Toward a Usage-Based Security Framework for Collaborative Computing Systems

XINWEN ZHANG
Samsung Information Systems America
MASAYUKI NAKAE
NEC Corporation
MICHAEL J. COVINGTON
Intel Corporation
and
RAVI SANDHU
University of Texas at San Antonio and TriCipher Inc.

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 article we propose a usage control (UCON) based security framework for collaborative applications, by following a layered approach with policy, enforcement, and implementation models, called the PEI framework. In the policy model layer, UCON policies are specified with predicates on subject and object attributes, along with system attributes as conditional constraints and user actions as obligations. 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. In the enforcement model layer, our novel framework uses a hybrid approach for subject attribute acquisition with both push and

This research was partially supported by the National Science Foundation and Intel Corporation.
An earlier version of this paper appeared under the title "A Usage-based Authorization Framework for Collaborative Computing Systems" in the *Proceedings of 11th ACM Symposium on Access Control Models and Technologies*. Lake Tahoe, California, USA, 2006.
Author's address: X. Zhang, Samsung Information Systems America, 75 West Plumeria Drive, San Jose, California 95134, USA; email: xinwen.z@samsung.com. M. Nakae, 1751, Shimonumabe, Nakahara-Ku, Kawasaki, Kanagawa 211-8666, Japan; email: m-nakae@bp.jp.nec.com. M. J. Covington, 2111 NE 25th Avenue, JF3-206, Hillsboro, Oregon 97124, USA; email: Michael.J.Covington@intel.com. R. Sandhu, Institute for Cyber-Security Research, Univ. of Texas at San Antonio, One UTSA Circle, San Antonio, TX 78249, USA; email: ravi.sandhu@utsa.edu.
The work of X. Zhang and M. Nakae was done while at George Mason University, Fairfax, VA, USA.

pull modes. By leveraging attribute propagations between a centralized attribute repository and distributed policy decision points, our architecture supports decision continuity and attribute mutability of the UCON policy model, as well as obligation evaluations during policy enforcement. 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

Additional Key Words and Phrases: Authorization, access control, usage control, UCON, collaborative computing, security architecture

ACM Reference Format:
Zhang, X., Nakae, M., Covington, M. J., and Sandhu, R. 2008. Toward usage-based security framework for collaborative computing systems. ACM Trans. Inform. Syst. Secur. 11, 1, Article 3 (February 2008), 36 pages. DOI = 10.1145.1330295.1330298. http://doi.acm.org/10.1145.1330295.1330298.

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 [Johnston 2002] 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 [Foster et al. 2001], 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 system-level resources that are managed by the VO; RPs are responsible for respecting predefined 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 among all of these entities to protect the resources in a VO and ensure their availability through access control mechanisms [Bertino et al. 2004].

Current authorization solutions for collaborative systems focus on centralized policy management with *privilege credentials*. In many Grid systems, for example, administrators of a VO issue credentials to users that determine

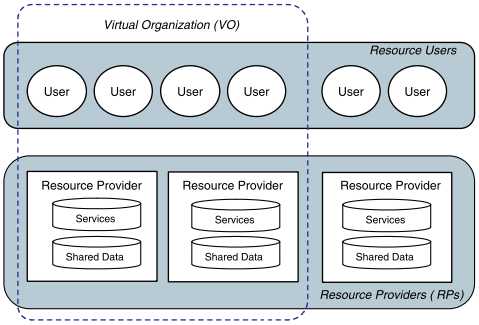


Fig. 1. An overview of virtual organization.

the resources and permissions a user can hold. That is, the permissions of a user are preassigned, 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. For example, a user's available usage quota on a particular resource could change dynamically according to his or her status; preassigned 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 objects). For example, a scientific instrument can be shared within a research VO, but critical functions may be used by only 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 do not 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 preexisting VO management infrastructure is in place, subject authentication is not available, and authorization decisions are, instead, dependent on contextual information [Covington et al. 2001], such as the location and time of the access request. Existing approaches lack flexibility to support context-based authorizations in collaborative systems.

In this article we propose a generalized security framework for collaborative systems by following the layered policy-enforcement-implementation (PEI) framework [Sandhu et al. 2006]. In the policy model layer, in accordance with dynamic authorization requirements in collaborative systems, we present an access control model based on the UCON model [Park and Sandhu 2004; Zhang et al. 2005]. By leveraging the policy specification flexibility and attribute mutability of UCON, our model supports not only VO-level security policies but

also usage constraints defined by each RP. In the enforcement layer, 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 identify several implementation issues and build a prototype system to show the feasibility and performance of our framework. Effectiveness of usage control is demonstrated with access control policies based on subject and object attributes, which are specified with the extensible access control markup language (XACML) [OASIS XACML]. The performance of the system is also studied.

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 with this model. Section 4 describes the proposed architecture. Section 5 presents a prototype that we have implemented according to our framework and details some experimental results. Section 6 presents some related work in Grid-like and other general collaborative computing systems. Finally, section 7 summarizes this paper and discusses ongoing and future work.

2. SYSTEM FRAMEWORK OVERVIEW

We develop our authorization framework by following the PEI methodology [Sandhu et al. 2006]. Although PEI was originally applied for the controlled information sharing problem, it provides principles for general security system design and development. This section briefly introduces the concept of the PEI framework and discusses the problems encountered at each PEI layer within the context of collaborative computing systems.[1]

2.1 PEI Framework

Traditionally, security systems distinguish policy from mechanism. The general goal for a security system has been to build flexible and robust mechanisms that can conveniently support a wide range of policies. Policy is concerned with "what" security needs to be enforced while mechanism is concerned with "how" the security is being enforced. Modern systems are distributed and have multiple trust and service dependencies. Trying to close the policy-mechanism gap in a single step is hardly viable in such a complex environment. Also, early literature treats the policy layer as being rigorous and well defined, whereas in reality the highest level of policy is often informal and fuzzy. One of the most difficult steps is to take informal policy requirements and provide sufficient rigor, structure, and detail in the next level of policy requirements so that they can be effectively handed off to security engineers to enforce.

The PEI framework seeks to bridge the gap between the what (policies) and the how (enforcement and implementation) by introducing three additional

[1]Note that PEI is an evolution of the earlier OM-AM framework originally applied to role-based access control (RBAC) [Sandhu 2000]. The model layer of OM-AM maps to the policy model layer of PEI, whereas the architecture layer of OM-AM is split into the enforcement model and implementation model layers of PEI.

