 60, 7, 48, 5] depending on credential sensitivity, on familiarity with the opponent or domain/environment, type of resources being negotiated upon etc.

The proposed negotiation protocol serves as a policy enforcement engine over the access and disclosure policies of an opponent, i.e. the protocol is data-driven by the deduction and abduction reasoning. In that sense, on top of the missing credentials computed one can additionally impose a strategy controlling the disclosure of these credentials. Here we note that such a strategy could be encoded directly in the disclosure policy (in [26] the authors define a stepwise reasoning on the disclosure policy structure) so that the protocol enforces the strategy directly. We also note that multiple disclosure policies can be defined for a given access policy thus encoding different possible strategies.

Since strategies depend on multiple factors and may take different sequences of credential exchange, so the goal we approach is to analyze the time boundaries the negotiation protocol performs by abstracting from a specific strategy. Essentially, we wanted to examine the protocol behavior on two extreme negotiation modes with the possibility of concurrent credential negotiations.

Figure 11 shows the time area resulted from our experimentations where all negotiation strategies, our prototype can perform, fall into.

The upper time bound, named as *mutual suspicious mode*, is implied by a stepwise disclosure of credentials where each entity discloses missing credentials consecutively one after the other. Thus, one credential by the server side and one by the client side in response, where the next credential is requested (negotiated) if the previous negotiation has been completed. The negotiation process is run in a single system thread on a client side and on a server side without any concurrent requests.

We assume the following negotiation scenario of this mode. A client and a server have the same number of 11 credentials protecting sensitive resources. We chose 11 credentials to obtain a reasonable abduction computation close to the estimated threshold noted above. The client requests for a resource with no input of credentials. The server computes number N of missing credentials ($N \leq 11$) and negotiates on them in suspicious mode. For each of the missing credential the client has a counter request with a credential to the server side. The server grants the requested credential and the client, on its turn, grants the respective credential to the server.

In this way, for each of the server's credential we model a negotiation round where the client runs abduction reasoning to find a missing credential and successfully negotiates on it. The negotiation scenario has $N+1$ invocation of abduction reasoning over fixed number of 11 hypotheses in any test.

The opposite to suspicious mode, is the *mutual greedy mode* of negotiations. This mode results in the bottom time bound in the figure. The minimum time for negotiation is determined by the strategy that implies complete disclosure of missing credentials in one round by client and by server side. Greedy mode utilizes multi-threaded concurrent exchange of missing credentials on each side.

To obtain the minimum negotiation time, we modify the above scenario with the assumption that the client and the server have recently been in contact for that resource and, being in greedy mode, the client along with the request for the resource runs N requests for the credentials necessary to grant server's N missing credentials. On receiving request for a resource, the server runs N threads requesting the missing credentials. In this scenario, we avoid client running abduction reasoning but only deduction to check if server's requests are granted by the already run client's requests.

In the greedy mode, the client and server run $2N$ system threads, while in the suspicious mode they run only 2 system threads. In greedy mode we loose in memory but profit in execution time.

Figure 11 illustrates how with increasing $1 \leq N \leq 11$ the two extreme modes perform in time. All possible negotiations lie in the hatched area. On x-axis we show the total number of credentials necessary to complete a negotiation process, where half of the credentials are requested by the server and the other half by the client side. With total of 22 credentials (the server and client ask each other for 11 credentials) any negotiation strategy would remain within 1.5 to 7.5 seconds. With 8 credentials the overall negotiation time is bound by 0.9 and 3.7 seconds.

In the following we briefly relate our results with those reported for the Traust service scenario [27, pp 14–16]. In the Traust scenario there are two subsequent negotiation sessions: one to disclose the request to access an information portal and one to access the portal as a rescue dog handler. The authors report an average execution time of the whole scenario to 4.04 seconds with total of nine disclosed credentials. In our case, the scenario in Figure 8(a) approximates the interactions of the first negotiation of the Traust scenario, while the mutual suspicious mode with total of 8 credentials approximates (upper bound) the message exchanges of the second negotiation phase. Our average time (sum of the two negotiations) to simulate the scenario would be an average of 4 seconds. Although, we experimented with slightly faster CPUs the obtained results shown that our system performs comparably to that of Traust.

