Monitoring Contract Enforcement within Virtual Organizations

Anna Squicciarini[1] and Federica Paci[2]

[1] College of Information Sciences and Technology
The Pennsylvania State University
asquicciarini@ist.psu.edu
[2] Computer Science Department
Purdue University
paci@cs.purdue.edu

Abstract. Virtual Organizations (VOs) represent a new collaboration paradigm in which the participating entities pool resources, services, and information to achieve a common goal. VOs are often created on demand and dynamically evolve over time. An organization identifies a business opportunity and creates a VO to meet it. In this paper we develop a system for monitoring the sharing of resources in VO. Sharing rules are defined by a particular, common type of contract in which virtual organization members agree to make available some amount of specified resource over a given time period. The main component of the system is a monitoring tool for policy enforcement, called Security Controller (SC). VO members' interactions are monitored in a decentralized manner in that each member has one associated SC which intercepts all the exchanged messages. We show that having SCs in VOs prevents from serious security breaches and guarantees VOs correct functioning without degrading the execution time of members' interactions. We base our discussion on application scenarios and illustrate the SC prototype, along with some performance evaluation.

Keywords: Virtual Organizations, Monitoring, Access Control, Collaboration.

1 Introduction

The Internet and the Web have enabled new ways for users, enterprises, and organizations to collaborate in a large number of application domains–from service provisioning and e-commerce to collaborative e-learning, entertaining, and cultural heritage. Virtual organizations (VOs for short) represent a new collaboration paradigm where the participating entities (enterprises or individuals) pool resources, services, information, and knowledge in order to achieve a common goal. Researchers as well as practitioners have recognized the advantages of VOs and have explored a number of possible approaches to facilitate their formation and management, especially in grid computing systems [2,4].

VOs vary tremendously, according to the specific goals, context and infrastructure. As such, a single architecture is not sufficient to fit all the possible organization types. Nevertheless, despite the many differences among VOs, research studies have identified

E. Bertino and J.B.D. Joshi (Eds.): CollaborateCom 2008, LNICST 10, pp. 563–577, 2009.

a broad set of common concerns and technology requirements. For example, Foster et al. in [4], identify a number of relevant requirements, the first of which is the need for highly flexible sharing relationships. They also pointed to the need of sophisticated and precise levels of control over how shared resources are used, including fine-grained and multi-stakeholder access control, delegation, and application of local and community policies and issues of quality of service, scheduling, and accounting. With regards to issues related to access controls, several approaches have been proposed [10,11,14].

A widely adopted approach, that we take into account in this work, is to share services and/or resources according to a set of policies of two main types, global to the VO and local to the VO members. The VO community policies are specified in the VO *contract*, representing the collaboration agreement established among all the VO members. Members join the VO aware of the contract regulating the system, and define their own local policies accordingly. The local policies represent the plan according to which the VO members will collaborate within the VO, and they must be compliant with the VO contract. Because many relevant operations by members depend on such policies, monitoring that they are properly enforced is crucial for the development of safe and well grounded VOs. In fact, it is possible that, unless strict enforcement mechanisms are in place, VO members do not act as stated and thus violate such contract.

Previous work has tackled the issue of policy enforcement within VOs from different angles, resulting in new access control mechanisms [10,11] and monitoring systems [9,14]. However, most of the proposed approaches either rely on a centralized entity, causing bottlenecks and scalability issues, or they are specific to a given domain, and cannot be easily applied to others.

In this paper we propose a new decentralized mechanism to monitor VO members' behaviors and compliance with global policies. Our work builds on a framework for secure specification and distribution of VO contracts previously proposed by us [9,11]. The framework includes a model for specifying VO contracts and policies of VO members, along protocols addressing integrity and confidentiality of the policy publication process [11], and is completed by the protection mechanism discussed in this paper.

The core component of our solution introduced in this paper is the Security Controller (SC), a fully implemented monitoring system, developed using SOA architecture. The SC has a number of interesting features. First, the SC enables secure distribution of the VO contract in a private fashion and ensures that VO members' local policies are compliant to the VO contract. Second, it monitors messages among VO members and ensures that entitled members receive the services as promised by the provider members; third it ensures non-repudiation, in that VO members cannot deny having receiving a request or a certain service and claim for additional ones (if not entitled). The SC is realized in a completely decentralized fashion, in that each VO member has an associated SC, which verifies if service requests are to be satisfied, To the best of our knowledge this is the first comprehensive solution for access control policy specification, monitoring and enforcement specific to VOs.

