Fine Grained Access Control with Trust and Reputation Management for Globus*

M. Colombo, F. Martinelli, P. Mori, M. Petrocchi, and A. Vaccarelli

IIT-CNR, Pisa, Italy
{maurizio.colombo, fabio.martinelli, paolo.mori,
marinella.petrocchi, anna.vaccarelli}@iit.cnr.it

Abstract. We propose an integrated architecture, extending a framework for fine grained access control of Grid computational services, with an inference engine managing reputation and trust management credentials. Also, we present the implementation of the proposed architecture, with preliminary performance figures.

1 Introduction

Grid computing represents a large scale, on-demand, cooperative communication and computation infrastructure. It can be considered as an aggregation of both physical and logical resources, for a given purpose. It offers resources for sharing, thus it requires a shift in the vision of usage control of the resources. Previous security mechanisms are not flexible, being based on the assumption that who use the resource is known a priori. Furthermore, technology may be not flexible enough to allow a fine grained control. Our work tries to address this problem.

One very commonly used toolkit for Grid computing is Globus [2]. In particular, most of the work has been devoted to access control techniques for accessing a service at the coarse grain level, *e.g.,* see [10, 12].

In [1, 7], we started a fine grained run-time access control framework for Grid services. It was based on a behavioral policy language that describes the correct sequence of actions that the applications are allowed to perform on the computational resource. Coarse grained access control, instead, only determines whether a given user can execute a given application, but once the execution right is granted, no further controls are performed during the execution.

In this work, we significantly enhance our fine grained run-time access control framework, by integrating it with a powerful trust management framework, *i.e.,* the Role-based Trust Management Language (RTML, [5, 14]).

Main contributions of this paper are: *i)* an implementation of the RTML framework, embedded in our behavioral policy, to perform fine grained access control and trust management in Grid; *ii)* an extension of the RTML framework

* Work partially supported by the EU project IST-3-016004-IP-09 Sensoria (*Software Engineering for Service Oriented Overlay Computers*), and by the EU STREP project IST-033817 GridTrust (*Trust and Security for Next Generation Grid*). We thank Ninghui Li for providing the Java parser for the RTML family of languages.

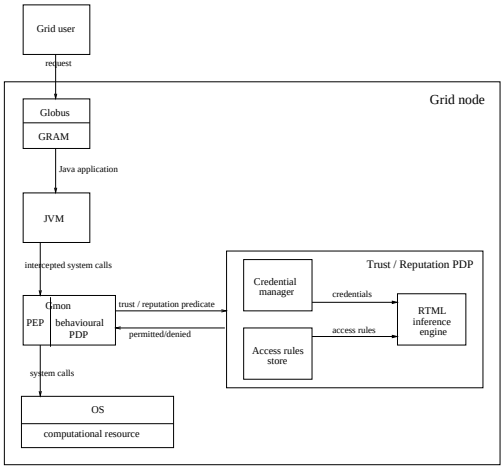

Fig. 1. Global architecture

to allow reputation management. We conceived its model and we integrated it in the above implementation.

A previous attempt to integrate trust management with fine grained access control in Grid can be found in [3]. The novelty of our approach is that we exploit RTML to deal with both trust and reputation management. Also, some examples may be found in literature in the area of reputation management for Grid, *e.g.,* see [11]. However, none of them is actually designed for fine grained level. It is also worth noticing that the usual approach is trying to help the user in selecting trusted services (*e.g.,* [13]). Our point of view is the opposite, because we are interested in protecting the service from the untrusted user.

The structure of the paper is as follows. Section 2 defines our integrated architecture. Section 3 recalls our integrated policy language, consisting of a behavioral policy language together with a trust and reputation management one. Section 4 describes the implementation. Finally, some performance experiments are reported in Section 5.

