A Fine-grained and X.509-based Access Control System for Globus*

Hristo Koshutanski[1], Fabio Martinelli[2], Paolo Mori[2], Luca Borz[1], and Anna Vaccarelli[2]

[1] CREATE-NET
Via Solteri 38, Trento 38100, Italy
{hristo.koshutanski, luca.borz}@create-net.org

[2] Istituto di Informatica e Telematica
Consiglio Nazionale delle Ricerche
Via Moruzzi 1, Pisa 56124, Italy
{fabio.martinelli, paolo.mori, anna.vaccarelli}@iit.cnr.it

Abstract. The rapid advancement of technologies such as Grid computing, peer-to-peer networking, Web Services to name a few, offer for companies and organizations an open and decentralized environment for dynamic resource sharing and integration. Globus toolkit emerged as the main resource sharing tool used in the Grid community.
Access control and access rights management become one of the main bottleneck when using Globus because in such an environment there are potentially unbounded number of users and resource providers without a priori established trust relationships. Thus, Grid computational resources could be executed by unknown applications running on behalf of distrusted users and therefore the integrity of those resources must be guaranteed.
To address this problem, the paper proposes an access control system that enhances the Globus toolkit with a number of features: (i) fine-grained behavioral control; (ii) application-level management of user's credentials for access control; (iii) full-fledged integration with X.509 certificate standard; (iv) access control feedback when users do not have enough permissions.

1 Introduction

Grid computing is a technology to support resource sharing among a very large dynamic and geographically dispersed set of entities. Entities participating in a Grid environment are organized in Virtual Organizations (VOs) and could

* This work is partially funded under the IST program of the EU Commission by the 2003-S116-00018 PAT-MOSTRO project, the STREP-project "ONE" (INFSO-IST-034744), the NoE-project "OPAALS" (INFSO-IST-034824), the STREP-project "S3MS" and the STREP-project "GRIDTrust".

be companies, universities, research institutes and other organizations. Grid resources could be computational resources, storage resources, software repositories and so on. Each entity exploits a Grid toolkit to set up a Grid service, i.e. the environment to share its resource on the Grid. Nowadays the most used Grid toolkit is Globus [1, 2].

Security, in particular access control issue, is one of the main concerns in Grid mainly because users participating in such an environment are potentially unknown with almost no a priori established trust relationships, which belong to distinct administrative domains. Computational resources and other Grid services are usually shared and used by applications running on behalf of unknown (not trusted) Grid users.

To improve the security of computational services, this paper describes the implementation of an enhanced access control system for the Globus toolkit that results from the integration of two security tools: Gmon and *i*Access. The key features of the new system are the enhanced fine-grained monitoring control on applications' behavior, policy-based access control decisions, integration with X.509 digital certificate standard for users' access rights management and access control feedback on application execution failure.

The adoption of X.509 standard [3] allows for cross-domain certification and identification. The X.509 framework is widely used standard nowadays by many vendors, companies and service providers.

The access control feedback is when the Grid user is denied to execute an application because of not enough access rights. The feedback allows the user to submit a proper set of credentials next time when he exploits the same Grid service.

The paper outline is the following. Section 2 provides a description of the main access control approaches used with the Globus toolkit. Section 3 shows the system underlying logical model. Next, Section 4 presents the the system architecture and its functional description. Section 5 emphasizes on the implementation details of the components that compose the system. Section 6 shows our preliminary system evaluation and Section 7 concludes the paper.

2 Related Work

As mentioned, security is a challenging issue in Grid because of the dynamic, collaborative and distributed nature of the environment. Particularly, establishing trust relationship among partners in Grid, as well as, controlling access to Grid's resources has emerged as a key security requirement. If an adequate security support is not adopted applications could perform dangerous and malicious actions on Grid resources. Since Globus [4] is a widely used Grid toolkit, most of the approaches that have been proposed to improve the Grid security are related to it.