The paper is organized as follows. Section 2 introduces the concept of VO contract, while Section 3 overviews the main phases of the VO lifecycle. Section 4 describes the VO creation process and how the rules specified in the contract are enforced. Section 5 describes the SC architecture, possible application scenarios and some relevant

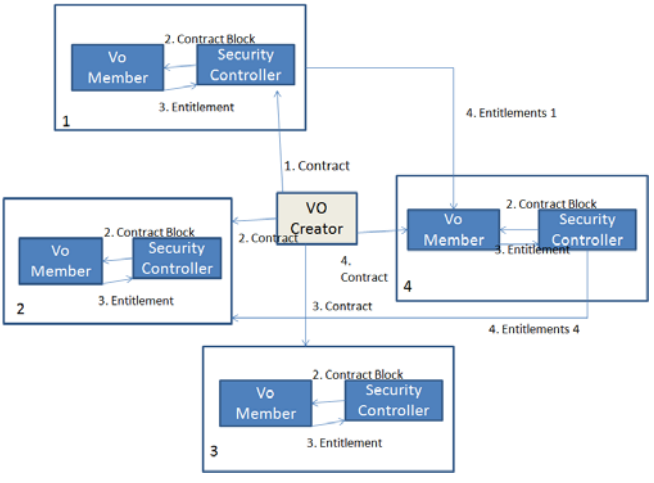

Fig. 1. Contract Publishing and Entitlements Distribution

implementation details. It also shows results of experiments we carried out to benchmark the overhead of the system. Section 6 gives an overview of related works. Section 7 concludes the paper.

2 Contract Specification and Local Policy Representation in Virtual Organizations

In this section we present an overview of a framework for regulated sharing of computational resources in a Virtual Organization[9] (VO). The type of organization we focus on is characterized by VO members, denoted as $VoSet$ representing groups of end users, also referred to as enterprises. We do not make any assumption about enterprise internal regulation and structure, and assume that the entities interested in participating to the VO are represented by service providers (SP). In a VO, different $VoSet$ members are pooled together for sharing a set of resources, referred to as $ResSet$, according to a community policy, also referred to as a *contract* among VO members.

The building block of a contract is represented by obligations. Obligations dictate exactly when and for what amount a member has the right to use the resources provided by the other members and the obligations a member has in terms of resources it has to provide to other community members. Precisely, we represent an *obligation* by a logic predicate of the form $Obl(VoM, VoS, R, I)$, where $VoM, VoS \in VoMSet$, R is a resource in $ResSet$ and I is a temporal expression. A VO member VoM has to make available to members VoS a total amount of a resource R over a time interval I. The resource can also be made available to all possible members, in which case Vos is omitted. We represent contracts in terms of two building blocks: the *obligation sequence* and the *contract block*.

An obligation sequence is a sequence of obligations $Obl(VoM, Vos, R_1, I_1,\ldots,R_k, I_k)$, such that $R_1,...,R_k$ are all instances of the same resource type and each $I_i, i = 1,\ldots,k$, is an ordered, contiguous time interval. Each contract block lists a set of obligation sequences, in order to allow the bearer VoM to choose which component of the obligation sequences to fulfill. A contract is a finite set of contract blocks $\{CB_1,...,CB_n\}$ $(n \geq 1)$, specified in such a way that it is not possible for a resource bearer to fulfill several obligations at once.

Each VO member, upon joining the VO, agrees on sharing its resources with other partners. The starting point for the fulfillment of a VO contract by a VO member is the publishing of a local policy for its resources. Indeed, the VO member keeps control of its resources and it autonomously decides which is the policy to use for them. As such, a local policy serves two main purposes: specifying how the VO member intends to grant access to its resources and, second, publishing the VO member's plan to comply with its obligations in a contract. Vo members define such local policies as strong permissions to a set of entitled members, denoted by expressions of the form $Ent_{VoM_k}(VoM_i, R, I)$. By publishing an entitlement the VoM_k promises that it will make R available over time interval I to the VO member VoM_i. The local policy is publicly available for other VO members. A local policy has to comply with community policies, that is, with the obligation sequences of the contract blocks referring to the resource bearer.

We say that a set of entitlements *Ep complies* with an obligation $Obl(VoM, VoS, R, I)$ if, for every enterprise $E \in VoS$ and every time point $t \in I$ there exists an entitlement $Ent'(E, R', [t_s, t_e])$ such that $t_s \leq t \leq t_e$ and $R \subset R'$. We denote with $R \subset R'$ as two comparable instances of a certain resource of the same type, wherein R is less than or equal to R' according to a given metric. Finally, we say that an access request is *supported* if there is a corresponding entitlement granting it.