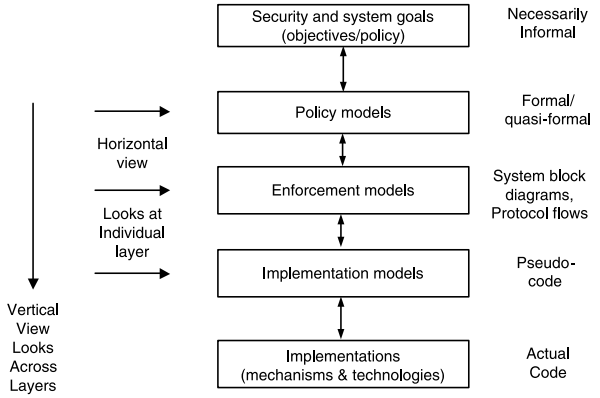


Fig. 2. The PEI models framework.

layers of models. As indicated in Figure 2, at the top there is a layer of informal security and system policy requirements. This layer is necessarily informal and is the layer at which business issues dominate. The bottom layer is actual implementation. Between the top and bottom layers there are three layers of models: the policy model, the enforcement model, and the implementation model. These layers focus on different aspects of a security system. The purpose of a policy model is to take informal high-level objectives and flesh out rigor and detail using a formal or quasi-formal notation. A well-known example of a successful policy model is the lattice model for mandatory access control [Bell and LaPadula 1975; Denning 1976; Sandhu 1993], which captured the informal concept of enforcing information flow using security labels that had been in use in military and national security arenas in the paper world. The enforcement and implementation models address the "how" aspect of enforcing a desired policy. The enforcement models address the big picture of the "how" question, at the level of system architectures, block diagrams, and protocol flows. The protocol flows can be formalized and analyzed to establish various security properties of the system. At the same time, they leave a certain level of detail unspecified, which is then elaborated in the implementation models. The implementation models are focused on specific issues identified in the enforcement models layer. These issues need to be resolved in sufficient detail to require description at a pseudocode level of detail and precision.

PEI is absolutely not intended to be a top-down waterfall-style software engineering methodology. Each layer focuses on a particular system design aspect, and inevitably all these aspects require compromises, decisions about trade-offs, and adjustments as the system gets built and deployed. The relationship between adjacent layers is many-to-many. A single policy model can be enforced by multiple enforcement models with possibly different trade-offs between security, trust, performance, cost, convenience, and the like. Likewise, a single enforcement model may support multiple policy models. There is similarly a many-to-many relationship between enforcement models and implementation models.

In the rest of this article we follow this framework to investigate the authorization management in collaborative computing systems, beginning with an overview of each layer in the following subsection.

2.2 Overview of PEI Models in Collaborative Computing Systems

The basic requirement of an authorization component in collaborative systems is to control access by users to shared resources. Furthermore, with the heterogeneous computing environments in these systems, the control should be scalable, dynamic, and fine grained. Previous work has 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 attributes 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. Furthermore, in many collaborative computing systems, a user's permission may not be determined only by his or her attributes and system conditions but also by other activities. For example, a user's write permission to a shared document may require the approval of a high-level manager in the organization.

Based on these requirements, usage control (UCON) [Park and Sandhu 2004; Zhang et al. 2005] is used in the policy model layer of our framework, as it is an attribute-based access control model and comprehensively considers authorizations, obligations, and conditions in access control decisions. In UCON, authorizations are predicates defined on subject and object attributes. Obligations are actions that have to be performed by the requesting subject or by other subjects before or during an access. Conditions are environmental restrictions represented by system attributes, such as time, location, load, and the like. UCON uses the real-time values of subject and object attributes for authorization decisions in a *session-based* manner. Usage decisions are enforced not only when a subject generates an access request but also during the ongoing stage of the *usage session,* which is referred to as *decision continuity*. As side effects of the usage, subject and object attributes can be updated. This is referred to as *attribute mutability*. Previous work has shown that decision continuity and attribute mutability can provide flexible, fine-grained, and dynamic access control [Park 2003; Zhang et al. 2005].

From the viewpoint of enforcement 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 [Lorch et al. 2003; OASIS XACML]. Figure 3 indicates the conceptual data flow of these two modes. In push mode, each subject presents his or her related information (e.g., identity and attribute certificates[2]) to the PDP, and the decision is sent to PEP; while in pull mode, a PEP collects the related information of a subject and queries the PDP for a policy decision. Considering the

[2]Here we refer to a certificate as a digitally signed statement of the value of a subject identity or a subject attribute by a trusted authority.

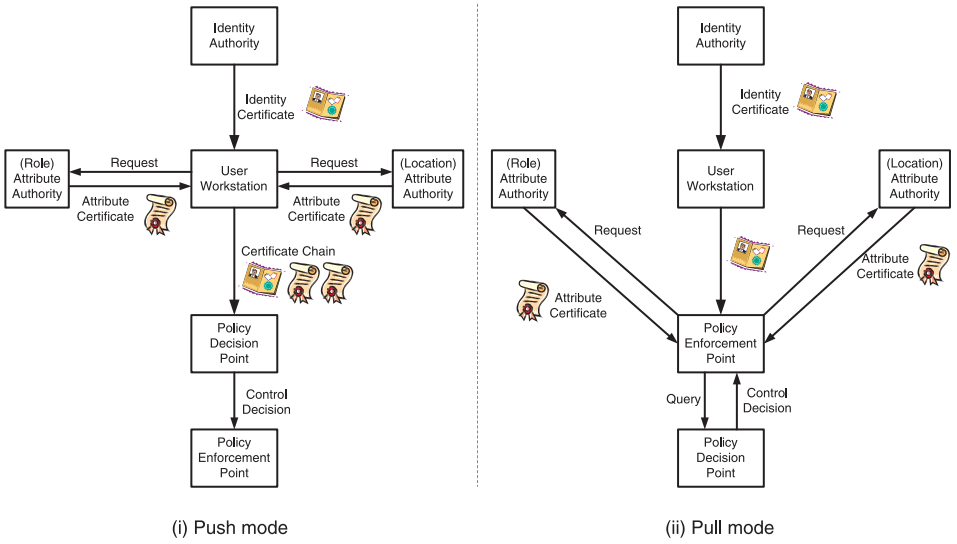


Fig. 3. Push and pull modes of authorization architecture.

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 result in attribute updates, which may affect other ongoing accesses. In a pure push-based architecture, this requires the subject to continuously obtain the latest attribute value and report it to the PDP. In a pure pull-based architecture, on the other hand, the PEP needs to keep querying the PDP with up-to-date attribute values, so extra communication overhead between them is introduced.

In the enforcement model of our framework, both PDP and PEP are located on the resource provider 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* (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 triggers the

reevaluation of the policy by the PDP according to the ongoing usage session and may result in revocation of the ongoing usage or update of attributes if necessary. This approach seamlessly supports decision continuity and attribute mutability of UCON within concurrent usage sessions.

In the implementation model layer, implementation techniques in several aspects are considered in our framework, such as policy specifications, subject and attribute authentications, trusted update of attributes, evaluation of obligations, and secure communications between computing components in the system. The mechanisms of a security system need not only to achieve security objectives but also to 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 describes the details of the techniques that are used in our prototype and provides performance results.

