Policies and Security Aspects For Distributed Scientific Laboratories

Nicoletta Dessí, Maria Grazia Fugini, R. A. Balachandar

Abstract Web Services and the Grid allow distributed research teams to form dynamic, multi-institutional virtual organizations sharing high performance computing resources, large scale data sets and instruments for solving computationally intensive scientific applications, thereby forming Virtual Laboratories. This paper aims at exploring security issues of such distributed scientific laboratories and tries to extend security mechanisms by defining a general approach in which a security policy is used both to provide and regulate access to scientific services. In particular, we consider how security policies specified in XACML and WS-Policy can support the requirements of secure data and resource sharing in a scientific experiment. A framework is given where security policies are stated by the different participants in the experiment, providing a Policy Management system. A prototype implementation of the proposed framework is presented.

1 Introduction

Web Services (WS) and the Grid have revolutionized the capacity to share information and services across organizations that execute scientific experiments in a wide range of disciplines in science and engineering (including biology, astronomy, high-energy physics, and so on) by allowing geographically distributed teams to form dynamic, multi-institutional virtual organizations whose members use shared community tools and private resources to collaborate on solutions to common problems. Since WS have been recognized as the logical architecture for the organization of

Nicoletta Dessí and R. A. Balachandar
Dipartimento di Matematica e Informatica, Universitá degli Studi di Cagliari, Via Ospedale 72, 09124 Cagliari, Italy, e-mail: dessi@unica.it, balsonra@yahoo.co.in

Maria Grazia Fugini
Dipartimento di Elettronica e Informazione, Politecnico di Milano, piazza L. da Vinci 32, 20133 Milano, Italy, e-mail: fugini@elet.polimi.it

Grid services, they can enable the formation of *Virtual Laboratories*, which are not simply concerned with file exchange, but also with direct access to computers, software, and other resources, as required by a dynamic collaboration paradigm among organizations [6].
As the community of researchers begins to use Virtual Laboratories, exploiting Grid capabilities [16], the definition of secure collaborative environments for the next generation of the science process will need further potentialities. In order to extend common security mechanisms such as certification, authorization or cryptography. These new functions include, for example, the definition and the enforcement of policies in place for single Virtual Laboratories in accordance with dynamically formed Virtual Organizations (VOs), and the integration of different local policies, in order to make the resources available to the VO members, who deploy their own services in the VO environment. These considerations motivate the approach that we propose in this paper, whose aim is to explore the security of environments supporting the execution of scientific experiments in a Virtual Laboratory. Specifically, the paper elaborates on extending usual control access mechanism by defining a general approach in which security policies are expressed and enforced to regulate *resource sharing* and *service provisioning*. In detail, the paper proposes a reference framework for secure collaboration where security policies can be formulated in order to regulate access to scientific services and to their provisioning. Since each Virtual Laboratory has a set of local security policies, we examine how these polices can be expressed and enforced such that the allocation process of resources to a distributed experiment is made aware of security implications. As a sample application of the proposed approach, some implementation hints are presented for distributed experiments that incorporate security policies. This paper is structured as follows. Section 2 reviews related work. Section 3 addresses requirements to be considered when security policies for experiments are defined. Section 4 presents our reference framework for Virtual Laboratories, with emphasis on security issues. Section 5 details our approach to Policy Management, giving a component architecture and providing implementation aspects. Finally, Section 6 contains the conclusions.

2 Related Work

A Virtual Laboratory for e-Science can be viewed as a cooperative System where WS are dynamically composed in complex processes (experiments) and executed at different organizations. WS security [19] is assuming more and more relevance since WS handle users' private information. WS-Trust [9] describes a framework for managing, assessing and establishing trust relationships for WS secure interoperation. In WS-based systems, security is often enforced through security services [20], for which new specifications have been developed to embed such security services in the typical distributed and WS-based elements, considering also security policies [18]. Examples are the SOAP header [19], the Security Assertion Markup Language (SAML) [12], XML Signature [4] and XML Encryption [14]. WS-Security [3] ap-

plies XML security technologies to SOAP messages with XML elements. Based on SOAP e-Services, [8] proposes an access control system, while XACML (XML Access Control Markup Language) [2] allows fine-grained access control policies to be expressed in XML. However, all these mechanisms prove useful in specifying specific aspects of security, but need to be selected first, and integrated later, into a uniform framework addressing all issues regarding e-collaboration.