Example 1. We consider a VO called Genome where the members are enterprises Hospital of Chicago (HPC), Department of Biochemistry California University (UBC), Hospital of Seattle (HSE), Department of Computer Science of University of Illinois (CSI) and cooperate for a project whose goal is to study the structure of human genomes. The resources to be shared are two databases conveying human genome data, called DB_1 (provided by the HPC) and DB_2 (provided by the HSE) and two servers $Serv_1$ and $Serv_2$ (provided by the CSI) where special software implementing dynamic programming techniques to elaborate data runs. Further, assume that DB_1 and DB_2 can be accessed either with read(r), or write(w) option, while $Serv_1$ and $Serv_2$ can be accessed for running (e) existing software or for updating(u)/deleting (d) software components[1]. The following is HPC's contract block: CB_1= ($Obl(HPC, \{HSE, UBC\}, \{DB_1, w\}$, [(1-05-2007,10:00:00),(5-05-2007,22:15:00)]). Once received CB_1, HPC defines its own local policies by specifying the following set of entitlements: $Ent_{HPC}(HSE,$ $\{DB_1, r\}$,[(1-05-2007,10:00:00),(1-05-2007,15:10:00)]),
$Ent_{HPC}(HSE, \{DB_1, a\}$,[(1-05-2007,15:20:00),(1-05-2007,22:15:00)]),
$Ent_{HPC}(UBC, \{DB_1, r\}$,[(1-05-2007,13:16:00),(2-05-2007,15:10:00)]),
$Ent_{HPC}(UBC, \{DB_1, r\}$,[(2-05-2007,15:11:00),(5-05-2007,21:36:00)]).

[1] Here, we assume that access rights follow the following order: $r < w$ for databases, whereas $e < u < d$ for server management.

3 Virtual Organization Lifecycle

In order to discuss the role of SCs in the VO management, we summarize the main phases in a VO lifecycle.

- *Preparation for participation in the VO*. This is a preliminary phase and reflects the necessary steps that a SP has to take in order to participate to the VO. SPs publish their resources' functionalities in a public repository. The resources' description provides detailed information about resources' capabilities, the resources' interaction means and other information like the resource quality. This information allows one to select a SP for inclusion in the VO.
- *Identification*. This phase is considered as the first major phase in the VO lifecycle and starts when an organization, referred to as VO Creator, identifies a business goal and thus defines a contract to fulfill it. The contract, as discussed in the previous section, states the roles and the requirements that each member has to fulfill in order to be part of the VO.
- *Formation*. The VO Creator queries public repositories to retrieve the information published during the Preparation phase. The Creator uses such information to select a set of potential VO members that match the contract's requirements. The VO Creator then sends them an invitation to join the VO containing the terms of the contract they have to fulfill. If they accept the invitation, they become members of the VO.
- *Operation*. Once the VO is set up, its members cooperate according to the collaboration rules specified in the contract. The operation phase has several critical security issues. VO members may exploit their privileges and misuse the resources available, gather information about other enterprises for personal gain or fail to fulfill the contract rules, and even take advantage of the resources made available to perpetrate crimes. All the interactions must be monitored, ruled by security policies and any violation must be notified.
- *Dissolution*. This phase takes place when the objectives of the VO have been fulfilled. The VO structure is dissolved and final operations are performed to nullify all contractual binding of the VO members.

4 Security Enhanced Virtual Organization Lifecycle

The SC monitors and coordinates the main operations of each of the VO lifecycle's phases. In particular, the SC has two main functions 1) during the creation phase, it distributes the VO contract in a selective manner and it controls compliance of the VO member's local policies, and 2) during the operational phase it monitors VO member's message exchanges to detect contract breaches. We elaborate on such functions in the remaining of this section.

4.1 Virtual Organization Creation and Contract Distribution

In this section we focus on the main operations of the VO creation phase in the in case the SC is set up. The SC interleaves with the conventional contract distribution

and policy publishing operations, so to ensure that the VO members access only the correct amount of data and that the members publish local policies compliant with the VO contract.

Upon defining the contract, the VO Creator is in charge of setting up the VO. It thus invites a number of potential members, selected on the basis of the services (or resources) they could offer to the VO. Potential members are selected based on the information related to the provided resources made available during the preparation phase.

If the potential member accepts, the VO Creator requests to a Certificate Authority (CA) to issue an identity certificate to it. This certificate is typically encoded by an X.509 credential that binds the VO's public key to the VO member identity. At this point, in conventional VO systems, the contract is distributed and the related local policies published by the new VO members. To ensure the correctness and compliance of VO contracts, we require the VO Creator to publish a list of endpoints where the SC components are up and running[2]. Each VO member then selects the SC that will monitor its message exchanges during the whole VO lifetime. The same SC can be the monitor of more than one VO member. Once all VO members have been identified, the VO creator distributes the contract to the VO members through a software component that selectively distributes obligations and entitlements [1]. Distribution is performed in such a way that members will access only the obligations of which they are bearers and the entitlements that grant them the access to other VO members' resources (Figure 1-step 1).

Each SC component facilitates the tasks of the VO members, in that it intercepts the contract and forwards to the associated VO member only the contract blocks of its interest (Figure 1-step 2). Upon receiving the contract blocks, the VO member defines the set of entitlements specifying how it will fulfill its obligations towards the community (Figure 1-step 2) and returns them to the SC component (Figure 1-step 3). The SC is in charge of verifying that the entitlements fulfill the member's obligations, and of distributing these entitlements (Figure 1-step 4) to the guaranteed members-identified by the VOs field of the entitlement (see entitlement definition in Section 2). Once obligations and entitlements have reached all the respective VO members, the VO set up process is complete and the VO enters the operational phase. SC components play a crucial role in this phase of the VO lifecycle: they check that their VO member submits only requests supported by entitlements, and in case of VO provider members that the provided resources respect the entitlements in place.

We elaborate on how SC components guarantee that there are no VO contract breaches in the subsequent section.

4.2 Contract Enforcement and Monitoring

Within the operational phase, the VO members' interaction consists of resources and services provided upon request, according to the specified entitlements. Intuitively, if no monitoring system is in place, a number of security breaches could arise. For example, a VO member could simply deny supported requests, or it could falsely claim of not having received the requested services. A trivial solution to avoid these issues would be

[2] SCs are by assumption trusted software components that run on trusted platforms.

to rely on a centralized controller. However, this would likely result in delays and the centralized monitor be a bottleneck. When the SC components are employed, instead, these type of situations are prevented without causing significant overhead.

The interaction among VO members can proceed with no third party in between. When a request is rejected, two different -and trusted- SCs double-check it, to ensure that the denial is grounded. Furthermore, a requesting member cannot claim of not having received a service, if this was actually granted, as by the data logged by the SC. Figure 2 summarizes how the SC components affect the message flow between two VO members. The SC filters intercepts the messages exchanged, to detect potential contract breaches. Precisely, a SC accepts two types of incoming messages:

- a message $msg = (Cert, Resource)$ that requests a resource $Resource$ that is provided by the monitored VO member;
- a fault message $msg = (Resource, FaultReason)$ that communicates that the monitored VO member's request of $Resource$ cannot be granted because is not available.

As shown in Algorithm 1, If SC receives a resource request message $msg = (Cert, Resource)$ it first verifies the validity of $Cert$ in msg. Then, it derives from $Cert$ the identity of the resource requester and controls the existence of an entitlement Ent_{VoM_i} (VoM_k, R, I) that obliges the monitored VO member to provide the resource $Resource$ to requester VO member for the whole time interval I, where I includes the time in which the request is received. If such an entitlement exists, SC forwards the resource request to VO member it controls that, in turn, returns the $Resource$ to the requester VO members (lines 1-11, Algorithm 1). On the contrary, if SC receives a fault message $msg = (Resource, FaultReason)$ it checks if the denial of Resource is motivated or not. SC looks for an entitlement Ent_{VoM_i} $(VoM_k, Resource, I)$ that grants to the monitored VO member the access to $Resource$. If such an entitlement does not exist, then the denial was motivated and the SC forwards the fault message to the

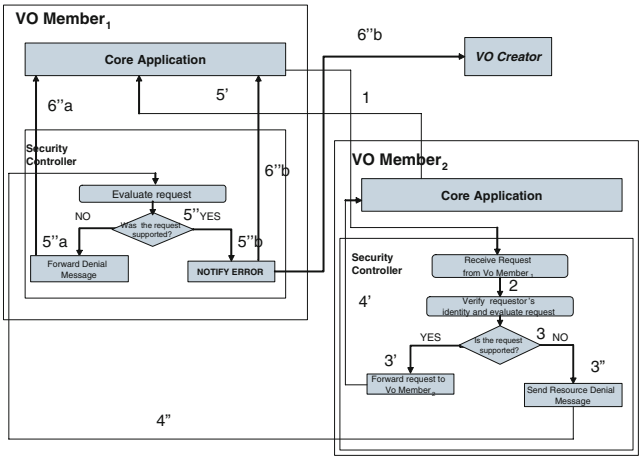

Fig. 2. Contract monitoring flow

controlled VO member. Otherwise, the provider's VO member violated the entitlement and the SC forwards the message $msg = (Resource, FaultReason)$ to the VO member it monitors, and to the VO Creator, which can apply some punishing actions[3].

The SC also detects the situation in which a requested $Resource$ that is available, is not released to a requestor whose request is granted by entitlement $Ent_{VoM_i}(VoM_k, R, I)$. The SC stores locally all the messages received and sent by the monitored VO member within a certain time frame (e.g., a few sessions). Such message history can be consulted at any time by the VO member, as it is made available to it upon request.

In this case, the SC checks that a reply message has been sent by the VO member providing resource R within the time interval I. If this is not the case, the SC notifies to thirds, typically the VO Creator. This simple mechanism ensures non-repudiation; the VO member cannot claim it has been denied a resource that it has actually received.

