A security architecture for a semantic Grid registry

Victor Tan, Weijian Fang, Sylvia C. Wong, Simon Miles, Luc Moreau
School of Electronics and Computer Science
University of Southampton
vhkt@ecs.soton.ac.uk

Abstract

Existing registry technologies such as UDDI can be enhanced to support capabilities for semantic reasoning and inquiry, which subsequently increases its usability range. The Grimoires registry was developed to provide such support through the use of metadata attachments to registry entities. The use of such attachments provides a way for allowing service operators to specify security assertions pertaining to registry entities owned by them. These assertions may however have to be reconciled with existing registry policies. A security architecture based on the XACML standard and deployed in the OMII framework is outlined to demonstrate how this goal is achieved in the registry.

1. Introduction

Registries are important in a large scale, distributed environment (such as the Grid) as they provide the necessary functionality that allows service providers to expose information about their services to potential users. Existing service registry technologies such as UDDI are however limited by several shortcomings with regards to the service descriptions they support, generally pertaining to the constraints in the information model for these service descriptions. Such constraints make it difficult to satisfy sophisticated user requirements for service discovery based on service characteristics inexpressible in the UDDI model, or for useful semantic reasoning on these descriptions. The Grimoires registry, which was designed and developed in the context of the myGrid project [6] and is now part of the OMII project [3], seeks to provide a solution to these issues through the use of metadata attachments.

As is the case for all Grid-based services, some form of security architecture is needed to protect the contents of the registry. In this paper, we describe the use of metadata for semantic descriptions and their requirements for security. We then discuss the potential use of metadata attachments as a way of enhancing the flexibility of expressing access control to the registry contents.

2. Using metadata for registry publications

Metadata are extra pieces of data giving information about existing entities in the registry. Currently, entities to which metadata can be attached are UDDI BusinessEntity, BusinessService, tModel and BindingTemplate and WSDL operation and message part. For example, BusinessEntities can be annotated with appropriate ratings; functionality profiles can be added to BusinessServices; and semantic types of operation arguments can be attached to WSDL message parts. A piece of metadata is in the form of an RDF [9] triple — the subject is the entity to be annotated, the predicate is the type of the annotation, and the object is the value. The metadata value can be a string, a URI, or structured data in RDF. A unique key is assigned to every piece of metadata published allowing metadata attachments to be updated without republishing the service. This presents an efficient way of capturing ephemeral information about services that change often, such as current load of a service.

There is no limit to the number of attachments each entity can have. Since each piece of metadata has its own unique key, it can be updated without republishing other metadata attached to the same entity. Support is provided for third party annotations, i.e. the ability to publish metadata is available to both service operators and third parties. This provides the flexibility of allowing users with expert knowledge to enrich service descriptions in ways that might not be conceivable to the original publishers. For instance, users can provide their personal ratings on services.

Multiple search patterns are also supported in Grimoires. The simplest form of query returns a list of all metadata attached to the specified entity. A more complex search pattern is supported using the operation find entityByMetadata, which takes a sequence of metadata (type, value) pairs or an RDQL statement. The operation returns a list of entities annotated by metadata matching the query. Search patterns such as this can also be combined with standard UDDI-type queries.

3. Authentication in Grimoires

Security for a registry in a distributed environment is generally concerned with establishing identity (or role, for a role-based system) correctly in the authentication process and using that identity/role to determine access control decisions to the registry contents. In UDDIv2 and v3, an arbitrary XML element termed an authentication token is transmitted by a user by embedding it within publisher API calls. This token is intended for use by the registry in an access control decision. It is generated by the registry and transmitted to a user when the user successfully authenticates to the registry. In implementations such as jUDDI [4], this initial authentication process is achieved using a username/password combination sent via the get_authToken message. The username/password will map to an internal identity known to the registry in a successful authentication, and a suitable authentication token can then be returned to the original user.

This username/password authentication approach however does not scale well for most Grid environments, which generally require that identities be determined in a more globally consistent manner. This is typically achieved utilizing certificate-based authentication schemes such as that found in the Globus Security Infrastructure [1] or OMII [3].

Grimoires is currently deployed within the OMII framework [3] as a Web Service, using Apache Axis and Tomcat as the foundational deploying environment. The OMII framework provides an implementation of applying digital signatures to SOAP messages and verifying these signatures in accordance with WS-Security standards [2]. The framework can extract the Distinguished Name (DN) from the submitted X509 client certificate in an incoming SOAP call, which can be subsequently used by the registry in the appropriate access control. This negates the need for additional functionality in the registry to generate a new authentication token to be returned to the user, as well as the need at the user end to embed this token in all subsequent publisher API calls.