Policies, as an increasingly popular approach to dynamic adjustability of applications, require an appropriate policy representation and the design and development of a policy management framework. Considering that security policies should be part of WS representations, [19] and [10] specify the Web Services Policy Framework (WS-Policy). Policy-based management is supported by standards organizations, such as the Internet Engineering Task Force (IETF). The IETF framework [13] defines a policy-based management architecture, as the basis for other efforts at designing policy architectures.

Existing technology for the Grid (e.g., see [11]) allows scientists to develop project results and to deploy them for ongoing operational use, but only within a restricted community. However, security is still implemented as a separate subsystem of the Grid, making the allocation decisions oblivious of the security implications. Lack of security [20] may adversely impact future investment in e-Science capabilities. The e-Science Core Programme initiated a Security Taskforce (STF) [http://www.nesc.ac.uk/teams/stf/], developing a Security Policy for e-Science (http://www.nesc.ac.uk/teams/stf/links/), while an authorization model for multi-policies is presented in [17]. An approach combining Grid and WS for e-Science is presented in [5, 1].

Authorizations in distributed workflows executed with their own distinctive access control policies and models has been tackled in [7]; security is handled through alarms and exceptions. In [15] access control for workflows is described explicitly addressing cooperation. However, decentralization of workflow execution is not explicitly addressed nor security policies handling is specifically tackled.

3 Basic Security Aspects for Virtual Laboratories

At least for certain domains, scientific experiments are cooperative processes that operate on, and manipulate, data sources and physical devices, whose tasks can be decomposed and made executable as (granular) services individually. Workflows express appropriate modeling of the experiment as a set of components that need to be mapped to distinct services and support open, scalable, and cooperative environments for scientific experiments [5]. We denote such scientific environments as Virtual Laboratories (VLs) or eLabs.

Each VL node (or *eNode*) is responsible for offering services and for setting the rules under which the service can be accessed by other *eNodes* through service invocation. Usually, the execution of an experiment involves multiple *eNodes* interacting to offer or to ask for services. Services correspond to different functionalities

that encapsulate problem solving and data processing capabilities. Services can be designed to use of VOs resources while the network infrastructure promotes the exploitation of distributed resources in a transparent manner. This offers good opportunities for achieving an open, scalable and cooperative environment.
We classify services in:

- *Vertical services*, that include components for a range of scientific domains, including various software applications.
- *Horizontal services*, that provide adaptive user interfaces, plug-and-play collaborative work components, interoperability functions, transaction co-ordination, and security.

Vertical services expose interfaces that convey information about specific application functions. Their interfaces are implemented from within the component embedding them and are assembled in a workflow that globally expresses the *experiment model*. Horizontal services allow for easier, more dynamic and automated *eNode* integration and for more precise run-time integration of remote services. They are designed to facilitate collaboration.
A VO member plans a complex scientific experiment by repeatedly choosing a sequence of services and including these services in a workflow. He can wait for the fulfilment of a specific workflow and/or choose the next service to invoke on the basis of the returned information. The workflow execution may require the collaboration of various services spread over different VLs whose owners must be confident that users accessing their software or data respect fine-grained access restrictions controlling the varying levels of access to the resource that a user may be granted for. For example, a service may require commodity processors or may have a limited choice of input data (possibly requiring a specific file-format or database access). Similarly, a scientist executing a service on a remote *eNode* must trust the administrator of the *eNode* to deliver a timely and accurate result (and possibly proprietary data sent to the site).
This requires the extension of security aspects related to resource sharing to those related to *service sharing*.
However, security is still currently unsupported in an integrated way by any of the available WS technologies, nor a standard method to enforce Grid security is defined. Moreover, security policy requirements have to be considered. The approach of this paper regards the definition of the basic aspects to be tackled when extending WS and Grid security infrastructures to VLs environments.