Community Authorization Service (CAS) [5, 6] has been proposed by the Globus team in order to improve the Globus authorization system. CAS is a service that stores a database of VO policies, which determine what each Grid

user is allowed to do on the VO resources as a VO member. This service issues user's proxy certificates that embed CAS policy assertions. The Grid user that wants to use a Grid resource contacts the CAS service to obtain a proper credential in order to execute an action on the resource. The credentials (certificates) returned by a CAS server are to be presented to the local Grid manager hosting the desired service. This approach requires CAS-enabled services, i.e. services that are able to understand and enforce the policies included in the credentials released by a CAS server.

A more advanced effort for enforcing and administrating authorization policies across heterogeneous systems is the OASIS eXtensible Access Control Markup Language (XACML) framework [7]. The main actor here is the Policy Decision Point (PDP) responsible for retrieving the relevant policies with respect to the clients request, evaluating them and rendering an authorization decision. The work also considers the combination of different policies from various partners using some policy combining algorithms and taking an authorization decision on the base of evaluating them. As such XACML is a good candidate for an XML-based representation of access policies across Grid domains.

The other key approach of OASIS consortium is the Secure Assertion Markup Language (SAML) standard [8]. The main objective of the approach is to offer a standard way for exchanging authentication and authorization information between trust domains. The basic data objects of SAML are assertions. Assertions contain information that determines whether users can be authenticated or authorized to use resources. The SAML framework also defines a protocol for requesting assertions and responding to them, which makes it suitable when modeling interactive communications between entities in a distributed environment such as Grid.

Keahey and Welch [9, 10] propose an approach that integrates an existing authorization system, called Akenti [11], in the Grid environment. Similarly, Stell, Sinnot and Watt [12] integrate a role based access control infrastructure, called PERMIS [13], with the Globus toolkit to provide fine-grained access control.

The main functionality behind PERMIS and Akenti is that the information needed for an access decision, such as identity, authorization and attributes is stored and conveyed in certificates, which are widely dispersed over the Internet (e.g., LDAP directories, Web servers etc.). The authorization engine has to gather and verify all user's related certificates and to evaluate them against the access policy in order to take a decision.

One of the advantages of PERMIS infrastructure with respect to other Grid access control systems is its integration with X.509 certificate framework. X.509 identity and attribute certificates are used to attest and convey users' identities and attributes. When PERMIS access decision function (ADF) gathers all user's certificates it checks them against time validity period and trusted certificate authorities. Once ADF validates and verifies the certificates it takes an access decision according to the local (resource specific) access policy and the user's access rights. The access control policy is also loaded and extracted from X.509 attribute certificate.

Since X.509 [3] is de facto the main and widely used digital certificate standard and since it was also adopted within the Globus toolkit so it is necessary for an access control system to be fully integrated with the standard.

The access control system proposed in this paper has the following characteristics over the existing approaches:

- Fine-grained application monitoring – applications executed on the computational service are not considered as atomic entities but all the interactions with local resources are monitored according to a behavioral policy;
- History-based application monitoring – permissions to execute actions depend also from the actions that have been previously executed by the application, i.e. the application behavior is taken into account;
- X.509 access rights management – permissions to execute actions depend also from the credentials that a Grid user has submitted to Globus. Full-fledged integration with X.509 Identity and Attribute Certificates;
- Fine-grained user's access rights management – users can control the scope of their credentials. We provide general or application specific granularity of credential management;
- Access control feedback – digitally signed SAML assertion is returned back to the user in case he needs more access rights.

3 The System Underlying Model

Each Grid service provider protects its computational resources by means of logical policies. We have defined three logical policies: Behavioral Policy, Access Policy and Disclosure Policy. These policies and the logical models for enforcing them are described in more details in [14]. For clarity of the presentation we will briefly recall them here.

The behavioral policy describes the allowed behavior of an application in terms of sequences of actions that the application is allowed to perform during its execution. For each action, the behavioral policy can also define a set of predicates that must be satisfied in order to authorize the execution of the action itself. The behavioral policy is evaluated and enforced by the Gmon module, as explained in the next section.