2 Architecture

The system architecture resembles the one in [7], and it is shown in Figure 1. It is designed to monitor the behaviour of Java applications executed on behalf of Grid users on Grid computational services. A Grid user U submits a request to execute a Java application A on a computational service S. The Globus Container [2], that runs the service S, receives the request, verifies the identity of U, and the Grid Resource Allocation and Management component (GRAM) submits A to the local resource scheduler for the execution. During the execution of A, our monitoring component, Gmon, acts as policy enforcement point (PEP), and intercepts any action that A tries to execute on the underlying com-

putational resource. Gmon actually executes the action only after evaluating a certain predefined security policy (see Section 3). The evaluation of the security policy is performed through the internal behavioral Policy Decision Point (PDP) of Gmon only for what behavioral aspects are concerned. In particular, Gmon checks whether the sequence of actions that the application is performing matches the permitted behaviour defined by the policy.

In this paper we contribute by allowing the policy to evaluate also reputation and trust attributes of U, on the basis of RTML, [5, 14], a family of languages suitable to represent policies and credentials. To this aim, the original architecture has been extended to include two specific components, dealing with trust and reputation management. According to the policy, Gmon may invoke one of these components, or both. Hence, Gmon behavioral policy languages may be considered as policy orchestrators. To take their decisions, the reputation PDP and the trust PDP exploit distinct credentials, since the reputation credential include also a weight (see Section 3). They are however elaborated by the same RTML-based inference engine, that we have extended for dealing with quantitative notions. Actually, both the trust and the reputation management PDPs are built around RTML. In particular, their main components are: *i) the credential manager*, that stores the set of credentials related to each user, in order to prove rights to access some specific resource; *ii) the access rules store*, that contains the local access rules that have been defined by the resource provider; *iii) the RTML inference engine*, that exploits the user credentials and the access rules to determine whether the user is entitled or not to have the specific attribute (or role) that has been requested by Gmon. In the case of reputation PDP, this engine also computes the weight associated with the requested attribute.

When dealing with reputation credentials, the credential manager is conceptually divided into two sub-components (according to a scheme proposed by [6]): *i) Experience manager*: it is in charge of recording direct experiences with users; *ii) Recommendation manager*: it implements three functions: storing recommendations from other providers, managing reputation of recommenders and exchange recommendations with other providers.

3 Policy languages

Behavioral policy language. In the original framework, the security behavioral policy, describes the permitted behaviour, *i.e.,* the sequences of security relevant actions that the Grid applications are allowed to execute on the Grid computational service. The exploited language is described in [7].

Here, we describe an extended framework, where the policy can state that some actions can be performed on the local resource only if the set of trust relations that the user has established in previous experiences grants him a given attribute in a given domain, probably the local one, or can state that the reputation of the user for this attribute is greater of a given threshold. We introduce two predicates, `repmaxof()` and `trust()`, to be used in the behavioral

policy. `repmaxof()` is used to check the reputation of the Grid user. The following example:

[repmaxof(UniPi.files(USER),0.6)].open(x_1,x_2,x_3,r)

states that the action `open` can be executed only if the reputation given by the service provider `UniPi` to the Grid user USER for the attribute `files` is greater than 0.6. User attributes have been introduced to assign distinct reputation to the same user, depending on the action to be performed. As an example, the attribute `files` refers to file accesses, the attribute `sockets` refers to network accesses through sockets, and so on. `UniPi.files(USER)` is a RTML statement, whose notation will be clarified in next subsections. The other predicate defined in this paper is `trust()`. This predicate evaluates the trust relationships that the user has collected in past experiences. The following example:

[trust(UniPi.guest(USER))].open(x_1,x_2,x_3,r)

declares that the action `open` can be executed only if the service provider `UniPi` grants to the Grid user USER the attribute guest. The predicate is satisfied if the service provider `UniPi` grants the attribute guest to the Grid user directly, or if the attribute is granted by a set of credentials properly combined.

Proper components of the architecture are in charge of evaluating these predicates, and proper policies are needed, Our language may be considered as an orchestrator of several policies. The interesting feature is that all these orchestrated policies can be modelled as inference systems. This clearly does not prevent to integrate other kind of policies, but defines a compact framework.

RTML with trust measures. RTML[14, 5] is a language defining credentials through roles, *i.e.,* authorities assign to someone roles, or attributes. Roles may be parameterized, *e.g.,* a basic credential of the form $A.r(p) \leftarrow D$ means that A assigns to D the role r with parameter p. In the following credential, organization IIT assigns the role of IIT researcher to Paolo, whose distinguished name adopted on the Grid is "CN=Paolo, OU=IIT, O=CNR, L=Pisa, C=IT".

IIT.researcher('CN=Paolo, OU=IIT, O=CNR, L=Pisa, C=IT') $\leftarrow$ Paolo

Enriching this language with trust means enhancing credentials in order to express that a principal trusts someone for performing some functionality f, or for giving a recommendation regarding a third party able to perform f. Thus, credentials can specify the degree of the assignment or trust.

We recall the language in [8], enriching RTML with trust measures v.

- (**simple member**) $A.r(p, v) \leftarrow D$. The role $A.r(p)$ has weight v.
- (**simple containment**) $A.r(p, v) \leftarrow_{v_2} A_1.r_1(p_1, v_1)$. According to A, all members of role $A_1.r_1(p_1, v_1)$ with weight v_1 are members of role $A.r(p, v)$ with weight $v = v_1 \otimes v_2$. v_2 is a constant filtering A_1's authority with A's authority.
- (**linking containment**) $A.r(p) \leftarrow A.r_1(p_1).r_2(p_2)$. If B has role $A_1.r_1(p_1)$ with weight v_1 and D has role $B.r_2(p_2)$ with weight v_2, then D has role $A.r(p)$ with weight $v = v_1 \otimes v_2$.
- (**intersection**) $A.r(p) \leftarrow A_1.r_1(p_1) \cap A_2.r_2(p_2)$. This statement defines that if D has both roles $A_1.r_1(p_1)$ with weight v_1 and $A_2.r_2(p_2)$ with weight v_2, then D has role $A.r(p)$ with weight $v = v_1 \odot v_2$.

We do not explicitly express weights in the linking and intersection containment statements. Indeed, these statements combine basic credentials (the simple member ones) and they determine how weights presented in the basic credentials must be combined too.

Operators $\otimes$ and $\odot$ combine the trust measures. Generally speaking, $\otimes$ combines opinions along a path, *i.e.,* A's opinion for B is combined with B's opinion for C into one indirect opinion that A should have for C, based on what B thinks about C. The latter, $\odot$, combines opinions across paths, *i.e.,* A's indirect opinion for X through path p1 is combined with A's indirect opinion for X through path p2 into one aggregate opinion that reconciles both. To work properly, these operators must form an algebraic structure called a c-semiring, [9].

In our framework, reputation of a user can be calculated based on past experiences of other services *w.r.t.* that user. The more the user has been well-behaved with that service, the more the service will positively recommend interactions with that user. Services emit two kinds of credentials. The first kind expresses trust towards a functionality, *e.g.,* towards good behaviours, and we denote them by $A.f(v) \leftarrow D$, *i.e.,* A trusts D for performing functionality f with degree v. The others are credentials of recommendation, denoted as $A.rf(v) \leftarrow D$. They express the fact that A trusts D as a recommender able to suggest someone for performing f.

Recommendations can be transitive. Transitivity is encoded by a linking containment of the form $A.rf \leftarrow A.rf.rf$. This statement says that if A defines B to have property $A.rf$, and B defines D to have property $B.rf$, then A defines D to have role $A.rf$, *i.e.,* A trusts D as a recommender.

Intuitively, A will trust the recommended party. This can be encoded into the following statement: $A.f \leftarrow A.rf.f$. This statement says that if B has role $A.rf$ and C has role $B.f$ then C has role $A.f$. B, that has the role $A.rf$, is the recommender, *i.e.,* A trusts B for choosing someone that is trusted for performing f. C, that has role $B.f$, is trusted to perform f by B. Hence, C is *indirectly* trusted to perform f by A. This resembles somehow the simple delegation statement of [4].

We can define a set of functionalities, *i.e.,* a range of values for f, *e.g.,* :

- $A.files(p, v) \leftarrow$ D. A trusts user D with degree v for *operating on a file.*
- $A.socket(p, v) \leftarrow$ D. A trusts user D with degree v for *operating on a socket.*

In the following, the parameter p will be used to denote the distinguished name of the user, as specified into his Grid certificate.

3.1 Security Policy Example

We consider a security policy consisting of a behavioural policy, a trust management policy and a reputation management policy. The behavioural policy is directly enforced by Gmon, and includes some predicates that, to be evaluated, could require the usage of the other two policies. These are evaluated through the trust PDP or by the reputation PDP that will exploit the credentials for trust

and reputation management. Since a Grid application interacts with the computational resource through operative system calls, we assume that the security relevant actions composing the policy are system calls. We give a simple example of a behavioural policy that includes the trust and reputation evaluation.

l1: $S1$:= {file0.txt, file1.txt}
l2: $S2$:= {file2.txt, file3.txt}
l3: ([in(x_1,$S1$), eq(x_2, READ), trust(Unipi.guest(USER))].open(x_1,x_2,-,fd).
l4: i([eq(x_5, fd)].read(x_5, -, -, -)).
l5: [eq(x_9, fd)].close(x_9, -))
l6: par
l7: ([in(x_{10}, $S2$), eq(x_{11}, WRITE), repmaxof(Unipi.files(USER),0.8)].open(x_{10},x_{11},-,fd).
l8: i([eq(x_{14}, fd)].write(x_{14}, -, -, -)).
l9: [eq(x_{18}, fd)].close(x_{18}, -))

This policy defines two sets of files, S1 and S2 (lines l1 and l2). The first rule of the policy (lines l3 - l5) defines the behaviour in reading files. Line l3 allows to execute the `open` system call if the three predicates `[in(`x_1`,`$S1$`), eq(`x_2`, READ), trust(UniPi.guest(USER))]` are satisfied. The first two predicates requires that the file that the application wants to open belongs to the set S1 and that the file is opened in READ mode. The third predicate, `trust(UniPi.guest(USER))`, requires that the user holds the attribute guest in the UniPi domain. Hence, only files that belong to the set S1 can be read in this system by users who holds a specific attribute, guest, in the UniPi domain. The evaluation of the user attributes is executed exploiting the trust management policy described in the following. The other rule of the policy (lines l7 - l9) defines the allowed behaviour of the applications in writing files. The main difference from the previous one is that, it requires that the user reputation for the attribute file, i.e. for operating on files, is at least 0.8. The reputation of the user for the attribute file is evaluated using the reputation policy described in the following of this section.

The trust management policy consists of a set of credentials. Some of them define the attributes of a user for a given service provider. The first two credentials give to the user Paolo the attribute of collab (*i.e.,* collaborator) for the service provider UniGe, and the attribute of researcher for the service provider IIT. The parameter "CN=Paolo, OU=IIT, O=CNR, L=Pisa, ST=PI, C=IT" is the distinguished name (DN) that appears in the identity certificate of Paolo. The third credential, instead, assigns to Unige the attribute of university for the MIUR authority. The other three credentials are different from the previous ones, because they allow to infer new attributes from the attributes that a user already has. The fourth credential give the attribute guest for the service provider UniPi to a user that already has the attribute of researcher for IIT and the attribute of collaborator for UniPi. The fifth credential says that UniPi gives the attribute university to all the subjects that already has the attribute university for MIUR. The last credential, instead, says the UniPi gives the attribute collaborator to the subjects that already has the attribute collaborator given by subjects that have the attribute university for UniPi.

1) UniGe.collab('CN=Paolo, OU=IIT, O=CNR, L=Pisa, ST=PI, C=IT') ← Paolo.
2) IIT.researcher('CN=Paolo, OU=IIT, O=CNR, L=Pisa, ST=PI, C=IT') ← Paolo.

3) Miur.university('CN=University of Genoa, OU=Security Lab, O=CS Department, L=Genoa, ST=GE, C=IT') ← UniGe.
4) UniPi.guest(name) ← IIT.researcher(name) ∩ UniPi.collab(name).
5) UniPi.university(uname) ← Miur.university(uname).
6) UniPi.collab(name) ← UniPi.university(uname).collab(name).

The following is the reputation management policy. The first two credentials say, respectively, that UniGe gives to Paolo the attribute files with reputation 0.7 and that IIT gives to Paolo the attribute files with reputation 0.8. The third credential says that UniPi accepts recommendations for the attribute files from UniGe. Hence, if a user has a credential that gives him the attribute files in UniGe, than the third credential of the policy gives him the attribute files also in UniPi (according to the fifth credential). The fourth credential states that UniPi accepts recommendations for the attribute files from IIT too. Results in combining these credentials are that: 1) Paolo has been recommended for files from UniGe, with weight $1 \otimes 0.7$. This is a consequence of combining the first, third and fifth credentials; 2) Paolo has been recommended for files from IIT, with weight $1 \otimes 0.8$. This results from the second, fourth and fifth credentials. Thus, Paolo can present one of the two, according to some requested threshold.

1) UniGe.files('CN=Paolo, OU=IIT, O=CNR, L=Pisa, ST=PI, C=IT', 0.7) ← Paolo.
2) IIT.files('CN=Paolo, OU=IIT, O=CNR, L=Pisa, ST=PI, C=IT', 0.8) ← Paolo.
3) UniPi.rfiles('CN=UniversityGenoa, OU=Miur, O=Unige, L=Genoa, ST=GE, C=IT', 1) ← UniGe.
4) UniPi.rfiles('CN - InstituteInformaticsTelematics, OU=IIT, O=CNR, L=Pisa, ST=PI, C=IT',1) ← IIT.
5) UniPi.files(userName) ← UniPi.rfiles(recName).files(userName)

4 Implementation

This section describes the ongoing implementation of our framework. It focuses on the integration of the reputation management system with the Grid computational resource monitoring system of [7]. We use the Java language, suitable for developing Grid applications, for the platform independence addressing the Grid interoperability problem. In our architecture, Gmon is both PEP and PDP for decisions concerning the behaviour of the applications. To evaluate the user reputation, instead, Gmon exploits another component, the Reputation PDP, that is invoked each time the policy requires to evaluate the user reputation to allow the current action. Gmon and the Reputation PDP runs on the same computational resource. Since the former is developed in C, while the latter in Java, the Reputation PDP is invoked by Gmon through the Java Native Interface. The Reputation PDP Java class has two main methods: `initialization` and `isPermitted`. The `initialization` method is invoked by Gmon in the initialization phase, with a set of parameters that indicates the Credential Repository where to retrieve credentials and certificates. The method `isPermitted` is invoked by Gmon during the execution of the Java application each time the security policy requires to evaluate the user reputation for a given attribute.

The Credential Manager manages the Credential Repository. The Recommendation Manager, that is part of the Credential Manager, is in charge of collecting Grid users credentials from a set of Grid providers that act also as user recommenders. The Recommendation Manager also updates the Credential Repository periodically. Instead, the Experience Manager, that is part of the Credential Manager too, stores the credentials that have been created on this node. Credentials are written using RTML. The RTML code is embedded in X509 Certificates. The Reputation PDP verifies the credentials signature, extracts the related XML code and is passes it to the RTML framework. The RTML framework has been implemented by Ninghui Li et al. [5], and it consists of a credential Parser and Engine. The Parser is a DOM-based parser, it parsers the received credentials and the access rules keeping the information into a complex data structure, the CredentialStore. The RTEngine implements the algorithm described in 4.1 and, once invoked, it uses the CredentialStores to evaluate the credentials and the access rules. Its output is a new CredentialStore containing the set of credentials physically owned by the user and the ones virtually owned, which are granted by the evaluation. The new set of credentials represents the trust of the Grid user on the node. Each secure action has a credential associated to it which represents the reputation required for the execution. The method `isPermitted` evaluates all the credentials in the user CredentialStore to verify if one of them satisfies the requirements for the action, *i.e.,* if one of the virtual credentials is compatible with the one associated to the action and its level of reputation (weight) is greater then the one requested.