4 A Reference Framework for Virtual Laboratories

Based on what illustrated so far, we now introduce some basic modeling elements for the context of VLs security, by defining as an actor each relevant *subject* capable of executing experiments supported by *networked resources*, which we consider as *objects*. In detail:

- *Subjects* execute activities and request access to information, services, and tools. Among subjects we mention the remote user of a WS/Grid enabled application, which would generally be composed of a large, distributed and dynamic population of resources. Subjects may also include organizations, servers and applications acting on behalf of users. In this paper, we consider only *trusted* groups which are not requested to exchange security tokens or credentials during a scientific experiment, since they know and trust each other, and received authentication and authorization to access resources when first joining the VL.
- *Objects* are the targets of laboratory activities. Services are considered as objects. Methods are also regarded as objects, which can be grouped together to form experiments. Fine-grained access control would thus be required over input and output parameters, methods, WS and groupings among WS (to form a process) and among WS and other applications (e.g., legacy software or device control software). Other objects are the server hosting the WS, an IP address, or the URI of a WS. Internal data, kept in a database and other objects accessed by the WS, should also be considered as part of the list of objects to be managed.
- *Actions* that can be performed are various, depending on the type of subject issuing a request. Remote users or applications would generally be allowed to execute a WS method, or access a server hosting a number of WS objects or an application. Rights corresponding to actions such as place_experiment, or view_results, update_data could be granted.

The identification of subjects and objects in a scientific environment defines a framework for secure collaboration based on the idea of integrating components that control the workflow execution through a set of specific security components. Such framework, depicted in Fig. 1 comprises components (diamonds), their specific activities (ovals) and specific security aspects (double-border boxes).

The framework elements are as follows:

Process Manager - Each process manager supervises the execution of the workflow modeling the scientific experiment. It is responsible for the transformation of the abstract workflow into a concrete plan whose components are the executions of specific tasks/tools and/or actual accesses to data repositories. This planning process can be performed in cooperation with a service broker, acting as a mediator, in that it supplies, at run time, the description and location of useful resources and services.

Task Manager - This is in charge of executing a particular set of activities which are instances of the workflow plan. It is also responsible for collaborating with others components for managing the service execution. In fact, execution involves contacting data sources and components and requesting the appropriate execution steps.

Service Manager - This supervises the successful completion of each task request. In case of failure, the service manager takes appropriate repair actions. Repair may involve either restarting the task execution or re-entering the configuration component in order to explore alternative ways of instantiating the task execution to avoid service failures, e.g., due to a security attack or service misuse. In that case, the service flow can be rerouted to other services able to provide substitute functionalities, thus allowing redo or retry operations on tasks that were abnormally ended before

rerouting. Moreover, this component waits until completion of the task request, and notifies to the task manager the end of the activity.
Policy Manager - This component supports and updates the resource provision policy that regulates the flow of information through the applications and the network, and across organizational boundaries, to meet the security requirements established by members who are in charge of deploying their own services under their own policies that assert privileges and /or constraints on resource and services utilization.

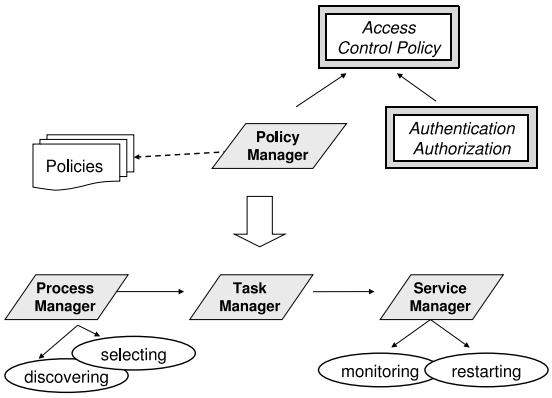

Fig. 1 Security Aspects and Related Components of a Virtual Laboratory

Two major concerns in this framework are: *structural and dynamic* concerns, and *security* concerns. i) *Structural and dynamic concerns* deal with the execution of a scientific experiment in a VL and incorporate controls on vertical services. ii) *Security concerns* refer to horizontal services supporting privileges and constraints on the of VL resources, and may differ from user to user for each individual service. The sequel of the paper presents how these policies can be implemented and how fine-grained constraints can be defined in the VL to gain restricted different access levels to services according to a policy that is fully decided by software owners themselves.