4. Enhancing access control assertions

The access control component of a security architecture for a data store can be policy-driven. Such a policy would typically contain assertions, which describe certain restrictions on granting request to data resources. In many database implementations, these assertions assume the form of mappings between identities and corresponding privileged operations (such as creating, deleting, etc) that can be performed on specific portions of a database.

For the case of a UDDI registry implementation, there are two major sets of operations that are offered to potential clients: the inquiry and the publication API. In the simplest case, implementing access control functionality would involve dividing the potential pool of known clients (i.e. clients that are able of authenticating successfully to the registry in question) into groups which are permitted to perform operations from either one or both of the two main API sets. This can map in a straight forward manner to the RBAC mechanisms (with the groups corresponding to roles) offered by many database systems.

There is however likely to be a need to further refine such access control assertions. For example, it might be desirable in some UDDI applications to restrict the ability to modify or delete a registry entry to only the user who published that entry in the first instance. In the case of Grimoires, the use of third party annotations would also in addition introduce the requirement that a service operator be able to specify the third parties that are permitted to publish metadata relating to the original entry of the service operator. In both cases above, restricting access solely on the basis of operations is insufficient; the notion of identities must be incorporated to some extent in the access control assertions. In addition, from an efficiency consideration, these access control assertions should be articulated directly on registry entity without the need for an additional communication between the service publisher and the registry access control security policy enforcer. This enhanced ability for the service publisher to make its own assertions may however in some circumstances need to be balanced against the security policies operating on the registry.

We describe a simple motivating scenario to illustrate these issues as a context for this paper. Consider service operators that offer a variety of tools and services; such parties may welcome metadata attachments to their registry entries from users in order to enhance the visibility and reputation of their tools. Towards this end, operators would seek to ensure that the users desiring to annotate their service descriptions are sufficiently qualified within a given context to do so. On the other hand, operators could contrive so that only users partial to their services are ever allowed to provide annotations. If it is in the interests of the system to support a balanced third party view of any particular service description, then the overall registry security policy can be modified to reflect this. For example, the policy could specify that a specific group of users will always be allowed or denied annotation capabilities, regardless of any forthcoming access control assertions from a given service operator.

From this sample scenario, we can ascertain two primary requirements: 1) a means for service publishers to include access control assertions relating to their entries, and 2) a way to reconcile these assertions and those from the registry security policy in a consistent way so that potential conflicts do not affect the overall access control functionality offered. The first requirement can be satisfied in Grimoires by specifying an access control assertion in a suitable format as a metadata attachment to a registry entity. For example, the identity of the third parties permitted to further annotate a service description could be specified in the metadata at the point when the entity is published to the registry.

For the second requirement, we have chosen the XACML [7] framework that supports a standardized method of expressing access control assertions as well combining these assertions from differing policies. XACML is an XML-based language for access control policies as well as a request/response language for expressing queries based on those policies. We briefly describe XACML in general and XACML policies specifically in the next sections and then detail how they are used for access control in Grimoires.

5. XACML

The policy language in XACML 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 the formation of a query to ask whether or not a given action should be allowed, and interpret the result.

The typical setup involves an individual requiring some action on a resource. A request is made to whatever actually protects that resource (like a filesystem or a web server), which is called a Policy Enforcement Point (PEP). The PEP forms a request based on the requester's attributes (such as the requester’s identity, assigned group, etc), the resource in question, the action, and other information pertaining to the request. The PEP then sends this request to a Policy Decision Point (PDP), which looks at the request, finds some policy that applies to the request, and applies it accordingly. The evaluation result is returned to the PEP, which can then allow or deny access to the requester.

6. XACML policies

A policy is a combination of several subcomponents: target, rules, rule-combining algorithm, and obligations. The functionality of these subcomponents are as follows:

Target. Each policy has only one target, which determines whether the policy is relevant for the request. This is achieved by defining attributes of three categories in the target: subject, resource, and action, along with their values. The values of these attributes are compared with the values of the same attributes in the request; if they match according to some specified function, then the policy is considered relevant to the request and is evaluated.

Rules. Multiple rules can be associated to a policy. Each rule is composed of a condition, an effect, and a target. Conditions are statements about attributes that upon evaluation return either True, False, or Indeterminate. Effect is the intended consequence of the satisfied rule. It can either take the value Permit or Deny. Target, as in the case of a policy, helps in determining whether or not a rule is relevant for a request. The mechanism for achieving this is also similar to how it is done in the case of a target for a policy. The final outcome of the rule depends on the condition evaluation.

Rule-combining algorithm: A policy can have multiple rules. It is possible for different rules to generate conflicting results. Rule-combining algorithms are responsible for resolving such conflicts to arrive at one outcome per policy per request. XACML defines the following rule-combining algorithms (permitting for user-defined combinations as well):

- Deny-overrides: If any rule evaluates to Deny, then the final authorization decision is also Deny.
- Ordered-deny-overrides: Same as deny-overrides, except the order in which relevant rules are evaluated is the same as the order in which they are added in the policy.
- Permit-overrides: If any rule evaluates to Permit, then the final authorization decision is also Permit.
- Ordered-permit-overrides: Same as permit-overrides, except the order in which relevant rules are evaluated is the same as the order in which they are added in the policy.
- First-applicable: The result of the first relevant rule encountered is the final authorization decision as well.

