A Usage-based Authorization Framework for Collaborative Computing Systems

Xinwen Zhang
George Mason University
Fairfax, Virginia, USA
xzhang6@gmu.edu

Masayuki Nakae
NEC Corporation
Kawasaki, Kanagawa, Japan
m-nakae@bp.jp.nec.com

Michael J. Covington
Intel Corporation
Hillsboro, Oregon, USA
michael.j.covington@intel.com

Ravi Sandhu
George Mason University
and TriCipher Inc., USA
sandhu@gmu.edu

ABSTRACT

Collaborative systems such as Grids provide efficient and scalable access to distributed computing capabilities and enable seamless resource sharing between users and platforms. This heterogeneous distribution of resources and the various modes of collaborations that exist between users, virtual organizations, and resource providers require scalable, flexible, and fine-grained access control to protect both individual and shared computing resources. In this paper we propose a usage control (UCON) based authorization framework for collaborative applications. In our framework, usage control policies are defined using subject and object attributes, along with system attributes as conditions. General attributes include not only persistent attributes such as role and group memberships, but also mutable usage attributes of subjects and objects. Conditions in UCON can be used to support context-based authorizations in ad-hoc collaborations. As a proof-of-concept we implement a prototype system based on our proposed architecture and conduct experimental studies to demonstrate the feasibility and performance of our approach.

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and Protection—*Access controls*; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection—*Unauthorized access*

General Terms

Security

Keywords

authorization, access control, usage control, UCON, collaborative computing, security architecture

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SACMAT'06, June 7–9, 2006, Lake Tahoe, California, USA.
Copyright 2006 ACM 1-59593-354-9/06/0006 ...$5.00.

1. INTRODUCTION AND MOTIVATION

Collaborative systems are becoming a popular means of providing efficient and scalable access to distributed computing capabilities. This is particularly true for applications with significant processing demands or large storage requirements. In collaborative systems, a set of nodes or organizations share their computing resources, such as compute cycles, storage space, or online services, to establish virtual organizations (VOs) aimed at achieving a particular task. These specific tasks often include large-scale distributed computing or scientific research projects [12] and may be serviced by VOs comprised of heterogeneous computing platforms. In such collaborative systems, authorization management is a fundamental problem as resource owners must 1) *prevent* unauthorized access; 2) *monitor* the legal use of their resources; and 3) *ensure* that all users abide by the agreements of the VO to which the resource has been allocated.

In collaborative systems such as Grids [11], general entities include resource users, a set of resource providers (RPs), and virtual organizations (VOs), as indicated in Figure 1. A VO is responsible for managing resources and providing some services to end-users. RPs provide the system-level resources that are managed by the VO; RPs are responsible for respecting the pre-defined access control policies (e.g., through service level agreements) that specify how resources are to be used within the VO. Finally, Resource users are provided access privileges to resources within a virtual organization. The authorization management of a collaborative system involves security relationships between all of these entities to protect the resources in a VO and ensure their availability through access control mechanisms [9].

Current authorization solutions for collaborative systems focus on centralized policy management with *privilege credentials*. In many Grid systems for example, administrators of the VO issue credentials to users that determine the resources and permissions a user can hold. That is, the permissions of a user are pre-assigned and the authorization only checks the validity of the credentials.

With dynamic user participation and resource-consuming requests, these access control solutions are neither flexible nor fine-grained. As a simple example, a user's available usage quota for a particular resource could change dynamically according to his status; pre-assigned permissions specified in credentials cannot capture the real-time properties of a user submitting a task to an RP. Also, current approaches do not consider the usage status of a shared object in authorization (e.g., the usage context or constraints of resource

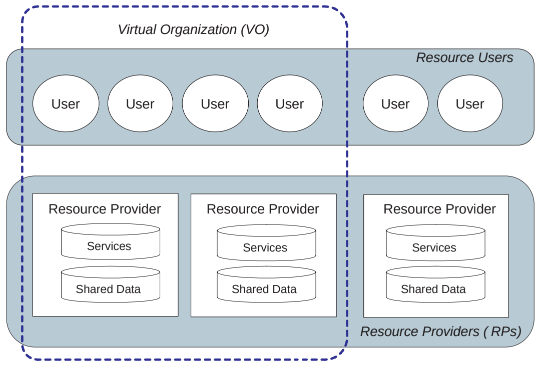

Figure 1: An example virtual organization (VO)

objects). For example, a scientific instrument can be shared within a research VO but critical functions may only be used by a single organization or group at any time. Attributes can be defined to specify that remote users are allowed to use the instrument by submitting corresponding attributes, such as role name (e.g., PI in the research project), while the administrator of the instrument can determine the permission that a user can have at the moment of access, thus ensuring, in real-time, that other accesses are not in conflict with the request. Section 3.5 gives additional examples of dynamic access control policies for collaborative systems. For these purposes, authorizations based on general and real-time attribute values of subjects and objects are required.

In addition, ad-hoc and pervasive collaborations bring new challenges for authorization management. In ad-hoc collaboration, where no pre-existing VO management infrastructure is in place, subject authentication is not available and authorization decisions are, instead, dependent on contextual information [8], such as the location and time of the access request. Existing approaches lack flexibility to support context-based authorizations in collaborative systems.

In this paper we propose a generalized authorization framework for collaborative systems following the objective-model-architecture-mechanism (OM-AM) approach [18]. In accordance with dynamic authorization requirements in collaborative systems, we present an access control model based on UCON [15, 25]. By leveraging flexible policy specification and attribute mutability of UCON, our model not only supports VO-level authorization policies, but also usage constraints defined by each RP. We propose a security architecture for Grids and other general collaborative computing systems that leverages a centralized attribute repository in each VO and a usage monitor in each RP for attribute management. Finally, we implement a prototype system to show the feasibility and performance of our framework. The effectiveness of usage control is demonstrated with access control policies based on subject and object attributes, which are specified with extensible access control markup language (XACML) [14]. The performance of the system is studied with the dynamic properties of decision processes.

The remainder of this paper is organized as follows: Section 2 provides an overview of our proposed framework. Section 3 presents the usage control model and various policies that can be specified in collaborative systems using this model. Section 4 describes the proposed architecture. Section 5 illustrates an implemented prototype according to our framework and presents some experimental results. Section 6 presents some related work in Grid-like and other general collaborative computing systems. Section 7 summarizes this paper and discusses ongoing and future work.

2. SYSTEM FRAMEWORK OVERVIEW

We develop our authorization framework by following the OM-AM methodology [18]. In this four-layer approach, the objective and model layers articulate *what* the security objectives and trade-offs are, while the architecture and mechanism layers address *how* to meet these requirements. The distinction between security objectives (policies) and mechanisms has been long recognized. OM-AM seeks to bridge the gap between the what (policies) and how (mechanisms) by introducing two additional layers of models and architectures.