5 Policy Management

Policy management in VLs, as the ability to support an access control policy in accordance with the resource access control goals, should support dynamically changing decentralized policies, policy administration and integrated policy enforcement. A typical policy management system would include two components, namely the *Policy Enforcement Point* (PEP), and the *Policy Decision Point* (PDP), as shown in

Fig. 2. The *PEP* is the logical entity, or location within a server, responsible for enforcing policies with respect to authentication of subscribers, authorization to access and services, accounting and mobility, and other requirements. The *PEP* is used to ensure that the policy is respected before the user is granted access the WS resource. The *PDP* is a location where an access decision is formed, as a result of evaluating the user's policy attributes, the requested operation, and the requested resource, in the light of applicable policies. The policy attributes may relate to authorization and authentication. They may also refer to the attributes related to Quality of Service (QoS), or to service implementation details, such as transport protocol used, and security algorithms implemented. The *PEP* and the *PDP* components may be either distributed or resident on the same server. In our VL, access control works as follows. A user who wants to perform an experiment submits a request to the appropriate resource(s) involved in the experiments through a set of invocations to WS providers. The Policy Manager (see Fig. 2) located in each of the requested resources, implements the *PEP* and the *PDP* to take the access decision about the user access request. The *PEP* wraps up an access request based on the user's security attributes or credentials, on the requested resource, and on the action the user wants to perform on the resource. It then forwards this request to the *PDP*, which checks the request against the resource policy and determines whether the access can be granted.

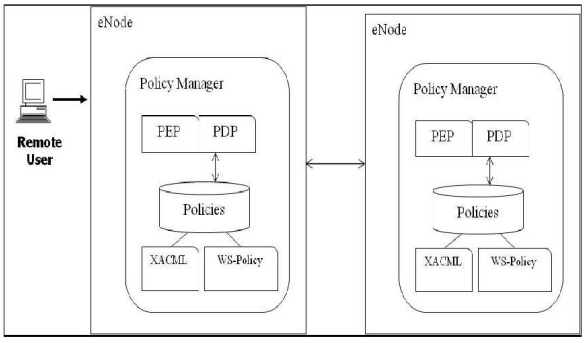

Fig. 2 Policy Management System

There is no standard way of implementing the *PDP* and *PEP* components; they shall either be located in a single machine or be distributed in the different machines depending on the convenience of the Grid Administrator and of the resource provider.

The *Policy Manager* (see Fig. 2) has the ability to recognize rules from the WS requestor and provider of relevant sources, and is able to correctly combine applicable access rules to return a proper, enforceable access decision.

Generally, policies are defined for access to a single resource; hence, the *PEP* and the *PDP* can be contained in a single eNode or be distributed. VL resources may

be part of more than one application and therefore there should be a defined *access control service*. Further, these resources can be used contemporaneously by different applications with different associated policies; hence they will be processed by the applicable Policy Managers. In that case, the applications have their own *PEP* and *PDP*, which control user access to the applications. Further, the *Policy Manager* must be able to recognize the policy attributes related to access control, as well as, the information related to QoS. In the following subsection, we describe the implementation methodology employed for the *Policy Manager* and the standard specification used to express the access policy requirements for a resource.

The described access control mechanisms of the Policy Manager can be implemented using XACML, which includes both a policy language and an access control decision request/response language (both encoded in XML). The policy language is used to describe general access control requirements, and has standard extension points for defining new functions, data types, combining logic, etc. The request/response language allows queries on whether a given action should be allowed, and the interpretation of the result. The response always includes an answer about whether the request should be allowed using one of four values: Permit, Deny, Indeterminate (in case of error or required values missing, that so a decision cannot be made) or Not Applicable (the request can't be answered by this service). A Policy represents a single access control policy, expressed through a set of Rules. Each XACML policy document contains exactly one Policy or a PolicySet, that contains other policies or a reference to policy locations. For example, consider a scenario where a user wants to access and read a web page available in a resource. The XACML representation of this request in the PEP is as follows:

```
< Request >
  < Subject >
    < Attribute AttributeId = "urn : oasis : names : tc : xacml : 1.0 : subject : subject − id"
        DataType = "urn : oasis : names : tc : xacml : 1.0 : data − type : rfc822Name" >
    < AttributeValue > www.unica.it < /AttributeValue >
    < /Attribute >
    < /Subject >
  < Resource >
    < AttributeAttributeId = "urn : oasis : names : tc : xacml : 1.0 : resource : resource − id"
        DataType = "http : //www.w3.org/2001/XMLSchema#anyURI" >
    < AttributeValue > http : //webmail.dsf.unica.it/userGuide_gLite.html < /AttributeValue >
    < /Attribute >
    < /Resource >
  < Action >
    < AttributeAttributeId = "urn : oasis : names : tc : xacml : 1.0 : action : action − id"
        DataType = "http : //www.w3.org/2001/XMLSchema#string" >
    < AttributeValue > read < /AttributeValue >
    < /Attribute >
```

```
    </Action>
</Request>
```