When the application finishes to execute, or it is interrupted for a policy violation, Gmon communicates to the Experience Manager about this execution. The Experience Manager issues to the current Grid user a new X509 Certificate embedding the RTML code representing credentials associated to the correctly executed actions These credentials could be used by the Reputation PDP the next time the same Grid user executes an application on this computational resource. Also, these are exploited by the Recommendation Manager to recommend this user to other Grid service providers.

4.1 An implementation of RTML with trust measures

The algorithm calculates a minimal set of simple member credentials, starting from two sets of available credentials, simple and not simple credentials. Without considering trust measures, the algorithm basically builds the resulting set by iteratively applying the inference rules for each kind of credential. If the inferred credential does not belong yet to the set of computed basic credentials, then it is added to this set. The procedure is iterated until no new credentials are found. If the algorithm is applied to a finite set of credentials, it correctly terminates.

Adding weights is possible. In this case the algorithm is a variant of the Floyd algorithm for calculating minimal/maximal weighted paths among all the nodes in a graph. Indeed, $A.r(v) \leftarrow C$ states that between A and C there is an arc labelled r and with measure v. We consider order $\leq_w$, defined as $v_1 \leq_w v_2$ iff $v_1 \odot v_2 = v_2$. Then, the algorithm computes the maximal weighted path (*w.r.t.* $\leq_w$) in the graph. We remind that in c-semiring $\otimes$ is an inclusive operation.

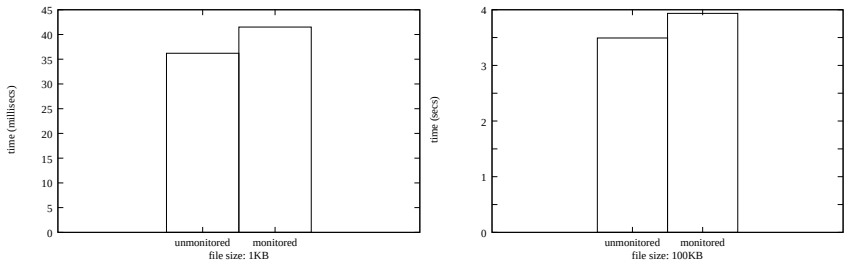

Fig. 2. Experimental Results

Trust Calculations (basic creds, rules)= {
  Results:=basic creds; Changed := true;
  While(Changed) {
    Changed:=false;
    For each credential $A.r \leftarrow_{v_2} A_1.r_1$ in rules and for each credential
    $A.r_1(v_1) \leftarrow C$ in basic creds
      if $A.r \leftarrow C$ not in basic creds, or $A.r(v) \leftarrow C$ in basic creds with
      not $v_1 \otimes v_2 \leq_w v$
        then {remove from basic creds all the creds like $A.r \leftarrow C$;
        insert $A.r(v_1 \otimes v_2) \leftarrow C$ in basic creds; Changed:=true};
    For each credential $A.r \leftarrow A.r_1.r_2$ in rules and for each credential
    $A.r_1(v_1) \leftarrow B$, $B.r_2(v_2) \leftarrow C$ in basic creds
      if $A.r \leftarrow C$ not in basic creds, or $A.r(v) \leftarrow C$ in basic creds with
      not $v_1 \otimes v_2 \leq_w v$
        then {remove from basic creds all the creds like $A.r \leftarrow C$;
        insert $A.r(v_1 \otimes v_2) \leftarrow C$ in basic creds; Changed:=true};
    For each credential $A.r \leftarrow A_1.r_1 \cap A_2.r_2$ in rules and for each credential
    $A_1.r_1(v_1) \leftarrow C$, $A.r_2(v_2) \leftarrow C$ in basic creds
      if $A.r \leftarrow C$ not in basic creds, or $A.r(v) \leftarrow C$ in basic creds with
      not $v_1 \odot v_2 \leq_w v$
        then {remove from basic creds all the creds like $A.r \leftarrow C$;
        insert $A.r(v_1 \odot v_2) \leftarrow C$ in basic creds; Changed:=true}; }

5 Performance experimentation

This section evaluates the overhead introduced by our authorisation framework. We performed some experiments to measure the execution time of a test application with and without our framework. We used the security policy of Section 3.1, and a very simple application, that opens a file, writes a set of data, and closes the file. Concerning performances, this is the worst case, because this application does not perform any computation, each performed action is monitored by our framework and, consequently, it introduces overhead. Figure 2 reports the execution time and the overhead for writing files of 1Kbyte and 100Kbytes.

The overhead measured writing a file of 1Kb is about 13% of the computational time: 2% is due to the credential evaluation, while 11% is due to check

the behavioural policy. Instead, in the second experiment, the overall overhead is 11% of the total execution time, and it is mainly due to the behavioural controls. The overhead introduced by our framework depends on several factors. One of this factor is the security policy, because simpler security policies take less time to be evaluated. Another factor that determines the impact of the introduced overhead is the application. Actually, if the application is computational-intensive, *i.e.,* it executes mainly computation, interacting a few times with the underlying resource, the overhead for monitoring refers to large computation times, and it is typically negligible. Otherwise, if the application mainly performs interactions with the resource, like in the above example, the overhead for monitoring them heavily impacts on the final execution time.

6 Conclusions

In this paper, we have enriched our framework for fine-grained access control on the Grid, by adding a RTML-based inference engine, managing trust and reputation credentials. We plan to evaluate the performances of the overall system, by considering more complex case studies.

References

1. F. Baiardi, F. Martinelli, P. Mori, and A. Vaccarelli. Improving grid services security with fine grain policies. In *OTM Workshops*, pages 123–134, 2004.
2. I. Foster. Globus toolkit version 4: Software for service-oriented systems. In *IFIP NPC*, pages 2–13. Springer, 2005.
3. H. Koshutanski, F. Martinelli, P. Mori, L. Borz, and A. Vaccarelli. A fine grained and x.509 based access control system for globus. In *OTM*, pages 1336–1350. Springer, 2006.
4. N. Li et al. Rtml: A role-based trust-management markup language. Technical report, CERIAS 03, 2004.
5. N. Li, J.C. Mitchell, and W. H. Winsborough. Design of a role-based trust management framework. In *S&P*, pages 114–130. IEEE, 2002.
6. J. Liu and V. Issarny. Enhanced reputation mechanism for mobile ad hoc networks. In *iTrust*, pages 48–62, 2004.
7. F. Martinelli, P. Mori, and A. Vaccarelli. Towards continuous usage control on grid computational services. In *ICAS/ICNS*, page 82, 2005.
8. F. Martinelli and M. Petrocchi. On relating and integrating two trust management frameworks. In *VODCA*, pages 191–205, 2007.
9. G. Rote. Path problems in graphs. *Computing Supplementum*, 7:155–189, 1990.
10. R.O. Sinnott, A.J. Stell, D.W. Chadwick, and O.J. Otenko. Experiences of applying advanced grid authorisation infrastructures. In *Advances in Grid Computing*, pages 265–274. Springer, 2005.
11. J.D. Sonnek and J.BF Weissman. A quantitative comparison of reputation systems in the grid. In *Gris Computing*, 2005.
12. M. Thompson, A. Essiari, K. Keahey, V. Welch, and S. Lang S. Fine-grained authorization for job and resource management using akenti and the globus tookit. In *CHEP*, 2003.
13. J. Weng, C. Miao, and A. A robust reputation system for the grid. In *CCGRID*, pages 548–551. IEEE, 2006.
14. W. H. Winsborough and J.C. Mitchell. Distributed credential chain discovery in trust management. *JCS*, 11(1):35–86, 2003.