3. USAGE CONTROL MODEL FOR COLLABORATIVE SYSTEMS

In a collaborative system, the resource provider (RP) typically maintains 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 temporary collaborative application, an object shared by its owner can be accessed only 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 policy 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

As depicted in Figure 4, a usage control system has six components: subjects and their attributes, objects and their attributes, rights, authorizations, obligations, and conditions. The authorizations, obligations, and conditions are components of usage control decisions. Authorizations are predicates based on the subject and/or the object attributes; obligations are activities that have to be performed by subjects before or during an access; and conditions are system environment restrictions that are imposed either before or during an access.

The most important properties that distinguish UCON from traditional access control models and trust management are the continuity of usage decisions and the mutability of attributes. Continuity means that control decisions can be determined and enforced not only before an access but also during the period of the access. Figure 5 shows a complete usage process consisting of

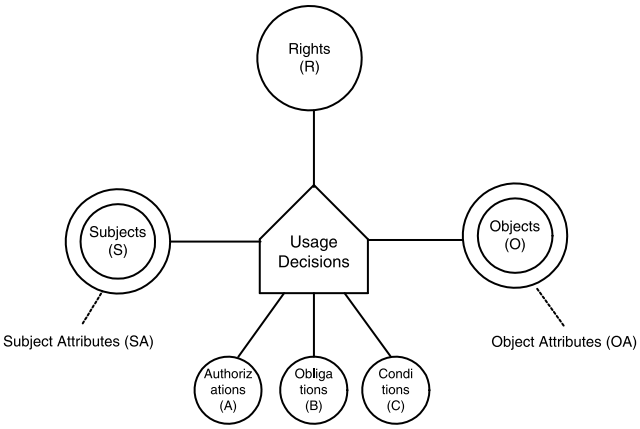


Fig. 4. Usage control model.

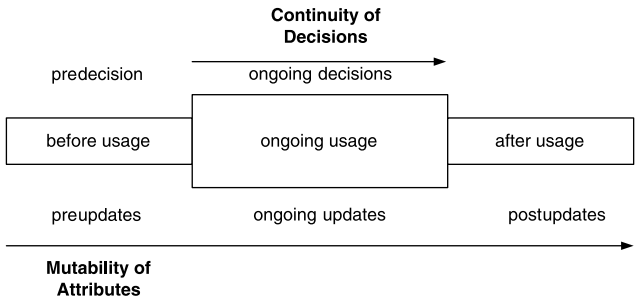


Fig. 5. Continuity and mutability properties of UCON.

three phases along the time line: before usage, ongoing usage, and after usage. The control decision components can be checked and enforced in the first two phases, named predecisions and ongoing decisions, respectively. UCON is a session-based access control model because it controls the current access request and ongoing usage. Therefore, obligations and conditions in the after-usage phase are not considered in UCON policies. The obligations and conditions after an access are regarded as long term obligations and conditions, which are not included in the core UCON models but should be included in related administrative models. In this article we focus only on the core aspects of UCON, while administrative models will be developed in the future.

Mutability means that subject and/or object attributes can be updated as the result of an access. Along with the three phases, there are three kinds of updates: preupdates, ongoing updates, and postupdates. All these updates are performed and monitored by the system. An update of a subject or an object attribute in an access may result in a system action to allow or revoke current access or another access, according to the authorizations of the access. An update on the current usage may generate cascading updates, while an update on another access can act as an external event that would cause a change of the

usage status, such as revocation. These are unique features of UCON models because of attribute mutability and decision continuity.

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. Strictly speaking, a subject is associated with a user and generates access requests on behalf of the user. For simplicity, we adopt the common, but technically inaccurate, terminology of saying that a subject is a user. Administrators in the VO and RPs are also subjects who 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 properties such as role, 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 assigned to a subject for some resources in a VO, membership in conflict-of-interest groups, and so on. For example, when an application has a shared resource that can be used only by subjects from nonconflicting groups, a subject attribute specifying its conflict groups should be defined. Object attributes can include general object properties such as type, ownership, and the like. In addition, application-specific attributes can be defined by system administrators or designers and include properties such as usage status, inclusive/exclusive accesses, and the like. Previous work has shown the expressive power of UCON to specify various access control models and policies with different attributes [Park and Sandhu 2004; Zhang et al. 2005].

As discussed in Park and colleagues [2004], attributes that are updated according to UCON policies are mutable or system controlled, while attributes managed by an 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, for example, because of the change of the user's job functionality.

Because UCON conditions impose 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 particular policy, may affect the result of a subsequent or an ongoing access.

Obligations can be used to specify prerequisite actions required before a resource can be used in collaborative systems. These can be simple actions,

such as clicking an agreement button before gaining access to an object, or a general access action such as that in a task-based security system [Thomas and Sandhu 1997]. Also, an obligation can be an action performed by subjects other than the requesting subject. For instance, in a collaborating research group between institutes, a user's access to a shared facility may require approval from his or her advisor or the principal investigator (e.g., represented as a digital signature for the access request with a short valid period).

3.3 UCON Policies and Scheme

A permission in UCON is a tuple (s, o, r), where s is the requesting or accessing subject, o is the target object, and r is the access right. A UCON policy specifies the authorizations, obligations, and conditions requirements before and during a usage process.

Definition 3.1. A UCON policy maps a permission of (s, o, r) to a tuple (P_{pre}, P_{on}, OB_{pre}, OB_{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; OB_{pre} and OB_{on} are sets of obligations that have to be satisfied before or during the 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 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 regarded as authorization predicates and condition predicates, respectively. A predicate takes one or more attribute values and constants and returns boolean values. P_{pre} and OB_{pre} are called predecision components, and P_{on} and OB_{on} are ongoing decision components. If any predicate in P_{pre} or action in OB_{pre} is not satisfied when the access request is generated, a usage cannot be granted; after granting, if any predicate in P_{on} or action in OB_{on} is not satisfied during a usage process, the ongoing access is revoked by the system. Note that an obligation action in OB_{on} can be a one time action, a continuous action, a periodic action, or an action under particular conditions [Zhang et al. 2005]. Without loss of generality, we assume that, in a general UCON policy, an obligation in OB_{pre} is a one-time action while an obligation in OB_{on} is a conditional action that is required when some attribute predicates are satisfied during an ongoing usage process, because continuous and periodical obligations can be regarded as special cases of conditional actions.

UP_{pre}, UP_{on}, and UP_{post} are pre-, ongoing, and postupdate actions on subject and/or object attributes, respectively. 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. Similar to an ongoing obligation action, an ongoing update action can be one time, continuous, periodical, or under specific conditions. For example, an update of total 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

postupdate actions in a policy, one for updates after a subject ends an ongoing access (with *endaccess* action) and the other for updates after an ongoing access is revoked by the system (with *revokeaccess* action), for example, because of a subject/object attribute change or a system condition change that makes 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 Zhang and colleagues [2005].

In a high-level view, a UCON policy evaluates a set of predicates and obligation actions initially and grants and continuously allows a permission if all of them are satisfied. All predicates are based on attributes of the requesting subject or the target object or the system, while obligations may be required from this subject or others. As the side effect of granting access, some attributes may be updated. In UCON, a policy specifies the attribute updates only of the accessing subject and target object but not system attributes (which are updated automatically by the system).

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

Definition 3.2. A UCON *scheme* is a 6-tuple (ATT_a, ATT_c, R, P, OB, 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, OB is a fixed set of obligation actions, 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 article supports condition checks during an ongoing access, system attributes are considered to be mutable in our framework.

Note that all policies in a UCON scheme are defined for positive permissions (e.g., to enable access). 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

The concepts of decision continuity and attribute mutability are based on a continuous 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. In general, for a particular permission, it is possible that there is more than one policy that can be applied for an access request. For example, a school facility can be accessed only by faculty or any member from a lab. Two policies can be defined for these two cases, which require different credentials.[3] In a real access request, a user can satisfy both policies: He or she is faculty and works in the lab. Only one policy is applied during a single usage session, and corresponding updates are performed based on this "selected policy." UCON does not cover the mechanism to select which policy is used. That is, a usage session refers to a specific usage process with (s, o, r) that

[3]Note that according to Definition 3.1, only conjunctions of predicates are used in a UCON policy.

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 regard 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. For system attributes, the details of how they change is not specified in UCON policies. However, 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 [Park and Sandhu 2004; Park et al. 2004; Zhang et al. 2005], such as role-based access control (RBAC), dynamic separation of duty, Chinese Wall policy, and policies with low or high watermark properties. For collaborative computing systems in particular, we summarize different types of policies that can be specified with different attribute predicates and obligation actions in UCON as follows:

—*Consumable resource management.* Subject attributes can be defined to specify dynamic resource management policies. For example, consider a policy that a subject can have only a fixed amount of total storage in a VO. An attribute can be used to record its currently used or available amount. Whenever a subject generates an access request to an RP, the RP uses this attribute value to determine whether 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 by an RP in concurrent usage sessions (in different RPs) should invoke rechecks by other RPs. A similar mechanism can be used to control the usage of other resources such as CPU cycles, network bandwidth, and the like.

—*Credit or reputation management.* A subject's access may generate credits or reputation points, which in turn can enable other permissions in a VO. This can be used in many incentive-based content sharing systems such as BitTorrent [Cohen 2003] peer-to-peer (P2P) downloading, where a peer's downloading priority and speed are determined by its uploading speed to other peers. Similar to above, subject attributes can be defined to capture these aspects, such as a peer's overall contribution in the system.

—*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 the object, in order to preserve its integrity. Also, a subject's particular operation on a collaborative task may require that some necessary preoperations have been performed by other subjects. Attributes can be defined to monitor the shared object or task's status and used to determine the just-in-time permissions of a subject.

—*Prerequisite or concurrent actions for ongoing accesses.* In many collaborative systems, a user's access requires prerequisite actions. For instance, in task-based systems a user needs to finish a task before getting the permission to do another task. Similar issues exist in workflow systems. Also, an ongoing access may require a user or his or her collaborators to perform some other actions concurrently. For example, an ongoing usage of a sensitive object requires that another subject (e.g., system administrator) log all the operations during the usage. UCON preobligations and ongoing obligations can be used to specify these requirements.

—*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. For example, in a collaborative task, an operation with high privileges requires at least two subjects' involvement at the same time. Also, for the purpose of avoiding conflict-of-interest scenarios, some operations may not be performed when two subjects are present.

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

All of these policies can be defined with the conceptual or formal model proposed in previous work [Park and Sandhu 2004; Zhang et al. 2005]. Detailed policy specifications are outside the scope of this paper.

4. ENFORCEMENT ARCHITECTURE

This section introduces the overall architecture of our framework and describes the features to support attribute mutability and decision continuity; we also explain how to support obligations within the enforcement architecture. Because it serves as the enforcement model in our PEI framework for collaborative computing, the goal of our architecture is to support as many features of a general UCON model as possible. We use the terms *enforcement architecture* and *enforcement model* interchangeably and often simply refer to it as *architecture*.

4.1 Overview

Figure 6 gives an overview of the enforcement architecture in the context of Grid Security Infrastructure (GSI) [Foster et al. 1998]. Typically, the archi-

tecture 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 [Welch et al. 2003] (step 1).

—Persistent subject attributes are pushed by the requesting subject to the policy decision point (PDP) on the RP side (step 2).

—After receiving the request, the PDP contacts the AR and retrieves the mutable attributes of the requesting subject (steps 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, object, and system attributes) and evaluation of policies. The decision is forwarded to the PEP and enforced in the execution environment of the task (step 6).

As the side effect of making a usage decision, attribute updates are preformed by the PDP according to corresponding policy. New subject attribute values are sent back to the 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 the latest attribute values when a new access request is generated. The process for obtaining the latest attribute values during ongoing access is described in Section 4.3.

In a collaborative system, a subject can initiate multiple usage sessions to different RPs or to a single RP with different objects. Also, multiple subjects can gain access to a shared resource. By centralizing 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.

Within a single VO, although there is only one AR depicted in Figure 6, physically there can be multiple ARs for different subjects or different types of attributes. For an access request, a PDP needs to acquire all necessary attributes from different ARs. For simplicity, we describe only interactions with single AR in one VO in this article.

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.

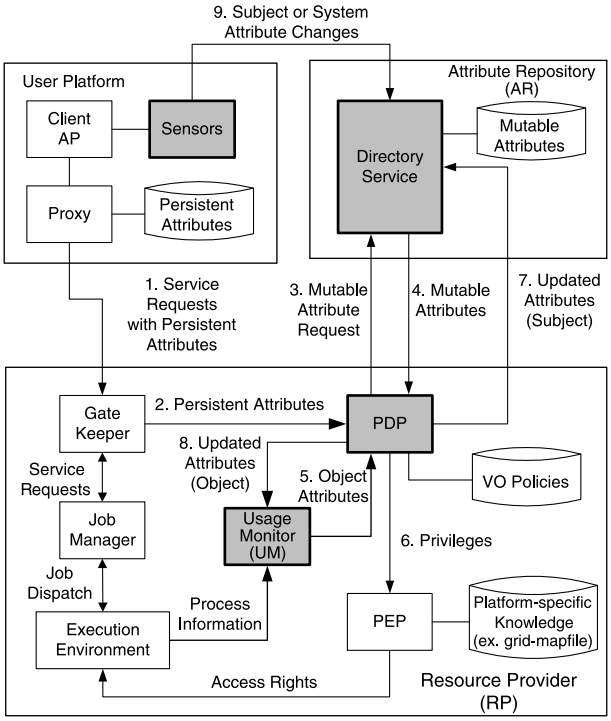


Fig. 6. Usage-based authorization architecture.