As an example, a behavioral policy could define the admitted behavior of an application when using client sockets. In this case, the admitted sequence of actions could be: "open the socket", "connect to the server", "send data" or "receive data" and "close the socket". The policy could define the set of servers the application can connect to. To this aim, the predicate paired with the action "connect to the server" includes a check on the server name. The policy could also state that, when connected to a server included in a given set, the application is allowed to receive data but is not allowed to send data. In this case, a predicate paired with the action "'send data" checks whether the server currently connected belongs to this set of hosts. The behavioral policy also defines when access decisions need to be taken by pairing access requests to some of the actions in the sequences. With reference to the previous example, the behavioral

policy could state that only trusted users can send data on sockets because the local resource stores some critical data. In this case, the predicate paired with the action "send data" includes an access request "send data" that must be evaluated by the access policy. In this way, the access request is evaluated when the application tries to execute the action "send data". When an access decision is needed the evaluation is performed according to the access and disclosure control policies.

The access control policy specifies what requirements a user (an application running on behalf of a user) has to satisfy in order to perform some actions on local Grid resources. Rather, the disclosure control policy specifies which credentials among those occurring in the access policy can be disclosed at any time so that if needed can be demanded from a client. The two logical policies are used by *i*Access module for taking access decisions. The intuition behind *i*Access logical module is the following. Whenever *i*Access is requested for an access decision, first it checks whether the user has enough access rights to perform the requested operation according to the access policy. If yes then *i*Access simply returns grant. If no then *i*Access steps in the interactive part of the computation where it first computes from the disclosure policy the set of disclosable credentials. Then among the disclosable credentials *i*Access computes a set of *missing credentials* that added to the user's set of active credentials grant the requested operation according to the access policy. If such a set is found then *i*Access returns those missing credentials otherwise returns deny. We refer the reader to [15] for more details on the model.

With reference to the previous example, the access policy could include the following three rules. A rule stating that the "send data" action requires the Grid user to present the role of "trusted user". The second rule could state that the role "admin" dominates the role "trusted user". The third rule could state that the role "trusted user" dominates the role "grid user".

While the disclosure policy could say that the need for a "admin" and "trusted user" certificate is disclosed only to clients that have presented a certificate attesting them as a "grid user".

Now, if a user presents a certificate for a "trusted user" (or "admin") the action "send data" will be granted (authorized) by the *i*Access module. Instead, if the user presents a "grid user" certificate then the action "send data" will be denied and a feedback to the user will be returned asking for an additional credential of "trusted user" or "admin". Here we note that the choice of either "trusted user" or "admin" depends on the minimality criteria used by *i*Access algorithm and refer the reader to [15] for details.

4 System Architecture and Functionality

4.1 The Architecture

Figure 1 shows the overall system architecture. The main components of the architecture are `Gmon`, `User's Profile` and *i* `Access`. Each of them represents

a software module that has one or more interfaces to communicate with the other modules or with Globus.

