An Authorization Framework for a Grid Based Component Architecture

Lavanya Ramakrishnan[1], Helen Rehn[2], Jay Alameda[2], Rachana Ananthakrishnan[1], Madhusudhan Govindaraju[1], Aleksander Slominski[1], Kay Connelly[1], Von Welch[2], Dennis Gannon[1], Randall Bramley[1], Shawn Hampton[2]

[1]Department of Computer Science, Indiana University, Bloomington, IN 47404, USA
{laramakr, ranantha, mgovinda , aslom, connelly, gannon, bramley}@cs.indiana.edu

[2]National Center for Supercomputer Applications, Champaign, IL 61820, USA
{hrehn, jalameda, shawn}@ncsa.uiuc.edu, welch@mcs.anl.gov

Abstract. This paper[1] presents an architecture to meet the needs for authentication and authorization in Grid based component systems. While Grid Security Infrastructure (GSI) [1] is widely accepted as the standard for authentication on the Grid, distributed authorization is still an open problem being investigated by various groups [2],[3],[4]. Our design provides authentication and fine-grained authorization at the interface, method and parameter levels. We discuss the various ways in which internal and external authorization services can be used in a component framework. The design is flexible to allow the use of various existing policy languages and authorization systems. Our prototype is based on XCAT, an implementation of the Common Component Architecture (CCA) specification.

1. Introduction

Computational grids combine computational and data resources across organizational boundaries. The sharing of code and data on the Grid gives rise to the need for security. Desired characteristics of Grid security include:

- The ability to verify the identity of an individual. Authentication ensures that the individual is who he/she claims to be.
- The capacity to allow or restrict access to some or all resources on the Grid, i.e. authorization.

The specific problem that this paper addresses is how does one provide secure authorization mechanisms for controlling access to distributed scientific application that are executing as Grid Services. This question is motivated by the emergence of two Grid technologies. The web services model that has been developed recently by industry has now been recognized as the logical architecture for the organization of Grid services. In this model, called the Open Grid Service Architecture (OGSA) [5], [23], a Grid service is described by extensions to the Web Services Description Language [6] and accessed by protocols like the Simple Object Access Protocol (SOAP). The second technology driving this work is an application construction framework called the Common Component Architecture (CCA) that was originally developed by the DOE for large-scale parallel computation. An implementation of the CCA specification, called XCAT [7], has been built using the same web service technology as used in OGSA. While OGSA provides a model for building Grid Services, XCAT provides a model for building Grid applications by composing components that are located on remote resources on the Grid. These applications may be distributed, coupled multidisciplinary simulations, or dynamic networks of data-mining agents. Unlike the persistent Grid services such as a directory service, these distributed applications are stateful, transient services that are executing on behalf of a single user or a group of users. The users can steer the application by adjusting the parameters at run-time and monitor the application by examining its output stream. Also, users can create a persistent service, called an application factory, which can be used by collaborators to launch instances of other applications. In this context, the security problem can be stated as follows:

[1] This research was supported by NSF grant ASC 9619019, NCSA Alliance

- What mechanisms can a group of scientists use to control access to running Grid computations that expose a Grid service interface?

Typically, authentication of Grid users is handled using GSI. It was designed with the notion of a global identity and a robust security scheme based on Public Key Infrastructure that allows the mapping of global identities to local identities. It enables single sign-on for multiple connected resources, while maintaining local institutional control over the individual resources. In this paper we discuss the use of existing technology to provide authentication and propose a solution for handling authorization for the problem described above.

2. Requirements

2.1 User and Developer Roles

Our work with application scientists has helped us identify the roles of people involved in the lifecycle of applications. Individuals may fall into more than one group. These groups have different requirements on how and why they need to specify access control.

Component Architects: These are the developers who provide mechanisms as part of a framework (library that implements the component specification) for the application developers, primary collaborators, users and resource/service providers to describe and implement authorization control for their resources.

Application Developers: These are the application scientists who develop application code and use the library developed by Component Architects to incorporate a security system in their code.

