MANAGEMENT OF SECURITY POLICIES IN VIRTUAL ORGANISATIONS

Benjamin Aziz, Alvaro Arenas, Ian Johnson
e-Science Centre, STFC Rutherford Appleton Laboratory, Oxfordshire OX11 0QX, U.K.
benjamin.aziz@stfc.ac.uk,alvaro.arenas@stfc.ac.uk,ian.johnson@stfc.ac.uk

Matej Artač, Aleš Černivec
XLAB d.o.o., Pot za Brdom 100, SI-1000 Slovenia
matej.artac@xlab.si,ales.cernivec@xlab.si

Philip Robinson
SAP, Belfast, Northern Ireland
philip.robinson@sap.com

Keywords: Grid Computing, Security Management, Security Policies, Virtual Organisations

Abstract: Grid-based virtual organisations facilitate the sharing of computational resources among users belonging to different organisations and working towards a computationally-intensive goal within some project. The selection, access, usage and release of such resources is usually controlled through the enforcement of security policies that express what is acceptable behaviour by the resources and their users at each stage. This process is complex to manage in large-scale Grid-based systems, therefore, a solution tackling the management of VO policies is desireable. In this paper, we propose one such solution that provides policy management capabilities at each phase of the VO lifecycle. We discuss full aspects of the solution starting from the context and requirements analysis, use cases, design, implementation and finally, qualitative and quantitative evaluation.

1 INTRODUCTION

Exchange of ideas and cooperation in terms of both human and computational resources provide benefits to research and industry projects. There are many advantages of cooperation between organisations of different types, consisting of universities, research institutions and industries. Many of these organisations own computational facilities composed into Grids, which can be used to their advantage by partners collaborating in formal projects. However, the diversity of systems, the time and effort required to configure them for access from external users as well as administrative constraints all can hinder exploitation of such large-scale computational resources.

To overcome these problems, we work with the notion of a Virtual Organisation (VO) (Foster et al., 2001). A VO is a dynamic entity composed of users and computational resources regardless of which institution or organisation they belong to. A system implementing VOs therefore provides the means to a simple and efficient collaboration beyond the boundaries of the physical organisations. Such a system is only complete when it also provides the mechanisms to assert the policies imposed within the physical organisations and restrictions agreed on in the projects utilising the VOs.

In this paper we propose a solution for managing security policies in VOs. The solution takes into consideration policies that the VO actors, such as its users and its resources, are subjected to outside the VO itself and addionally, it accommodates for additional policies that arise from the nature of VO itself. The main novelty in our solution is that it considers security information at the point of resource selection. This is achieved by applying standard policy enforcement machinery.

The solution takes into account the phases each VO goes through within its lifetime. The implementation follows a set of requirements which make sure that, first of all, the Grid administration staff in the physical organisations will be willing to adopt the system implementing VOs. Considering the security and maintainability of the active resources is critical, this is an important issue to be addressed. Furthermore, the impact of the policy enforcement should not be apparent on the actual usability of the VOs. Finally, the policy system should not enable situations where a user with less rights would gain more access than permitted, or to prevent the access to the system altogether, unless specifically required to do so.

For the proposed VO policy management system

we will systematically show its usage in the scenarios that are common in the lifecycle of the VOs. The policies have their own lifecycle within that of the VOs, hence we present the scenarios accordingly. We have developed our VO policy management system for the XtreemOS operating system[1](Coppola et al., 2008). Our evaluation of the proposed solution is based on our experience with this reference implementation.

The rest of the paper is structured as follows. In Section 2, we give some background discussing the context and the requirements motivating our work. In Section 3, we define the phases of the policy management lifecycle and demonstrate the use cases involved. In Section 4, we introduce our VO policy management system and discuss its design and implementation. In Section 5, we evaluate our service in meeting its requirements and highlight its performance. Finally, in Section 6, we review related work and in Section 7, we conclude the paper giving directions for future work.

2 VIRTUAL ORGANISATION POLICY MANAGEMENT

The VO policy management problem concerns the assurance that policies governing a collection of users, services and resources from different organisations are consistently enforced accross the distributed collaborative environments. This section describes the context of this problem, providing a background on VO lifecycle, security policies, certificates and credentials, and administrative domains. It also discusses the requirements for addressing the problem.

2.1 Context

In XtreemOS, VOs are created for the purpose of enabling distributed collaborations among different organisations such that users from one organisation can access resources owned and administered by other organistions based on a Grid infrastructure. The VO scenario is used to describe the activities and information that compose the context of VO policy management and to motivate our requirements.

2.1.1 Virtual Organisation Lifecycle

The VO lifecycle in XtreemOS consists of four main phases. The *Grid Set-Up and Management* phase involves the setting up of the infrastructure underlying the Grid and populating the Grid with resources and prospective users of those resources. Despite the fact that this phase is the entry point to the lifecycle, it is visited again whenever new resources or users are added to the Grid and existing ones removed. The *VO Creation and Management* phase is concerned with creating VOs on top of the Grid infrastructure and populating those VOs with resources and users. Once a VO has been created and populated, it then becomes operational, which means that the lifecycle moves to the *VO Operation* phase. Users can then order, access and use resources. From the operational phase, it is always possible to perform VO management operations such as replacing VO users and resources. Finally, once the VO has reached its goal and its purpose has expired, it then enters the *VO Termination* phase, in which the VO state is dissolved.