The basic requirement of the authorization problem in collaborative systems is to control accesses to shared resources by users. Furthermore, with the heterogeneous computing environments in these systems, the control should be scalable, dynamic, and fine-grained. Previous work emphasized authorizations based on user identity or group membership, while the usage properties of the shared resources - such as the status of shared objects and the dynamic parameters of subjects in a VO - were not considered. Also, context-based authorizations are not well supported in existing approaches, such as in ad-hoc collaborations without well-established authentication infrastructures.

Based on these requirements, usage control (UCON) [15, 25] is used in the model layer of our framework, as it is an attribute-based access control model and comprehensively considers authorizations and conditions in access control decisions. In UCON, authorizations are predicates defined on subject and object attributes, while conditions are environmental restrictions represented by system attributes, such as time, location, load, etc. UCON uses the real-time values of subject and object attributes for authorization decisions in a *session-based* manner. Authorizations and conditions are enforced not only when a subject generates an access request, but also during the whole ongoing stage of the *usage session*, which is referred to as *decision continuity*. As the side-effects of the usage, subject and object attributes can be updated; this is referred to as *attribute mutability* in UCON. Previous work have shown that decision continuity and attribute mutability can provide flexible, fine-grained, and dynamic access control [15, 25].

From the point of view of architecture, a typical authorization system includes a policy decision point (PDP) and a policy enforcement point (PEP). Some authorities may exist, such as identity and attribute authorities, either inside a VO, or externally. Existing solutions in authorization management can be divided into two types of architectures: pull mode and push mode [13, 14]. Figure 2 indicates the conceptual data-flow of these two modes. In the push approach, each subject presents his related information (e.g., identity and attribute certificates) to the PDP and the decision is sent to the PEP; while in the pull approach, a PEP collects the related information of a subject and queries the PDP for policy decision. Considering the temporal and dynamic attributes of subjects and objects, pure push or pull based architecture is neither efficient nor scalable for collaborative systems. For example, if a subject attribute is mutable, then an access request or an ongoing access may update the attribute, which affects other ongoing accesses. In a pure push-based architecture, this requires that the subject continuously obtain the latest attribute value and report it to the PDP. While in a pure pull-based architecture—since the PEP should keep querying the PDP with new attribute values—extra communication overhead between them is introduced.

In the architecture layer of our framework, both PDP and PEP are located on the RP side. For an access, the PDP collects the subject and object attributes, as well as system attributes provided by supporting services in the VO, and makes the control decision, which is enforced by the PEP. For attribute acquisition, *immutable*

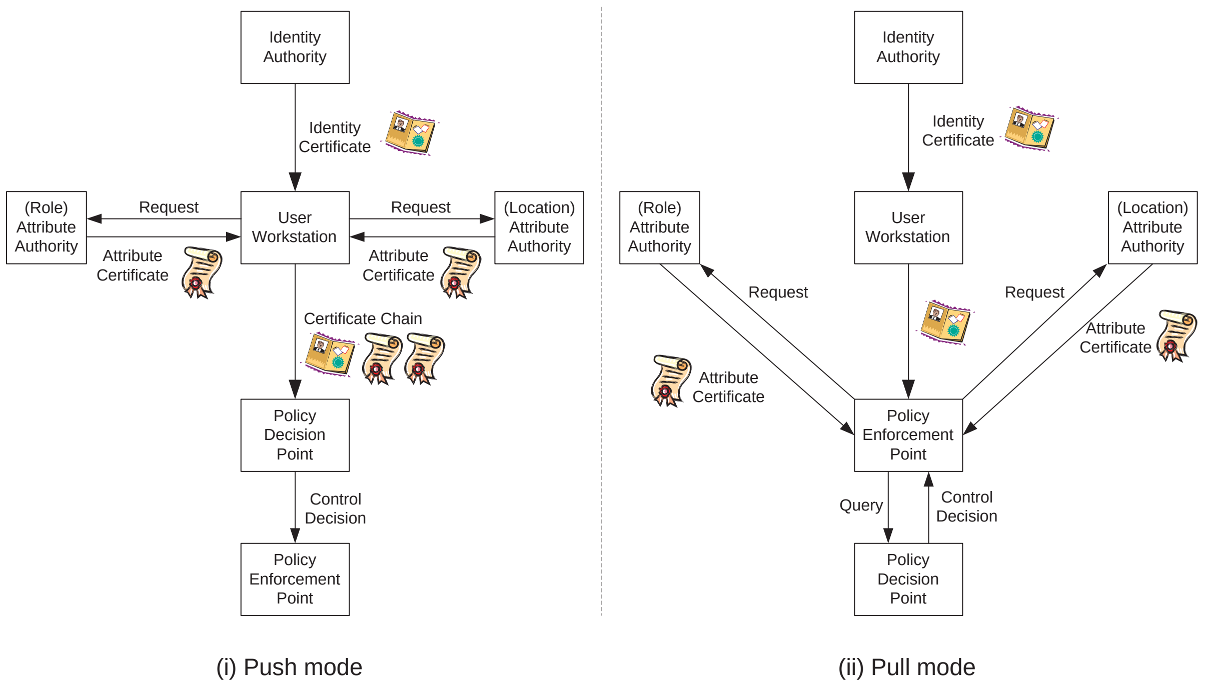

Figure 2: Push and pull modes of authorization architecture

(persistent) subject attributes (e.g., role and group membership) are *pushed* to the PDP by the requesting subject. *Mutable* subject attributes are *pulled* by the PDP from the VO's centralized *attribute repository*, and mutable object attributes are pulled by the PDP from the local RP's *usage monitor*, which records the temporal and dynamic properties of the object. This *hybrid* approach with push and pull mode for attribute acquisition and management improves the efficiency and scalability of our framework.

The updates of mutable subject attributes are performed by the PDP and reported to the centralized repository, and the updates of mutable object attributes are captured by the local usage monitor. As the conditions of UCON are built on system attributes, which can be changed and reported in a VO, condition checks are similar to the mutable subject attributes, which are stored in the centralized attribute repository and pulled by the PDP. Any update of subject or object attributes and any change of system conditions trigger the re-evaluation of the policy by the PDP according to the ongoing usage session, and revoke or update attributes if necessary. This approach seamlessly supports decision continuity and attribute mutability of UCON within concurrent usage sessions.

In the mechanism layer, implementation techniques in several aspects are considered in our framework, such as policy specifications, subject and attribute authentications, trusted update of attributes, and secure communications between computing components in the system. The mechanisms of a security system need to not only achieve the security objectives, but also meet the performance requirements of different usage and collaborative scenarios. As a proof-of-concept we implement a prototype system based on our proposed framework. Section 5 shows the details of the techniques that are used in our prototype and provides performance results.