Example 2. UCB wants to examine some data contained in the database DB_1 of HPC. It decides to access the database DB_1 the 2nd of May 2007 at 10 p.m. The HPC's SC checks if there are entitlements granting the access to UCB. The access to DB_1 is granted to UBC in times different from the time of the request as stated by the following entitlement: Ent_{HPC} $(UBC, \{DB_1, r\}$,[(1-05-2007,13:16:00), (2-05-2007, 15:10:00)]), Ent_{HPC} $(UBC, \{DB_1, r\}$,[(2-05-2007,15:11:00),(5-05-2007,21:36:00)]). Thus, the HPC's controller sends a fault message to the UBC's controller to communicate the denial of the request. UBC's SC verifies that the denial was well motivated and forwards the fault message to UBC.

5 System Implementation

In this section we provide a sketch of the architecture which implements the proposed SC along with interesting details of the SC monitoring component.

The SC component is realized by three main modules. First is the *message handler* that handles incoming and outcoming messages during the distribution phase of the contract and the VO operational phase. The second component is the *distribution module* that is realized by the $\mathcal{X}$-Seal system [1], for the distribution of signed contract blocks and encrypted documents. Last but not least is the *compliance checker* module. The compliance checker module is used for checking whether the received entitlements at the time of contract distribution or contract update are actually fully compliant with obligations or not. Running this module is a crucial precondition for ensuring monitoring correctness since only compliant entitlements and obligations must be usefully distributed and enforced within the VO [11].

The core of the SC is the monitoring system. This module is activated during the VO formation phase and it is used during the operational phase. The monitoring system is realized as a proxy that enforces the local entitlements on the incoming messages. We focus on technical details related to this component in the remainder of this section.

[3] Punishing actions can be of several type, from banning to the VO to decreased reputation and services' availability. We do not further elaborate on this aspect as it is out of the scope of the current work.

Algorithm 1. Monitor Contract Enforcement

Require: A resource request message msg = $(Cert, Resource)$ where $Cert$ is the sender's identity certificate and $Resource$ is the requested resource **OR** a fault message msg = $(Resource, FaultReason)$ where $Resource$ is the resource requested but not available and $FaultReason$ specifies the reason why $Resource$ is not granted

if msg is a msg =$(Cert, Resource)$ **then**
  if $Cert$ is valid **then**
    $VOMemberID$:= $ExtractIdentity(Cert)$;
    if $(Ent_{VoM_i}(VoM_k, R, I)$ s.t VoM_k, == $VOMemberID$ $\wedge$ R == $Resource$ $\wedge$ $t_{msgsend} \in I)$ **then**
      Send msg=$(Cert, Resource)$ to Resource's provider Member
    else
      $FaultReason$:= "No Entitlement Supports Resource Request";
      Send msg=$(Resource, FaultReason)$ to Resource VOMemberID 's SC
    end if
  end if
end if
if msg is a msg =$(Resource, FaultReason)$ **then**
  if $(Ent_{VoM_i}(VoM_k, R, I)$ s.t VoM_k,== VOMemberMonitored $\wedge$ R = = Resource $\wedge$ $t_{msgsend} \in I)$ **then**
    $FaultReason$:="Resource's provider violates the contract";
    Send msg=$(Resource, FaultReason)$ to VOMemberMonitored and VO Creator
  else
    Send msg =$(Resource, FaultReason)$ to VOMemberMonitored
  end if
end if
if !(Receive $msg \wedge t_r \in I$) **then**
  Send msg=$(Resource, Contract\_Violation)$ to VO Creator
end if