Primary Collaborator: The primary collaborator is the person who owns the process for the application code. He/she instantiates components in the framework and uses an authorization system to set access control policies for the instantiated components.

Users: These secondary collaborators use the software components to gain access to codes and data.

Resource/Service providers: This includes the class of people who own the physical resources that the component requires. They may have constraints on who can use their resources.

2.2 Application Security Requirements

The following are some characteristics that are important for running applications on the Grid.

- There may be different levels of security required depending on the sensitivity of the application.
  - Security may not be important for all applications and there may be some components with no restrictions.
  - Highly sensitive data and computations have greater security considerations.
- The authorization framework should not impact performance when it is not needed.
- Different versions of digital certificates or various identity mechanisms are used on the Grid. An authorization scheme has to take these differences into account .

2.3 Policy Management Requirements

A basic authorization framework should have the following capabilities:

- Policy Representation and Management: It is necessary to be able to express and manage policy about trusted users and their allowed actions.
- Policy Evaluation: The policy needs to be evaluated at the time the action is requested. The result of the evaluation is based on the user's identity and the security policy governing the use of the resources.

Policy management in an authorization framework can be either distributed or centralized and the choice is dictated by the characteristics of the application and the environment. Consider the drawbacks in each case: first, in the case of distributed policy management/evaluation, where an access control list is associated with each individual component in an application, a change in policy needs to be propagated across each component. This would impose a large administrative overhead. Secondly, in the case of a

centralized policy management/evaluation system a large number of requests for authorization can negatively affect the system performance.

3. Common Component Architecture

3.1 Component Framework Overview

The Common Component Architecture (CCA) [8], [9], [10], [11] is an architecture that describes how to compose, create and connect applications that are wrapped as components in a distributed environment. The external interfaces of CCA components are called ports. A component provides and uses services through ports. According to the CCA specifications there are two types of ports:
Provides Port: A provides port is the interface through which the component makes available a set of services that can be used by other components.
Uses Port: A uses port acts as a reference to the remote service from the component using the service.

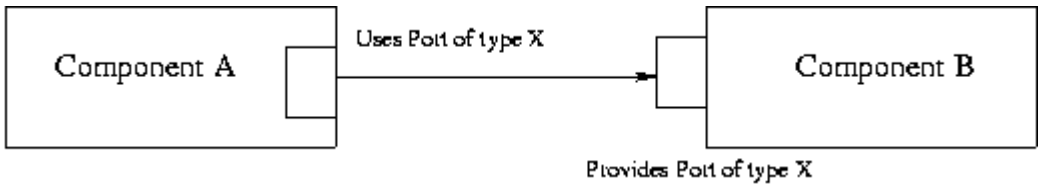

Fig. 1. A uses port of type X is connected to a provides port of the same type.

As shown in Figure 1, a component with a uses port type "X" can use the services provided by a provides port of the same type of another component.

3.2 Grid Tools - XCAT and XSOAP

XCAT [12], [7] is a component system that wraps high performance application code and provides an elegant scripting based programming model to compose and run applications on the Grid. XCAT components can be represented as web services, which facilitates easy, standards-based access to the components. The XML descriptions of the component and port characteristics are parsed to extract the information pertaining to the instantiation environment.

The component framework provides some services by default to all components. These services include creation, connection, registry and directory services that provide standard ways a component instance can use to discover, instantiate, connect and advertise other component instances. To facilitate code reusability, the framework provides these services as "pseudo-components" that exist either as a separate, remote component or within the process that owns the component instance. We intend to add authorization to this list of services provided by the component framework.

The communication protocol between the ports is based on XSOAP [13]. XSOAP is an RMI system implemented both in Java and C++ that is based on SOAP-RPC. SOAP [14] defines XML based communication and SOAP RPC precisely states the protocol for using XML as the data format and HTTP as the network protocol. XSOAP uses GSI [1] to handle authentication [15].

4. Related Work

GSI enables secure authentication using public key encryption. It uses X.509 certificates as mechanism for identity and the Secure Sockets Layer (SSL) as the communication protocol. Support for delegation of credentials and single sign-on is provided as part of GSI.

In the Community Authorization Service (CAS) [3], resources and users are classified into groups called communities. CAS maintains authorization information for all entities in the community. Communities run

CAS servers, which keep track of fine-grained access control information and grant restricted proxies to community members. The CAS server has its own authorization model that determines who can change policy information maintained by the server.

Akenti [2] is a security model based on a distributed access control mechanism. The policy engine gathers the use-condition certificates from the various stakeholders and the attribute certificates from trusted directories and grants access to a resource based on the matching of the attributes in these two certificates.

Legion [16] is an object based system used to access resources on the Grid. In Legion an object has a MayI layer [4] that verifies the credentials received with a method call using the access control list stored as part of the object to determine if access to the resource may be granted.

The Oasis Security Assertion Markup Language (SAML) [24] defines a protocol by which clients can request and receive assertions from SAML authorities concerning authenticated identities [17]. A SAML authority could be contacted to receive information regarding the authorization of the client attempting to access the component. XACML is an XML policy language from OASIS that expresses policies for information access over the internet [18].

The XML Trust Assertion Service (XTASS) is designed to manage long-term trust relations between principals [19]. In the XTASS architecture, authorization data is bound to a cryptographic key by means of a URI that corresponds to a set of resources for which access is granted. An assertion service can be queried to determine authorization.

WS-Security from Microsoft and IBM provides mechanisms for transferring credentials through SOAP and ensuring the integrity and confidentiality of SOAP messages [20].

The digital signature security extension to SOAP allows SOAP messages to be signed [21]. In addition, XKMS and XML Signature Syntax and Processing deal with the representation and processing of digital signatures in XML [22].

The Object Management Group (OMG) Security Specifications [27] describes the service needed to acquire authorization tokens to access a target system. These tokens are used to gain access to a CORBA invocation on the target.

We have mentioned the related security work in the Grid and Web services. We believe that these and other systems can be used with our model for security in the component framework.

5. Authorization Framework

We have designed an authorization framework that is tailored for Grid based component systems. Our authorization framework contains the following

- A standard way to intercept method invocations on components that require authorization checks.
- A standard mechanism for a component to indicate choice of an internal or external authorization service.
- A mechanism for using authorization services that evaluates the policy once method invocations have been intercepted.
- Standard information the authorization services can expect to receive in order to make authorization decisions.

5.1 Authorization Model

The architecture of our framework can be described using the stack abstraction shown in Fig 2. Our system employs a simple request response protocol. The layers of the stack are:

Communication Layer: The communication layer is analogous to the traditional transport layer in the network model, and includes the physical socket connections. The communication layer handles the authentication when it receives a request.

Framework Layer: The framework layer is an abstraction of the implementation of an architecture for instantiating applications. The authorization sub-layer embedded in this layer, intercepts the incoming call, checks authorization, and either allows (Fig 2b) or denies (Fig 2c) the call to the method in the component. Fig 2a shows the request-response path without authorization.

Application Layer: The application layer contains components that manage applications. It has hooks to the framework layer, so that the application can dynamically affect the authorization information stored by the authorization sub-layer.

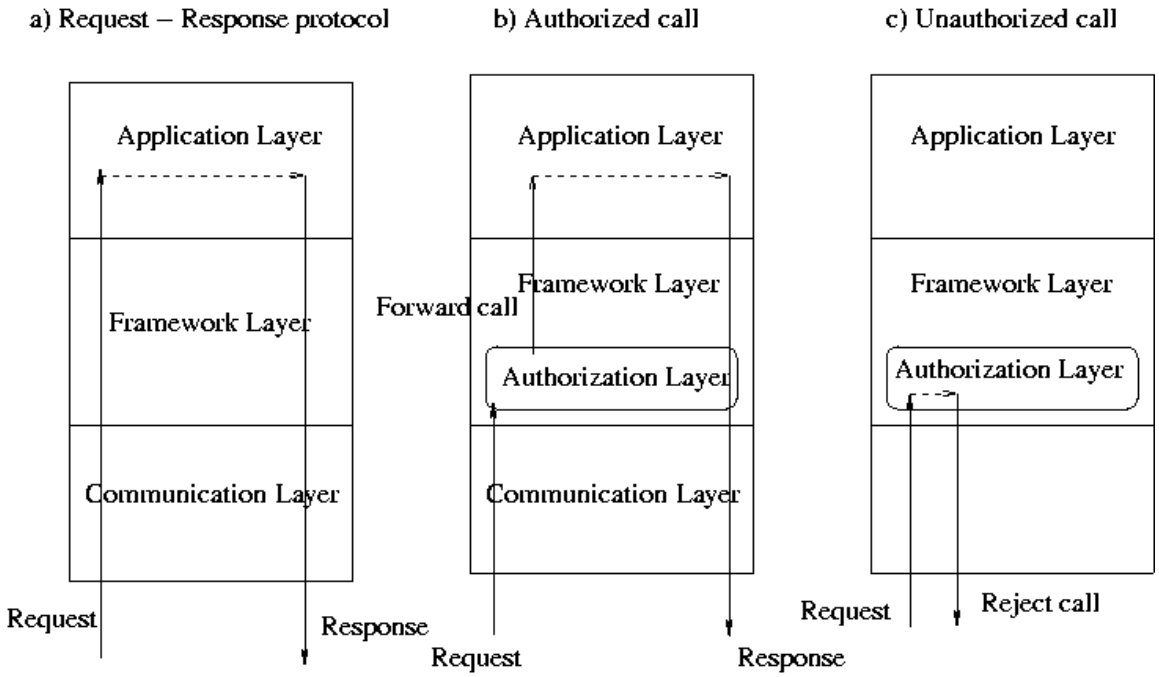

Fig. 2. Stack Abstractions in our Component Framework.

The implementation of the policy management and evaluation are abstracted into an authorization service which may be an internal implementation or an external authorization service, as can be seen in Figure 3. The authorization service provides the following features for policy management and evaluation.

- A method to dynamically add a credential or list of credentials that can be trusted for a particular action.
- A method to dynamically delete a credential that is not allowed to perform an action.
- The ability to verify that the presented credential allows the user to access the requested resource.

The authorization layer need not evaluate the policy, but just needs to know which service to contact. The internal structure of the authorization layer is separated from the details of the implementation of the authorization service. The authorization sub-layer in the framework contacts the authorization service to perform the policy check during the request-response protocol. The advantages of the model are

- The authorization model is pluggable into existing applications thus allowing for the flexibility of having secure and un-secure applications
- The authorization service acts as a bridge between the diverse security subsystems that may exist on the Grid.
- The intricacies of the policy representation, management and evaluation are abstracted away from the application.
- The model provides the user the flexibility to choose between any pre-existing security mechanisms or any application-specific mechanisms. For example if an application needs to restrict or grant access at the parameter level of a method, the authorization service module could be customized for the same.

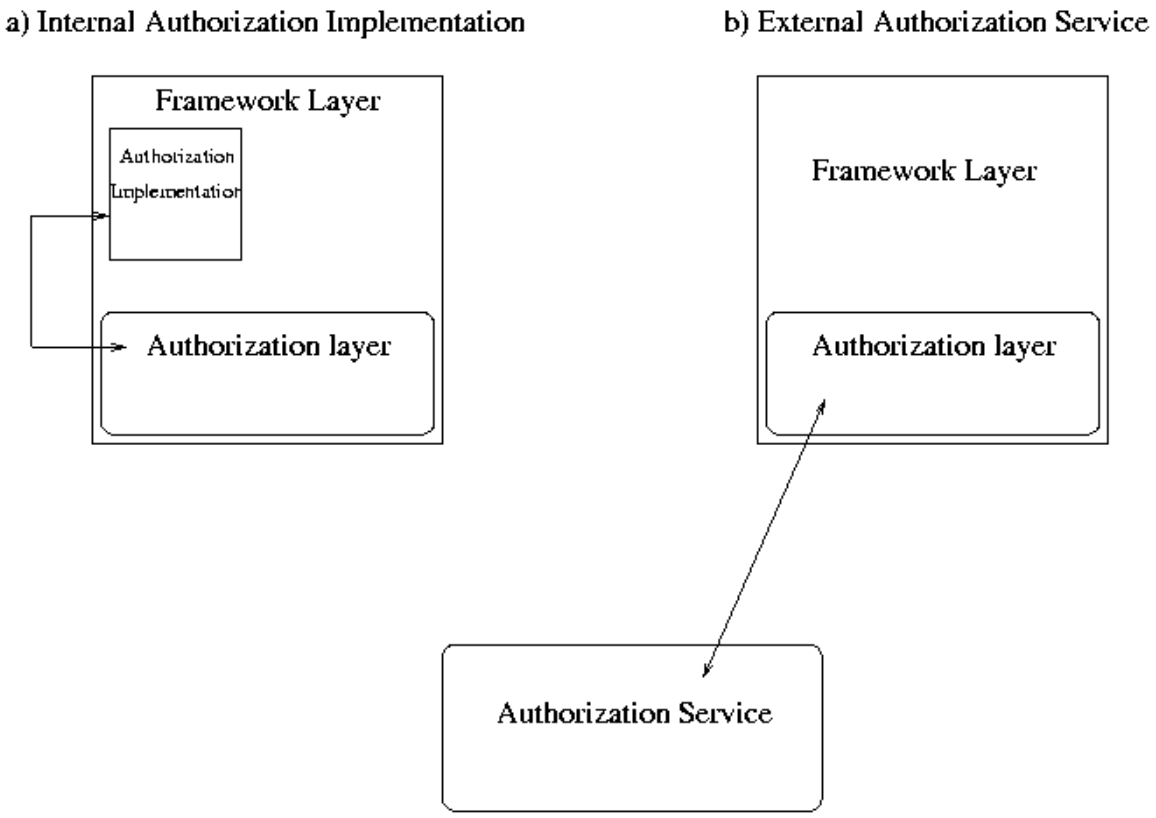

Fig. 3. Implementation of Internal and External Authorization Services.

5.2 Case Scenarios

We describe two scenarios to show how the model uses an external and internal authorization service.
Scenario 1: Component A provides a service. The component architect has designed component A to use an external authorization service X to manage restricted access to its methods.

Initialization
1. Component A registers a provides port for the service being provided.
2. Component A registers a uses port that it can use to contact the authorization service.
3. Component A's uses port is connected to the authorization service.
4. The policy information about trust relations in the environment is initialized with the authorization service X.

Policy Check
1. The call received from the communication layer is intercepted and required information (target method, port and component, identity of the caller) is extracted.
2. The authorization layer recognizes that an external authorization service needs to be contacted for policy evaluation.
3. The authorization layer invokes the policy evaluation method on the external service including the information required for the authorization check.
4. The authorization layer forwards or rejects the intercepted call depending on the result of the policy check.

Policy Management
Since the authorization service is external, the changes to the policy management are done by directly interacting with the external authorization service.

Scenario 2: Component A provides a service. The component architect has designed component A to use an internal authorization service

Initialization
1. Component A registers a provides port for the service being provided.
2. Component A registers the internal authorization service with the authorization layer.
3. Component A registers a policy management provides port.
4. The policy information about trusted users is initialized.

Policy Check
1. The call received from the communication layer is intercepted and required information (target method, port and component, identity of the caller) is extracted.
2. The authorization layer identifies that an internal authorization service exists.
3. The authorization layer invokes the authorization method, which in this case is an internal call. The information required for the authorization check is included in the call.
4. The authorization layer forwards or rejects the intercepted call depending on the result of the policy check.