3. USAGE CONTROL FOR COLLABORATIVE SYSTEMS

Generally in a collaborative system, an RP has the ultimate control of its shared resources. At the same time, as a member of a VO, an RP should respect the VO's community policies (e.g., to allow the accesses of shared resources and services to authorized subjects and provide the expected quality of services). For ad-hoc group collaborations, some policies should be specified by the group administrator or owner and followed by each participant for a particular task. For example, in a temporal collaborative application, an object shared by its owner can only be accessed by devices in the same room as the owner. An access control model should provide a comprehensive and systematic view of the security requirements in a collaborative system and should be configurable to support individual policies.

We use UCON as the access control model in our framework due to its strong expressive power and policy specification flexibility. Starting with a brief introduction of the concept of UCON, in this section we present a UCON model for collaborative systems by leveraging the features of decision continuity and attribute mutability. We then discuss various policies in collaborative systems that can be specified with this model.

3.1 Overview

In a usage control (UCON) system [15, 25], each entity (subject or object) is represented using a set of attributes, depending on application and security requirements. A system state is specified by attribute-value assignments for each entity in the system. Predicates can be defined based on attributes as variables and constants. In each system state, a predicate is evaluated by using the values of the attributes in the state.

Besides subject and object attributes, UCON considers obligations and conditions in access control decisions. Obligations are actions that have to be performed by a subject before or during a usage. Conditions are system restrictions under which a usage can be allowed (e.g., system status and environmental information). Conditions are represented as predicates built on some system or environmental attributes [25]. The UCON model used in this paper only has authorizations and conditions, which are specified by subject/object attributes and system attributes. We leave the inclusion of obligations for future work.

The most important properties that distinguish UCON from traditional access control models are the continuity of usage decisions and the mutability of subject and object attributes. In UCON, control decision components (authorizations, obligations, and conditions) are not only evaluated before an access, but also during the usage process of an access. If any of them does not hold during the usage process, the ongoing access is revoked by the system. That is, an access control in UCON is not a one-time check and enforcement, but a continuous process. Mutability means that for a usage process, subject and/or object attributes can be updated as the side-effect results of granting of the access and processing of the usage. Attribute updates can be performed before, during, or after a usage process. Due to the existence of concurrent accesses in a system, attribute updates may invoke cascading authorization checks (e.g., an attribute change in a usage involving a particular subject and object may affect another concurrent usage of the same subject or another subject's access to the same object). Besides subject and object attribute updates, decision continuity also captures the changes of system attributes, such as system time and load, although how system attributes change are not defined in the core models of UCON.

3.2 UCON Model for Collaborations

In collaborative systems, an object is a sensitive shared resource or a service in a VO such that particular actions (rights) can be performed by a subject. A subject is a user that can generate access requests to objects provided by a VO. Administrators in the VO and RPs are also subjects that can define or change security policies. In this paper we focus on the control of general resources where the subjects are resource consumers, while the administrative aspect of the system is not considered.

Subject attributes include role or security clearance in a VO, or a group membership within a VO. Additionally, specific attributes can be defined depending on the requirements of a particular application, such as the quota of a subject for some resources in a VO, conflict of interest groups, etc. For example, when an application has a shared resource that can only be used by a single group at a time, a subject attribute specifying the conflict groups should be defined. Object attributes can include general object properties such as type, ownership, etc., but also, application specific attributes can be defined by system administrator or designer, such as usage status, inclusive/exclusive accesses, etc. Previous work has shown the expressive power of UCON to specify various access control models and policies with different attributes [15, 25].

As discussed in [16], attributes that are updated according to UCON policies are mutable or system-controlled, while attributes managed by administrator are persistent or administrator-controlled and generally do not change as a result of accesses. For example, a subject's role name generally is assigned by the security officer of an organization according to a user's job functionality, and does not change because of an access requested by the subject. In general, only the security officer can update the role name of a subject for organizational purposes, e.g., because of the change of the user's job functionality.

As conditions are environmental restrictions, system attributes are introduced. Specifically, a condition is a predicate built on system attributes to specify the restriction that has to be satisfied before or during a usage process. Although system attributes are not updated in a UCON policy, they change due to the changes of the system environment. For example, the system attribute $location$ changes when an accessing device moves out of a room, and this change, according to a policy, may affect the result of a subsequent or an ongoing access.

3.3 UCON Policies and Scheme

With authorizations based on subject and object attributes and conditions based on system attributes, a UCON policy can be defined as the following shows.

DEFINITION 1. *A policy maps a usage permission of* (s, o, r) *to a tuple* $(P_{pre}, P_{on}, UP_{pre}, UP_{on}, UP_{post})$*, where* P_{pre} *and* P_{on} *are sets of attribute predicates that need to be satisfied before and during a usage process, respectively;* UP_{pre}*,* UP_{on} *and* UP_{post} *are sets of update actions that are performed on the attributes of* s *and* o *before, during, and after the usage process, respectively.*

In this definition, s and o are parameters of the policy, r is a generic right, P_{pre} and P_{on} are conjunctions of predicates built on s's and/or o's attributes or the system attributes, which are called pre-decision components and ongoing decision components, respectively. A predicate takes one or more attribute values and constants, and returns boolean values. For a particular policy, the pre-decision components and ongoing decision components may or may not be the same. If any predicate in P_{pre} is not satisfied when the access request is generated, a usage cannot be granted; after granting, if any predicate in P_{on} is not satisfied during a usage process, the ongoing access is revoked by the system. Note that a disjunctive form of predicates is not needed explicitly since we can have one policy for each disjunctive component. UP_{pre}, UP_{on}, and UP_{post} are pre, ongoing, and post update actions on subject and/or object attributes. An update action returns a new value to a specific attribute, which can be a constant, a function of its old value, or a function of other attributes' values. An update in UP_{on} can be a one-time action, a continuous action, a periodical action, or an action under particular conditions. For example, an update of usage time during access is a continuous action, while an update of idle time is a conditional action only when a subject is idle during access. There are two types of post update actions in a policy, one for updates after a subject ends an ongoing access (with $endaccess$ action), the other for updates after an ongoing access is revoked by the system (with $revokeaccess$ action), e.g., because of a subject/object attribute or a system condition changes that make any predicate in P_{on} invalid. In a single policy these two types of update actions may or may not be the same. Formal semantics and policy specification of UCON are described in [25].

In a high-level view, a UCON policy evaluates a set of predicates built on subject and/or object attributes and system attributes and grants or continuously allows a permission if all of them are satisfied. As the side-effect of granting access, some attribute values may be updated. In UCON, a policy only specifies the attribute updates of the accessing subject and target object, but not system attributes (which are updated automatically by the system).