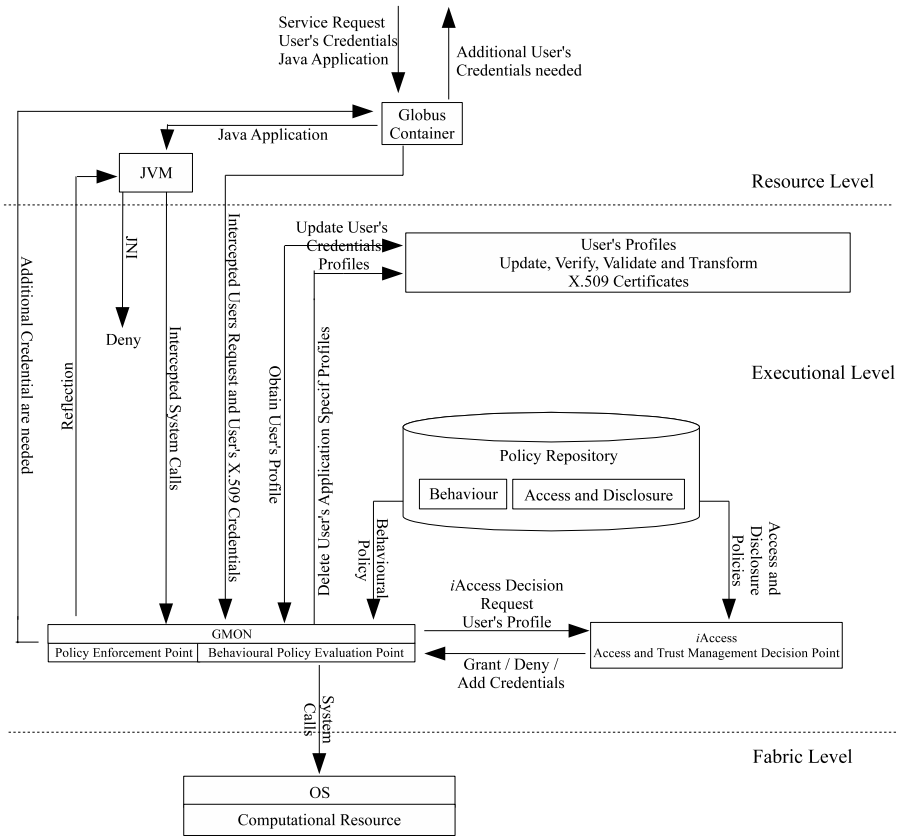

Fig. 1. System Architecture

The Gmon module is the access control enforcement point and the behavioral policy evaluation point of the architecture. As so it is the component integrated and connected directly with the Globus toolkit. In the standard Globus toolkit when a job request is received first the Grid user is authenticated by his proxy certificate and then the job request is passed to Managed Job Factory Service (MJFS) implemented by the Grid Resource Allocation and Management (GRAM) component. For each job request MJFS creates a distinct Managed Job Service (MJS) that manages the application execution on the local resource [4]. In our implementation we focus our attention on Java applications because the portability of Java is an interesting feature for the Grid environment mainly because Grid is a collection of heterogeneous resources. The integration of our

system with Globus requires that the MJFS interacts with Gmon to pass the job request together with the additional user's credentials and the MJS launches the Java application using a customized Java Virtual Machine. The resource requirements defined in the job request such as the main memory size or the CPU time are enforced by Gmon along with the behavioral policy while the user's credentials are passed to the User's Profile module. The customized JVM that runs Grid applications is monitored by Gmon. In particular, the JVM was modified in a way such that during a Java application execution whenever JVM invokes a system call (interacting with the underlying operating system) Gmon intercepts the call and suspends the execution. The execution is resumed only if the outcome of all the controls performed by our system is positive.

Gmon is also the behavioral policy evaluation point in the system. As such, each time the JVM is suspended Gmon evaluates the behavioral policy in order to determine whether the current system call is allowed or not. The evaluation takes also into account the application history, i.e. the system calls previously invoked by the application.

The positive evaluation of a system call (allowing it for execution) is only if it is included in (at least) one of the sequences of actions that represent the behavioral policy and if all the system calls that precede the current one in this sequence have been already executed by the application. Moreover, the behavioral policy can also associate to a system call some predicates that have to be verified to allow the system call. These predicates include conditions on the system call parameters and results, as well as, conditions on the overall system status. These predicates can also include a request for an access decision that Gmon forwards to the *i*Access module by passing it also the set of credentials submitted by the user. A detailed description of Gmon is given in [16].

Gmon also communicates with the User's Profile module. Whenever a user submits a set of credentials together with the job request, Gmon invokes update user's profile interface presenting the user's submitted credentials and as a result of the execution Gmon receives the local user's profile (actually a path to its location) suitable for *i*Access execution. When Gmon obtains the local profile location it starts monitoring and enforcing the application behavior.

The User's Profile module provides a full-fledged integration with X.509 standard. The module is responsible for managing and to storing credentials submitted by a Grid user. It has two interfaces: `Update` User's Profiles and `Delete` User's Profiles. The former performs validation and verification of user's presented certificates (X.509 format) and generates a user's profile suitable for the underlying logical model of the *i*Access module. The latter interface deletes user's profiles as following. User's job specific profiles are deleted by Gmon when an application finishes its execution while user's general profile is deleted by Gmon when the Grid general policy specifies.

As an outcome of the Update User's Profile interface it is returned the location where the user's profile (the one suitable for the logical model) is locally stored.