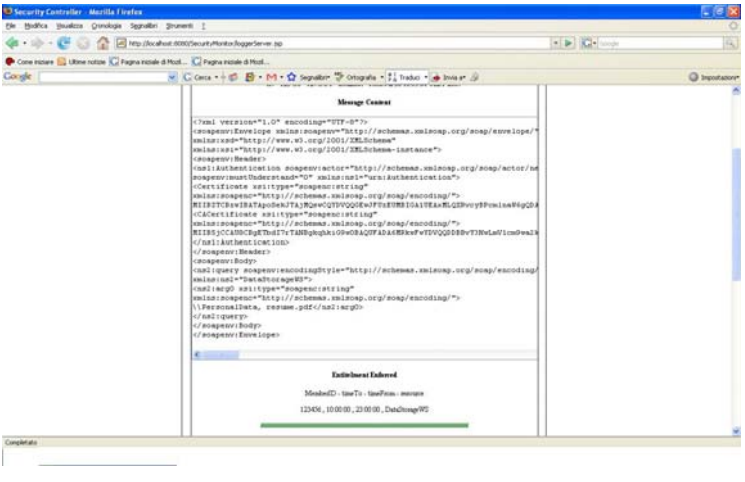

Fig. 3. Security Controller main interface

Security Controller Monitoring System. The SC monitoring system has been implemented in Java (JDK 1.5) and Java Server Pages (JSPs), the Apache Tomcat Application Server and MySQL database to store the entitlements and certificates. We have deployed the SC component in a SOA-based VO infrastructure where the services provided by the VO members are Web services and the messages exchanged are SOAP messages. The three most relevant operations of the monitoring system are realized by the `checkRequestResource`, `checkReplyMessage` and `checkReceivedMessage` java methods. `checkRequestResource` is executed upon receiving of the SOAP message from a VO member to invoke a Web service operation. This method implements the checks performed on a resource message listed in Algorithm 1 (lines 1-11). We report the code snippet of the `checkRequestResource` method in Appendix. `checkReplyMessage`, on the contrary, is executed when the request of a Web service's operation cannot be granted and hence a SOAPFault message is received or when no reply SOAP message is received. The method implements the second half (lines 12-21) of Algorithm 1. `checkReceivedMessage` controls that for a given request message, the corresponding reply message has been received (see Figure 4).

Entitlements and information about the VO Web service providers and the url to locate them are stored respectively into the ENTITLEMENTS and RESOURCES tables. Table CERTIFICATES contains the VO membership certificate and the certificate of the issuer CA. The messages intercepted by the SC are stored in the table MESSAGES.

The monitoring system has been suited with a JSP interface that displays the SOAP messages sent from or to its VO member. Possible displayed messages are: the SOAP message sent by the VO to invoke the operations of a Web service of another VO member, the reply received to the VO member as output of an executed operation request, and

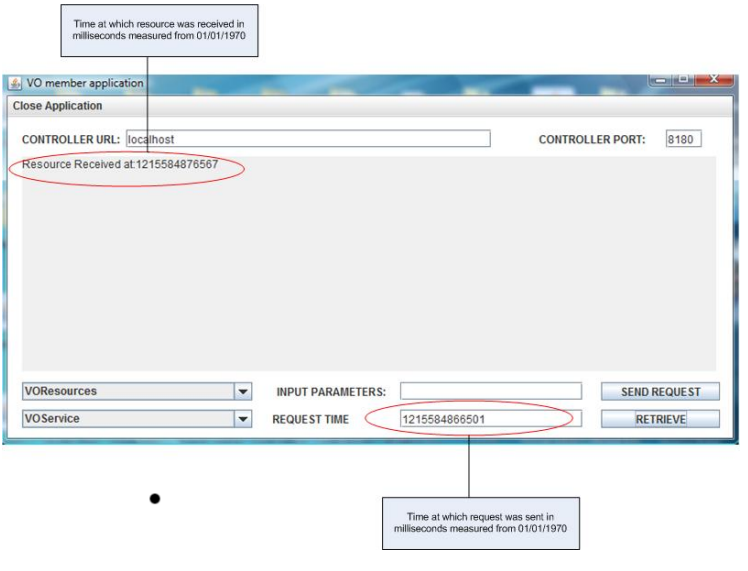

Fig. 4. Message receival interface

the reply SOAPFault message, if the operation cannot be invoked. Figure 3 represents an example of SOAP message sent by the VO member 123456 to invoke the operation `query` offered by the Web service `DataStorageWS`. The message Header contains an Authentication element including a Certificate which includes the VO member's 123456 identity certificate, while the Body of the message contains an element query specifying the name of an XML file name and the XPath query to execute on the file. The entitlement that supports this request states the VO member 123456 is allowed to invoke any of the `DataStorageWS`' operations between 10:00 a.m to 11:00 p.m.

Notice that the SC implementation is modular: checking if a resource message request is supported by an entitlement is independent from the type of message. Therefore, our SC could be integrated in any VO infrastructure -like grid infrastructure- by only implementing the necessary parsers to comply with the format exchanged in that VO, if it does not support SOA messages. Similarly, to facilitate future integrations we chose not to encode the entitlements in any proprietary format. To this extent, we also used the standard and well established X.509 encoding for our VO member identity certificates. This certificate should be released on behalf of the VO Creator by a CA. In order to verify the validity of this certificate, also the CA public key certificate is needed. Both the certificates have to be sent with the resource request messages to authenticate the resource requester. For testing purposes, we assumed that both the membership certificate and the CA certificate are stored locally into the database of the VO member.

Monitoring System Evaluation. We present different test cases to evaluate the performance of the SC component. For each scenario we have compared the interaction time between two VO members with and without the intervention of the SC Components. In all the test cases we have assumed that: a) VO member HPC provides the `DataStorageWS` Web service; b) HBC previously defined an entitlement that grants to VO member UCB the right to invoke the query operation between 10:00 a.m. and 3:00 p.m. Specifically, we have conceived the following cases:

1. UCB requests the operation query within 10:0 a.m-3:00 p.m. a. HPC's SC allows the execution of query operations and the query result is returned to UCB b.