A set of attributes, predicates, and policies make up the authorization scheme of a system.

DEFINITION 2. *A UCON authorization and condition* scheme *is a 5-tuple* (ATT_a, ATT_c, R, P, C)*, where* ATT_a *is a fixed set of subject and object attribute names,* ATT_c *is a fixed set of system attribute names,* R *is a fixed set of generic rights,* P *is a fixed set of predicates built on* ATT_a *and* ATT_c*, and* C *is a set of policies.*

In a UCON scheme, if an attribute appears in UP_{pre}, UP_{on}, or UP_{post} of any policy, it is *mutable*; otherwise, it is *immutable* or *persistent*. As the architecture proposed in this paper supports condition checks, system attributes are considered as mutable in our framework.

Note that all policies in a UCON scheme are defined for positive permissions (to enable permissions). For an access request, if there is no policy to enable the permission according to the predicates, the access is denied by default.

3.4 Continuity and Mutability

Both the concepts of decision continuity and attribute mutability are based on a continuous ongoing accessing process. A *usage session* is defined as an accessing process initiated by a subject s to an object o with a generic right r, according to a UCON policy. Note that in a UCON scheme, there may be more than one policy regarding a single permission (s, o, r), each of which can have different attribute predicates and update actions. A usage session refers to a specific usage process with (s, o, r) that follows a particular policy, where the attribute predicates are evaluated based on s, o, and the system conditions, and updates are performed on s and/or o's attributes.

Continuous security check in a single session is invoked by two different types of attribute mutability: the updates of subject or object attributes which are specified by the policies in a scheme, and the updates of system attributes which are the changes of environmental or contextual information. As defined in a UCON policy, subject and object attributes can be updated before, during and after a usage session. Also, concurrent usage sessions with regarding to the same subject or object can affect each other, as the attributes are shared variables between usage sessions. That is, an update in one usage session may enable or revoke another usage session. On the other hand, although how system attributes change is not specified in UCON policies, any change that causes the condition predicates to become invalid should be reflected during the usage session (e.g., the system should revoke the ongoing access).

3.5 UCON Policies for Collaborative Computing

With the properties of decision continuity and attribute mutability, flexible policies can be defined, as conceptually and formally studied in previous work [15, 16, 25], such as role-based access control (RBAC), dynamic separation of duty, Chinese Wall, and policies with low or high watermark properties. Particularly for collaborative computing systems, we summarize different types of policies that can be specified with different attributes and predicates in UCON as follows.

- *Consumable Resource Management* Subject attributes can be defined to specify dynamic resource management policies. For example, a policy defines that a subject can only have a fixed amount of storage totally in a VO. An attribute can be used to record its current used or available amount. Whenever a subject generates an access request to an RP, the RP uses this attribute value to determine if the request can be approved. During the usage session, the RP should update this attribute with the storage that the subject has used in this session. Also, any change of this attribute in concurrent usage sessions should invoke the re-checking by the RP. Similar mechanism can be used to control the usage of other resources such as CPU cycles, network bandwidth, etc.

- *Credit or Reputation Management* A subject's access may generate credits or reputation points, which in turn can enable other permissions in a VO. Similar to above, subject attributes can be defined to capture these aspects.

- *Status of Shared Objects and Collaborative Tasks* In a cooperative environment, when a subject is doing *write* operations on an object, other collaborative subjects cannot write or modify it to preserve the integrity. Also, a subject's particular operation on a collaborative task may require that some necessary pre-operations have been performed by other subjects. Attributes can be defined to monitor the shared object or task' status and used to determine the just-in-time permissions of a subject.

- *Exclusive/inclusive Collaborations* Mutable attributes can be used to enforce exclusive or inclusive rights for shared objects and resources. Exclusive attributes are used to resolve conflict of interests while inclusive attributes can be used to resolve consolidated interests. In collaborative systems, an operation or task may require concurrent involvement from multiple subjects with particular attributes, while exclude other subjects.

- *Constraints of Collaborations* Constraints can be defined for fine-grained and flexible collaborations with subject/object attributes or system attributes. For example, an RP's resource can be accessible to a subject only during a particular period. As another example, the access permission of a subject can be temporally delegated to another subject by the collaborative relationship between them.

All these policies can be defined with the conceptual or formal model proposed in previous work [15, 25]. Detailed policy specifications are out of the scope of this paper.

4. ARCHITECTURE

This section first introduces the overall architecture of our framework, followed with the features to support attribute mutability and decision continuity. The goal of our architecture is to support the general UCON model introduced in the previous section.

4.1 Overview

Figure 3 shows the architecture overview of our framework in the context of Grid Security Infrastructure (GSI) [10]. Typically, the architecture includes three main components within a VO: user platforms, individual resource providers (RPs), and an attribute repository (AR). AR is a centralized service to store and push mutable subject and system attributes in a VO. Object attributes are stored in a usage monitor (UM) on each RP side. For simplicity identity and external attribute authorities are not included here.

A usage session is initialized by a subject (e.g., a resource consumer) and works as follows. First the subject generates an access request from its platform, and the request is submitted to an RP with the client-side proxy [23] (step 1). Persistent subject attributes are pushed by the requesting subject to the policy decision point (PDP) in the RP side (step 2). After receiving the request, the PDP contacts the AR and retrieves the mutable attributes of the requesting subject (step 3 and 4) and the object attributes from the UM (step 5). The access control decision according to VO policies is issued by the PDP after collecting all related information (subject and object attributes and system attributes) and evaluations of policies. The decision is forwarded to the PEP and enforced in the execution environment of the task (step 6).

As the side effects of making a usage decision, attribute updates are preformed by the PDP according to corresponding policy. New subject attribute values are sent back to AR (step 7), and the object attributes are updated to the UM (step 8). As a result, updated attributes can be shared between different usage sessions, and the PDP always checks the AR and UM for latest attribute values when a new access request is generated.