*i*Access is the access control decision point of the architecture. Once invoked, *i*Access takes an access decision according to the access and disclosure policies. The *i*Access module replies to Gmon with the result of the evaluation: grant, deny or a set of additional credentials. The set of additional credentials is in the form of SAML assertion digitally signed by *i*Access.

Whenever Gmon receives the *i*Access response, it enforces the decision according to the behavioral policy. In case of missing credentials Gmon stops the application and returns the need of missing credentials back to the user. Gmon also deletes the *user's job specific profile* when an application terminates. It does so by invoking the proper interface of the User's Profile component and specifying as input the application session id.

4.2 Functionality

Following are the key functionalities our system offers with respect to the already existing approaches.

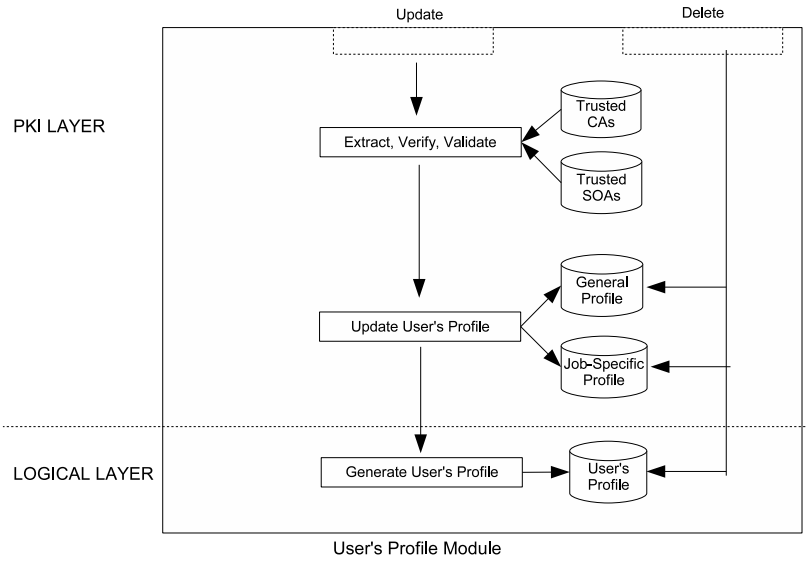

Fig. 2. User's Profile Module Architecture

– *User general profile vs multiple job specific profiles.* User general profile allows sharing of credentials among all applications running on behalf of the user. Job specific profiles allow for independent credential management, i.e. only visible to a specific application session and not shared among other user's application sessions.

We point out that user's general profile is kept much longer than the job specific one as it is used across multiple job related executions. For each user the system keeps one general user profile and multiple user's job specific profiles.

- *Digitally signed access decision feedback* in the form of digitally signed SAML assertion [8]. *i*Access lists all missing credentials as SAML attributes wrapped into a SAML assertion.
- *Full integration with X.509 standard.* As part of *i*Access project [3] this modules integrates X.509 certificate framework [3]. Figure 2 shows the architecture of the User's Profile module. The module has two interfaces: `Update` and `Delete`. `Update` interface takes as input the user's set of X.509 certificates and the user ID. The user ID is mandatory for the interface execution. In our case, the user ID is extracted from the user's (proxy) certificate that has been submitted to the Globus system. Rather, `Delete` interface takes as input either a session ID related to a particular application execution or a user ID related to a particular user. In the former case, the interface deletes the user job specific session info together with the user's profile located at the logical layer. In the latter case the interface deletes the user general profile located at the PKI layer together with (if exists) the user's job specific profile and the respective one at the logical layer.

The certificate management is divided in five steps:

1. Extract and parse X.509 certificates,
2. Validate X.509 certificates against expiration time and data corruption,
3. Verify X.509 certificates against trusted Certificate Authorities (CAs) if identity certificate, or trusted Source of Authorities (SOAs) if attribute certificate,
4. Update User's profile according to certificates' scope,
5. Transform all user's certificates to predicates suitable for the *i*Access logical model.

The part Update takes the X.509 certificates and saves them in session (XML) files. If a session info already exists then it is updated with the newly presented certificates. The Update functionality creates a separate session for "job" marked certificates and an application specific session for each "user" marked certificates. The functionality presented above are contained in the PKI Session Layer.