Policy Management
Policy Management is done by making method calls on the policy management port. These calls have authorization policies set and controlled in the same fashion as any other provides port.

5.3 Implementation of Authorization Model in XCAT

We have a prototype implementation of the model with an internal authorization service in the XCAT framework. Some of the elements in our prototype are
Identity: We obtain the user's credential and use the Distinguished Name from the user's credential for policy information.
Policy type: We use a simple access control list.
Policy initialization: We can specify the initial policy, which includes information about the identity of the user and the actions he is allowed to perform, in XML. In our prototype the information specifies the trusted user's Distinguished Name and the methods he/she is allowed to access. This information is parsed when the component is instantiated and an access control list is initialized.
Policy check: When a method is invoked on the provides port of a component, the interceptor in the XCAT framework intercepts the method call. It parses the method name and parameters and forwards the call to the corresponding method on the provides port of the component. We have introduced some additional functionality to the interceptor by having it enforce a policy check before the call is forwarded, thus making it equivalent to the authorization layer of the stack abstraction. The user credentials required to evaluate the policy is extracted from the communication layer. The interceptor calls the authorization method and accepts or rejects the method call based on the result of the policy check. The interceptor also has the ability to exercise various levels of access control – at the method, port and parameter level. For example: if sensitive parameters are being passed to a method, the interceptor can authorize the call based on the actual values to those parameters according to the policy access information. Furthermore, the interceptor detects the presence of an external authorization service or an internal authorization service for the policy check by determining whether the component has registered a uses port for an external authorization service, or has registered its use of an internal authorization service.

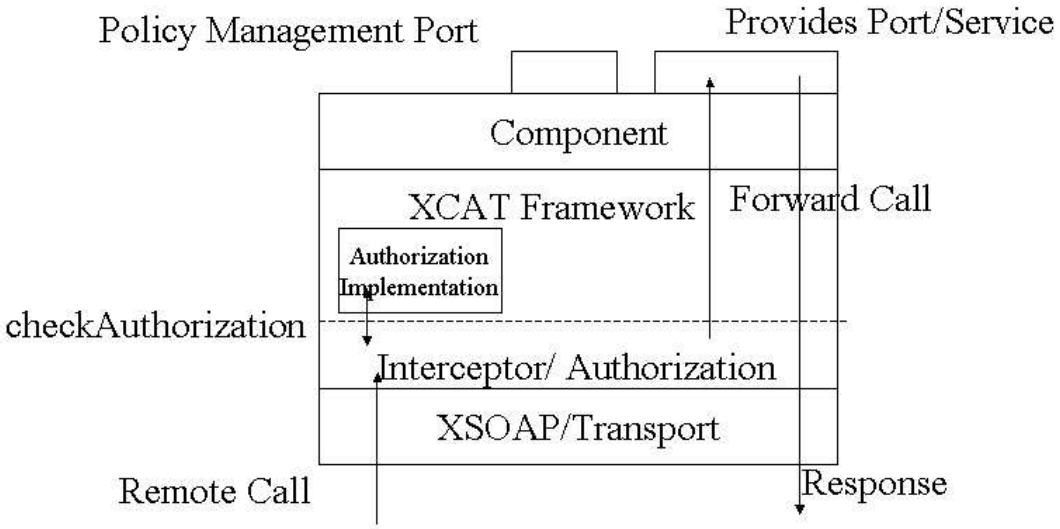

Fig. 4. Schematic of XCAT Implementation

Policy management: The policy management port adds and deletes trusted user Distinguished Names as trust relations change during the lifetime of a component. The policy management port is a provides port with methods to add and delete credentials from the access control list. By specifying the policy information for these methods, tasks of policy management are restricted to authorized users.