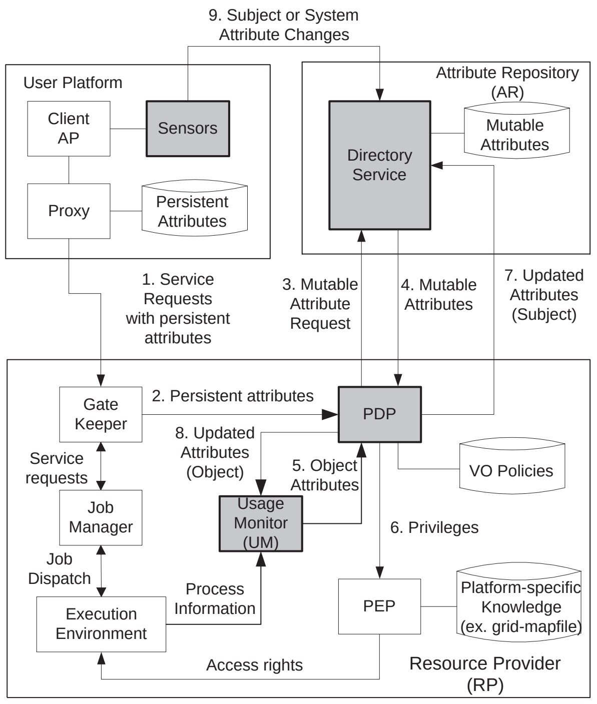

Figure 3: Usage-based authorization architecture

In a collaborative system, a subject can initiate multiple usage sessions to different RPs or to a single RP with different objects. Also, a shared resource can be accessed by multiple subjects. By centralized mutable subject attributes, our architecture supports concurrent usage sessions from a single user. At the same time, object attributes are monitored by the UM and captured by the PDP when new usage requests are generated. Therefore our architecture can support concurrent usage sessions to a single object.

4.2 Attribute Acquisition and Management

As UCON is attribute based, a critical requirement to correctly enforce access control policies in a UCON system is to get just-in-time attribute values. One of the novelties of our architecture is that it supports different modes for attribute acquisition and management with regarding to different types of attributes.

Push Mode for Persistent Attributes For the persistent attributes of a subject, since there are no policies in a UCON scheme that update their values, they are pushed by the subject to the PDP of the RP to which it submits an access request. As these attribute values do not change during the usage session, they are only pushed and evaluated once. Typical persistent subject attributes include those provided by external authorities, such as identity or attribute certificates issued from other organizations. Also, attribute certificates issued by internal authorities are also persistent attributes, such as a subject's role or group in a VO, since they are not updated as results of any usages. An object also can have persistent attributes, which are maintained by its RP and checked by the PDP when an access is generated to it.

Note that as aforementioned, persistent attributes can be updated by system administrators or security officers for organizational purposes. There should be parallel update and management mechanisms for these attributes. For example, a simple option may require that a subject's persistent attributes can be updated only if the subject is not accessing any object. In this paper we focus on the core aspects of UCON, and the administrative updates of persistent attributes are not considered here.

Pull Mode for Mutable Attributes When a subject generates an access, mutable subject attribute values are pulled by the PDP from the centralized AR. For mutable object attributes, they are provided by the UM of the RP.

Update Propagation for Mutable Attributes Because of attribute mutability, an ongoing usage session needs to repeatedly check the attribute values and evaluate the policy. As mutable subject attributes are maintained in a centralized AR, when an update happens, e.g., an update from another usage session, the new attribute value is propagated by the AR to the PDP. Similarly, mutable object attributes can be propagated by the local usage monitor to the PDP. As update propagation is related to ongoing decision checking, the details are described in next subsection.

4.3 Mutability and Continuity

4.3.1 Updates of Attributes

As described in Section 4.1, in a usage session, pre-updates are results of the approval of the access request performed by the PDP, according to a particular usage policy. That is, pre-updates are triggered by an access request based on a policy. During a usage session, ongoing and post updates are triggered by some events, either from the subject or from the system.

Ongoing Updates Ongoing attribute updates are invoked by some events in a system, such as time events and events that change system status. For example, for a subject's time slice which is repeatedly decreased in a usage session, the update is invoked by the time event of the system. For another example, a subject's attribute update about the usage status (e.g., from $busy$ to $idle$) is invoked by a system event that monitors the status. This kind of event can be monitored by the UM and reported to the PDP. Once the PDP receives an event, the attribute values of the object and subject are retrieved and evaluated and corresponding policies are re-checked by the PDP if necessary (e.g., to allow an ongoing usage to continue or revoke it). For simplicity the event trigger and transmission are not indicated in Figure 3.

Post-updates Post updates can be triggered by two types of events: a subject's action to end an usage session, and the revocation of an ongoing usage session by the system [25]. For the first type, the UM reposts an $endaccess$ event to the PDP after a subject ends the usage. The PDP performs the updates according to the policy and reports new attribute values to AR and UM, respectively. For the second case, as the revocation of an ongoing access is the result of the decision continuity (described shortly) enforced by the PDP, the post-updates are performed by the PDP and reported to the AR and UM, respectively.

System Attribute Changes Besides the PDP-performed attribute updates for subjects and objects, system attributes are changed by external events. For example, as the mobility of a portable device accessing an object, the sensor of the device or connection service provider reports its location information to the attribute repository whenever the device is moving (step 9 in Figure 3). In this paper we assume that event detection and reporting mechanisms for system attribute changes are provided by the functional components of a system, which are not explicitly included in our architecture.

4.3.2 Continuous Enforcement of Policies

During a usage session in UCON, a policy is checked and the decision is enforced repeatedly if there are ongoing decision components in this policy. Since a decision component is built with an attribute predicate, ongoing checks, in practice, are triggered by attribute changes during a usage session (e.g., the updates of mutable subject and object attributes and the changes of system attributes, as discussed above).

Due to the existence of concurrent usage sessions in a collaborative system, attribute values have to be synchronized between different RPs and an update event in one RP has to be propagated to other RPs. In our architecture, centralized AR acts as a bridge to forward real-time attribute values to RPs. Specifically, when a PDP of an RP (say RP_1) contacts the AR for a usage request (step 3 in Figure 3), the AR logs this request with related attribute names. Upon receiving a newly updated attribute (e.g., from RP_2 or a system attribute change event), the AR issues an updated attribute certificate with the new value to RP_1, which can re-evaluate the policy of the ongoing usage session with the new attribute value. When the predicates of the policy are still satisfied, the ongoing access is allowed to continue; otherwise, a revocation event is generated by the PDP and the decision is enforced by the PEP. For the change of an object attribute, since it is monitored by the UM, the ongoing decision check can be locally implemented.

4.4 Other Related Issues

4.4.1 Authenticity of Attribute Values

The authenticity of a subject's attributes depends on three aspects: the authentication of the subject, the binding between a subject's identity and its attributes, and the integrity of the attribute values. In general the identity of a subject is a certificate, such as a public-key certificate. Also, persistent attributes and their values are certificates or credentials, signed by some authorities. Since in general an identity authority is different from attribute authorities, a mechanism is needed to bind identity certificates and attribute certificates.

For mutable subject attributes, the AR of a VO is the authority to ensure their authenticity and integrity. This implies that the attribute certificates issued by the AR should be trusted by individual RPs. As managed inside an RP, the authenticity and integrity of mutable object attributes can be easily achieved, e.g., by existing trust infrastructure in the same organization.