Summary of experiments. We evaluated the overhead of trust negotiations on the coarse-grained level against network delay, credential verification and access decision time split in deduction and abduction reasoning. The prototype implementation shown reasonable and affordable time performance. The protocol has more sensitivity to network delay than to certificate validation. Deduction reasoning as part of the logic access decision has more influence on the overall negotiation time than network delay and certificate validation. The prototype performance is foremost sensible to the number of disclosable credentials due to the abduction reasoning used as part of the logic access decision.

To mitigate the exponential time growth of the computation of missing credentials, one can divide and specify a single disclosure policy per resource so that the system dynamically loads the relevant disclosure policy on request.

An interesting idea in Bertino et al. [7] work, that could be well used in our settings, is the use of trust tickets. A trust ticket keeps information on recent successful negotiations of credential exchange for a given resource. Thus, if a trust ticket is introduced initially (during an introductory phase) for the same resource then the negotiation process can be speeded up, i.e. omitted negotiations on those requirements indicated in the trust ticket.

9.2 Fine-grained Performance

The security checks performed by the fine-grained monitoring support introduce an overhead in the application execution time. The main factor concerning the execution overhead is if a java application performs more computational operations with only few system calls – case where the overhead of security monitoring is negligible with respect to the overall execution time, or if an application mainly performs system calls – case where security checks impacts on the overall time.

The test presented in this section evaluates the execution time of a Java application by adopting a standard JVM and a security enhanced JVM with the Behavioral and Property PDPs. In particular, we adopted the Jikes RVM Java Virtual Machine developed by the IBM T. J. Watson Research Center [2] run on Linux operating system. This JVM has been modified to embed our security support.

The test scenario is the one described in Section 7, where the Java application submitted by the Grid user exploits an utility library. In this case, the utility library consists of a collection of benchmarks that belong to the Ashes Suite Collection benchmarks[3]. In particular, user's remote application uses a library to convert an mp3 file to a wav one. Both, the Grid user's application and the utility library, were monitored by our system. System calls defined by the policy are (rf. Figure 7): read the library, read an input file (in mp3 format), and write a result file (in wav format). This application is well-suited for our test because it performs a large number of security relevant system calls during its execution: around 1500 calls for about 5 seconds.

The fine-grained system controls if a user submitted the application has a proper set of credentials to open the

[3] http://www.sable.mcgill.ca/ashes

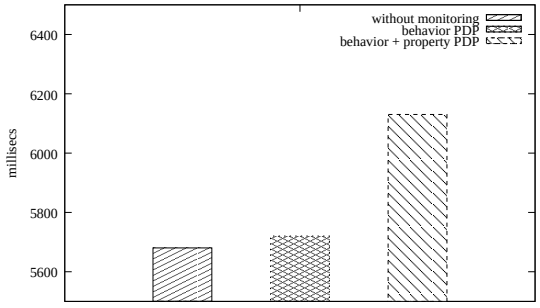

Fig. 12 Fine-grained monitoring performance

requested utility library, and if access to the library and to the data files is according to the admitted behavior.

Figure 12 shows the results of the evaluation as an average of ten trials. The average execution time of the library without any security control is 5680 milliseconds, while the execution time with the enforcement of the behavioral policy only is 5720 milliseconds, and the execution time with the enforcement of both the behavioral and the user property policies is 6130 milliseconds. The Behavioral PDP has been invoked 1489 times, while the Property PDP has been invoked only 1 time - when the application opened the utility library file. The overhead introduced by the Behavioral PDP is about 1%, while the overhead by the Behavioral and Property PDPs is about 8% of the application execution time.

The results shown the practical aspects of the Behavioral PDP while the Property PDP shown that the system overhead is sensitive against logic access decisions at this level. A possible solution to mitigate the overhead is to encapsulate the behavioral and property policies, and refine a common policy framework and its evaluation that handles both sides requirements.

10 System Architecture Security Analysis

This Section discusses the security aspects of the system architecture and its components. The purpose of the analysis is to show, in a rather informal way, what potential treats the system mitigates, and how its architectural components provide security and trust to the overall framework.