With reference to the example scenario that we introduced in Section 4, a situation might arise where the registry operator may desire to ensure that users from a specific organization are always banned from making third party annotations, regardless of any assertions from the individual service publishers. Here, the primary policy to be evaluated will initially contain the rule that expresses the desired constraint on the specific users. Subsequent rules appended to the policy will be derived from the security assertions provided by service publishers. By specifying the ordered-deny-overrides mode of rule combining, the evaluation of the policy will always ensure that the designated users will always be banned regardless of any assertions made by the service operators.

7. XACML access control in Grimoires

We utilize the example that we had described in Section 4 to demonstrate a simple example of XACML based access control in Grimoires. This example is detailed in relation to the security infrastructure of Grimoires shown in Fig. 1.

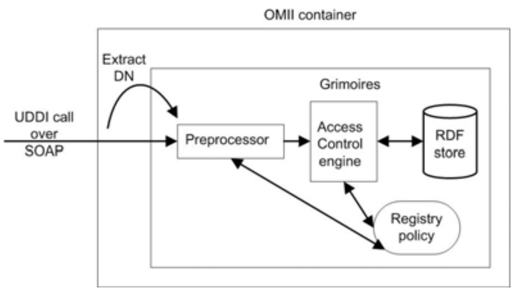

Fig 1. Grimoires security infrastructure.

We consider a simple scenario where all authenticated individuals known to the registry are divided into several groups or roles; each role being permitted a specific set of registry operations on metadata attachments and registry entries. This is expressed in XACML as a registry wide policy that applies to all incoming requests. Assume a user attempts to publish a new businessService entity to the registry through a save_service API call. In the OMII framework, the outgoing UDDI SOAP message contents are signed with the user's private key utilizing the OMII client side libraries, and appended to the SOAP message in accordance with WS-Security standards. At the container end, the signature is verified and the X500DN is extracted and passed over to the pre-processor, which is implemented as an Axis handler.

The pre-processor then determines validity of the X500DN identity within the context of the registry security domain, maps the X500DN to an assigned role and formulates an appropriate XACML request from the SOAP message. It effectively implements the PEP functionality within the XACML architecture configuration. The XACML request, along with the original message contents, are passed onwards to the access control engine. This engine provides the PDP functionality of evaluating the permissibility of the request. If the assigned role is permitted the requested save_service operation, a check is first made to ensure that no entities with the same key already exist in the registry. Once this is achieved, a businessService entity is created in the RDF backend store and a corresponding key is returned to the invoking user. At the same time, a new default metadata attachment for this entity is created consisting of a single XACML rule fragment that specifies that all future publish and metadata API operations on this entity is confined only to the requests originating from the X500DN of this entity.

Consider now a second user (with a different X500DN) that wishes to annotate this newly published entity via a metadata attachment. The incoming request from this user is processed in the same manner as previously up until the point when the access control engine ascertains that the entity already exists. In this case, it retrieves all related metadata attachments for the entity consisting of XACML fragments (one at this period of time) and accumulates them into a single policy. The request then is evaluated by combining the original registry wide policy with this dynamically constructed policy. Here, the requested operation will be denied as the newly constructed policy restricts all metadata operations to the original publisher. This original publisher could broaden the access rights by simply publishing further XACML rule fragments which specify different groups or users permitted to perform specific metadata, publish or inquiry operations. In a situation like this where there are multiple rules to be evaluated in combination, the publisher could also specify the actual XACML rule combining algorithm to be used as a separate metadata attachment.

To ensure consistency in combining the registry policy and the constructed policy, the registry administrator can seek to impose an appropriate policy combining algorithm. For example, setting the PolicyCombiningAlgId to Ordered-Deny-Overrides and evaluating the registry policy first in this combination mode, ensures that any restrictions expressed there always takes precedence regardless of any other assertions in the dynamically constructed policy. Thus, if a metadata XACML rule fragment asserts that a certain user is permitted to perform a specific metadata operation but the operation is forbidden to the group that the user is classified in, then a request from that user for this operation will be denied. The registry policy and the combination rule is published as a separate entity description in the registry that is accessible to all potential users, who can decide accordingly on how to make appropriate security assertions in order to provide the desired level of access control on entities owned by them.

```
<Subjects>
  <Subject>
   <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
     <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">GoodGroup</AttributeValue>
     <SubjectAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:2.0:example:attribute:role"
DataType="http://www.w3.org/2001/XMLSchema#string"/>
   </SubjectMatch>
  </Subject>
</Subjects>

<Actions>
  <Action>
    <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
      <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">addMetadataToEntity</AttributeValue>
      <ActionAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema#string"
AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"/>
    </ActionMatch>
  </Action>
  <Action>
    <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">deleteMetadata</AttributeValue>
      <ActionAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema#string"
AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"/>
    </ActionMatch>
  </Action>
</Actions>
```