One of the novelties of our architecture is that it supports different modes for attribute acquisition and management with regard to different types of attributes:

—*Push mode for persistent attributes.* Because there are no policies in a UCON scheme that update their values, the persistent attributes of a subject are pushed by the subject to the PDP of the RP to which the subject submits an access request. As these attribute values do not change during the usage session, they are pushed and evaluated only 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 persistent attributes, such as a subject's role or group in a VO, because they are not updated as a result of usage. An object also can have persistent attributes that 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 has no current access to any

object.[4] In this paper we focus on the core aspects of UCON; 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. Values for mutable object attributes 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. Because mutable subject attributes are maintained in a centralized AR, updates (e.g., an update from another usage session) trigger the new attribute value to be 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 Supporting Attribute Mutability and Decision Continuity

4.3.1 *Updates of Attributes.* As described in Section 4.1, in a usage session, preupdates are results of the approval of the access request performed by the PDP, according to a particular usage policy. That is, preupdates are triggered by an access request based on a policy. During a usage session, ongoing and postupdates are triggered by some specific events, either from the subject or from the system.

—*Ongoing updates.* Ongoing attribute updates are invoked by events in a system, such as time events and events that change system status. For example, the update for a subject's time slice, which is repeatedly adjusted in a usage session, 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 rechecked by the PDP if necessary (e.g., to allow an ongoing usage to continue or to revoke it). For simplicity, the event trigger and transmission are not called out in Figure 6.

—*Postupdates.* Postupdates 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 [Zhang et al. 2005]. For the first type, the UM reports 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. In the second case, the revocation of an ongoing access is the result of decision continuity enforced by the PDP; the postupdates are performed by the PDP and reported to the AR and UM, respectively.

[4]This may lead to a denial of administrative update if the subject has indefinite access to an object. Mechanisms for "update on the fly" are needed for these attributes. For simplicity, we do not explore the details of this mechanism in this article.

—*System attribute changes.* Besides the PDP-performed attribute updates for subjects and objects, system attributes are changed by external events. For example, to account for 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 6). 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. Because a decision component is built from attribute predicates, in practice ongoing checks 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, the 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 6), the AR logs this request with related attribute names. On 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 reevaluate the policy of the ongoing usage session with the new attribute value. If the predicates of the policy remains 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. Whenever the PDP has no interest in the subject attributes, e.g., when the access is ended or revoked, it informs the AR such that no updated attribute values are sent from AR to RP_1. For the change of an object attribute, because it is monitored by the UM the ongoing decision check can be locally implemented.

4.4 Supporting Obligations

As discussed in Section 3, there are two types of obligations in a general UCON model, preobligations and ongoing obligations. For simplicity, in this section we consider only how preobligations can be supported in our architecture, although we conjecture that ongoing obligations can be supported similarly.

To support obligations, we need a mechanism to evaluate whether an obligation action has been performed before the PDP allows an access. In centralized policy enforcement systems, monitoring technologies such as logging can be used for this purpose. But it is not scalable for decentralized systems such as distributed collaborations. In this article we propose an attribute-based obligation evaluation mechanism by leveraging the attribute propagation in our architecture. Specifically, for each obligation action we define a subject attribute that specifies whether the obligation has been performed by the subject

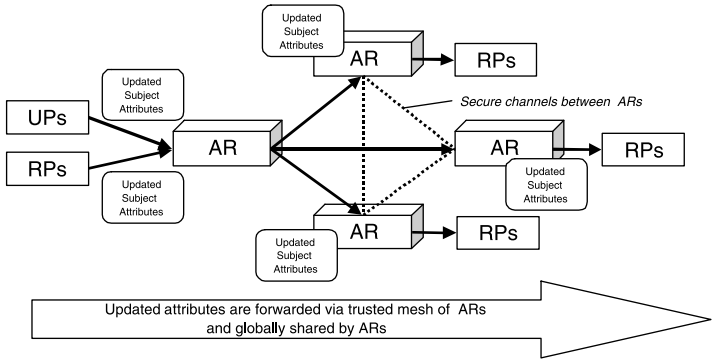


Fig. 7. Attribute sharing over trusted mesh.

or not. This *obligation subject* may be the same as the access requesting subject or different, depending on the corresponding policy. When the obligation subject has successfully performed the action, the attribute value is updated in AR. On receiving an access request from a subject, the PDP pulls the attribute corresponding to the obligation action from AR, similar to retrieving the mutable subject attributes aforementioned. If the attribute value indicates that the obligation has been satisfied, the access can be allowed. We explain the details of this process in Section 5 with a preobligation policy example.

4.5 VO Federations

So far we have introduced the enforcement architecture and control flow for a single VO, in which only a single centralized AR is used. In a realistic usage setting, however, there are many cases where users and resource providers in different VOs collaborate, requiring the PDP of a VO to obtain attributes from another VO to enforce policies, typically including the attributes of an access requesting subject and the attributes indicating whether obligations are satisfied by that subject.

To enable such *VO federations* with scalable attribute sharing between ARs, a "trusted mesh" is defined as a graph M = (V, E), where the vertics V correspond to a set of ARs in a VO federation and the edges E to a set of secure communication channels between the ARs. As shown in Figure 7, each AR has the reference information (referral) of every connected AR. Whenever a PDP of a VO (say, VO2) needs an attribute value of a subject from another VO (say, VO1), ARs and RPs cooperatively perform the operation with M. Specifically, the PDP of a local RP in VO2 requests an attribute value from the AR of VO2, which is actually stored on the AR of VO1. The AR of VO2 looks up the requested value with the internal directory service and obtains a referral to the remote AR of VO1. A secure channel is built between two ARs, and the AR of VO2 forwards the attribute request through the secure channel. Finally, the AR of VO1 queries the attribute value and sends back to the AR of VO2.

Similar to DNS services, the scheme for VO federations is scalable and can ensure the freshness of shared attribute values. In addition, this approach

allows the ARs to be completely symmetric in the mesh; this symmetry provides a new level of robustness that accommodates temporary federation, in which an AR leaves the VO federation after a limited time served.

4.6 Other Related Issues

4.6.1 *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. Because in general an identity authority is different from attribute authorities, a mechanism is needed to bind identity certificates and attribute certificates. Existing mechanisms proposed in Park and Sandhu [2000] can be used.

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 they are managed inside an RP, the authenticity and integrity of mutable object attributes can be easily achieved, for example with existing trust infrastructure in the same organization.

4.6.2 *Trust of System Attributes.* Policy decisions in our framework rely on the trusted values of system attributes, which specify condition components in a UCON policy. For example, the location-based access control needs input from sensors deployed on user platforms (refer to step 9 in Figure 6). As this paper focuses on architecture and enforcement work flow, the detailed requirements of the client platform and other system components for condition check are not explored. Trusted Computing technologies enabled by Trusted Platform Module (TPM) [TCG TPM 2003] specified by Trusted Computing Group (TCG) provide mechanisms for this purpose and, generally, can support trusted attributes and client integrity attestation [Covington et al. 2006; Sailer et al. 2004]. With widely deployed TPM on laptops and emerging Trusted Computing technology on mobile devices (e.g., TCG's Mobile Trusted Module [TCG MTM 2006]), we believe that this requirement can be satisfied.

4.6.3 *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. A 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 have implemented a Web-based prototype system, which enables a group of software

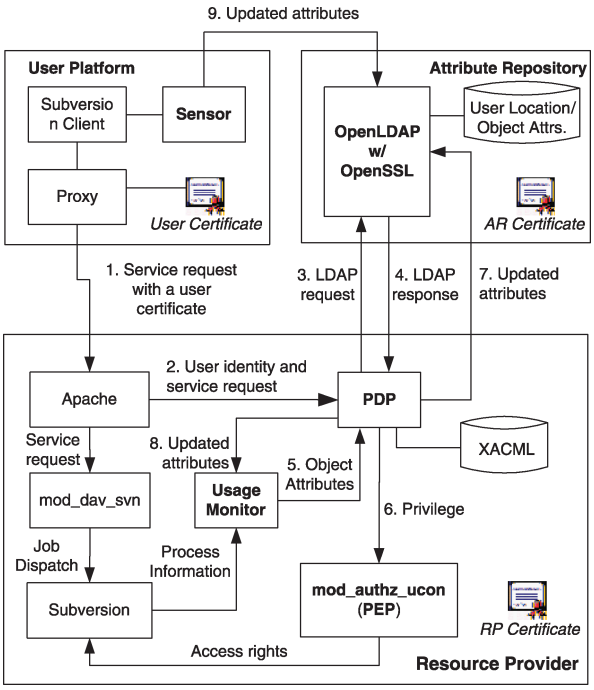


Fig. 8. Prototype system architecture.

developers to share and collaboratively develop application code from different user locations. This section first introduces the overview of the prototype architecture, then demonstrates three 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 [Subversion], which is integrated with Apache WebDAV module (*mod_dav*) [mod_dav]. As shown in Figure 8, the prototype architecture is similar to the general architecture proposed in the previous section. Specifically, the AR is a directory service built with OpenLDAP 2.0.27 [OpenLDAP]. OpenSSL 0.9.6b [OpenSSL] 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 a user is able to download and/or upload to a resource provider. Because 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 that 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 [DB4Object] 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 that 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) [OASIS XACML] to specify UCON policies. XACML is an open-standard format to specify access control policies and is expected to be widely used with the properties of interoperability and extensibility. Using Sun's XACML library [XACML], 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. Note that the concept of obligation in XACML does not mean the same as that in UCON model. Later in this section we show how to extend XACML to specify obligations in a UCON policy.

As an implementation-oriented policy specification, an XACML policy does not exactly map to an abstract UCON policy defined in Section 3.3. Specifically, the ongoing check 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 reevaluation 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`. Because 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.* When a user's platform changes the value of an attribute such as location, the updated value must be propagated to other interested platforms, as described in Sections 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 Enforcing UCON Policies

To demonstrate the feasibility of usage control with the prototype, we consider three scenarios with location-based, task-based, and preobligation 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 one another.

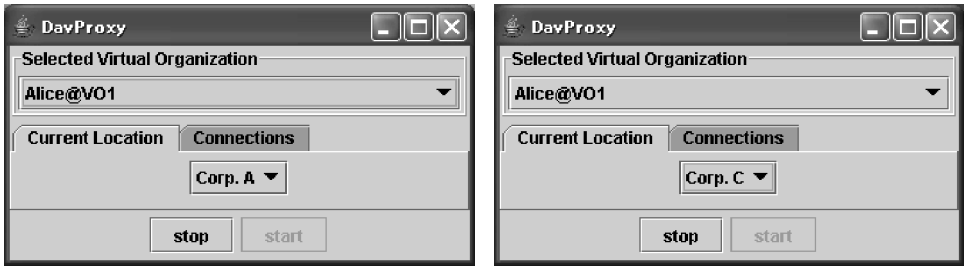
5.2.1 *Location-Based Access Control.* Although Internet-based collaborative systems enable developers to work together remotely, face-to-face meetings are still required to define and confirm application specifications in depth. This requirement is satisfied with mobile technologies such as laptop and 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, because VO1-related data might include sensitive information of Corp. B.

A UCON policy (see Figure 9) is defined according to this requirement in XACML format, restricting 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 (Figure 10) 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 (Figure 11[i]), and otherwise rejects it (Figure 11[ii]).

5.2.2 *Task-Based Access Control.* In collaborative software development, workflow 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

```
<Policy PolicyId="VO1-policy-1"
        PolicyCombinationAlg="rule-combining-algorithm:permit-overrides">
  <Target>
    <Subjects>
      <Subject>
        <!-- The subject identity must include "OU=VO1". -->
        <SubjectMatch MatchId="function:x500Name-match">
          <AttributeValue DataType="string">OU=VO1</AttributeValue>
          <SubjectAttributeDesignator AttributeId="subject-id" DataType="x500Name"/>
        </SubjectMatch>
        <!-- The subject is located in Corp. A or B. -->
        <SubjectMatch MatchId="function:regexp-string-match">
          <AttributeValue DataType="string">^Corp. [AB]$</AttributeValue>
          <SubjectAttributeDesignator AttributeId="subject-location" DataType="string"/>
        </SubjectMatch>
      </Subject>
    </Subjects>
    <Resources>
      <Resource>
        <!-- The object must be created by a VO1 member. -->
        <ResourceMatch MatchId="x500Name-match">
          <AttributeValue DataType="string">OU=VO1</AttributeValue>
          <ResourceAttributeDesignator AttributeId="creator-id" DataType="x500Name"/>
        </ResourceMatch>
      <Resource>
    </Resources>
    <Actions>
      <!-- "GET" represents the read privilege. -->
      <Action>GET</Action>
    </Actions>
  </Target>
  <Rule effect="permit"/>
</Policy>
```

Fig. 9. Location-based access control policy of VO1.



(i) In Corp. A (ii) In Corp. C

Fig. 10. Location change detection by the sensor in Alice's platform.

value FOR_DEVELOPMENT, which means that any developer can gain access to the object. The policy is defined in Figure 12.

As shown in Figure 13, 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 (e.g., Alice) cannot have access to the objects. The policy is shown in Figure 14. Another policy is needed to allow the tester's access on the object being tested, as shown in Figure 15. In this policy, the <Condition> element is used to compare the subject identity with an object attribute last-accessor-id, who

(i) In Corp. A (ii) In Corp. C

Fig. 11. Results when Alice tries to gain access to shared data in VO1 from different locations.

```
<Policy PolicyId="VO1-policy-2"
        PolicyCombinationAlg="rule-combining-algorithm:permit-overrides">
  <Target>
    ... <!-- Same as the location-based policy of VO1. --> ...
      <Resource>
        ... <!-- Same as the VO1-policy-1. --> ...
        <ResourceMatch MatchId="string-match">
          <AttributeValue DataType="string">FOR_DEVELOPMENT</AttributeValue>
          <ResourceAttributeDesignator AttributeId="InUse" DataType="string"/>
        </ResourceMatch>
      </Resource>
    ... <!-- Same as the location-based policy of VO1. --> ...
</Policy>
```

Fig. 12. Task-based access control policy of VO1: Allow any access to an object when its *InUse* = FOR_DEVELOPMENT.

indicates that only the subject who has locked the object can access it. The policy to unlock an object can be specified in a similar way.

5.2.3 *Preobligation Policy*. Consider a software development project under an extreme programming scheme. All developers are required to have associated reviewers to review their code before releasing [Beck 1999]. In this case, a preobligation is used to make sure that a reviewing process for a new code has been done and, if not, to notify an associated reviewer to review the code.

In the original XACML specification, however, obligations dictate only "postobligations," which are performed after decision making by PDP [OASIS XACML] and used to specify postupdates in our implementation.

To introduce preobligations into XACML, we extend our prototype as follows:

—Each obligation element in XACML has a *Type* attribute that indicates that the obligation is a UCON preobligation, or a "postobligation." For the above example, the modified XACML describes the prerequisite as follows:

```
<Obligation ObligationId="EnsureReviewProcess" Type="Pre"/>
```

—On an access request, a PDP first performs an action to invoke the preobligation, which may result in updated attribute(s) to an AR as a side effect.

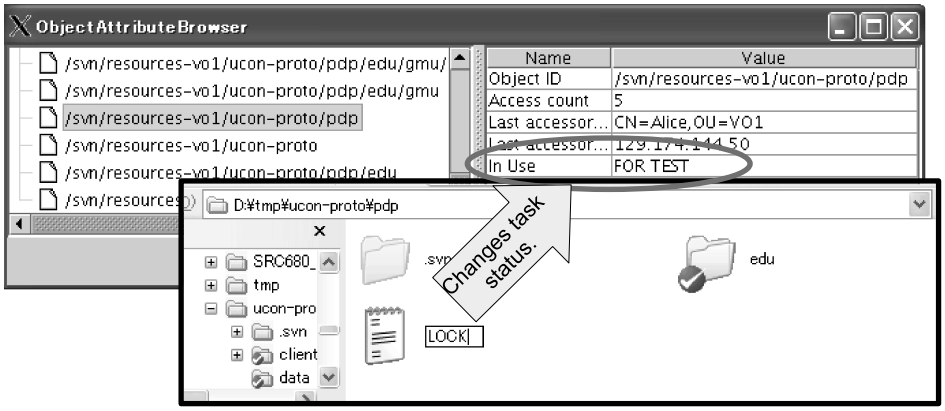


Fig. 13. An object attribute change after LOCK file creation.

```
<Policy PolicyId="LOCK policy"
        PolicyCombinationAlg="rule-combining-algorithm:permit-overrides">
  <Target>
    <Subjects>
      ... <!-- Same as the location-based policy of VO1. --> ...
    </Subjects>
    <Resources>
      <Resource>
        <ResourceMatch MatchId="string-match">
          <AttributeValue DataType="string">LOCK</AttributeValue>
          <ResourceAttributeDesignator AttributeId="resource-id" DataType="string"/>
        </ResourceMatch>
      </Resource>
    </Resources>
    <Actions>
      <Action>PUT</Action> <!-- object creation -->
    </Actions>
  </Target>
  <Rule effect="permit"/>
  <Obligations>
    <Obligation ObligationId="lock-action" FulfillOn="permit"/>
  </Obligations>
</Policy>
```

Fig. 14. Task-based access control policy of VO1: Allow to test an object by creating a LOCK file.

—Then, the PDP fetches (possibly updated) attribute values from the AR and makes a decision for the access request.

—Any postupdates are performed as specified by the original XACML obligations.

Note that these modifications violate the original XACML specification in terms of control flow. To support preobligations, we modify a main event loop, known as a context handler, in the Suns XACML library. Following the previous section, we discuss the support only of preobligation policies in this section.

By the extension of preobligations, we specify a policy for the aforementioned prerequisite of code review, as shown in Figure 16.

```
<Policy PolicyId="test-policy"
        PolicyCombinationAlg="rule-combining-algorithm:permit-overrides">
  <Target>
    ... <!-- Same as the location-based policy VO1. --> ...
      <Resource>
        ... <!-- Same as the location-based policy of VO1. --> ...
        <ResourceMatch MatchId="string-match">
          <AttributeValue DataType="string">FOR TEST</AttributeValue>
          <ResourceAttributeDesignator AttributeId="InUse" DataType="string"/>
        </ResourceMatch>
      </Resource>
    ... <!-- Same as the location-based policy of VO1. --> ...
  </Target>
  <Rule RuleId="AllowAnAccessFromTheTester" effect="permit">
    <!-- The subject must be identical to the last accessor of the locked objects.
    <Condition FunctionId="any-of">
      <Function FunctionId="x500Name-match"/>
      <Apply FunctionId="x500Name-one-and-only">
        <ResourceAttributeDesignator AttributeId="last-accessor-id" DataType="x500Name"/>
      </Apply>
      <SubjectAttributeDesignator AttributeId="subject-id" DataType="x500Name"/>
    </Condition>
  </Rule>
</Policy>
```

Fig. 15. Task-based access control policy of VO1: Allow the testing subject to access an locked object.

```
    <Resources>
      <Resource>
          <!-- Allows registration only for the files that have been reviewed. -->
          <ResourceMatch MatchId="string-match">
            <AttributeValue DataType="string">True</AttributeValue>
            <ResourceAttributeDesignator AttributeId="reviewed" DataType="string"/>
          </ResourceMatch>
      </Resource>
    </Resources>
    <Actions>
      <Action>PUT</Action>
    </Actions>
    <Rule effect="permit"/>
    <Obligations>
      <!-- Ensure the review process for new code. -->
      <Obligation ObligationId="EnsureReviewProcess" Type="Pre"/>
    </Obligations>
    </Policy>
```

Fig. 16. Preobligation policy: Allow to release an object if review process has been done.

When a developer (say Alice) releases new code, the PDP first invokes the preobligation function named <EnsureReviewProcess>. Then, the function confirms the status of the relevant review process with a project management system that we implement as a simple Perl-CGI application for the evaluation. If the review process is done by Alice's reviewer (say Bob), the function updates the value of the new code's attribute reviewed to *True*. Otherwise, the function leaves the attribute value be *False* and encourages the associated reviewer to review the new code by sending a notification email (see Figure 17). Because the policy specifies a condition that the release is permitted only when the reviewed attribute is *True,* the PDP rejects the access request unless the review process is done (see Figure 18).