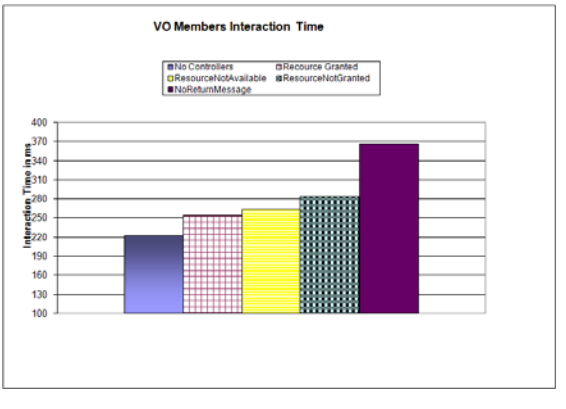

Fig. 5. VO Members Interaction Times

HPC's SC allows the execution of query operations but the `DataStorageWS` is not available due to a deployment error. Therefore, the query result is not returned to UCB.

2. UCB requests the operation query at 4:15 p.m. HPC's SC does not grant query operation invocation. It returns a SOAP fault message to UCB's SC. It double-checks that the request to invoke query operation is in fact not supported and forwards the message to UCB.
3. UCB requests the operation query at a time t within 10:0 a.m-3:00 p.m. a. HPC's SC allows the execution of query operations but the query result is not returned to UCB. UCB's SC checks that a return message with the query results has been received by UCB in the time interval that goes from t to 3:00 p.m.
4. UCB invokes directly the query operation and receives the result. In this case the SC components are not involved.

We have performed our experiments on a Genuine Intel CPU T2300@ 1.66 GHz processor and with 1 GB of RAM, under Microsoft Windows XP Home Edition. The performances have been measured in terms of CPU time (in milliseconds) and are reported in Figure 5.

We notice that the interaction time is always between 200 and 400 ms, regardless of the considered case. The variance for each case is relatively small, and it is about 22. Therefore we can conclude that having SCs which mediate the collaborations among the VO members is worth the cost of the added overhead as it prevents from serious security breaches and guarantees the correct functioning of a VO. Additionally, we expect the time required for the monitoring process to become lower once the SC will be deployed in a real setting, where more efficient and faster processors can be used.

6 Related Work

Virtual Organizations have been thoroughly explored in the realm of grid computing systems [4,2], where technology and resources enable the formation of virtualized environments of users and relative resources belonging to different administrative domains.

The TrustCom project [12] have produced a framework for trust, security and contract management of service oriented architectures, web services and grid technologies to manage the formation, operation and dissolution of virtual organisations and supply chain business relationships.

Another interesting project about virtual organizations is [5]. The project focus on dynamic coalitions, namely, coalitions where member domains may leave or new domains may join during the life of the coalition. In dynamic coalitions, the sharing of resources by autonomous domains is achieved by the distribution of access permissions to coalition members based on negotiated resource-sharing agreements. In the context of the project, a set of tools have been developed that integrate security services for dynamic coalitions, namely, services for (1) private and shared resource management, (2) identity and attribute certificate management, (3) secure group communication, and (4) joint administration for enforcing joint-action policies on shared critical resources. A number of tools and techniques to support the monitoring of network performance and

Grid resources and services have been proposed: for example, NetLogger[13], Autopilot [8] and Remos [3]. These systems incorporate a range of often sophisticated sensor interface, instrumentation, data collection, data filtering, and data summarization techniques that have proven invaluable in a range of application experiments. These systems differ from ours in their main goals and application domains. We do not require specific underlying infrastructure and do not focus on monitoring performance of distributed applications. Rather our goal is to create a self-monitored VO, where members' fulfillment to security and sharing policies is proactively executed. Our SC could be integrated with a performance monitoring tool, to also tackle the problem of monitoring services performance and quality of service. A related contract monitoring system intended to provide automated checking of business to business contracts has been proposed in [6]. It introduces a novel modeling approach to obligations, unifying the treatment of both permissions and obligations by refyining both. The closest work to ours is the Law governed interaction (LGI) proposed by Minsky et al. [14]. LGI is a decentralized coordination and control mechanism for distributed systems. It enables a distributed group of software actors, which may be heterogeneous, open, and large, to engage in a mode of interaction governed by an explicitly specified policy, called the interaction law of this group. The law-enforcement is done in a logically decentralized manner, by associating with each actor a generic component called controller, which is trusted to mediate the interaction of its actor with others. The implementation of this coordination and control mechanism is called Moses [7]. Similarly to our work, the community is governed by global policy and each actor is governed by a local policy, which must conform to. Our concept of contract is however different. In [14] local laws are obtained as refinements of community laws, whereas we consider local policy as a plan for enterprises to allocate resources under their own control while complying with coalition policy as expressed in the contract. Moreover, the implementation of the Moses' controller and of our SC differs in two aspects: 1) the Moses controller allows the member/actor to chose the law that he wants to be applied, while in our controller this is not possible, since it is the single member that has the ability to specify its own entitlements; 2) the Moses controller applies a law enforcing strategy that prevents contract violations, while our controller lets the members autonomy to operate, and detects entitlements violations.

7 Conclusions

Virtual Organization represents a new collaboration paradigm in which the participating entities pool resources in order to achieve a common goal. Ensuring trustworthiness of the members is a fundamental aspect for the VO success. In this paper, we addressed two main problems related to Virtual Organizations's secure deployment: the VO policies' enforcement and the monitoring of VO members' interactions. We have proposed a decentralized monitoring system realized by a software component called Security Controller. We have also evaluated the system overhead introduced by the presence of SC components. The overhead introduced by the SC is reasonably low. Therefore we conclude that having SCs in VOs prevents from serious security breaches and guarantees VOs correct functioning without degrading the time required for members' interactions. We plan to extend the SC implementation with new components to provide

monitoring of service performance. We are investigating which of the existing performance monitoring tools could possibly be employed for integration. We also plan to deploy our SC component in the realm of grid environment.

References

1. Bertino, E., Ferrari, E., Paci, F., Parasiliti Provenza, L.: System for Securing Push Based Distribution of XML Documents. International Journal of Information Security 6(4), 255–284 (2007)
2. Czajkowski, S., Foster, I., Kesselman, C., Sander, V., Tuecke, S.: Snap: A protocol for negotiating service level agreements and coordinating resource management in distributed systems. In: Feitelson, D.G., Rudolph, L., Schwiegelshohn, U. (eds.) JSSPP 2002. LNCS, vol. 2537, pp. 153–183. Springer, Heidelberg (2002)
3. Dewitt, T., Gross, T., Lowecama, B.: Remos-A Resource Monitoring System for Network Aware Applications. Carnegie Mellon School of Computer Science, CMU-CS-97-194
4. Foster, I., Kesselman, C., Tuecke, S.: The Anatomy of the Grid: Enabling Scalable Virtual Organizations. International J. Supercomputer Applications 15(3) (2001)
5. Integrated Security Services Dynamic Coalition Management, `http://www.ece.umd.edu/gligor/ISSDCM2003/ISSDCM2003.html`
6. Linington, P., Neal, S.: Using policies in the checking of business to business contracts. In: Proceedings of the 4th IEEE Workshop on Policies for Distributed Systems and Networks, Como, Italy, June 2003, pp. 207–218 (2003)
7. Moses-Law Governed Interactions (LGI), `http://www.moses.rutgers.edu/`
8. Nastel AutoPilot Overview White Paper. Published by Nastel Technology
9. Sadighi Firozabadi, B., Sergot, M., Squicciarini, A.C., Bertino, E.: A Framework for Contractual Resource Sharing in Coalitions. In: Proceedings of IEEE 5th International Workshop on Policies for Distributed Systems and Networks, New York, USA, pp. 117–126 (2004)
10. Sadighi Firozabadi, B., Sergot, M.: Contractual Access Control. In: Proceedings of IEEE Security Protocols,10th International Workshop, Cambridge, UK, pp. 96–103 (2002)
11. Squicciarini, A.C., Bertino, E., Paci, F.: A Secure framework for Virtual Community Contracts. International Journal of Web based Communities (IJWBC), Inderscience 2(2), 237–255 (2006)
12. TrustCom project, `http://www.eu-trustcom.com/`
13. Tierney, B., Gunter, D.: NetLogger: A Toolkit for Distributed System Performance Tuning and Debugging, `http://dsd.lbl.gov/publications/NetLogger.overview.pdf`
14. Xuhui, A., Minsky, N.H.: Flexible Regulation of Distributed Coalitions. In: Snekkenes, E., Gollmann, D. (eds.) ESORICS 2003. LNCS, vol. 2808, pp. 39–60. Springer, Heidelberg (2003)

A CheckRequestResource's Code

In this appendix we report code snippet of method `CheckRequestResource` introduced in Section 5.

```
public boolean checkRequestResource() {

  settings = Proxy.getInstance().serverSettings;
  this.direction = "IN";
  this.destinationHost = settings.getHost();
  this.senderHost = settings.getSenderHost();
  boolean result = false;
  boolean valid;
  try {
   (SOAPobject.getCertificate()).checkValidity(new Date());
   valid=true;
  } catch (CertificateExpiredException e1) {
   valid=false;
   e1.printStackTrace();
  } catch (CertificateNotYetValidException e1) {
   valid=false;
   e1.printStackTrace();
     }

  if(valid==false){
   msg="Error: Your Certificate is not Valid!";
   result = false;
  }else{
   result = findEnt(SOAPobject.getResource(), memberID);
    }
  if(result){
   Proxy.getInstance().getInfoServer().add(getInfoLog());
   }
return result;
}
```