The role of Generate User's Profile module is to transform and prepare the information contained in X.509 certificates in a from suitable for the *i*Access logical model, located on the Logical Layer.

5 System Implementation

Section 4 shows that our framework integrates three software components: Globus, Gmon and *i*Access. This section describes the implementation details that concern the integration of those components.

[3] www.interactiveaccess.org

5.1 Globus

The Globus Toolkit v4 (GT4) provides a set of Grid infrastructure services that implement interfaces for managing computational, storage and other resources [4]. The Globus Container runs services provided by local resources including the standard Globus services and the custom services developed by Grid resource providers. The Grid Resource Allocation and Management (GRAM) is the Globus service that is in charge of execution management of applications submitted by remote Grid users.

The integration of our framework with the Globus toolkit was easy due to the high modularity of the Globus architecture. The Globus toolkit was modified basically for two tasks:

- defining and managing the `java` job type;
- transferring submitted credentials by Grid users to the Grid services and managing them to support the access control decisions.

Grid users issues a job request for a service in order to submit an application to Grid. The job request is an XML document that conforms to the Resource Specification Language (RSL) to describe the job features through two type of information:

- the resource requirements such as the main memory size or the CPU time
- the job configuration such as the file transfers to be executed to get the application code and its data, the name of the executable file, the input parameters, the file for the input and the output of the application, and so on.

The standard Globus job description schema defines the following job types: mpi, single, multiple and condor. We defined the `java` job type. To this aim, we modified the job description schema by adding the `java` job type in the Job Type Enumeration field. To manage the `java` job type, instead, we modified the Job Manager Perl script included in the Managed Job Service (MJS) that submits the application executable code to the scheduler of the local resource. The modified script, in the case of `java` jobs, submits the application specified in the executable field of the job request to our customized JVM. In this way, when a Grid user wants to execute a Java application he specifies `java` as a job type in its job request. Then on the computational service side Globus automatically invokes the security enhanced JVM in order to execute the submitted application.

The credentials submitted by a remote Grid user are also included in the job request by exploiting the extension field. In Globus, the job request is examined by the Managed Job Factory Service (MJFS) of the GRAM component to extract the requirements to instantiate a proper MJS for managing the request. We modified the MJFS to export the job request to Gmon. In this case the MJFS exports the job request on a local file that will be loaded by Gmon. At this stage Gmon is not started yet. The jobDescriptionType and RSLHelper classes of the

Globus package provide simple methods to convert the job description in a text string to be saved in a file.

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. It is exactly the case when the credentials submitted by a user are not enough to allow user's application to execute an action. In this case the error message specifies the credentials that the user additionally needs to provide. 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.

5.2 Gmon

The successful Gmon integration within the framework requires Gmon to interact with the User's Profile component and with *i*Access. Gmon interacts with Globus in order to get the service request submitted by the remote Grid user. This interaction is simply implemented through an XML file that is generated by Globus as described in the previous section. Gmon simply loads it during the initialization phase. Once it is loaded, Gmon extracts from the job request the resource requirements along with the submitted credentials. Gmon also imports from Globus the user ID extracted from the proxy certificate that has been submitted by the user.

Next, Gmon invokes the User's Profile module as an external library function in order to update the user profile by passing the submitted credentials (in X.509 format) and the user ID.

Since the behavioral policy defines the predicates that Gmon has to evaluate for each action, access requests are simply represented as new predicates in the policy and the evaluation of those predicates consists of the invocation of the *i*Access component with input the action request and the link to the user's local profile. The *i*Access (as well as User's Profile component) are invoked by Gmon as an external library functions.

The interactions between Gmon and *i*Access are the most critical part of the architecture in term of performance. Since *i*Access module is Java based and Gmon is C based, we have to provide a way to invoke *i*Access without loading the JVM every time. Since *i*Access is an internal and trusted application it is executed by exploiting a standard JVM. We use the Java Native Interface (JNI) library [17] that allows 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 needed for the execution of a Java class and also commands to obtain links to a Java class and its methods. A separate command is provided by JNI for releasing the JVM. In this case, at initialization time, Gmon loads the JVM and creates the links to the *i*Access methods and keeps them in memory for all the subsequent invocation of *i*Access. When the application terminates Gmon release the JVM before it terminates.