4.4.2 Concurrency Control for Updates of Mutable Attributes

With concurrent usage sessions in a system, mutable attributes can be updated in multiple sessions simultaneously, therefore concurrency control should be considered to maintain the integrity of their values. Specifically, when a mutable subject attribute is going to be updated in a session, it cannot be pulled by other sessions until the update operation be finished. Similar problem exists in the updates of mutable object attributes between concurrent sessions. As traditional mechanisms can be used in our architecture, such as two-phase locking protocol, the details of concurrency control are not included in this paper.

5. PROTOTYPE IMPLEMENTATION AND EXPERIMENTAL STUDY

To show the feasibility and performance of our framework, we implement a web-based prototype system, which enables a group of software developers to share and collaboratively develop application code from different user sites (locations). This section first introduces the overview of the prototype architecture, then demonstrates two implemented UCON policies with our prototype, and finally presents some performance results.

5.1 Prototype Overview

The RP in our prototype provides a platform for developers from different corporations to develop applications collaboratively. The core building block of the RP is a concurrent revision control system called Subversion [5], which is integrated with Apache WebDAV module (mod_dav) [2]. The prototype architecture is similar to the general architecture proposed in previous section. Specifically, the AR is a directory service built with OpenLDAP 2.0.27 [3]. OpenSSL 0.9.6b [4] is used to build mutual authentication and secure communication channels between AR, user platforms, and RPs. The "sensor" program in the user platform simulates a component to detect the platform's location information as the subject's attribute and update its value to the AR.

Subversion has an ACL-based access control mechanism, which controls what data user is able to download and/or upload to a resource provider. Since the access control is performed without referring to any attribute changes, it is hard to enforce general usage control policies such as those discussed in Section 3.5. In our prototype, we implement an Apache authorization module (mod_authz_ucon) as an interface from Apache to the PDP module. For every access request to the Subversion, mod_authz_ucon forwards it to the PDP module, and approves the request if the PDP module allows it.

The RP is built on a Linux-2.4.18 box which has Pentium IV 1 GHz CPU and 1 GB memory, and uses Apache 2.0.54 with mod_ssl and mod_dav, and Subversion 1.2.3. The mod_authz_ucon is written in C (with the gcc-4.0.1 optimization level -O2), and the other components (PDP and UM) are in Java 1.4.2. These modules are connected with a local loopback network interface (`lo`). The UM uses DB4Object [1] as an object-oriented database to store the object attributes that the UM has captured. The user platform used in the prototype system is built on a Windows XP machine which has Pentium M 1 GHz CPU and 768 MB memory. The sensor is written in Java 1.4.2 which simulates to monitor the user-location information and sends changes to the AR.

The following subsections describe individual implementation issues.

5.1.1 Policy Specification

Our prototype uses the extensible access control markup language (XACML) [14] to specify UCON policies. XACML is an open-standard format to specify access control policies, and expected to be widely used with the properties of interoperability and extensibility. Using the Sun's XACML library [6], the PDP module interprets XACML policies and makes access decisions.

A UCON policy can be described in XACML format as the following shows:

```
<Policy PolicyId="(policy-name)"
        PolicyCombinationAlg="rule-combining-algorithm:permit-overrides">
  <Target>
    <Subjects>(predicates over subject attributes)</Subjects>
    <Resources>(predicates over pure-object attributes)</Resources>
    <Actions>(predicates over access rights such as read and write)</Actions>
  </Target>
  <Rule effect="permit"/> (Specification that this policy is positive)
  <Obligations>(Specification of attribute-update actions)</Obligations>
</Policy>
```

where the predicates in the UCON policy are described in `<Subjects>` and `<Resources>` elements, the rights are in `<Actions>` element, and the update actions are defined in `<Obligations>`

element [1]. Note that an XACML policy only specifies attribute requirements before an access and possible updates after an access (P_{pre} and UP_{post} defined in Section 3.3). In our prototype, subject attribute changes during a usage session are implemented with the sensor program in the user platform. Also, the ongoing check P_{on} in a UCON policy is achieved by individual checks according to an XACML policy, triggered by corresponding attribute update events or system condition changes. For example, to capture the ongoing check based on a user's location during a session, the sensor program generates an event to update the subject attribute corresponding to the movement of the user platform, which in turn triggers the policy re-evaluation by the PDP.

5.1.2 Usage Monitor

In our prototype, the UM captures an object-attribute change such as the creator's identity from Subversion. The identity (X.509 distinguished name) of a user is presented in the user's certificate, e.g., `CN=Alice, OU=VO1`. Since the identity includes a VO name (`VO1`) to which the user belongs, the PDP module can identify `VO1` as the object's assignment, and enforce a `VO1`-dependent policy to the object. This means that an RP can host several VOs without any interference between them.

5.1.3 Attribute Update Propagation

Whenever a user's platform location changes occurring in an individual platform, the updated value has to be propagated to other related platforms, as described in Section 4.2 and 4.3. To ensure the attribute value's authenticity and integrity, the AR, UM and PDP perform mutual authentication with SSL v3 to build a secure channel for every communication. By using VO-based X.509 certificates in the authentication protocol, the communication is restricted to a particular VO. In this way, we can build the trusted path for the attributes to be propagated securely.

5.2 Enforce UCON Policies

In order to demonstrate the feasibility of usage control with the prototype, we consider two scenarios with location-based and task-based access control policies, respectively. As a use case, consider two collaborative software development projects: one between Alice and Bob from different corporations (Corp. A and B, respectively), and one between Alice and Chris (from Corp. C). The project between Alice and Bob is performed in VO1, and the other in VO2. Also, Corp. B and C are assumed to have a conflict of interest to each other.

5.2.1 Location-based Access Control

Although Internet-based collaborative systems enable developers to work together remotely, face-to-face meetings are still required so as to define and confirm application specifications in depth. This requirement is satisfied with mobile technologies such as laptop and hand-held computers with wireless network capability. Such technologies, however, pose a security threat regarding confidentiality of the application code. Now suppose that Alice visits Chris (in Corp. C) for the VO2 project. In this context, Alice should not be able to refer to the VO1-related data in Corp. C, even though Alice is also a valid member of VO1, since VO1-related data might include sensitive information of Corp. B.

A UCON policy is defined according to this requirement with XACML format, which restricts the user's location to Corp. A or B to access any objects in VO1. In our prototype, Alice's location change is detected by the sensor equipped on Alice's platform , and reported to the VO1's AR. With this policy, the PDP module accepts a request for the VO1 data when Alice is in Corp. A , otherwise rejects it.

5.2.2 Task-based Access Control

In collaborative software development, work-flow management is crucial to preserve the integrity of the whole development process. For example, when Alice is testing a software module, Bob has to suspend his modification to it. In our prototype, a task's progress is kept as an object attribute $InUse$. By default, the $InUse$ attribute takes the value `FOR_DEVELOPMENT`, which means that any developer can access the object.

When Alice tries to test a module by creating a `LOCK` file in the module's source directory, the $InUse$ value of the relevant objects is updated to `FOR_TEST`, which means that any developer except the tester (i.e., Alice) cannot access the objects. Another policy is needed to allow the tester's access on the object being tested , in which a `<Condition>` element is used to compare the subject identity with an object attribute `last-accessor-id`, which indicates that only the subject which has locked the object can access it. The policy to unlock an object can be specified in a similar way.

5.3 Performance Evaluation

As a usage control decision is dynamically determined by subject and object attributes, which are either pulled or pushed to the PDP of each collaborative arena for evaluation, the performance of the system should be considered. According to our proposed architecture, persistent attributes are pushed by the requesting subject and this can be a one-time operation in a single usage session, which does not affect the runtime performance. The main overhead of the system introduced by usage control is the overhead of the PDP module, which consists of mutable attribute acquisitions, XACML policy interpretations and evaluations, and the updates of mutable attributes. Performance study with our implemented prototype system was conducted in a closed 100Base-TX network, which consists of a Linux server hosting an RP and an AR, and a Windows client machine as a user platform. The RP holds the UCON policies specifying the above location and task-based access control.

As the prototype system is for sharing application codes of collaborative software development, object attributes are defined based on the software package or module, e.g., the existence of a `LOCK` file under a module directory in RP. For simplicity we assume that a subject only participates in a single software module at the RP. Table 1 shows the PDP performance for updating (`import` command of Subversion) the code (files) of a software module. We ran the experiment with different average size of files (10KB, 100KB, and 1000KB) and different number of files (1, 10, 100) in a module, and measured the processing time of the PDP. The average time per access varies in the range 45.30 - 99.43 msec, which does not depend on either the number of files to transfer, or the file size, as the UM keeps the object attributes on a module (or directory) basis. The last column in Table 1 shows the total processing time of a single usage session on the client side, including uploading all files of the module. For example, updating a module with 10 files with average size 1000 KB takes about 12 seconds from the user platform to the server. The results show that the performance is acceptable for general collaboration requirements.

As aforementioned, the PDP's operation in a single usage process consists of several steps, including fetching subject and object attributes, object attributes updates, XACML policy interpretations, and communications to mod_authz_ucon (the PEP). Note that the policies evaluated in our prototype do not have the updates of sub-

[1] Note that the concept of obligation in XACML does not mean the same as that in UCON$_{ABC}$ model.

Access operations: update (# of files, avg. file size)	Total PDP processing (msec)	# of accesses	Avg. time per access (msec) (msec)	Session running time (sec)
1, 10 KB	2304	26	88.62	2.77
1, 100 KB	2307	26	88.73	2.68
1, 1000 KB	2473	26	95.12	3.28
10, 10 KB	3993	44	90.75	4.94
10, 100 KB	4375	44	99.43	5.47
10, 1000 KB	3506	44	79.68	11.96
100, 10 KB	12423	224	55.46	13.78
100, 100 KB	15958	224	71.24	19.97
100, 1000 KB	10147	224	45.30	85.33

Table 1: PDP performance overhead in file sharing prototype system

ject attributes [2]. In order to investigate possible mechanisms for better performance, we measured the processing time of each step in the PDP module. As shown in Figure 4, fetching subject attributes has the highest cost, which makes up 30-48% of the overall processing time. This results from the overhead of the SSL handshaking between the PDP module and the AR. In real applications, mechanisms such as *keep-alive* connections and attribute value cache on PDP side can be used to reduce this overhead and thus improve the overall performance of the system.

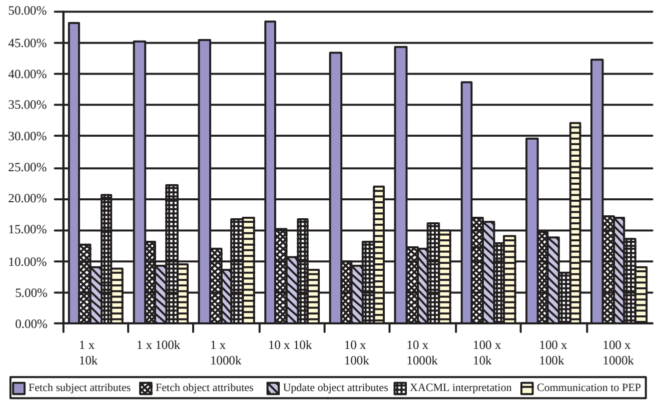

Figure 4: Micro benchmark of the PDP module. Each value shows the ratio of the stepwise running time to the entire PDP processing time.

6. RELATED WORK

Originally in some Grid systems, each RP uses a *grid-mapfile* to map external resource consumers to local identities and defines their permission. With dynamic property of user participation and resource sharing, this approach is not scalable.

The Community Authorization Service (CAS) [17] is a centralized approach, in which a CAS server maintains the access control policies and the PDP is deployed on the CAS side. Although this approach solves the scalability problem, it lacks flexibility for ad-hoc collaborations. For example, for a temporal group collaborating between some users with mobile devices, the authorization is based on the location of the platforms, e.g., only users in the same room can access the shared resources. Since there is no centralized point, CAS cannot solve this problem. Also, CAS lacks flexibility to support a new RP which has not established trust relationship with CAS, or an existing RP to change its policy regarding its shared resources.

[2]The change of the subject's location is not an update of the PDP, but a user's discretionary activity, and is not included in the UCON policy.

Instead of centralized authorization, the Virtual Organization Membership Service (VOMS) [7] describes an approach in which each RP has a set of local policies. To access shared resource, a user provides an attribute certificate issued from the VO to identify the role, group name, and capabilities of the user. By moving the PDP from centralized server to each RP's local site, VOMS can solve the scalability problem with gridmap file and the flexibility of CAS, but it cannot support collaborations without well-established infrastructure since it still requires a (globally) centralized attribute authority. Further, since an attribute in VOMS only includes role and group information in a VO, some policies cannot be implemented, such as user-level and VO-level delegation, and context-based authorization. That is, a user only can gain permissions from a VO administrator.

PRIMA [13] is a privilege management system which supports ad-hoc collaboration and permission delegation. To submit a request to an RP, a user provides a set of attributes, which define the privileges of the user, such as file access permissions, user quota, network access, etc. The RP assigns permissions to the user with these attributes, according to the local policies. A shortcoming with this approach is that, in a dynamic collaborative environment, the privileges of a user may change according to the resource consuming status in an RP, or some constraints with other concurrent jobs running in the RP. Therefore the pre-issued privilege attributes in PRIMA cannot support this dynamic and in-time permission assignments. A significant difference between PRIMA and our approach is that we use general attributes without any pre-assigned privileges, such such as context-aware attributes [19]. The permissions of a subject are granted just when the subject generates the requests and the corresponding attribute values are presented, either pushed by the requesting subject or pulled by the PDP. Also, our approach supports dynamic properties of collaborations, such as continuous control and attribute mutability during an access.