Fig. 2 Initial registry policy

An example of a rule in a system wide policy is shown in Fig. 2. Here a XACML request pertaining to the GoodGroup role is permitted to perform addMetadataToEntity and deleteMetadata operations. The policy may alternatively contain a list of other rules as well that articulate further restrictions on specific types of operations permissible to registry entries. An example of a XACML rule fragment is shown in Fig. 3. Here, a specific X500 CN identity is permitted to perform the operation of

```
<Rule RuleId="PermitRule" Effect="Permit">
  <Target>
    <Subjects>
      <Subject>
        <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:rfc822Name-match">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">CN=Bart
Simpson</AttributeValue>
        <SubjectAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
DataType="urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name"/>
        </SubjectMatch>
      </Subject>
    </Subjects>

   <Actions>
     <Action>
       <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
         <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">addMetadataToEntity</AttributeValue>
         <ActionAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema#string"
AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"/>
       </ActionMatch>
     </Action>
   </Actions>
  </Target>
</Rule>
```

Fig. 3 XACML rule fragments as metadata attachments

addMetadataToEntity. The entire fragment is expressed as the object value of the RDF triple that constitutes a single metadata attachment.

An example of XACML request requesting the attachment of a metadata entry to a service entity with the key value of 12345 is shown in Fig. 4. This request is created from the UDDI message contents by the pre-processor.

8. Conclusion

In this paper, we briefly describe the features of the Grimoires registry which include support for

```
<Request>
  <Subject>
    <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id" DataType="xs:string" Issuer="registry.com">
      <AttributeValue>CN=John Doe</AttributeValue>
    </Attribute>
    <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:subject:name-format" DataType="xs:anyURI">
      <AttributeValue>urn:oasis:names:tc:xacml:1.0:datatype:x500name</AttributeValue>
    </Attribute>
    <Attribute AttributeId="urn:oasis:names:tc:xacml:2.0:example:attribute:role" DataType="xs:string" Issuer="registry.com">
      <AttributeValue>GoodGroup</AttributeValue>
    </Attribute>
  </Subject>

  <Resource>
    <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
DataType="http://www.w3.org/2001/XMLSchema#anyURI">
      <AttributeValue>12345</AttributeValue>
    </Attribute>
  </Resource>

  <Action>
    <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
DataType="http://www.w3.org/2001/XMLSchema#string">
      <AttributeValue>addMetadataToEntity</AttributeValue>
    </Attribute>
  </Action>
</Request>
```

Fig. 4 XACML request

metadata attachments and third party annotation. A scenario that describes the motivating security requirements for such a registry is presented. We then briefly present XACML as an approach to implementing the access control engine for the engine towards fulfilling the required security requirements. A sample scenario illustrating how XACML policies and rules can be applied within the security infrastructure is described. We believe the approach of permitting users to specify security assertions pertaining to data owned by them represents a useful way forward towards articulating more flexible access control requirements domains involving large number of users with disparate security requirements.

Future work in this area could look at examining more standard ways of providing security assertions as metadata attachments. In addition to arbitrary XML fragments or direct XACML rules, it might be interesting to examine the use of SAML assertions in line with the SAML-XACML mapping profile in the XACML standard. The XACML-RBAC mapping profile could be used as a guide towards implementing a more comprehensive form of role based access control to supplant the simple identity-to-group mapping implemented by the pre-processor in the security architecture.

9. Acknowledgments

This research is funded in part by the Grimoires (EPSRC Grant GR/S90843/01) and myGrid (EPSRC Grant GR/R67743/01) projects.

REFERENCES

[1] The Globus Alliance. http://www.globus.org/.
[2] Web Services Security (WSS). http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss.
[3] Open Middleware Infrastructure Institute. http://www.omii.ac.uk/.
[4] An open source java implementation of the universal description, discovery, and integration (UDDI) specification for web services. http://ws.apache.org/juddi/.
[5] W3C. Rdf primer. http://www.w3.org/TR/rdf-primer/, 2004.
[6] R. Stevens, A. Robinson, and C.A. Goble. *myGrid: Personalised Bioinformatics on the Information Grid. Proceedings* of 11th International Conference on Intelligent Systems in Molecular Biology, Brisbane, Australia. Bioinformatics Vol. 19 Suppl. 1 2003, i302-i304
[7] XACML. Extensible Access Control Markup Language. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml