A PERMIS-based Authorization Solution between Portlets and Back-end Web Services

Hao Yin[1], Sofia Brenes Barahona[4,] , Donald F. McMullen[2], Marlon Pierce[2], Kianosh Huffman[3], Geoffrey Fox[2,4],

[1]*School of Computer Science and School of Electronic Information, Sichuan University, No. 24, South Section 1, Yihuan Road, Chengdu, 610065, China*

[2]*The Pervasive Technology Labs at Indiana University, 501 N. Morton St. Bloomington, Indiana 47404 USA*

[3]*School of Informatics, Indiana University, 901 E. 10th St., Bloomington, Indiana 47408*

[4]*Department of Computer Science, Indiana University, 215 Lindley Hall, 150 S. Woodlawn, Bloomington, Indiana 47405*

{hayin, mcmullen, marpierc, kihuffma, gcf, sbrenesb}@indiana.edu

Abstract:

A portal is a Web-based application that acts as an entry point to distributed resources. Individual portlets in a portal can be used to integrate information from a variety of back-end Web services. However, when Web services are deployed, they are available to unintended clients not related to the portal so a general solution for authorizing access to them is needed that is integrated with the portal's own authentication and authorization mechanisms. This paper investigates the feasibility of an implementation of a general purpose solution for authorization between portlets and their back end Web services based on Privilege and Role Management Infrastructure Standards (PERMIS) which uses Web services security standards such as WS-Security and SAML. This solution is also appropriate for authorization across organizational boundaries supporting the inclusion of service resources to a portal which are contributed by many different organizations. A motivating example of instrument sharing based on the CIMA remote instrument access protocol is presented.

Key Word: Web Portal, Web services, instrument middleware, WS-Security, Authorization, Role Based Access Control

1. Introduction

Web-based science portals [1, 2] have been increasingly used as gateways connecting users to a range of services. For example, the National Science Foundation's TeraGrid project funds numerous science gateways to provide higher level user interface and services to TeraGrid resources. Many other national Grid systems have adopted similar approaches. This work has been surveyed by the Science Gateways workshop held at Global Grid Forum 14 [3]. Portals are designed to aggregate and integrate content from different sources, possibly provided by Web services [4]. Figure 1 shows a common architecture of Web-based portals and their services. The Common Instrument Middleware Architecture (CIMA) portal system [5] discussed in more detail below provides a realistic example of this system.

One of the distinguishing features of science portals is that they must establish and manage a user's identity. In addition to this authentication process, they also typically provide a system for making authorization decisions about what a particular user can do. These approaches taken by portal system designers to authorization only have scope within a particular instance of a portal and do not extend to external services that are aggregated and presented in the portal, creating a serious gap in the overall design methodology. The key problem, then, in distributed systems such as shown in Figure 1 is that we must provide a "global" way of establishing and conveying identity and privilege to all components of the system.

In practice, end users can authenticate themselves to a portal through a login module. Modern Java portal systems, such as GridSphere[6]

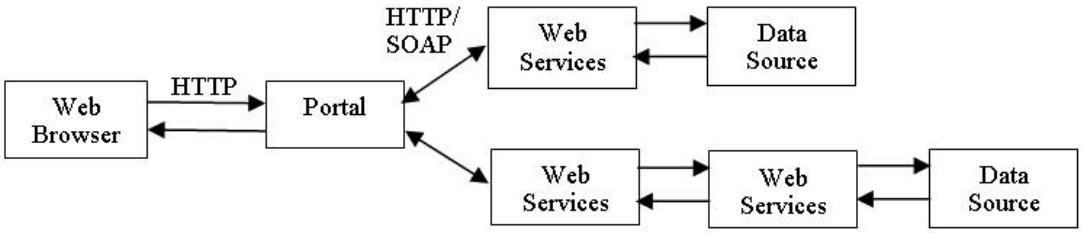

Figure 1. Architecture of Web-based Portal interacting with remote Web Services. The arrows represent network communication links (HTTP or SOAP over HTTP in the case of services). Each box represents a separate component running of the system, typically running on a separate service

are built to be extended by standard component, called portlets [7]. The portal can decide which portlets the user can access, and only indirectly which Web services can be used. Exposing Web services creates the potential for misuse: a user could invoke the Web service directly instead of logging in to the portal. It is possible to provide individual authentication mechanisms per Web service but this creates additional complexity. Specifically, then, there is an *authorization gap* between portlets and the services behind them, and a need exists for a reusable, scaleable role-based access mechanism by which identities and authorization schemes used by portals and their backing services can be meshed other than re-authenticating the user at each service-to-service interface.

This paper will focus on this problem and give a Privilege and Role Management Infrastructure Standards (PERMIS)-based [8] authorization solution, exemplified by a remote instrument access portal based on the Common Instrument Middleware Architecture (CIMA) we are developing. This scenario is described in more detail in Section 5. This portal provides access to shared instrument resources across a federation of laboratories with similar research interests. We begin by surveying general approaches to authorization mechanisms and discuss which ones are appropriate to our portal system.

2. Overview of Authorization Mechanisms

Four main approaches to authorization are available, each with advantages and drawbacks in a given application: access control lists, role-based access control, attribute-based access control, and capabilities-based access control. As we discuss, our motivating scenario is best implemented using authorization roles, which we want to contrast with other approaches.

An *access control list* (ACL) is a data structure, usually a table, containing an entry for each user with access privileges to a particular system object, such as a file or a directory. The most common privileges include the ability to read, write or execute a file. ACL is widely used on Microsoft Windows NT/2000 and UNIX based operation systems. A fundamental problem with ACLs is the *confused deputy problem* in which a process started under one set of permissions performs a task for a process with a lesser set of privileges, and does so under the least restrictive ACL [9].

Role-based access control (RBAC) [10] is an alternative approach to ACLs. Instead of assigning permissions to users directly, roles are created for various responsibilities and access permissions are assigned to specific roles. The assignment of permissions is fine-grained and meaningful, and users obtain the permissions to perform particular operations through their assigned role. A policy file contains the definition of roles related to protected resources and permissions related to a specific role. This mechanism has better scaling for large numbers of users and simplifies the authorization management.

Attribute-based Access Control. Most current authorization systems are based on identity, which means that the subject should be known to the system before their request for protected resources can be authorized. Attribute-based access control (ABAC) [11] is an approach to solve these scalability problems and establish mutual trust negotiation in the large distributed systems. ABAC enables authorization decisions to be made without subject identity by basing on attribute credentials, which contains the characteristics of the subject, such as name, job title, email, etc.

Capability-based access control. Another approach defines *capabilities*. L. Fang [12] defines a capability as "an identifier that carries a set of specific access permission policies on the referred objects." Capabilities combine user role, target service, and action in a single token. In a capability-based system the user presents a signed list of capabilities to a service and the service determines whether the user is authorized to perform the requested function based on this list. This is similar to attribute-based access except capabilities are directly related to the service being called, as opposed to attributes, the semantics of which can be shared by several services.

In addition to the methods discussed above there are several software systems available that are suitable for implementing and managing cross-organizational access schemes: Virtual Organization Membership Service (VOMS) [13], Community Authorization Service (CAS) [14], Shibboleth [15] and Privilege and Role Management Infrastructure Standards (PERMIS).

Our motivating application is a collection of portals used to access instruments and data across a global network of loosely federated, peer laboratories. Role-based authorization best models this environment. Individual users are assigned specific roles to accomplish different operations using shared resources accessible across the lab federation as Web services and accessed through JSR 168 portlets. To simplify authorization decisions users are assigned roles with respect to each shared resource. PERMIS was chosen as the decision engine primarily because it provides a concrete implementation of roles and policies for role-based authorization and can be easily integrated with back-end Web services through Apache Axis API handlers. Also PERMIS can use a Shibboleth identity provider.

3. An Authorization Solution for Portlets

As discussed previously, we need to combine authentication with authorization. Our proposed solution involves not only the usage of XML Signature [16], XML Encryption [17] to guarantee SOAP [18] message level security, but also the usage of WS-Security [19] and SAML [20] to provide security information so that Web services can know who the user is, and then make an authorization decision according to a role-based policy with the information of user, operation and resource extracted from SOAP message.

WS-Security is a formal OASIS standard, which applies existing XML security standards, such as XML-Signature, XML-Encryption and SAML, to SOAP message. The purpose of WS-Security is to provide a standard format in SOAP header to provide interoperable and secure SOAP message. Security tokens, XML encryption and XML signature are three major elements in SOAP header. Username token, binary token (e.g., X.509 certificate, Kerberos v5 ticket) and XML token are composed of security tokens.

Security Assertion Markup Language (SAML) is also an approved OASIS standard that defines XML structures for representing security-related information pertaining to user authentication and authorization. There are three kinds of SAML assertions: Authentication Assertion, Authorization Assertion and Attribute Assertion. SAML assertions can be used with WS-Security as an XML token.

Web services handlers can be adopted on the portal side to intercept SOAP message, add a SAML assertion in SOAP header and sign it on behalf of the issuer. On the Web services side, handlers can be used to verify the signature and extract the security information from the received SOAP message. By using handlers, only minimal changes need to be made to the portal side's code, and no changes need to be made to the Web services code. This solution is a general purpose approach and can be easily integrated with various applications. Figure 2 illustrates our solution.

Step 1: A user authenticates to the portal, and then the portal invokes Web services to acquire requested instrument data. Before the SOAP request is sent, the request handler on the portal side embeds security tokens in SOAP header with digital signature of the portal. The request handler creates a SAML authentication statement containing the name identifier of the user who logged in the portal, and signs it on behalf of the portal. If necessary, the SOAP message can be also encrypted.

Step 2: The request handler on Web services side validates the signature of the received SOAP message. After that it acts as an Authorization

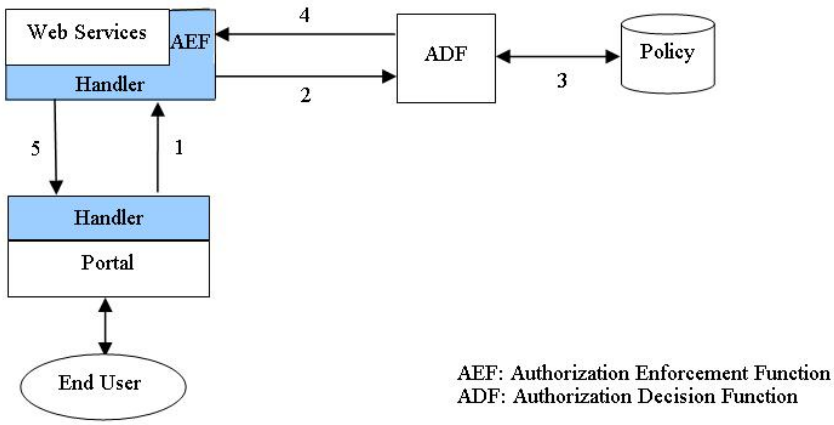

Figure 2. Our authorization solution model uses Web service handlers to transmit identity information between portlets and remote Web services. This identity is then mapped to a role and used to make an access decision on the service. The numbered steps are explained in the text.

Enforcement Function (AEF) [29] and constructs the essential factors for making an authorization decision. These factors are subject, action and target. Subject, also called user identity can be obtained from the SAML assertion in the SOAP header. Action can be deduced by the operation name in the SOAP body element. Target should be the qualified name of Web services, which is self-evident to Web services side.

Step 3: Authorization Decision Function (ADF) [29] gets the factors from AEF and verifies the access control according to the assigned role of the user and a role-based policy, which can be stored in files or in a LDAP [21] server.

Step 4: ADF returns the decision to AEF. If the user is granted the privilege, then AEF forwards the request to a Web service instance, or breaks off the processing.

Step 5: Web services return the result to portal.

There could be a pair of response handlers on both Web services side and portal side to encrypt or sign the SOAP message when Web services return the result. In addition, the functions of Web services handler can be divided into several modules, which consist of a handler chain. Take the request handler on portal side for example, one handler for adding SAML assertion; one for signing the SOAP message; and one for encrypting the SOAP message as needed.

4. Implementation

4.1 Architecture of Implementation

Our implementation of the authorization solution is designed in terms of Web services security standards. We adopt GridSphere as a JSR 168 compliant portlet container, Apache Axis [22] as Web services engine and PERMIS as ADF. If necessary, OpenLDAP [23] can be used to store the role-based policy. OpenSAML [24] is used to create a SAML assertion and WSS4J [25] is used to add security tokens such as SAML assertion in SOAP header and sign or encrypt SOAP message. Figure 3 shows the overall architecture.