First, we discuss the coarse-grained level security. Globus container and trust negotiation engine are the components of the framework that face external interactions (from outside). Globus container is accessed by Grid user to submit applications for execution. The security between Globus container and Grid User platform is discussed in Globus specification [14]. In this paper, we assume that this communication channel is secured by means of [14] to provide confidentiality and integrity of authorization requests and authorization decisions between Globus container and Grid user. Trust negotiation engine, instead, is accessed by trust negotiation client to perform credentials exchange and negotiations. In this case, security is achieved through the use of the SSL protocol. Grid user and resource provider authenticate each other and establish SSL connection using certified public keys issued by trusted authorities. We assume that communications over SSL are sufficiently secure for our model. Thus, Grid user and resource provider achieve confidentiality and integrity of authorization messages and credentials transmitted during trust negotiations. Credentials encode attributes of peers, and are digitally signed by trusted authorities.

Credentials are compliant with X.509 (v3) attribute certificate standard. All credentials are associated with a particular identity (public key) presented for the SSL handshake to proof their ownership, and vanish fake credentials. We have addressed a verification issue of certificates that may expire during a negotiation process. During a negotiation process some of the already presented credentials may expire before the negotiation is completed. To deal with that, the credential manager (refer to Section 3) keeps two sets of user profiles: raw X.509 certificates, and their logic based equivalent. The former one is re-evaluated on any negotiation step to be performed (i.e., before an access decision is taken), so that if any of the presented certificates expires the negotiation engine aborts the negotiation process.

Unfortunately, trust negotiation might become a subject of denial-of-the-service attacks since it is a relatively heavy computational process (due to its design nature). Anyway, one could define a short enough bounded session time for a round of negotiations to mitigate this kind of attacks. When the session time expires, the session automatically is terminated. Finally, for the coarse-grained level we need secure communications between our Authorization service and the Globus container. In the current implementation, we define these components to reside on a same node and, consequently, confidentiality and integrity of access requests and authorization decisions are preserved. In case where the authorization service resides on a remote node, security for the interactions between the Globus container and the authorization service can be achieved through the standard GSI mechanisms.

On the fine-grained level, all security threats (e.g. to subvert system functionality and block up computational resources) come from applications executed by the computational service. An application executed on a Grid platform can not perform dangerous or forbidden operations on platform's resources, since every action performed by the application is intercepted and checked before the actual execution. Moreover, the application cannot bypass the monitoring mechanism, because the Behavioral PEP was integrated inside the JVM, and the execution of the application is completely mediated by the JVM. Additionally, we disabled the Java Native Interface support for executing arbitrary code from a Java application.

We also emphasize on the importance of trust between policy decision and enforcement components on the fine-grained level. We define that the Property PDP and the Behavioral PDP components (evaluating and enforcing fine-grained security policies) are settled on a same platform and are not accessible from outside. Thus, interactions between them is considered as trusted, and confidentiality and integrity of access requests and authorization decisions are guaranteed. Finally, integrity of data used on, both, coarse- and fine-grained levels, such as policies, user profiles, credentials, is preserved by placing them into a secure local storage (not accessible by ordinary applications).

11 Conclusions and Future Work

We have presented a system for enhancing Grid security by integrating a negotiation-based authorization service for dynamic access rights establishment with a resource monitoring and enforcement service for fine-grained application behavioral control. We have also presented system architecture, functional description, implementation and performance evaluation. The implementation and experimental results shown the scope and practical aspects of the system. Out of the evaluation results, we have concluded the following main issues:

- the importance of well-designed coarse-grained access and disclosure policies for efficient negotiation. The main factor (posing an upper-bound) is the number of potentially disclosable credentials for a given service.
- the importance of well-integrated behavioral and property policies for fine-grained enforcement. The main factor here is the intensity of performed system calls and (user) property evaluation.

A prerequisite for adoption of the system is the definition of high-level tools for policy specification and automated translation to their formal representation. We refer the reader to ASP RuleML[4] for an approach that adapts high-level RuleML[5] language to express answer-set programs and its automated transformation to logic programs. One can define coarse-grained security policies in RuleML language with the help of GUI for XML editing and generation, and perform an automated transformation to their respective logic program format. Also this tool can be used for defining and integrating the property policy requirements with those of behavioral policy model.