Akenti [21] is a distributed policy management system, where a set of stakeholders define conditions for a resource usage. An RP makes authorization decisions based on all these conditions in attribute certificate format. Condition certificates are pulled by the PDP, which is similar the mutable attribute acquisition in our approach model, while the mutability of conditions are not supported in Akenti. Also, since it is extensively dependent on public key infrastructure (PKI), Akenti cannot support ad-hoc collaborations without pre-established infrastructure.

Context-aware authorizations have been studied by several researchers. In [8] the security-relevant context of access request environments are captured by environment roles in RBAC. A context-aware access control model based on RBAC is presented in [24] for pervasive Grid applications, where a context agent collects environmental information and dynamically enforces user-role assignments and permission-role assignments. In [22], access control models are reviewed and compared in the context of collaborative systems, and a set of assessment criteria is proposed to consider access control in collaborations. Very recently, a framework for secure collaboration between domains is proposed in [20], where each domain uses RBAC and policies are locally enforced by individual domains in a mediator-free manner.

7. CONCLUSION AND FUTURE WORK

An authorization framework is proposed in this paper for collaborative computing systems following the OM-AM approach. To meet scalable, dynamic, and fine-grained authorization requirements, in the model layer, the recently developed UCON model is used to support various authorization policies for collaborations. Our proposed architecture can support attribute mutability and decision continuity by leveraging a hybrid approach of attribute acquisitions and event-based updates. An implemented prototype for group-based collaborative software development demonstrates the feasibility of our framework, and the performance study shows that our framework can be used for general collaborations.

The access control model in this paper only includes authorizations and conditions of UCON, and we are going to capture the obligation aspect in the future work. As the obligations of a usage session are actions that have to be performed by the requesting subject or some other subjects in the system, the architecture layer will need mechanisms to monitor and propagate obligation satisfactions. Also, XACML will be extended to support UCON obligations in policy specifications.

8. REFERENCES

[1] DB4Object, http://www.db4o.com/.

[2] mod_dav: a DAV module for Apache, http://www.webdav.org/mod_dav/.

[3] OpenLDAP, http://www.openldap.org/.

[4] OpenSSL, http://www.openssl.org/.

[5] Subversion, http://subversion.tigris.org/.

[6] Sun's XACML implementation, http://sunxacml.sourceforge.net/.

[7] R. Alfieri, R. Cecchinib, V. Ciaschinic, L. dell'Agnellod, A. Frohnere, K. Lorenteyf, and F. Spatarog. From gridmap-file to voms: Managing authorization in a grid environment. *Future Generation Computer Systems 21*, 2005.

[8] M. J. Covington, W. Long, S. Srinivasan, A. K. Dey, M. Ahamad, and G. D. Abowd. Securing context-aware applications using environment roles. In *Proceedings of the 9th ACM Symposium on Access Control Models and Technologies*, pages 10–20, Chantilly, Virginia, USA, May 3-4 2001.

[9] E. Bertino et al. Panel: Security for grid-based computing systems issues and challenges. In *Proceedings of the Symposium on Access Control Models and Technologies*, pages 125–125, Yorktown Heights, New York, USA, June 2-4 2004.

[10] I. Foster, C. Kessekan, G. Tsudik, and S. Tueckel. A security architecture for computational grids. In *Proceedings of ACM Conference on Computer and Communications Security*, SanFrancisco, California, USA, pages =.

[11] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organization. *International Journal of Supercomputing Applications*, 15(3), 2001.

[12] W. E. Johnston. The computing and data grid approach: Infrastructure for distributed science applications. *Computing the Informatics, Special Issue on Grid Computing*, 2002.

[13] M. Lorch, D. B. Adams, D. Kafura, M. S. R. Koneni, A. Rathi, and S. Shah. The prima system for privilege management, authorization and enforcement in grid environments. In *Proceedings of the 4th International Workshop on Grid Computing*, pages 109–116, Nov. 17 2003.

[14] OASIS XACML TC. *Core Specification: eXtensible Access Control Markup Language (XACML)*, 2005.

[15] J. Park and R. Sandhu. The $\text{UCON}_{\texttt{abc}}$ usage control model. *ACM Transactions on Information and Systems Security*, 7(1):128–174, February 2004.

[16] J. Park, X. Zhang, and R. Sandhu. Attribute mutability in usage control. In *Proceedings of the Annual IFIP WG 11.3 Working Conference on Data and Applications Security*, pages 15–29, Sitges, Catalonia, Spain, July 25-28 2004.

[17] L. Pearlman, V. Welch, I. Foster, and K. Kesselman. A community authorization service for group collaboration. In *Proceedings of IEEE Workshop on Policies for Distributed Systems and Networks*, pages 50–59, Monteray, California, USA, June 5-7 2002.

[18] R. Sandhu. Engineering authority and trust in cyberspace: The OM-AM and RBAC way. In *Proceedings of Fifth ACM Workshop on Role-based Access Control*, pages 111–119, Berlin, Germany, 2000.

[19] Manoj R. Sastry and Michael J. Covington. Attribute-based authentication using trusted platforms. In *Proceedings of Wireless Personal Multimedia Communications*, Aalborg, Denmark, September 18-22 2005.

[20] M. Shehab, E. Bertino, and A. Ghafoor. Secure collaboration in mediator-free environments. In *Proceedings of the 12th ACM Conference on Computer and Communication Security*, pages 58–67, Alexandria, Virginia, USA, 2005.

[21] M. Thompson, A. Essiari, and S. Mudumbai. Certificate-based authorization policy in a pki environment. *ACM Transactions on Information and System Security*, 6(4):566–588, 2003.

[22] W. Tolone, G. Ahn, and T. Pai. Access control in collaborative systems. *ACM Computing Surveys*, 37(1):29–41, March 2005.

[23] V. Welch et al. Security for grid services. In *Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing*, pages 48 – 57, June 22-24 2003.

[24] G. Zhang and M. Parashar. Dynamic context-aware access control for grid applications. In *Proceedings of the 4th International Workshop on Grid Computing*, pages 101–108, Nov. 17 2003.

[25] X. Zhang, F. Parisi-Presicce, R. Sandhu, and J. Park. Formal model and policy specification of usage control. *ACM Transactions on Information and Systems Security*, 8(4):351–387, November 2005.