4.2 Handlers

We present in this section some implementation details so that the reader can reproduce our approach. In order to complete the steps mentioned above, a request handler or handler chain can be deployed on both portal and Web services side with minimal changes to the source code. On the portal side it can be implemented using the Axis 1.x API. In the code below a WS-Security token handler is added to the SOAP message processing chain, which embeds a SAML authentication assertion in the SOAP header containing the user identity.

```
SimpleChain sc = new SimpleChain();
sc.addHandler(new WssTokenHandler());
sc.addHandler(new WssSignHandler());
// adds a handler to the end of the chain
call.setClientHandlers(sc, null);
```

The first argument of the method *setClientHanders* is for request handler, the second is for response handler. The request

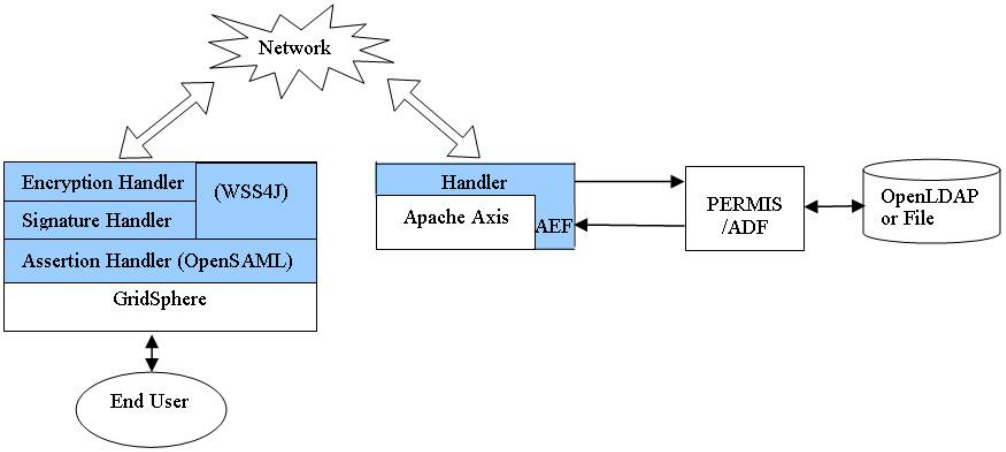

Figure 3. Our implementation of the general architecture shown in Figure 2 uses the software packages shown.

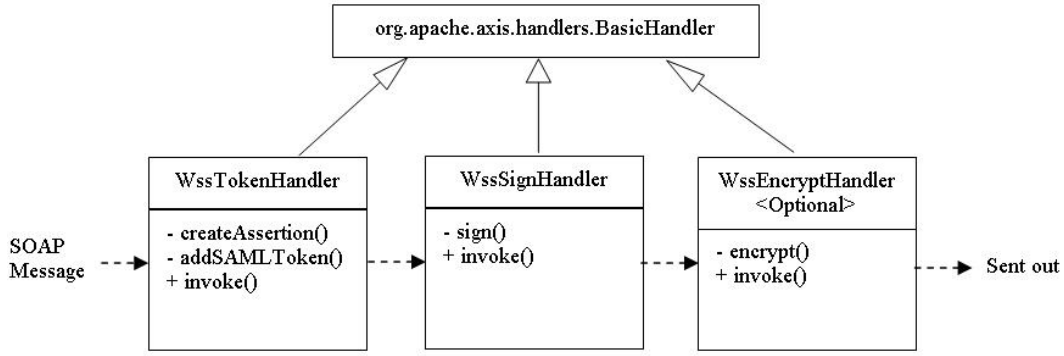

Figure 4. Request Handler Chain on Portal Side

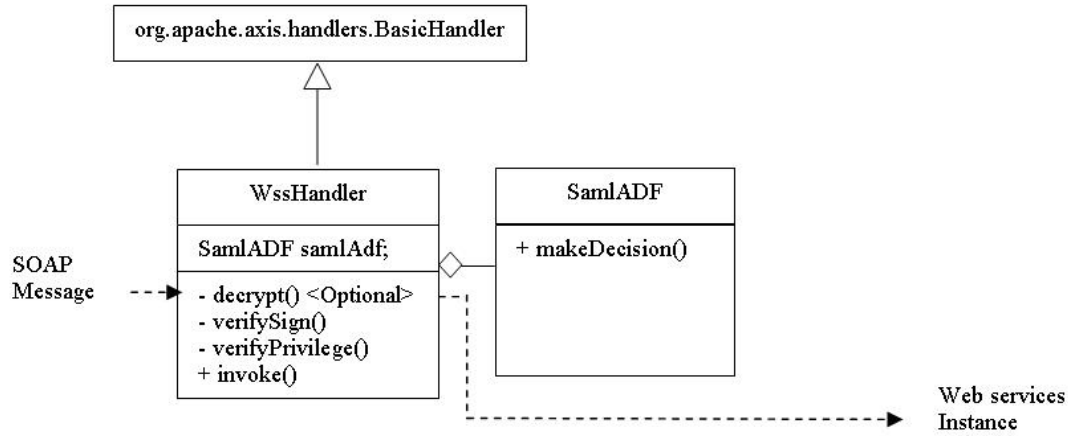

Figure 5. The Request Handler in the Axis service container processes the SAML assertions in the SOAP header to extract the portal user's identity. This identity will then be mapped to a role.

handler chain on the portal side is showed in Figure 4:

On the Web services side, handlers are added in the configuration file, server-config.wsdd:

```
<service name="WssService" provider="java:RPC">
  <requestFlow>
    <handler name="wssHandler"
                type="java:examples.WssHandler"/>
  </requestFlow>
```

…

</service>

The Axis engine will load the handler determined by the deployment configuration. Figure 5 shows the request handler on Web services side.

WssHandler first verifies the signature of the received SOAP message, and then extracts the user identity from SAML authentication assertion, after that maps it to LDAP Distinguished Name (DN), which is the value of subject. Action can be obtained from the element in the SOAP body. Target is set up by the name of the invoked Web services. These factors are essential for *SamlADF* to make a decision on the basis of entries stored in the LDAP. *SamlADF* acts as an authorization engine by using PERMIS API. A fragment of the source codes is shown below to initialize a generic Role-Based Access Control interface:

```
DirContext dirCtx[] = new DirContext[1];
Hashtable env = new Hashtable();
  //LDAP connection and configuration information
      placed in the Hashtable.
dirCtx[0] = new InitialDirContext(env);
AttributeRepository repository =
  (AttributeRepository)                    new
LDAPRepository(dirCtx);
    //Construct the PBAAPI object using the
      AttributeRepository. Specific constructor
    arguments omitted.
PBAAPI pbaApi = new PermisRBAC(…);
  //Use the PBAAPI object to make an authorization
      decision.
boolean result = pbaApi.decision(subject, action,
  target, null);
```

If *result* is true, the current user is granted the privilege to invoke the Web services, or terminate the processing and return an AxisFault exception.

In our implementation, AEF and ADF are built as a single module to avoid an additional Web services invocation, which consists of a SAML request with authorization query and a SAML response with authorization statement

4.3 Setup of the PERMIS System

The previous discussion illustrates how a configured PERMIS system can be used to make authorization decisions. Prior to this, PERMIS itself must be properly configured with roles and rules (or policies) by an administrator. This administrator creates an XML-based policy using Policy Editor provided by the PERMIS software package. The policy includes the following:

- object ID, which acts as a handle, or name, for the policy instance;
- Source of Authority (SOA), a signing certificate for all role and service certificates;
- roles, which are specified with X.509 certificates;
- protected targets, which are the X.509 certificate identifiers for Web services;
- actions, which are methods of the Web service that can be invoked; and
- privilege allocation, i.e. which roles can do specific actions on a specific target.

We take as an example the CIMA portal, which provides access to data collection services running at several facilities, including the Indiana University Molecular Structures Center (IUMSC). User roles consist of *IUMSC_Researcher* and *IUMSC_Member*, the target service base name is “OU=IUMSC, O=CIMA”, and service actions (i.e. Web service operations) include *RequestSession* and *Register*. The administrator creates X.509 Attribute Certificates (ACs) for users using either the Attribute Certificate Manager or the Privilege Allocator tools provided by PERMIS. For each general user, the AC contains the user identity and assigned roles with the signature of the SOA. For the SOA, the AC contains the policy mentioned above with self-sign. The SOA is required to hold a PKCS#12 file to store a pair of PKI keys, which is used to sign the ACs. The administrator can optionally use an LDAP server to store the ACs.. In order to load ACs into LDAP, the following schema is defined to support the Attribute Certificate attribute for entries.

```
attributetype ( 2.5.4.58
   NAME 'attributeCertificateAttribute'
   DESC 'A binary attribute certificate'
   SYNTAX 1.3.6.1.4.1.1466.115.121.1.5 )
objectclass ( 2.5.6.24 NAME 'pmiUser'
   SUP top AUXILIARY
```