2.1.2 Security Policies in VOs

We define a security policy, $p \in \mathcal{P}$, as a set of tuples $p = (u, r, a, q)$, where $u \in U$ is a user, $r \in R$ is a resource, $a \in A$ is the permission to execute an action on a resource and $q \in Q$ is a set of logical conditions. The policy hence states that u is permitted to apply action a to the resource r under the conditions q such that q must be true. When any of the users, resources, actions or set of conditions are unknown, we use the variables x, y, z to represent their values. In our VOs, we distinguish four categories of policies.

- *User Policies:* User policies are policies that are specified by the VO users in order to determine the suitability of VO resources to run their jobs. The purpose of these policies is to protect user data and jobs. We refer to a user policy as $p_u \in \mathcal{P}$. An example of such policy is "Run all my jobs on resources not shared with competitive users".
- *Resource Policies:* These are security policies set by the local resource administrators to control access and usage to their resources by VO users. We refer to these policies as $p_r \in \mathcal{P}$. An example of a resource policy would be that "this resource is available to the VO users only between the hours 5pm and 7am".
- *VO Policies:* VO policies are specified by the VO owners or managers to determine what is acceptable behaviour in a VO by its resources and users. VO policies are different from user or resource policies in that they provide general rules at the level of the whole VO rather than a specific user or resource belonging to a specific administrative domain. We shall refer to VO policies as $p_{vo} \in \mathcal{P}$. An example of VO policies is that "VO users can only submit a maximum of three jobs per day to the VO resources".

[1] www.xtreemos.org

- *Filtering Policies:* Filtering policies are created from matching user and VO policies. These policies are then evaluated at selection time to ensure that the resources selected are suitable. Filtering policies are refered to in what follows as $p_{fl} \in \mathcal{P}$. An example of a filtering policy would be to intersect the above user and VO policies to obtain the following policy "A user can submit a maximum of three jobs a day and all those jobs must be run on resources not shared with competitive users".

2.1.3 Policy Evaluation Strategies

We assume that there are four points in time at which security policies can be evaluated and enforced, relative to the timeline of computational jobs.

- *Selection-time:* Evaluating policies at this stage occurs at the time when VO resources are being selected by selection services, such as a scheduling service, for job executions. Hence, policies will guide the matching of resources to jobs.
- *Access-time:* Evaluation of policies at this stage represents classical access control, like for example the lattice-based and the role-based access control models (Sandhu et al., 1996).
- *Usage-time:* This stage is used to refer to the runtime control of the usage of resources by the security policies. A recent dominant example of usage-time policy evaluation is the usage control model proposed in (Park and Sandhu, 2004).
- *Release-time:* This stage refers to the evaluation of policies applying once resources have been released by the user and they include all the future obligations on the user (Bettini et al., 2002), such as the cleaning-up of the state of the resource or paying for its usage.

The main focus of the solution we discuss later is on the first two points: selection-time and access-time.

2.1.4 VO Certificates and Credentials

Certificates are tokens of trust that convey credentials about the identity and attributes of the certified entity. In our case, we define certificates as a set, $c \in C$, where $\{|c|\}_K$ denotes that the certificate is signed by the key, K. Certificates carry certain information, as in the case of X-509 certificates, such as the *issuer*, *subject*, *validity* and *attributes*. We shall refer to these whenever necessary using the in-notation, for example, $\{|issuer = CA, subject = Bob|\}_{K_{CA}}$, refers to a certificate whose subject is *Bob*, issuer is *CA* and it is signed by the private key of *CA*. We write private keys as K and their corresponding public keys as K^+.

2.1.5 Administrative Domains

Administrative domains represent divisions of trust and ownership among users and resource administrators and give some structure to the way resources are owned and managed by the owner organisations. We consider three administrative domains, called sites:

- *Resource Sites*: These are the sites in which resources offered to the Grid reside.
- *User Sites*: These denote sites to which users of VOs belong and who will eventually submit jobs to the resources of those VOs.
- *The Core Site*: This is a robust and well-protected site in which the security and VO management services will be running. From the trust point of view, the core site represents the root of trust for both the resource and user sites.

2.2 Requirements

The requirements for VO policy management can be derived from what Foster, Kesselman and Tuecke (Foster et al., 2001) define as the *"Grid Problem": flexible, secure, coordinated resource sharing among dynamic collections of individuals, institutions, and resources*. They state that resource sharing is, necessarily, *highly controlled*, with resource *providers and consumers* defining clearly and carefully *what is shared*, *who is allowed* to share, and *the conditions* under which sharing occurs. Wasson and Humphrey(Wasson and Humphrey, 2003) further suggest that VO policies should be sufficiently concrete in order to describe *actions that will be taken to maintain the desired VO state*. They identify three classes of VO Policy: (i) VO-wide operational policy, (ii) VO policy on resources and (iii) VO policy on users. Taking these past foundations for VO policy management as a basis, along with XtreemOS application-specific requirements, we condense the requirements into six main classes:

R1 **Administrative Autonomy**: organisations, owners and administrators must be able to maintain ultimate control over the usage of resources within their domains. Failure to do so results in political and legal problems such as information disclosure between organisations sharing resources.

R2 **Effective Conflict Handling**: conflicting policies (e.g. policies with the same subject and resource but with opposing permissions, i.e. grant versus deny) must not be present in the VO policy management service at enforcement time. VO policy management must have strategies for pre-enforcement resolution of policy conflicts.

R3 **Maintenance Efficiency**: VO policy management must introduce minimal administrative effort to resource providers and consumers, beyond the protection of their resources, data and jobs. Redundant interactions for dealing with policy or certificate expiration and update must be avoided.

R4 **Least Privileges**: any subject must only be able to access resources in the VO necessary for established or registered responsibilities, roles, tasks and conditions. If the case arises that subjects can access resources for which the access is not justified or sanctioned, they may maliciously or accidentally compromise the resource or make it unaccessible for legitimate subjects.

R5 **Non-intrusiveness**: the overhead required to enforce policies must have negligible impact on the performance of resource requests. Enforcement of policies requires two types of compute-intensive actions: (i) cryptographic operations and (ii) policy evaluation (filtering, data collection, result determination). These actions must be implemented efficiently so as they do not introduce unnecessary checks, operations or interactions that substantially slow the system's performance.

R6 **Expressiveness without Losing Enforceability**: it must be possible to codify rules that govern the behavior of VO users, resource access and usage in such a way that they are consistently and automatically enforceable. We advocate that enforcement of policies at the lowest level of the software stack (the Operating System) is a means of attaining flexibility and expressiveness with strong enforceability of policies.

In the evaluation Section 5, the above requirements *R1-R6* are used to demonstrate the validity of VOPS for addressing the "Grid Problem".

3 VO POLICY LIFECYCLE

In this section, we discuss the management of VO policies in each phase of the VO lifecycle as described in Section 2.1.1. First, however, we start by giving an overview of some of the XtreemOS services used in this lifecycle. The first group of these services are *trust management services* and they include:

- *A Root Certification Authority (Root CA)*: This is a service that creates the main trust anchor in the Grid and facilitates the certification of the identity of other core services within that Grid. This service can be performed offline to avoid compromise of the root private key, or online using the CDA service described next.
- *A Credential Distribution Authority (CDA)*: This service is responsible for distributing identity certificates to users. It is a subordinate of the Root CA and facilitates the separation by managing users' certification only.
- *A Resource Certification Authority (RCA)*: This service is responsible for distributing identity certificates to resources. It is a subordinate of the Root CA and facilitates the separation of concern by managing resources' certification only.

The next group includes the service responsible for managing policies in VOs.

- *A VO Policy Service (VOPS)*: This is the main service here, which is used to manage security policies in VOs as well as enforce those policies at the point of resource selection. We shall explain more in detail the architecture of VOPS in Section 4.

Finally, the last group includes services for managing job submissions and selecting resources, which are necessary in order to illustrate the operation of an active policy management system.

- *The Resource Selection Service (RSS)*: this is a distributed overlay service (Costa et al., 2009) that runs on every resource and that computes specific matching algorithms to decide whether or not the resource is suitable and available to run a job submitted by a VO user. Incorporated in this decision is the decision taken by a local policy decision point, part of the VOPS service, as to whether or not the resource satisfies the filter policy associated with the submitted job.
- *The Application Execution Manager (AEM)*: This is a job management service, which accepts job submissions from users and interacts with the RSS to determine the appropriate resources matching the job. The AEM then executes the job on the selected resources or may prompt the user to make the final decision.

Next, we describe the policy management lifecycle phases, which involve all the above services.

3.1 Grid Management

In this phase, site administrators willing to offer their site resources to the Grid advertise the general site and resource policies with the VOPS. At this stage, there are no VOs formed yet, so these policies will be generic enough at the level of the Grid.

3.1.1 Site Registration

During this phase, site administrators at individual sites register their sites with the VOPS as shown in

the scenario of Figure 1.

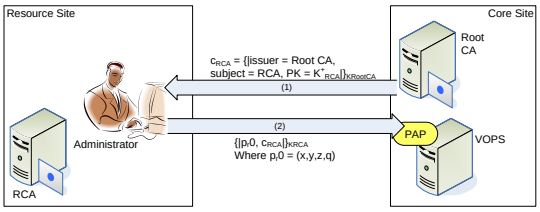

Figure 1: Adding a site domain to the Grid.

We assumed that prior to this step, the site administrator will have contacted the Grid administrators at the core site to perform a *vetting process*, in which the physical identity of the administrator is authenticated as well as any attributes related to their site.

After this vetting process, the site administrator (whom we assume also to be the administrator for all resources belonging to his site) will first register his site, by means of registering the site's RCA, with the Root CA trusted at the core site running the VOPS. This registration process results in the Root CA returning a certificate $c_{RCA} = \{|issuer = Root\ CA, subject = RCA, PK = K^{+}_{RCA}|\}_{K_{Root\ CA}}$. The certificate binds the site's RCA to its public key K^{+}_{RCA}. This certification of the RCA is needed by VOPS to authentically associate the site with its policies.