The *PEP* submits this request form to the *PDP* component which checks this request against the policy of the resource hosting the intended web page. For example, the following policy states that the ”developers” group is allowed to read the resource (i.e., the Web Page):

```
<RuleRuleId="ReadRule"Effect="Permit">
    <Target>
        <Subjects>
        <AnySubject/>
    </Subjects>
    <Resources>
        <AnyResource/>
    </Resources>
<Actions>
<Action>
<ActionMatchMatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
<AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">read</AttributeValue>
<ActionAttributeDesignatorDataType="http://www.w3.org/2001/XMLSchema#string"
AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"/>
</ActionMatch>
</Action>
</Actions>
</Target>
<ConditionFunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
<ApplyFunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
<SubjectAttributeDesignatorDataType="http://www.w3.org/2001/XMLSchema#string"
AttributeId="group"/>
</Apply>
<AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">developers</AttributeValue>
</Condition>
</Rule>
```

The *PDP* checks this policy against the request and determines whether the read request can be allowed for the web page. It then forms a XACML response and forwards it to the *PEP* which eventually allows the user to read the page. The implementation of XACML provides a programming interface to read, evaluate and validate XACML policies. It can also be used to develop the Policy Manager con-

taining the *PEP* and the *PDP*, and performs most of the functionalities of the Policy Manager. We can create a *PEP* which interacts with a *PDP* by creating requests and interpreting the related responses. A PEP typically interacts in an application-specific manner and there is currently no standard way to send XACML requests to an online PDP. Hence, we need to include code for both *PEP* and *PDP* in the same application. For instance, the following code snippet will create an XACML request and pass the same to the *PDP*.

$$RequestCtx request = newRequestCtx(subjects, resourceAttrs, actionAttrs, environmentAttrs);$$
$$ResponseCtx response = pdp.evaluate(request);$$

The XACML based Policy Manager can recognize policy attributes related to authentication and authorization. Hence, they can be used only for implementing access control mechanisms. However, such authorization policies do not express the capabilities, requirements, and general characteristics of entities (i.e., users and resources) in an XML WS-based system and there are some more attributes, different from the access control attributes, that need to be examined before accessing a WS.

For instance, one may need to negotiate QoS characteristics of the service, or privacy policies and also the kind of security mechanism used in the WS. Unfortunately, XACML does not provide the grammar and syntax required to express these policies. For this aspects, we use WS-policy specifications which provide a flexible and extensible grammar for expressing various aspects of policy attributes, such as the used authentication scheme, the selected transport protocol, the algorithm suite, and so on. For example, the following specification represents the policy for the algorithm suite required for cryptographic operations with symmetric or asymmetric key based security tokens (it is also possible to include timestamps to the policy specifications to prevent any misuse of the policies).

```
< wsp : Policy
xmlns : sp = "http : //schemas.xmlsoap.org/ws/2005/07/securitypolicy"
xmlns : wsp = "http : //schemas.xmlsoap.org/ws/2004/09/policy" >
< wsp : ExactlyOne >
    < sp : Basic256Rsa15/ >
    < sp : TripleDesRsa15/ >
< /wsp : ExactlyOne >< wsp : All >
    < sp : IncludeTimestamp/ >
< /wsp : All >
< /wsp : Policy >
```

The Apache implementation of WS-Policy provides versatile APIs for programmatic access to WS-Policies. Under this approach, we can implement a policy matching mechanism to negotiate security attributes, and other QoS attributes, be-

fore actual access to the WS. Moreover, WS-policy APIs are a flexible tool to read, compare and verify the attributes present in WS-Policies. For instance, the following code snippet shall be used for creating a Policy Reader object to access a WS-Policy (here Policy_A) and to compare this object with another policy (Policy_B):

$$
\begin{aligned}
&PolicyReaderreader = \\
&PolicyFactory.getPolicyReader(PolicyFactory.DOM\_POLICY\_READER); \\
&PolicyReaderreader = \\
&PolicyFactory.getPolicyReader(PolicyFactory.DOM\_POLICY\_READER); \\
&FileInputStreamPolicy\_A = newFileInputStream(\text{"}ResA.xml\text{"}); \\
&PolicypolicyA = reader.readPolicy(Policy\_A); \\
&FileInputStreamPolicy\_B = newFileInputStream(\text{"}ResB.xml\text{"}); \\
&PolicypolicyB = reader.readPolicy(Policy\_B); \\
&Booleanresult = PolicyComparator2.compare(Policy\_A, Policy\_B)
\end{aligned}
$$