DESC 'a pmi entity that can contain X509 ACs'
MAY attributeCertificateAttribute)

4.4 User Mapping between Portal and PERMIS

Generally, a user is required to provide username and password to authenticate to the portal. This user identity can be obtained through the portlet API. Meanwhile, PERMIS policy also defines users, roles and privileges. The user identity in PERMIS is in the form of an X.509 Distinguished Name (DN). Therefore portal user names need to be in correspondence with those in PERMIS. Because the user identity in portal is always a simple name or an email address, a mechanism is required to map it from portal to PERMIS. An easy and feasible way is to extract the simple name is to require it have the DN information in the simpler email-style through portal and construct the DN from this. For example, a user identity in portal is "hayin@iumsc.cima", the user identity in PERMIS can be "CN=hayin, OU=IUMSC, O=CIMA". This matching is currently done manually.

4.5 Federated Authorization for Multiple Organizations.

The CIMA portal and Web service software are decoupled. A lab site may run its own data services but use the main CIMA portal, or it can use its own copy of the portal software. As the CIMA portal system for federating crystallography labs has been adopted by several sites in the United States and abroad, a single role such as "administrator" is not sufficient. For example, users at two sites may each have the role "administrator", but the contents that should be displayed to each one are site- or instrument-specific and therefore different for each user. The "administrator" role provided by the portal is not enough to distinguish between these two types of administrators, so an authorization policy hierarchy as described below is used for this purpose.

The top level in the DN used by PERMIS is "O=CIMA". The second level can be used for different labs. For example, the DN for IUMSC could be "OU=IUMSC, O=CIMA" and the DN for the lab at Purdue could be "OU=Purdue, O=CIMA". In our implementation, the DN for each lab is the target name used in PERMIS. A user that belongs to these labs has a specific role related to each lab. For an Indiana University professor, the role may be "*IUMSC_Researcher*". If the same user on the same portal can also access services at Purdue, the role may be "*Purdue_Member*" for this service. When the user accesses the CIMA portal, the portal system knows the correct Web Service to invoke and to construct the necessary elements for the PERMIS authorization engine. With this solution, PERMIS can support authorization management across organizations.

5. A Use Case for the CIMA Portal

The CIMA project is a Web services-based approach to making instruments and sensor networks accessible in a standards-based, uniform way and for interacting remotely with instruments and the data they produce [26]. A high level overview of the CIMA architecture is shown in Figure 6.

The *CIMA Service* runs as a Web service, and its responsibilities include communicating with all available instruments through their corresponding plugin implementation. As each plugin is specific to an instrument or instrument variable, it contains code for interacting with the instrument and knows how to process each request and construct an appropriate reply, placing its results on a channel that identifies the result as its own.

On the portal side, the *CIMA Sink* is responsible for receiving real time data from instruments and displaying it as tables and graphs. In our application, the *CIMA Service* is deployed using the Apache Axis for Java API. Our PERMIS-based authorization solution is embedded in both *CIMA Sink* and *CIMA Service* by using Web services handlers as provided by the Axis API.

CIMA must include security in order to restrict access to an instrument, or to restrict the types of operations that particular users can perform. As SOAP messages are being used for communication, the PERMIS framework can be used as an authorization solution that will only allow requests verified by PERMIS to reach the *CIMA Service.*

The scenario implemented for this paper considers the case where a *Researcher* is leading

an experiment with the aid of a group of *Students*.

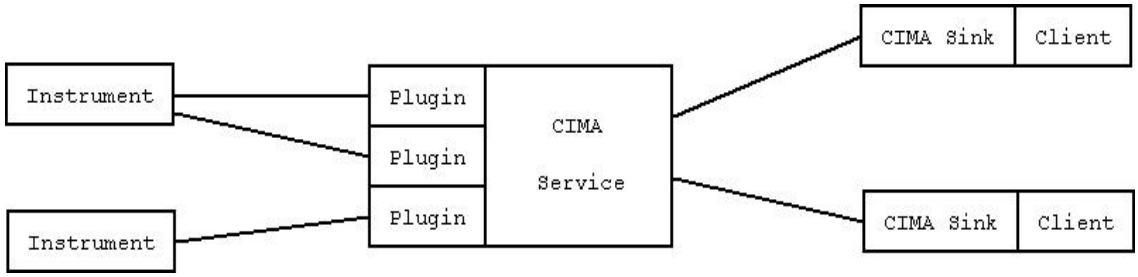

Figure 6. A high level overview of CIMA system components includes real-time instrument data sources that are proxied through Web services using instrument-specific plugins. Sinks receive data from the service. Specific clients can be built using these sinks to display or process the data.

The instrument in use requires a valid session key to be provided before data can be retrieved from it. This session key can only be generated by the *Researcher*, which in turn distributes it to the *Students*. The *Students* then use the session key to register with the instrument and start receiving data.

There are two possible actions, *RequestSession* and *Register*. The *Student* user is only authorized to perform the latter. If a *Student* issues a *RequestSession* request, the request will be denied by PERMIS before it reaches the *CIMA Service*.

In order to test the CIMA and PERMIS implementation, both the *CIMA Service* and the *CIMA Sink* were modified to use PERMIS as their authorization method. For the CIMA portal at Indiana University, a *Researcher* is assigned with the role *IUMSC_Researcher*, and a *Student* with the role *IUMSC_Member*. Integration was successful in that whenever the *Researcher* performed any of the operations, PERMIS would verify and allow the request to continue. In contrast, when the *Student* attempted to perform a *RequestSession* operation, PERMIS would deny the request, returning an appropriate message to the *Student*, indicating that an unauthorized operation was attempted. None of the denied operations ever reached the *CIMA Service*, as PERMIS intercepted and denied them.

6. Conclusions and Future Work

Access rights of portal users to interact with a particular portlet are qualitatively different from rights that a portlet may need in order to access a service on behalf of users. Hiding back-end services or securing them with an authentication layer leads to scaling difficulties, especially when services are "owned" by different organizations. We have shown that PERMIS provides a flexible and lightweight role-based way to manage permissions when portlets access Web services for content, both within and across organizational boundaries. Roles and authorization policies based on them are defined by the service owner and associated with user identities shared with the portal owner. The sharing of roles across services and the distribution of management between the service provider (defining roles and policies, associating roles with users) and the portlet provider (defining users and providing user identities when portlets access content provider services) gives a scalable solution with local control.

The PERMIS based authorization solution between portlets and Web services described here is built on industry standards. In conjunction with PERMIS, this solution provides both secure SOAP message transactions and an authorization mechanism to enhance the security of Web services. The limitation of our proof-of-concept implementation is that the portal user identities must be synchronized with PERMIS through external mechanisms and user name conventions. The username must match our X.509 DN structure (see 4.4), and the PERMIS administrator must assign that identity to one or more roles. This implementation shortcoming can be solved in the following general manner: the portal owner could create appropriate AC if the service provider gives the portal provider the Source of Authority certificates. Containers like GridSphere can have their user management system extended to support these additional actions (c.f. the Grid Account Management Architecture (GAMA) [27]). We also need to investigate the XML Key Management System [28] for managing the PKI

signing keys used in the system.

7. Acknowledgements

CIMA is supported by National Science Foundation cooperative agreements and grants SCI 0330568 and MRI CDA-0116050, respectively. OGCE development is supported by the National Science Foundation's Middleware Initiative, SCI 0330613.

We thank Professor Jiliu Zhou, School of Computer Science, Sichuan University, China for supporting H.Y.; Professor David Chadwick, Computer Science Department, University of Kent and PERMIS team for their invaluable help.

Reference:

[1] M. A. Smith, "Portals: toward an application framework for interoperability", *Commun. ACM* 47, 10 (Oct. 2004), 93-97. available http://doi.acm.org/10.1145/1022594.1022600

[2] Hey, A. and Fox, G., eds. Concurrency and Computation: Practice and Experience, Vol. 14, No. 13-15 (2002). Special Issue on Grid Computing Environments.

[3] Science Gateways Workshop, Global Grid Forum 14, July 27-30 2005, Chicago, Illinois. Workshop URL: http://www.gridforum.org/GGF14/ggf_events_next_schedule_Gateways.htm

[4] W3C Working Group, "Web Services Architecture", W3C, 2004, Available http://www.w3.org/TR/ws-arch

[5] R. Bramley, K. Chiu, T. Devadithya, N. Gupta, C. Hart, J.C. Huffman, K.L. Huffman, Y. Ma, D.F McMullen. (2006) "Instrument Monitoring, Data Sharing and Archiving Using Common Instrument Middleware Architecture (CIMA)", *Journal of Chemical Information and Modeling*, 46(3) p.1017-25, May-June 2006.

[6] J. Novotny, M. Russell, O. Wehrens, "GridSphere: a portal framework for building collaborations", *Concurrency - Practice and Experience*, 2004, 16(5), pp. 503-513.

[7] A. Abdelnur, S. Hepper, "Java Portlet Specification version 1.0", 2003, Available http://www.jcp.org/aboutJava/communityprocess/final/jsr168

[8] D.W. Chadwick, O. Otenko, (2002) "The PERMIS X.509 Role Based Privilege Management Infrastructure" In Proc 7th ACM Symposium On Access Control Models And Technologies (SACMAT 2002), Monterey, USA, pages 135-140, June 2002.

[9] N. Hardy, (1988) "The confused deputy", Operating Systems Review, **22**(4), pp. 36-38, Oct. 1988.

[10] D.F. Ferraiolo, J. A. Cugini, D. R. Kuhn, "Role-Based Access Control (RBAC): Features and Motivations," *11th Annual Computer Security Applications Proceedings*, 1995.

[11] W. Winsborough, J. Jacobs, "Automated trust negotiation in attribute-based access control," *DARPA Information*

[12] L. Fang, D. Gannon, F. Siebenlist, "XPOLA – An Extensible Capability-based Authorization Infrastructure for Grids." 4th Annual PKI R&D Workshop: Multiple Paths to Trust Proceedings, August 2005. NIST Interagency Report 7224. Available http://csrc.nist.gov/publications/nistir/ir7224/NISTIR-7224.zip.

[13] R. Alfieri, R. Cecchini, V. Ciaschini, L. dell'Agnello, A. Frohner, A. Gianoli, K. Lörentey, F. Spataro, (2003) "VOMS, an Authorization System for Virtual Organizations" European Across Grids Conference 2003: 33-40

[14] L. Pearlman, C. Kesselman, V. Welch, I. Foster, S. Tuecke, (2003) "The Community Authorization Service: Status and Future" In Proceedings of the Conference for Computing in High Energy and Nuclear Physics, La Jolla, California, USA, Mar. 2003.

[15] Shibboleth Project, Available at http://shibboleth.internet2.edu/

[16] M. Bartel, J. Boyer, B. Fox, B. LaMacchia, E. Simon, "XML-Signature Syntax and Processing", W3C, Available http://www.w3.org/TR/xmldsig-core/

[17] T. Imamura, B. Dillaway, E. Simon, "XML Encryption Syntax and Processing", W3C, Available http://www.w3.org/TR/xmlenc-core/

[18] Simple Object Access Protocol. http://www.w3.org/TR/soap

[19] OASIS Web Services Security TC, "WS-Security", OASIS, Available http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss

[20] OASIS Security Services TC, "Security Assertion Markup Language (SAML)", OASIS, Available http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security

[21] T. Howes, M.C. Smith, G.S. Good, T.A. Howes, "Understanding and Deploying Ldap Directory Services (MacMillan Network Architecture and Development Series)", Macmillan Technical Pub, 1998

[22] Apache Axis Project, http://ws.apache.org/axis

[23] OpenLDAP project, http://www.openldap.org

[24] OpenSAML project, http://www.opensaml.org

[25] Apache WSS4J project, http://ws.apache.org/wss4j/

[26] D. F. McMullen, T. Devadithya, K. Chiu. “Integrating Instruments and Sensors into the Grid with CIMA Web Services” In Proceedings of the Third APAC Conference on Advanced Computing, Grid Applications and e-Research. 2005.

[27] K. Bhatia, S. Chandra, K. Mueller (2005) “GAMA: Grid Account Management Architecture” In Proceedings of the First International Conference on e-Science and Grid Technologies (e-Science 2005), 5-8 December 2005, Melbourne, Australia. IEEE Computer Society 2005, ISBN 0-7695-2448-6 pp. 413-420.

[28] W. Ford, P. Hallam-Baker, B. Fox, B. Dillaway, B. LaMacchia, J. Epstein, J. Lapp, “XML Key Management Specification (XKMS), W3C, 2001, Available: http://www.w3.org/TR/xkms/ *Survivability Conference and Exposition (DISCEX III)*, April, 2003.

[29] M. Lorch et al, "Conceptual Grid Authorization Framework and Classification,” Global Grid Forum Document GFD-I.038. Available from http://www.ggf.org/documents/GFD.38.pdf.