The site's administrator then submits to the VOPS the default site's policy with regards to all Grid users, which includes rules on how to access and use the site's resources and the possible actions that can be performed on them. This policy is written as $p_r0 = (x,y,z,q)$, which only outlines the set of conditions, q, under which any Grid user attempting to perform any actions on any of the site's resources will be admitted. This policy is general in the sense that it does not specify any users, resources or actions at this stage and so it should cater for all future unknown VOs created within the Grid. The policy is the safest policy though other policies can also be specified. The association of p_r0 with c_{RCA} establishes the trust path with the Root CA.

3.1.2 Site Population

Once the resource site has been registered with the core site, it is then possible for the RCA to issue certificates to the individual resources in the site binding them with public keys of each resource and to inform the VOPS with the specific VO policy for the resources. This scenario is called *site population* and it is shown in Figure 2.

In the first interaction, the RCA issues a certificate $c_r1 = \{|issuer = RCA, subject = r, PK = K^{+}_{r}|\}K_{RCA}$,

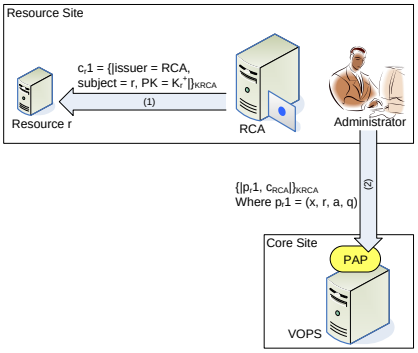

Figure 2: Adding a resource to the RCA.

which binds the resource r to its public key K^{+}_{r}. This allows r to be authentically identified in any future interactions assuming it does not leak its private key to unauthorised users.

In the next step, the site administrator creates a specific policy, p_r1, for the resource and submits it to the VOPS along with the certificate identifying the site, c_{RCA}. $p_r1 = \{x,r,a,q\}$, refers to the set of conditions, q, under which r will allow action a to be applied to it by any VO user x. This policy is more specific than p_r0 in that it refers to a specific resource in the site and the specific actions that could be applied to that resource.

3.1.3 User Registration

In parallel to the registration and population of resource sites, users can also be registered in the Grid as shown in Figure 3.

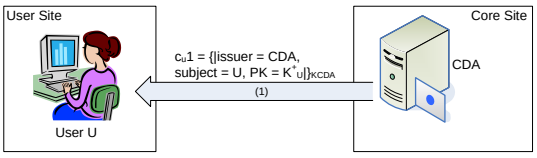

Figure 3: User Registration in the CDA.

However, from the perspective of VOPS, no user policies are introduced at this stage to its database. The reason is because in practice, users may have VO-specific policies to specify the conditions under which their jobs are run, however, they would rarely specify Grid-wide policies. The interaction will allow the user to obtain a certificate from the CDA, trusted by VOPS, to associate her with her public key. This certificate will be used in the next phase of VO creation and management to allow the user to include her VO policies in VOPS.

3.2 VO Creation and Management

After the previous phase, the Grid is assumed to have been set-up and it is ready to be utilised. Therefore, in this next phase, VOs will be created and populated with Grid sites, resources and users.

3.2.1 VO Creation

A user, already registered in the Grid, will create a VO as is shown in Figure 4.

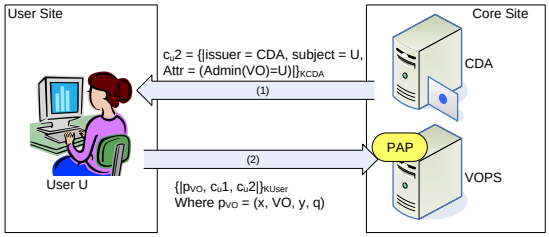

Figure 4: Creating a VO and Registering its Policies.

In order to achieve this, the user needs to obtain an attribute certificate c_u2 from the CDA stating that the user is the administrator of the VO. This certificate along with c_u1 is sent in the next interaction, which registers the VO policy of the new VO with VOPS. The two certificates together prove the identity of the user as well as the fact that the user is the VO administrator. The VO policy, $p_{VO} = (x, VO, y, q)$, states all the acceptable conditions q under which the VO must operate in the future. Example of such conditions include acceptable VO load-balancing conditions.

3.2.2 Adding Sites to a VO

After registering a site in the Grid, the site administrator may wish to include their site in some VO. This operation of adding sites to a VO is shown in the interactions of Figure 5.

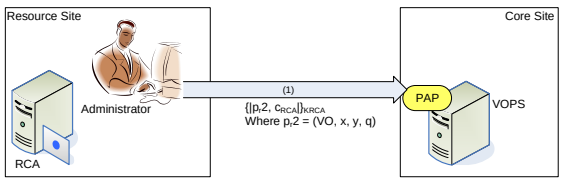

Figure 5: Adding a Site to a VO.

The site administrator sends the new site policy, $p_r2 = (VO, x, y, q)$, to VOPS along with the proof of the identity of the administrator. The policy gives the acceptable conditions, q, under which any users of the VO can access any of the resources offered by the site.

3.2.3 Adding Resources to a VO

Once a site administrator has joined their site to a VO, they can add site resources to the VO under a new policy, p_r3, as shown in Figure 6.

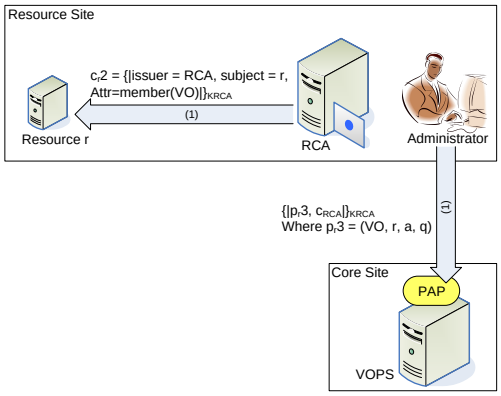

Figure 6: Adding a resource to a VO.

Prior to that, the local RCA at the resource site will issue the resource with an attribute certificate to certify the fact that the resource is now in the VO. This certificate is necessary for future resource selection processes. The policy p_r3 provides the acceptable conditions, q, under which r can be accessed using actions a by users of the VO.

3.2.4 Adding Users to a VO

Similar to the addition of sites and resources to a VO, users need to join VOs before they can utilise the VO resources for their jobs, as is shown in Figure 7. In

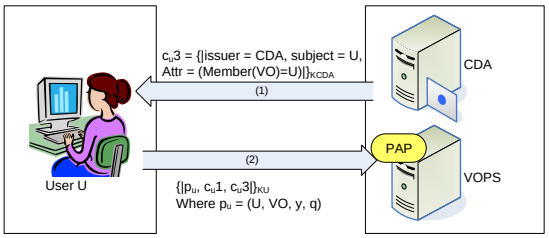

Figure 7: Joining Users to VOs.

this case, the first interaction allows the user to obtain an attribute certificate from the CDA to certify that the user is indeed a member of the VO. This certificate will allow the user to submit jobs in the future to VO resources. The user also can specify her policy,

$p_u = (U, VO, y, q)$, specifying the conditions q under which the user's jobs, may be computed over the VO resources. This policy is signed by the user and associated with the user's certificate showing her identity and the fact that it belongs to the VO. The policy p_u, which will be used at the operational phase to create the filtering policy, allows VOPS to determine the suitability of resources to run the user's jobs. For example, the resource may not have the minimum assurence level or QoS attributes.

3.3 VO Operation

The main functionality of VOPS during the operational phase of VOs is to aid the RSS in its resource selection process by making sure that only resources whose policies match the filtering policies in VOPS are selected for job executions. This operational phase is shown in Figure 8.

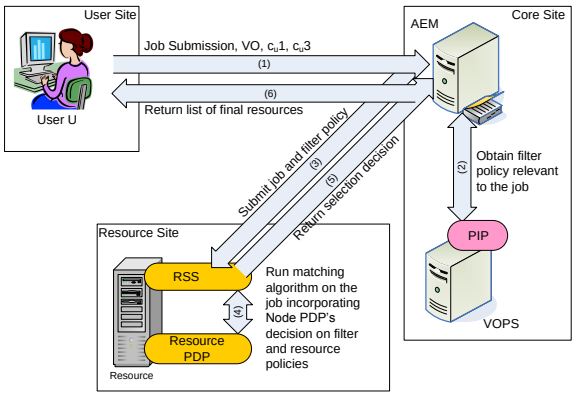

Figure 8: Resource Selection during VO Operation.

In this phase, the user submits a job to the AEM along with certificates proving her identity and the fact that it belongs to the VO. The job submission is then used to obtain, from VOPS, a copy of the relevant filtering policy (i.e. the policy union of the user and the VO policies.) Once this is done, the AEM then submits the job along with the filtering policy to the RSS, which is running on every resource in the VO. The RSS evaluates, using internal selection and matching algorithms, whether the resource can or cannot offer itself to run the job. This will depend on static and dynamic properties of the resource.

Part of this decision also is the decision from the local resource PDP, which will evaluate the request's attributes against the local and the filtering policies and return a security decision as to whether the resource is suitable or not to run the job. Once the final decision has been computed by the RSS, it returns the decision to the AEM, which in turn returns the list of all resources that have been selected to the user or alternatively, will schedule the job directly to run on a subset of those resources.

3.4 VO Termination

The final phase in the policy management lifecycle is the termination phase, which involves the VO administrator informing the VOPS of the termination of the VO that she owns, as shown in Figure 9.

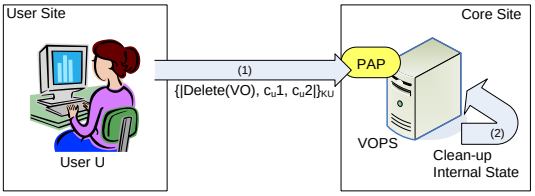

Figure 9: Termination of VOs.

In the first interaction, the VO owner needs to prove her identity and provide the attribute certificate showing that she owns the VO. Once these certificates are validated by the VOPS, the VOPS then proceeds to clean-up its internal state by removing policies associated with the VO.

4 VOPS: A VO POLICY SERVICE

4.1 Architecture

The VO Policy Management Service (VOPS) is structured logically as shown in Figure 10.

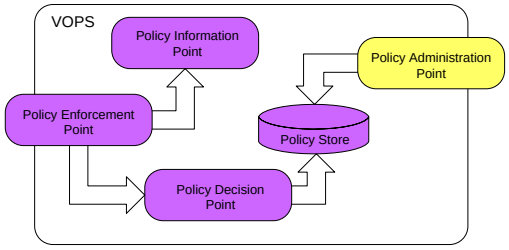

Figure 10: The structure of VOPS.

- *Policy Enforcement Point (PEP):* The PEP is where the users requests are intercepted in order for these requests to be checked and appropriate decisions enforced on the requests. The users requests may carry user credentials regarding their attributes and the attributes of the context.

- *Policy Decision Point (PDP):* The PDP is the component which enforces the security policies on user requests. The PDP contains the logic applied to policies and users' requests. There is one component of the PDP, the Resource PDP, which runs locally at the resource and assists the RSS during the selection process.
- *Policy Information Point (PIP):* The PIP is the component of VOPS that queries attributes about the request arriving from a user, additional user credentials and information about the context of the request (e.g. time and location).
- *Policy Administration Point (PAP):* The PAP allows site administrators and users to add, delete and update site, resource, VO and user policies in the policy store.
- *Policy Store:* The Policy Store is a database containing all the relevant policies.

4.2 Implementation

The implementation of VOPS includes a number of classes, as shown in Figure 11.

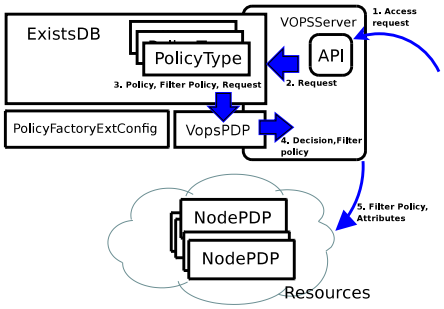

Figure 11: Dataflow in VOPS.

- *VOPSServer:* This is the main class containing server logic providing VOPS API. The API provides the enforcement point (where access requests are being processed) as well as methods for creating and removing policies, adding policy rules and creating filter policies.
- *EXistsDB:* As mentioned above, the PAP module accepts policies from resource administrators and makes them available to the PDP. The PAP therefore holds policies in an internal database implemented with eXist[2]. eXist is an XML database which provides easy and fast search engine over policies stored in XACML format. EXistsDB is an entity interfacing VOPSServer and providing access point to administrators and users.

[2]http://exist.sourceforge.net

- *PolicyType:* This is a collection of policies from a domain. It provides access to essential operations over policies e.g. listing, adding, removing. EXistsDB encapsules PolicyType as fields.
- *PolicyFactoryExtConfig:* This is a configuration class containing settings from a configuration file.
- *NodePDP:* This is a class used by resource selection services during selection process. Filtered policies are provided to selection service where enforcement is done in time. Subject, resource and action attributes are provided with corresponding certificates and job description file.
- *VopsPDP:* This class provides implementation of the decision engine. It constructs Policy Decision Point with different modules provided by the XACML implementation.

The implementation of the above classes was carried out in Java and uses Bouncy Castle provider[3] library for handling certificates. Policy handling is achieved with Sun's XACML implementation library[4] extended with modified PDP module. The service is hosted within a staging framework and messaging bus developed in XtreemOS. eXist is used as XML database which provides efficient way for storing XACML policies and provides XQuery processing for easier information extraction from policies stored in the database.

5 EVALUATION

The evaluation of the VOPS service is carried under two main criteria: The first is to determine how well the service meets the requirements set in Section 2.2, and the second is its performance overhead.

5.1 Meeting the Requirements

5.1.1 R1: Autonomy

Any system running on top of the existing Linux machines with the purpose of providing access to external users must first assure that the policies imposed by the resource administrator, who is the actual owner of the hardware are enforced first. We have built into VOPS this requirement by providing a hierarchy of policies by their type. This means that the resource policy will take precedence in the evaluation on VO or user policies.

[3]http://www.bouncycastle.org
[4]http://sunxacml.sourceforge.net

5.1.2 R2: Conflict Handling

In our VOPS implementation, we consider the conflicting policies in the design time already. Possible conflicting situations can arise only within a specific policy (i.e., between the rules that compose a policy) and not between policies themselves thanks to the hierarchical structure of the database. Since XACML is used to represent policies, **rule combining algorithms** are used at a policy level to automatically resolve conflicts among policy rules (Shu et al., 2009). Least privileged policies are user policies (each user of a specific VO possesses one user policy) and are considered in parallel with VO policies. VO policies are provided by VO administrators and have therefore higher privileges than user policies. Resource policies are considered at the last stage of resource selection through filtering policies and these comprise rules of all three kind of policies conforming with attributes presented with user, resource and job information.

5.1.3 R3: Maintenance efficiency

The VOPS provides a set of policy database operations, including policy deletions, insertions and policy retrieval, each available to user, VO and resource domains. The policies are referenced by their unique ID, reducing unnecessary traffic for deletions and modifications. Building up a policy consists of adding rules that encapsulate the conditions in the smallest amount of data. Furthermore, entities of the system request policy filters, which, in turn, contain only the relevant policies in a single compact form.

5.1.4 R4: Least privileges

High flexibility of the VOPS based on the XACML specification shows, in fact, that it is possible to provide either positive or negative policy definitions. In practice this will mean that the system, upon its bootstrap, will have a built-in policy denying any action to any actor. Only by adding new policy rules is it possible to enable the actions to the specified actors.

5.1.5 R5: Non-intrusiveness

To provide non-intrusiveness in a system, the design should be such that it reduces the load on a particular PDP. One way to achieve this is to have a central PIP, but to distribute the PDP to the entities that are subject to decision. In the XtreemOS reference implementation, the PDP is used during the resource discovery, and the resulting list must comply with the policies. This is a distributed operation that employs a peer-to-peer paradigm. We are thus able to utilise the filtering policies evaluated on the nodes themselves. As a result, handling the policies becomes a part of the operations that take place in any case, contributing to a negligible overhead.

5.1.6 R6: Expressiveness without losing Enforceability

Four types of policies are defined within our system: user, resource, VO and filtering policies. Rules in each type of policies can conform to any of the relevant user, resource or VO attributes provided through the use of certificates and job description. Due to this fact, any scenario concerning user-VO, user-resource and VO-resource relations can be expressed through the aforementioned policies within our system.

5.2 Performance Evaluation

Our reference for performance evaluation is a detailed comparison between different XACML engines described in (Turkmen and Crispo, 2008) where the choice of the Sun's XACML engine over other alternatives is justified. We add our results of timing submissions of generated user requests to VOPS PDP in the same manner, where the requests contain a varying number (10, 100, 1000 and 10000) of generic policies consisting of four rules of which three of them deny the access to the resource and one of the rules always permits the access. First policy conforming to the request and denying the action (no permissive rule resides in the policy) stops the whole evaluation by denying the evaluation. In order to time the evaluation of all policies in the storage, a permissive rule exists in each policy and, therefore, each policy conforms to the request.

The results in Figure 12 show that we introduced some overhead to the response times in our implementation consisting of the eXist engine for storing the XACML policies and the XtreemOS service staging the communication bus.

On the other hand, we gained the ease to manage the policies, extract the filter policies from the database and separate the communication part from the core of the service. The blue graph presents average times for XACML engine's PDP, the green graph presents times obtained by authors in (Turkmen and Crispo, 2008), which is shown as a comparison. The red graph presents average times of first user's access request evaluation, i.e., when VOPS is started. In case of 10000 policies, not taking into account the first access to PEP, which is the most expensive, we obtain average time of 174 ms with standard deviation of 88 ms (50 tests were performed). By that we get close to the desired blue graph with small addition due to

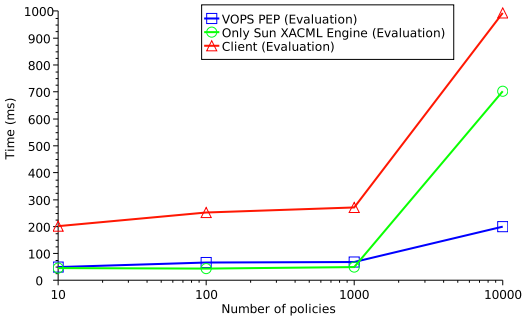

Figure 12: Times to evaluate user request with large number of policies in PDP.

usage of communication framework and network latency of around 40 ms.

Figure 13 depicts how much time is consumed for saving the policies to database and loading different amount of policies from database into main memory preparing the policies for evaluation.

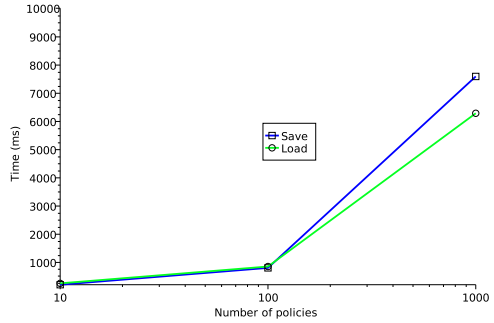

Figure 13: Time for loading large number of policies and PDP construction (green line) and time for saving policies from memory to database.

6 RELATED WORK

Standard policy management for Grids is specified in the OGF Security Architecture for Open Grid Services (OGSA) (Nagaratnam et al., 2003), describing how Grids can use Web services techniques to specify, publish and enforce security policies. Each node can use standard authorisation languages such as WS-Policy and XACML to specify its policies. Our approach is aligned with the OGSA specification; however, it differs in that we consider policy information when selecting resources. Most Grid systems use job-description languages to express resource attributes and perform matches between job requests and available resources at selection time. The primary focus is on attributes such as computational power and storage capacity, and thus they handle access control only during job execution. In our work, we argue that access control should be considered also in the resource selection process during the VO operation phase.

There have been other attempts to use security information at resource-selection time. (Mazzoleni et al., 2009), integrates policy information with resource selection, using XACML as the policy language. They compare a VO policy with a local resource policy in order to determine compatibility by applying policy-matching algorithms. Their technique differs from ours in that we do not use policy matching, instead we apply the classical machinery for policy evaluation/enforcement at selection time in order to determine whether a resource could enforce a VO policy. Application of numerical trust metric to integrate trust into resource management is presented in (Arenas et al., 2008). The limitation of such approach is that even though the idea of reducing and simplifying the notion of trust to a number is appealing, many aspects of trust in the real world are qualitative rather than quantitative.

For the evaluation and enforcement of policies at access time, we have followed the standard approach to access control (Chakrabarti, 2007), as other systems like PERMIS (Chadwick et al., 2008). Those systems differ in the language used to specify authorization (i.e., SAML, XACML, or ad-hoc language), the authorization model (i.e., pull or push), and functionalities (i.e., support for PDP only or both PDP and PEP).

7 CONCLUSION

We presented in this paper a VO policy management system, which provides capabilities for the management of security policies in Grid systems along a lifecycle including phases related to the setting-up of Grids, creation and management of VOs as well as their operation and termination. The system facilitates the selection of resources by evaluating policies specified by the site and resource administrators, users and VO owners.

There are several directions for future work. One problem we see is a lack of additional policy check at access time on the resource as described in this article. To solve this issue, some revocation mechanism should be incorporated into the system to assist with

the decision just before user's request is executed on the resource. We believe there are several possibilities to tackle this issue, however, more thorough and stringent checks are done on expense of time and communication complexity. Moreover, a threat analysis is needed to expose any vulnerabilities that VOPS may be susceptible to. One type of such vulnerabilities could arise from the problem of compromised certificates. Another interesting line of work is to experiment with the application of the VOPS system to the domain of Cloud computing. This will be based on the use of XtreemOS as an operating system enabling virtualisation platforms for Cloud service providers, as was recently envisioned by Morin et al. in (Morin et al., 2009). Finally, VOPS could be enhanced to construct dynamic enforcement mechanisms capable of enforcing runtime-based usage control policies.

8 ACKNOWLEDGEMENTS

This work was funded by the European FP6 project XtreemOS under the EC contract number IST-033576. The VOPS implementation was partially financed by the European Union under the European Social Fund.

REFERENCES

Arenas, A. E., Aziz, B., and Silaghi, G. C. (2008). Reputation Management in Grid-based Virtual Organisations. In *SECRYPT 2008, International Conference on Security and Cryptography*, pages 538–545.

Bettini, C., Jajodia, S., Wang, X., and Wijesekera, D. (2002). Obligation Monitoring in Policy Management. In *POLICY '02: 3rd International Workshop on Policies for Distributed Systems and Networks*. IEEE Computer Society.

Chadwick, D. W., Zhao, G., Otenko, S., Laborde, R., Su, L., and Nguyen, T.-A. (2008). PERMIS: A Modular Authorization Infrastructure. *Concurrency and Computation: Practice and Experience*, 20(11):1341–1357.

Chakrabarti, A. (2007). *Grid Computing Security*. Springer.

Coppola, M., Jégou, Y., Matthews, B., Morin, C., Prieto, L. P., Sánchez, O. D., Yang, E., and Yu, H. (2008). Virtual Organization Support within a Grid-wide Operating System. *IEEE Internet Computing*, 12(2):20–28.

Costa, P., Napper, J., Pierre, G., and van Steen, M. (2009). Autonomous resource selection for decentralized utility computing. In *29th International Conference on Distributed Computing Systems (ICDCS)*.

Foster, I. T., Kesselman, C., and Tuecke, S. (2001). The Anatomy of the Grid - Enabling Scalable Virtual Organizations. *International Journal of High Performance Computing Applications*, 15(3):200–222.

Mazzoleni, P., Crispo, B., Sivasubramanian, S., and Bertino, E. (2009). Efficient Integration of Fine-Grained Access Control and Resource Brokering in Grid. *The Journal of Supercomputing*, 49(1):108–126.

Morin, C., Jégou, Y., Gallard, J., and Riteau, P. (2009). Clouds, A New Playground for the XtreemOS Grid Operating System. *Parallel Processing Letters (PPL)*, 19(3):435–449.

Nagaratnam, N., Janson, P., Dayka, J., Nadalin, A., Siebenlist, F., Welch, V., Tuecke, S., and Foster, I. (2003). Security Architecture for Open Grid Services. OGF Document.

Park, J. and Sandhu, R. (2004). The UCON_{abc} Usage Control Model. *ACM Transactions on Information and System Security*, 7(1):128–174.

Sandhu, R. S., Coyne, E. J., Feinstein, H. L., and Youman, C. E. (1996). Role-based Access Control Models. *Computer*, 29(2):38–47.

Shu, C., Yang, E., and Arenas, A. (2009). Detecting Conflicts in ABAC Policies with Rule-Reduction and Binary-Search Techniques. In *Policy 2009: IEEE International Symposium on Policies for Distributed Systems and Networks*. IEEE Computer Society.

Turkmen, F. and Crispo, B. (2008). Performance Evaluation of XACML PDP Implementations. In *SWS 2008: ACM Workshop on Secure Web Services*, pages 37–44. ACM.

Wasson, G. and Humphrey, M. (2003). Toward explicit policy management for virtual organizations. In *POLICY '03: Proceedings of the 4th IEEE International Workshop on Policies for Distributed Systems and Networks*. IEEE Computer Society.