Through the combination of XACML and WS-Policy specifications, we can implement a full fledged Policy Management system for WS to manage authorization policies on resources as well as policies related to security and other QoS aspects. However, this Policy Management system cannot be used as such in Grid environments, considering the very nature of jobs and resources in the Grid. In fact, in the Grid, there are computationally intensive resources, such as clusters, that can either host an experiment as a service, or allow jobs to be executed in it. Hence, the policy requirements in this environment will be different from those of WS environments. For example, suppose that a resource wants to contribute up to (but not more than) 200MB of its memory for job execution in the Grid. To express such policy, currently existing policy languages do not offer enough grammar and syntax. Hence, we suggest to extend the existing policy language schema to include policies regarding elements typical of Grid Services, such as bandwidth information, memory, CPU cycle, etc. For our prototype implementation, we consider three attributes namely the memory, CPU cycle and the available nodes in the cluster resource and a schema is developed with these attributes. The APIs of the WS-Policy implementation are modified accordingly, to deal with this schema and be able to perform operations such as compare, read, normalize, and so on.

The schema that includes the attributes related to a Grid resource, and its usage in WS-Policy is as follows:

$$
\begin{aligned}
&< xs : schema \\
&targetNamespace = \text{"}http://unica.it/gridpolicy.xsd\text{"} \\
&xmlns : tns = \text{"}http://unica.it/gridpolicy.xsd\text{"} \\
&xmlns : xs = \text{"}http://www.w3.org/2001/XMLSchema\text{"} \\
&elementFormDefault = \text{"}qualified\text{"} \\
&blockDefault = \text{"}\#all\text{"} >
\end{aligned}
$$

$< xs : elementname = "Mem" type = "tns : OperatorContentType" / >$
$< xs : elementname = "ProcessorSpeed" type = "tns : OperatorContentType" / >$
$< xs : elementname = "DiskSpace" type = "tns : OperatorContentType" / >$

The following WS-Policy uses this schema to represent the capabilities and policy information of a Grid resource:

$wsp : Policy xmlns : sp = "http : //schemas.xmlsoap.org/ws/2005/07/securitypolicy"$
$xmlns : wsp = "http : //schemas.xmlsoap.org/ws/2004/09/policy"$
$xmlns : cs = "http : //schemas.mit.edu/cs" >< wsp : ExactlyOne >< wsp : All >$
$< cs : Mem > 1024 < /cs : Mem >$
$< cs : ProcessorSpeed > 2GHz < /cs : ProcessorSpeed >$
$< /wsp : All >$
$< wsp : All >< sp : Basic256Rsa15/ >< sp : TripleDesRsa15/ >< /wsp : ExactlyOne < wsp : All >$
$< /wsp : ExactlyOne >$
$< /wsp : Policy >$

Through this policy, the Grid resource wants to advertise that it can allocate no more than 1GB of its free memory to Grid job execution, and that it is able to provide 2GHz of its processor speed. This policy information can be read and compared with other policies using the WS-Policy implementation libraries.

This prototype implementation modifies the WS-Policy specification to deal with a larger number of attributes. To implement these issues in a real time dynamic environment, an extensive survey of Grid resource usage policies and their representation in a WS-policy schema are needed. Our future research will investigate the development of a Policy Management system working for both WS and Grid environments.

6 Implementation Hints

The illustrated framework has been the basis for developing a prototype VL which, in an initial validation stage, has been used to test secure cooperation from the perspective of one scientific server only, for which a Security Server has been implemented, containing security functions deployed as Security WS. The prototype (see Fig. 3) is built on top of Taverna[1], a workflow composer that allows designers to map the initial abstract workflow into a detailed plan. Each Taverna workflow consists of a set of components, called Processors, each with a name, a set of inputs and a set of outputs. The aim of a Processor is to define an inputs-to-outputs transformation. Vertical services can be installed by adding to Taverna new plug-in processors that can operate alone or can be connected with data and workflows through control links. When a workflow is executed and the execution reaches a security Proces-

[1] Taverna is available in the myGrid open source E-science environment http://www.mygrid.org.uk/

sor, an associated invocation task is called that invokes a specific horizontal service implementing security mechanisms. The Scufl workbench included in MyGrid provides a view for composition and execution of processors. The internal structure of a VL includes four components: a Security Server, a Front-End, a Back-End, a Workflow Editor.