5.3 User's Profiles

The Update User's Profile interface has as input the path of an XML file that contains the certificates presented by the user and the user ID extracted from the user X.509 proxy (identity) certificate.

A certificate sent to the system has an additional property scope indicating *user-* or *job-specific* lifetime. Following is an example of user input of identity and attribute certificates.

```
<UserInput>
    <X509V3Cert type="id" scope="user">
            MIIBtjCCAwIBAgICAN4wDQYJKoZIhvcNAQEFBQAwTTELMAkGA1UEBtTAPB
            XcKJyiqoOkY68DcwPVahmd ....
     </X509V3Cert>
     <X509V3Cert type="attr" scope="job">
            MIIBtjCCAR8CAQEwbKFqpGgwZjELMAkGA1UEBhMCSVQxDDAKBgNVByQzE
            VFJFTlRPMRIwEAYDVQQDEwlM ....
     </X509V3Cert>
</UserInput>
```

Identity Certificates are implemented with the standard library of Sun JDK while the Attribute certificates are implemented with the Bouncy Castle[4] security libraries.

Each identity certificate has a subject name and an issuer name while, in contrast, each attribute certificate has a holder name, attribute value and issuer name. Those are the relevant fields for the `Generate User's Profile` function. The role of this function is first to match identity certificates with attribute certificates in order to extract appropriate information regarding the user's access rights. The matching checks whether the serial number of an identity certificate is within the field of the holder name in an attribute certificate, as recommended by ITU-T for use of X.509 specification.

Once the matching is finished then the `Generate User's Profile` function generates appropriate predicates according to the matching outcome. The outcome of the generation is a set of logical predicates describing user's identity and attributes.

5.4 *i*Access

*i*Access is a Java-based access decision engine that implements the interactive access control model [15]. DLV system is used as an underlying logical engine for logical operations. Particularly, for the integration with Gmon *i*Access engine has been optimized only to the level of taking access decision without managing user's credentials in contrast to the model in [15].

[4] http://www.bouncycastle.org

6 System Preliminary Evaluation

Our preliminary results we performed to evaluate the impact on the adoption of our framework on the execution time of Java applications. This overhead strongly depends on the specific Java application, on the enforced policy and on the number of certificates submitted by the user.

If the application performs only few system calls or if the security policy is very simple the impact of the security checks on the overall execution time is negligible with respect to the application execution itself.

The first set of experiments evaluates the execution time of two Java applications adopting a standard JVM, a JVM enhanced with Gmon and a JVM enhanced with Gmon and *i*Access. We exploited the Jikes RVM Java Virtual Machine developed by the IBM T.J. Watson Research Center [18], running on Linux (FC4 with 2.6.13 kernel) operating system. The applications we chose are two benchmarks from the Ashes Suite Collection benchmarks[5], that are suitable for our tests because they perform a large number of system calls. In our experiments we adopted a simple policy similar to the one shown in [14]. The

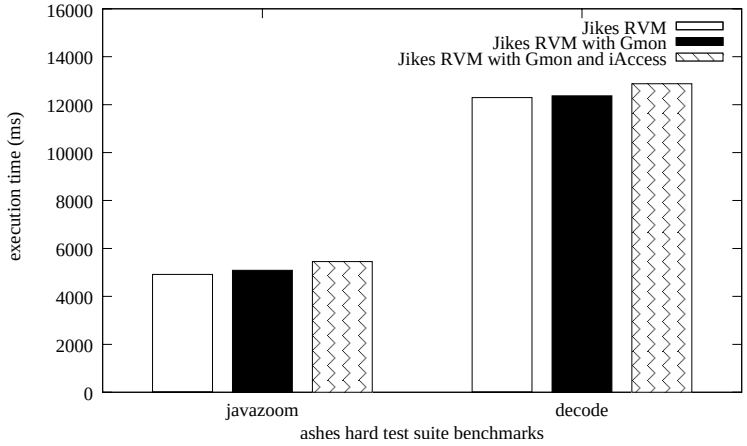

Fig. 3. Java benchmarks execution times

performances are shown in Figure 3. Javazoom is an mp3 to wav converter. It performs about 1500 system call in about 5 seconds. Most of these system calls are issued to read the mp3 file and to write the wav one. Our system controls whether javazoom has the credentials to open the mp3 file, to create the wav one and whether access to these files is compatible with the admitted behavior described by the behavioral policy. In this case, the overhead introduced by

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

Gmon only is about 4%, while the overhead by Gmon and *i*Access is about 10% of the execution time of the application.