An alternative solution to the policy specification problem could be the adoption of the eXtensible Access Control Markup Language (XACML) [58] as a policy language. XACML is a widely used standard for defining authorization policies. It is well-known by system administrators, and supported by a large community of users. There are available tools supporting system administrators in writing security policies, such as the graphical policy editor UMU-XACML-Editor[6].

We also point out the importance of a model-driven approach for policy transformation from high-level definitions to fine-grained policy specification (refinement), for example [44].

Future work has several potential directions. One direction is to apply the prototype implementation on real test-beds where one can evaluate its performance on practical case studies, including real-world Grid security policies and fine-grained monitoring requirements. Second direction is to apply the system model to the domain of Virtual Organizations. Re-designing the fine-grained behavioral and property policies into one unified policy framework for improving its evaluation performance. Defining suitable semantic characterization of a new policy framework allowing for effective policy and trust management by both platform owners and third-party providers (VO partners). Third direction is to research on how the coarse-grained authorization can be generalized to handle interoperability of negotiations with the other negotiation systems in Grid such as TrustBuilder [56], Trust-X [7], or PeerTrust [37].

[4] http://www.kr.tuwien.ac.at/staff/roman/asprulemL

[5] http://www.ruleml.org

[6] http://xacml.dif.um.es

References

1. Alfieri, R., Cecchini, R., Ciaschini, V., dell'Agnello, L., Frohner, A., Lőrentey, K., Spataro, F.: From gridmap-file to voms: managing authorization in a grid environment. Future Gener. Comput. Syst. **21**(4), 549–558 (2005).
2. Alpern, B., Attanasio, C., Barton, J., et al.: The jalapeño virtual machine. IBM System Journal **39**(1), 211–221 (2000)
3. Apt, K.: Logic programming. In: J. van Leeuwen (ed.) Handbook of Theoretical Computer Science. Elsevier (1990)
4. Barton, T., Basney, J., Freeman, T., Scavo, T., Siebenlist, F., Welch, V., Ananthakrishnan, R., Baker, B., Goode, M., Keahey, K.: Identity federation and attribute-based authorization through the globus toolkit, shibboleth, gridshib, and myproxy. In: 5th Annual PKI R&D Workshop (2006)
5. Baselice, S., Bonatti, P.A., Faella, M.: On interoperable trust negotiation strategies. In: Proceedings of IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY'07), pp. 39–50. IEEE Computer Society (2007)
6. Becker, M.Y., Nanz, S.: The role of abduction in declarative authorization policies. In: Proceedings of the 10th International Symposium on Practical Aspects of Declarative Languages (PADL'08), LNCS. Springer (2008)
7. Bertino, E., Ferrari, E., Squicciarini, A.C.: Trust-X: A peer-to-peer framework for trust establishment. IEEE Transactions on Knowledge and Data Engineering **16**(7), 827–842 (2004)
8. Chadwick, D.W., Otenko, A.: The PERMIS X.509 role-based privilege management infrastructure. In: Seventh ACM Symposium on Access Control Models and Technologies, pp. 135–140. ACM Press (2002).