The Security Server exposes various functionalities aimed at data privacy and security both in the pole and during the interaction among poles. It manages User Authentication, Validity check of Security Contracts, Trust Levels, Cryptographic Functions, and Security Levels. The Security Server service communicates with the front-end scientific services by sending them the local Security Levels and the list of remote poles offering a specific resource. User authentications occurs through insertion of a secret code by the user requesting the execution of a protected workflow. The Front-end of the scientific pole is a set of WS that can be invoked by a workflow editor, after negotiation. These WS interact with the Security Server, from which they require information related to the local pole access policy. The Front-end includes services that do not hold their own resource trust level, but rather inherit the clearance level of the user executing the WS. However, the Front-end service receives, at creation time, a threshold security level, reflecting the quality and sensitiveness of the service.

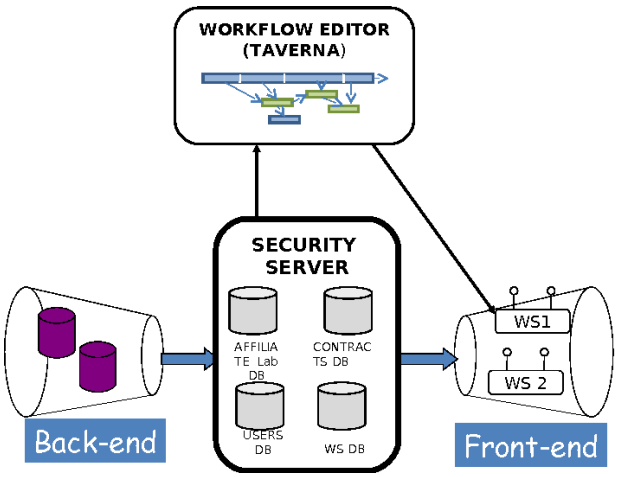

Fig. 3 Security Components Implementation Architecture

The Back-end of a scientific pole is constituted by the local resources of the scientific pole, e.g., calculus procedures or data stored in databases. All the resources in the Back-end are exposed as WS, and can be invoked by a remote Virtual Laboratory. Each resource has its own Resource Service Level assigned by an administrator. The applied policy is ”no read up, no write down”. The invocations of the Back-end services are protected via SSL. Finally, the scientific workflow is defined

using the Taverna workflow editor of MyGrid [2]. Upon proper negotiation of security contracts, a download of the workflow modifier tool and the encryption/decryption module from the provider pole is required. The modifier tool modifies the scientific workflow, by adding crypt and decrypt activities and the input data related to access codes of services. The crypt/decrypt module implements cryptographic functions on exchanged data (we use AES). These editors are designed to be used by scientists teams, generally co-ordinated by a Chief Scientist. However, a workflow is not associated to a whole, given global Security Level, but rather each service of the workflow has an associated Security Level depending on the qualification of the user requiring the service.

7 Concluding Remarks

This paper has highlighted the requirements that should be considered when access control policies of Virtual Laboratories are written. To allow an access control policy to be flexible and dynamic, it can no longer be a high-level specification, but must become a dynamic specification that allows real-time access control administration of WS and the Grid resources. To this aim, we have presented the security requirements of a cooperative environment for executing scientific experiments. Namely, we have illustrated XACML policy specifications, and the use of the WS-Policy to define scientific resource sharing requirements needed to securely activate a collaboration in experiments with negotiating of QoS policy attributes. A security framework and a prototype environment have been presented, with the purpose of providing a uniform view of Grid service policies for a dynamic environment where a set of nodes cooperate to perform a scientific experiment. Currently there exists no standardized access control for virtual applications implemented with WS on the Grid. We plan to extend the requirements presented in this paper and define a formal security model and architecture for WS and Grid enabled scientific applications. The model will be based on the security policy languages used in this paper, independently of specific technologies and configuration models. This should ensure industry-wide adoption by vendors and organizations alike to allow cross-organization business integration. Interoperation requires a standard-based solution. In fact, a Virtual Laboratory, created with WS and the Grid, where scientific relationships may frequently change, requires a highly flexible, but robust security framework, based on approval and universal acceptance of standards. This would allow business partners to avoid interoperability problems among their disparate applications and maintain a security context to allow interoperation.

Acknowledgements This paper has been partially supported by the Italian TEKNE Project.

[2] Taverna, and other e-Science management tools, are freely available on the Internet, but to ensure encryption, decryption and server authentication capabilities they require additional features.

References

1. Amigoni F., Fugini M.G., Liberati D., "Design and Execution of Distributed Experiments", Proc. 9th International Conference on Enterprise Information Systems, (ICEIS'07), Madeira, June 2007
2. Anderson A. et. al., XACML 1.0 Specification, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml, 2003
3. Atkinson B. et al., Web Services Security (WS-Security), 2002, Version 1.0 April 5, 2002, http://www.verisign.com/wss/wss.pdf
4. Bartel M., Boyer J., Fox B., LaMacchia B. and Simon, XML Signatures, http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/E
5. Bosin A., Dess N., Fugini M.G., Liberati D., Pes B., "Supporting Distributed Experiments in Cooperative Environments", in Business Process Management, Springer-Verlag Bussler C., Haller A. (Eds.), vol. 25, 2006, pp. 281 - 292
6. Camarinha-Matos L.M., Silveri I., Afsarmanesh H., and Oliveira A.I., "Towards a Framework for Creation of Dynamic Virtual Organizations", in Collaborative Networks and Their Breeding Environments, Springer, Boston Volume 186/2005, 2005, pp. 69-80
7. Casati F., Castano S., Fugini M.G., "Managing Workflow Authorization Constraints Through Active Database Technology", Journal of Information Systems Frontiers, Special Issue on Workflow Automation and Business Process Integration, 2002
8. Damiani E., De Capitani di Vimercati S., Paraboschi S., Samarati P., "Fine Grained Access Control for SOAP E-Services", in Proc. of the Tenth International World Wide Web Conference, Hong Kong, China, May 1-5, 2001.
9. Della-Libera G. et al., Web Services Trust Language (WS-Trust), available at http://www.ibm.com/developerworks/library/ws-trust/index.html
10. Della-Libera G., et al, "Web Services Security Policy Language (WS-SecurityPolicy," July 2005. (See http://www.oasis-en.org/committees/download.php/16569/)
11. Foster, I. 2006. "Service-Oriented Science: Scaling e-Science Impact", Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web intelligence, 2006
12. Hallam-Baker P., Hodges J., Maler E., McLaren C., Irving R., SAML 1.0 Specification, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security, 2003
13. IETF Policy Framework Working Group, A framework for policy-based admission control, available at http://www.ietf.org/rfc/rfc2753.txt, 2003
14. ImamuraT., Dillaway B., Simon E., XML Encryption, http://www.w3.org/TR/xmlenc-core/
15. Jiang H., Lu S., "Access Control for Workflow Environment: The RTFW Model", in Computer Supported Cooperative Work in Design III, LNCS Springer Berlin / Heidelberg, Volume 4402/2007, 2007, pp. 619-626
16. Kim K.H., Buyya R., "Policy-based Resource Allocation in Hierarchical Virtual Organizations for Global Grids", 18th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD'06), 2006, pp. 36-46
17. Lang B., Foster I., Siebenlist F., Ananthakrishnan R., Freeman T., "A Multipolicy Authorization Framework for Grid Security," Fifth IEEE International Symposium on Network Computing and Applications (NCA'06), 2006, pp. 269-272
18. Mohammad A.,Chen A.,Wang G. W., Changzhou C., Santiago R., "A Multi-Layer Security Enabled Quality of Service (QoS) Management Architecture", in Enterprise Distributed Object Computing Conference, 2007 (EDOC 2007) Oct. 2007, pp.423-423
19. Nadalin A., C. Kaler, P. Hallam-Baker, R. Monzillo (Eds.) Web Services Security, available at http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf
20. Welch V., Siebenlist F., Foster I., Bresnahan J., Czajkowski K., Gawor J., Kesselman C., Meder S., Pearlman L., Tuecke S., "Security for Grid Services", Proc. 12th IEEE International Symposium on High Performance Distributed Computing, 22-24 June 2003, pp. 48- 57