Decode is an algorithm for decoding encrypted messages that uses the Shamir's Secret Sharing scheme. It executes about 570 system calls in about 12 seconds. In this case, the overhead introduced by Gmon is about 1% while the overhead by Gmon and *i*Access is about 5% of the application execution time.

The second set of experiments concerns the evaluation of the performance of the `User's Profile` module. It has been done on Linux (FC4 with 2.6.13 kernel) operating system as well. The table below shows the several sessions we have performed. For each row we run the Update User's Profile prototype with a different number of certificates.

Number of Certs	5	10	20	50
Update User's Profile (execution time ms)	452	589	727	1884

All the times are expressed in milliseconds. The Update process has different phases (refer to Figure 2). The time table shows that having 50 certificates (which is quite a few for a user to have) the time is less than two seconds. It indicates that the time response of the `User's Profile` module is acceptable whit respect to the overall system performance.

7 Conclusions and Future Work

We have presented an access control system that enhances the access control mechanism in the Globus toolkit. The key system functionalities with respect to other access control systems are: (i) fine-grained behavioral control combined with a credential-based access control mechanism; (ii) integration with X.509 certificate standard and application-level granularity management of user's credentials; (iii) access control feedback when users do not have enough permissions to use Grid resources.

Even if the adoption of different policies, different benchmarks or different sets of user's certificates could result in very different performances, we believe that the results from the preliminary evaluation confirm that it is possible to define a fine-grained security policy that integrates behavioral and trust management aspects and that the overhead to enforce is affordable.

Future work is on improving the performance of the system and performing experimental assessments when applying it on real Grid scenarios.

References

1. Foster, I., Kesselman, C., Tuecke, S.: The anatomy of the grid: Enabling scalable virtual organizations. International Journal of Supercomputer Applications **15**(3) (2001) 200–222
2. Foster, I., Kesselman, C., Nick, J., Tuecke, S.: The physiology of the grid: An open grid service architecture for distributed system integration. Globus Project (2002) http://www.globus.org/research/papers/ogsa.pdf.

3. X.509: The directory: Public-key and attribute certificate frameworks (2001) ITU-T Recommendation X.509:2000(E) | ISO/IEC 9594-8:2001(E).
4. Foster, I.: Globus toolkit version 4: Software for service-oriented systems. In: Proceedings of IFIP International Conference on Network and Parallel Computing. Volume 3779., Springer-Verlag, LNCS (2005) 2–13
5. 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). (2002) 50–59
6. 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) TUBT003
7. XACML: eXtensible Access Control Markup Language (XACML) (2005) www.oasis-open.org/committees/xacml.
8. SAML: Security Assertion Markup Language (SAML) (2005) www.oasis-open.org/committees/security.
9. 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. Volume 2536. (2002) 199–206
10. 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)
11. 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). (1999) 215–228
12. 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. (2005) 195–201
13. Chadwick, D.W., Otenko, A.: The PERMIS X.509 role-based privilege management infrastructure. In: Seventh ACM Symposium on Access Control Models and Technologies, ACM Press (2002) 135–140
14. 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 (2006)
15. Koshutanski, H., Massacci, F.: Interactive access control for Web Services. In: Proceedings of the 19th IFIP Information Security Conference (SEC 2004), Toulouse, France, Kluwer Press (2004) 151–166
16. 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. (2005) 82
17. Liang, S.: Java(TM) Native Interface: Programmer's Guide and Specification. Addison-Wesley (1999)
18. Alpern, B., Attanasio, C., Barton, J., et al.: The jalapeño virtual machine. IBM System Journal **39**(1) (2000) 211–221