5.4 Authorization as an External Service

The service providers make available their resources or services on the Grid. So when authorization is presented as a service it abstracts away the details of the implementation of the authorization mechanism, from the component framework. Some of the characteristics of an authorization service are

- The authorization service will be registered at a well-known location. WSDL [5] references for the service will help in the discovery of this service.
- The service will need to be authenticated to discourage impostors. GSI's authentication protocol in the communication layer will satisfy this requirement.
- Important information such as credentials and the results of the policy evaluation will be exchanged between the service provider and the user. Encryption technologies that are part of GSI will be vital to protect this information.
- The authorization service will need to have its own scheme of authentication and authorization for allowing access to its services.
- There maybe different security sub-systems running on the Grid. The authorization system could act as the middle-ware to allow interaction between these sub-systems by doing credential conversion. A client may use Kerberos credentials while the service provider's security system may use X.509 certificates. The authorization service will take in the Kerberos credential and match it with some pre-defined protocol with the X.509 certificate expected by the service provider.

The authorization service may or may not be designed as a component in the framework. An authorization service component would provide its services, i.e. policy management and policy evaluation, through a provides port. The services would thus have authorized access as the calls to a provides port would pass through the interceptor or authorization layer.

6. Future Work

The OGSA specification will incorporate standard interfaces for authorization and policy management required by applications. We plan to make the required changes in our design so that our authorization model can be presented as an OGSA compliant service.

The use of Akenti and CAS in the given framework will give rise to interesting possibilities. For our initial implementation we are using a simple policy language specified in XML. We would like to experiment with other policy languages. We are also working towards authorization as a service with WSDL references. Currently the user needs to explicitly specify the authorization service or implementation to use. We propose to use the ongoing research on dynamic discovery services to be able to automate this process to some extent. The applicability of the model to other component models eg: CORBA Component Model (CCM) [28], Enterprise JavaBeans (EJB)[29] needs to be studied.

Authorizing every call has an immediate effect on performance. We need to measure the exact overhead and look at schemes to improve this. The possibility of using caching or buffering to improve performance needs to be investigated in greater detail.

7. Conclusions

We present an authorization model for a Grid based component-based system. The model concentrates on the authorization infrastructure in a distributed component environment. It allows the use of various levels of security. Apart from simple access or denial of access to components, it allows the establishment of fine grained access control at the method, port and parameter levels.

The design allows the use of internal or external authorization services. The model is flexible to allow the use of various existing policy languages and authorization systems to take care of advanced security needs of the user.

References

1. I. Foster, C. Kesselman, G. Tsudik, S. Tuecke. A Security Architecture for Computational Grids. Proc. 5th ACM Conference on Computer and Communications Security Conference, pp. 83-92, 1998.
2. W. Johnston, S. Mudumbai, and M. Thompson. Authorization and Attribute Certificates for Widely Distributed Access Control. Proceedings of the IEEE 7th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, WETICE '98.
3. L. Pearlman, V. Welch, I. Foster, C. Kesselman, S. Tuecke. A Community Authorization Service for Group Collaboration. Submitted to IEEE 3rd International Workshop on Policies for Distributed Systems and Networks, 2001. http://www.globus.org/research/papers/CAS_2002_Submitted.pdf
4. A. Ferrari, F. Knabe, M. Humphrey, S. Chapin, A. Grimshaw. A Flexible Security System for Metacomputing Environments. Technical Report CS-98-36. January 1998.
5. S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman. Grid Service Specification. February 2002. http://www.globus.org/research/papers/gsspec.pdf
6. Christenser E., Curbera, F., Meredith, G. and Weerawarana.,S. Web Services Description Language (WSDL) 1.1 W3C, Note 15,2001, http://www.w3.org/TR/wsdl
7. Madhusudhan Govindaraju, Sriram Krishnan , Kenneth Chiu, Aleksander Slominski, Dennis Gannon, Randall Bramley. XCAT 2.0: Design and Implementation. Technical Report 562. Department of Computer Science, Indiana University. June 2002.