9. Chervenak, A., Foster, I., Kesselman, C., Salisbury, C., Tuecke, S.: The data grid: Towards an architecture for the distributed management and analysis of large scientific datasets. Journal of Network and Computer Applications **23**, 187–200 (2001)
10. Constandache, I., Olmedilla, D., Siebenlist, F.: Policy-driven negotiation for authorization in the grid. In: Proceedings of the Eighth IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY '07), pp. 211–220. IEEE Computer Society (2007).
11. Eiter, T., Gottlob, G., Leone, N.: Abduction from logic programs: Semantics and complexity. Theoretical Computer Science **189**(1-2), 129–177 (1997)
12. Fang, L., Gannon, D., Siebenlist, F.: XPOLA: An extensible capability-based authorization infrastructure for grids. In: Forth Annual PKI Workshop: Multiple Paths to Trust. NIST (2005)
13. Feller, M., Foster, I., , Martin, S.: Gt4 gram: A functionality and performance study. In: Proceedings of the Teragrid 2007 Conference. Madison, WI, USA (2007)
14. Foster, I.: Globus toolkit version 4: Software for service-oriented systems. In: Proceedings of IFIP International Conference on Network and Parallel Computing, pp. 2–13. Springer-Verlag, LNCS 3779 (2005)
15. Foster, I., Kesselman, C.: The Grid: Blueprint for a Future Computing Infrastructure, chap. Computational Grids. Morgan Kaufmann (1998)
16. Foster, I., Kesselman, C., Pearlman, L., Tuecke, S., Welch, V.: A community authorization service for group collaboration. In: Proceedings of the 3rd IEEE Int. Workshop on Policies for Distributed Systems and Networks (POLICY 02), pp. 50–59 (2002)
17. Foster, I., Kesselman, C., Tsudik, G., Tuecke, S.: A security architecture for computational grids. In: Proceedings of the 5th ACM conference on Computer and communications security (CCS'98), pp. 83–92. ACM (1998)
18. Hoare, C.A.R.: Communicating sequential processes. Communications of the ACM **21**(8), 666–677 (1978). DOI http://doi.acm.org/10.1145/359576.359585
19. Hofmeyr, S.A., Somayaji, A., Forrest, S.: Intrusion detection using sequences of system calls pp. 151–180 (1998)
20. Kapadia, A., Sampemane, G., Campbell, R.H.: KNOW why your access was denied: regulating feedback for usable security. In: Proceedings of the 11th ACM conference on Computer and Communications Security, pp. 52–61. ACM Press, New York, NY, USA (2004)
21. Keahey, K., Welch, V.: Fine-grain authorization for resource management in the grid environment. In: GRID '02: Proceedings of the Third International Workshop on Grid Computing - LNCS, vol. 2536, pp. 199–206 (2002)
22. Keahey, K., Welch, V., Lang, S., Liu, B., Meder, S.: Fine-grained authorization for job execution in the grid: design and implementation: Research articles. Concurr. Comput. : Pract. Exper. **16**(5), 477–488 (2004).
23. Koshutanski, H., Martinelli, F., Mori, P., Borz, L., Vaccarelli, A.: A fine-grained and X.509-based access control system for Globus. In: Proceedings of the International Symposium on Grid computing, high-performAnce and Distributed Applications (GADA'06). Springer-Verlag press, Montpellier, France (2006)
24. Koshutanski, H., Martinelli, F., Mori, P., Vaccarelli, A.: Fine-grained and history-based access control with trust management for autonomic grid services. In: Proceedings of the 2nd International Conference on Autonomic and Autonomous Systems (ICAS'06). IEEE Computer Society, Silicon Valley, California (2006)
25. Koshutanski, H., Massacci, F.: Interactive access control for autonomic systems: from theory to implementation. ACM Transactions on Autonomous and Adaptive Systems (TAAS) (to appear)
26. Koshutanski, H., Massacci, F.: A negotiation scheme for access rights establishment in autonomic communication. Journal of Network and System Management (JNSM) **15**(1) (2007)
27. Lee, A.J., Winslett, M., Basney, J., Welch, V.: The traust authorization service. ACM Transactions on Information and System Security (TISSEC) **11**(1), 1–33 (2008).
28. Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., Scarcello, F.: The dlv system for knowledge representation and reasoning. ACM Transactions on Computational Logic (2006). Available via url http://www.arxiv.org/ps/cs.AI/0211004
29. Lepro, R.: Cardea: Dynamic access control in distributed systems. In: NAS Technical Report NAS-03-020. NASA Advanced Supercomputing (NAS) Division (2003)
30. Li, J., Cordes, D.: A scalable authorization approach for the globus grid system. Future Gener. Comput. Syst. **21**(2), 291–301 (2005).
31. Li, N., Mitchell, J.C., Winsborough, W.H.: Design of a role-based trust-management framework. In: Proceedings of the 2002 IEEE Symposium on Security and Privacy, pp. 114–130. IEEE Computer Society (2002)
32. Liang, S.: Java(TM) Native Interface: Programmer's Guide and Specification. Addison-Wesley (1999)
33. Lorch, M., Adams, D.B., Kafura, D., Koneni, M.S.R., Rathi, A., Shah, S.: The PRIMA system for privilege management, authorization and enforcement in grid environments. In: Proceedings of the Fourth International Workshop on Grid Computing, p. 109. IEEE Computer Society (2003)
34. Martinelli, F.: Towards an integrated formal analysis for security and trust. In: FMOODS, pp. 115–130 (2005)
35. Martinelli, F., Mori, P., Vaccarelli, A.: Towards continuous usage control on grid computational services. In: Proceedings of Joint International Conference on Autonomic and Autonomous Systems and International Conference on Networking and Services (ICAS-ICNS 2005), IEEE Computer Society, p. 82 (2005)
36. Nefedova, V., Jacob, R., Foster, I., Liu, Z., Liu, Y., Deelman, E., Mehta, G., Su, M.H., Vahi, K.: Automating climate science: Large ensemble simulations on the TeraGrid with the GriPhyN virtual data system. In: Proceedings of the Second IEEE International Conference on e-Science and Grid Computing (E-SCIENCE'06), p. 32. IEEE Computer Society (2006).
37. Nejdl, W., Olmedilla, D., Winslett, M.: PeerTrust: Automated trust negotiation for peers on the semantic web. In: VLDB Workshop on Secure Data Management (SDM), *Lecture Notes in Computer Science*, vol. 3178, pp. 118–132. Springer (2004)
38. Pearlman, L., Kesselman, C., Welch, V., Foster, I., Tuecke, S.: The community authorization service: Status and future. Proceedings of Computing in High Energy and Nuclear Physics (CHEP 03): ECONF **C0303241** (2003)
39. Provos, N.: Improving host security with system call policies. In: SSYM'03: Proceedings of the 12th conference on USENIX Security Symposium, pp. 257–272. USENIX Association, Berkeley, CA, USA (2003)
40. Randall, D.A., Ringler, T.D., Heikes, R.P., Jones, P., Baumgardner, J.: Climate modeling with spherical geodesic grids. Computing in Science and Engineering **4**(5), 32–41 (2002)
41. Saltzer, J.H., Schroeder, M.D.: The protection of information in computer systems. Proceedings of the IEEE **63**(9), 1278–1308 (1975)
42. Seamons, K., Winslett, M., Yu, T.: Limiting the disclosure of access control policies during automated trust negotiation. In: Proceedings of the Network and Distributed System Security Symposium (2001)

43. Seamons, K., Winslett, M., Yu, T., Smith, B., Child, E., Jacobson, J., Mills, H., Yu, L.: Requirements for policy languages for trust negotiation. In: Proceedings of the 3rd International Workshop on Policies for Distributed Systems and Networks (POLICY'02), pp. 68–79. IEEE Computer Society (2002)
44. Seehusen, F., Stølen, K.: A transformational approach to facilitate monitoring of high-level policies. In: 9th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY 2008), pp. 70–73. IEEE Computer Society (2008)
45. Sekar, R., Bowen, T., Segal, M.: On preventing intrusions by process behavior monitoring. In: ID'99: Proceedings of the 1st conference on Workshop on Intrusion Detection and Network Monitoring, pp. 29–40. USENIX Association, Berkeley, CA, USA (1999)
46. Shanahan, M.: Prediction is deduction but explanation is abduction. In: Proceedings of IJCAI'89, pp. 1055–1060. Morgan Kaufmann (1989)
47. Spencer Jr. B. et al.: Neesgrid: A distributed collaboratory for advanced earthquake engineering experiment and simulation. In: 13th World Conf. on Earthquake Engineering (2004)
48. Squicciarini, A., Bertino, E., Ferrari, E., Paci, F., Thuraisingham, B.: PP-trust-X: A system for privacy preserving trust negotiations. ACM Trans. Inf. Syst. Secur. **10**(3), 12 (2007).
49. Stell, A.J., Sinnott, R.O., Watt, J.P.: Comparison of advanced authorisation infrastructures for grid computing. In: Proceedings of High Performance Computing System and Applications 2005, HPCS, pp. 195–201 (2005)
50. Thompson, M., Essiari, A., Keahey, K., Welch, V., Lang, S., Liu, B.: Fine-grained authorization for job and resource management using akenti and the globus toolkit. In: Proceedings of Computing in High Energy and Nuclear Physics (CHEP03) (2003)
51. Thompson, M., Johnston, W., Mudumbai, S., Hoo, G., Jackson, K., Essiari, A.: Certificate-based access control for widely distributed resources. In: Proceedings of Eighth USENIX Security Symposium (Security'99), pp. 215–228 (1999)
52. Welch, V., Ananthakrishnan, R., Siebenlist, F., Chadwick, D., Meder, S., Pearlman, L.: Use of SAML for OGSI Authorization. Global Grid Forum, Open Grid Services Architecture Authorization Working Group (2005). http://forge.gridforum.org/projects/ogsa-authz
53. Welch, V., Siebenlist, F., Foster, I., Bresnahan, J., Czajkowski, K., Gawor, J., Kesselman, C., Meder, S., Pearlman, L., Tuecke, S.: Security for grid services. In: 12th IEEE International Symp. on High Performance Distributed Computing (2003)
54. Winsborough, W., Seamons, K., Jones, V.: Automated trust negotiation. In: Proceedings of DARPA Information Survivability Conference and Exposition (DISCEX), vol. 1, pp. 88–102. IEEE Press (2000)
55. Winslett, M.: An introduction to trust negotiation. In: First International Conference on Trust Management (iTrust'03), *LNCS*, vol. 2692, pp. 275–283. Springer (2003)
56. Winslett, M., Yu, T., Seamons, K.E., Hess, A., Jacobson, J., Jarvis, R., Smith, B., Yu, L.: Negotiating trust in the web. IEEE Internet Computing **6**(6), 30–37 (2002)
57. X.509: The directory: Public-key and attribute certificate frameworks (2005). ITU-T Recommendation X.509:2005 | ISO/IEC 9594-8:2005
58. XACML: eXtensible Access Control Markup Language (XACML) (2005). Www.oasis-open.org/committees/xacml
59. Yu, T., Ma, X., Winslett, M.: Prunes: an efficient and complete strategy for automated trust negotiation over the internet. In: Proceedings of the 7th ACM conference on Computer and communications security (CCS '00), pp. 210–219. ACM (2000).
60. Yu, T., Winslett, M., Seamons, K.E.: Supporting structured credentials and sensitive policies through interoperable strategies for automated trust negotiation. ACM Transactions on Information and System Security (TISSEC) **6**(1), 1–42 (2003).

Hristo Koshutanski received M.Sc. in Mathematics from Plovdiv University "Paisii Hilendarski" in 2001 and Ph.D. in Information and Communication Technology from University of Trento in 2005. He won the E-NEXT SATIN award (The European Doctoral School of Advanced Topics In Networking) for doctoral research in 2005 and he was a lecturer at ESSLLI'05 European summer school. He holds an EU Marie Curie EIF post-doc fellowship with a host institution the University of Málaga for 2007–2009. His research interests include access control models, trust management techniques for digital credential negotiation, semantics of access control, digital identity management. He has co-authored a number of scientific papers.

Aliaksandr Lazouski received M.Sc. in Electronics from Belorussian State University in 2006. He is currently a PhD student in computer science department at the University of Pisa in collaboration with IIT-CNR. His research interests include access control models, trust management, usage control, digital rights management.

Fabio Martinelli (M.Sc. 1994, Ph.D. 1999) is a senior researcher of IIT-CNR. He is co-author of more than 80 papers on international journals and conference/workshop proceedings. His main research interests involve security and privacy in distributed and mobile systems and foundations of security and trust. He serves as PC-chair/organizer in several international conferences/workshops. He is the co-initiator of the International Workshop series on Formal Aspects in Security

and Trust (FAST). He is serving as scientific co-director of the international research school on Foundations of Security Analysis and Design (FOSAD) since 2004 edition. He has been recently awarded by NATO as co-director for a Advanced Training Course. He chairs the WG on security and trust management (STM) of the European Research Consortium in Informatics and Mathematics (ERCIM). He usually manages R&D projects on information and communication security and he is involved in several EU projects.

Paolo Mori received M.Sc. in Computer Science from the University of Pisa in 1998, and Ph.D. in Computer Science from the same university in 2003. He is currently a researcher of IIT-CNR. He is (co-)author of more than 20 papers on international journals and conference/workshop proceedings. 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).