8. R. Armstrong, D. Gannon, A. Geist, K. Keahey, S. Kohn, L. McInnes, S. Parker and B. Smolinski. Toward a Common Component Architecture for High-Performance Scientific Computing. In Proceedings of the 8th IEEE International Symposium on High Performance Distributed Computation, August 1999.
9. B. A. Allan, R. C. Armstrong, A. P. Wolfe, J. Ray, D. E. Bernholdt and J. A. Kohl, The CCA Core Specification In a Distributed Memory SPMD Framework, submitted to Concurrency: Practice and Experience.
10. The Common Component Architecture Technical Specification, Version 0.5. See http://www.cca-forum.org.
11. R. Bramley, K. Chiu, S. Diwan, D. Gannon, M.Govindaraju, N. Mukhi, B. Temko, and M. Yechuri. A component based services architecture for building distributed applications. In Proceedings of Ninth IEEE International Symposium on High Performance Distributed Computing Conference,Pittsburgh, August 1-4 2000.
12. J. Villacis, M. Govindaraju, D. Stern, A. Whitaker, F. Breg, P. Deuskar, B. Temko, D. Gannon, R. Bramley. CAT: A High Performance, Distributed Component Architecture Toolkit for the Grid. Proceedings of Eighth IEEE International Symposium on High Performance Distributed Computing Conference, Redondo Beach, California. August 3-6 1999.
13. Indiana Extreme Lab. XSOAP Toolkit. http://www.extreme.indiana.edu/xgws/xsoap/
14. D. Box, et al. Simple Object Access Protocol (SOAP) 1.1. W3C Note. http://www.w3.org/TR/SOAP/
15. DOE Science Grid's GSI Enabled Java SOAP client/server project. http://doesciencegrid.org/Grid/projects/soap/index.html
16. A. S. Grimshaw, W. A. Wulf, J. C. French, A. C. Weaver, P. F. Reynolds Jr. Legion: The Next Logical Step Toward a Nationwide Virtual Computer. Technical Report CS-94-21. August 1994.
17. OASIS. Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML). Technical Specification. http://www.oasis-open.org/committees/security/docs
18. OASIS. Extensible Access Control Markup Language. http://www.oasisopen.org/committees/xacml/index.shtml
19. P. Hallam-Baker. X-TASS: XML Trust Assertion Service Specification. 2001. http://www.oasis-open.org/committees/security/docs/draft-xtass-v09.pdf
20. B. Atkinson, et al. Web Services Security (WS-Security). Version 1.0. April 5, 2002. http://msdn.microsoft.com/library/en-us/ dnglobspec/html/ws-security.asp
21. M. Bartel, J. Boyer, B. Fox, B. LaMacchia, E. Simon. XML-Signature Syntax and Processing. W3C Recommendation. http://www.w3.org/TR/xmldsig-core/.
22. W. Ford, P. Hallam-Baker, B. Fox, B. Dillaway, B. LaMacchia, J. Epstein, J.Lapp. XML Key Management Specification (XKMS). 2001. http://www.w3.org/TR/xkms/
23. Argonne National Lab. Globus. http://www.globus.org
24. D. Platt. Oasis Security Services Use Cases and Requirements. Oasis SSTC. 30 May 2001. http://www.oasis-open.org/committees/security/docs/draft-sstc-saml-reqs-01.pdf
25. Xuhui Ao, Naftaly Minsky, Thu Nguyen, Victoria Ungureanu. Law-Governed Communities Over the Internet. In Proc. of Coordination' 2000: Fourth International Conference on Coordination Models and Languages, LNCS, No. 1906, pages 133-147, Springer-Verlag, September 2000, Limassol Cyprus.
26. T. Bray et al. Extensible Markup Language (XML) 1.0 (Second Edition). W3C Recommendation. 6 October 2000. http://www.w3.org/TR/REC-xml
27. OMG Security http://www.omg.org/technology/documents/formal/omg_security.htm
28. Diego Sevilla Ruiz. CORBA and CORBA Component Model (CCM). http://ditec.um.es/~dsevilla/ccm/
29. Eneterprise JavaBeans (EJB). http://java.sun.com/products/ejb/