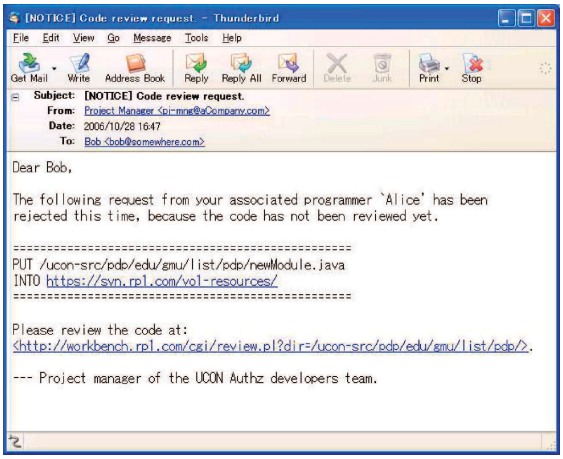


Fig. 17. Example of an obligation request mail sent to an associated reviewer.

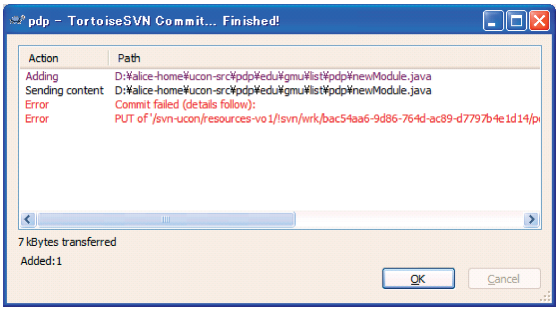


Fig. 18. Example of an error report showing that a PDP has rejected a release request for a file not reviewed.

5.3 Implementing Attribute Sharing Between ARs

As discussed in previous section, efficient attribute sharing is necessary for VO-level collaborations. For scalability and security, in our prototype a trusted mesh is built using VO-based referrals in conjunction with LDAP over SSL (LDAPS). A VO referral is a virtual LDAP entry including a URL to the actual entry on a remote server. When a LDAP client receives a referral from a local server, the client gains access to the actual entry accordingly to the URL. In our AR implementation, all attributes in a particular VO are descendants of the root entry identified as the VO's name (e.g., ou=VO1).

With the referrals we can establish a trusted mesh M = (V, E) in the following steps:

(1) Create a trusted certificate list of V and distribute it to individual ARs in V.
(2) For each AR in V, register individual referrals of other ARs in (V - {AR}) as shown in Figure 19. Note that each URL has the LDAPS prefix.

```
dn: ou=VO1
ou: VO1
objectclass: top
objectclass: organizationalunit
objectclass: referral
ref: ldaps://ar1.companyA.com/ou=VO1

dn: ou=VO2
ou: VO2
objectclass: top
objectclass: organizationalunit
objectclass: referral
ref: ldaps://ar2.companyB.com/ou=VO2

...
```

Fig. 19. Example of referral entries in an AR.

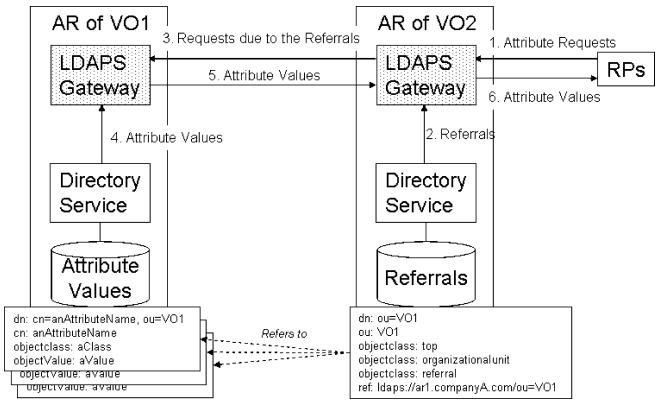


Fig. 20. Data flow for attribute sharing between ARs.

Based on this, attribute sharing can be performed cooperatively between ARs with the following steps (refer to Figure 20):

(1) In the local AR in a VO (e.g., VO2), an RP in the same VO requests an attribute value that is actually stored on a remote AR in another VO (e.g., VO1).
(2) By looking up the requested value with the internal directory service, the AR obtains a referral to the remote AR.
(3) The AR requests the attribute value with the LDAPS gateway, which establishes SSL channels with the LDAPS gateway of the remote AR in a mutual authentication mode and forwards the attribute request through the secure channel.
(4) The remote LDAPS gateway obtains the attribute value from the coexisting directory service.

Table I. PDP Performance Overhead in File Sharing Prototype System

Access operations: update (# of files, avg. file size)	Total PDP processing (msec)	# of accesses	Avg. time per access (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

(5) The remote LDAPS gateway sends the attribute value back to the local LDAPS gateway.
(6) The local LDAPS gateway forwards the attribute value to the RP.

5.4 Performance Evaluation

5.4.1 *Policy Enforcement Performance in Single VO.* 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 Figure 8, 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, for example the existence of a `LOCK` file under a module directory in RP. For simplicity we assume that a subject participates in only a single software module at the RP. Table I 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 of 45.30 to 99.43 milliseconds, 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 I 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 ten 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.

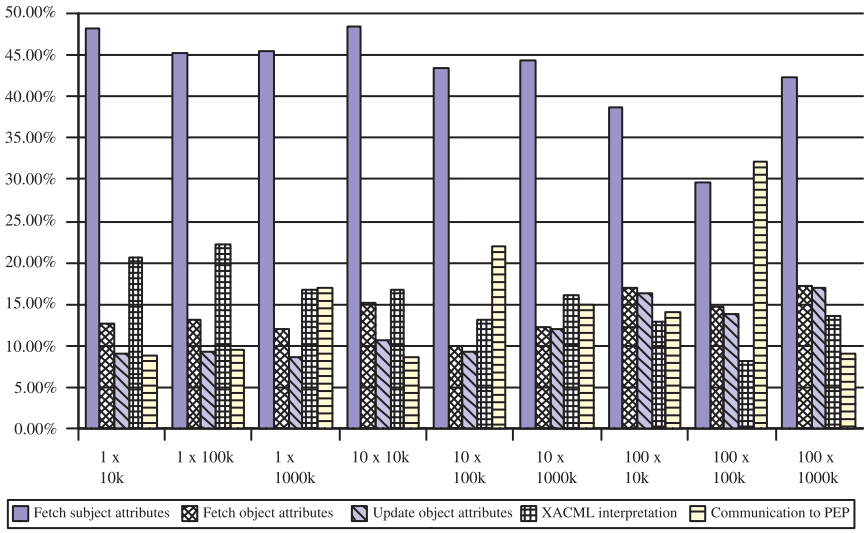


Fig. 21. Micro benchmark of the PDP module. Each value shows the ratio of the stepwise running time to the entire PDP processing time.

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 subject attributes.[5] To investigate possible mechanisms for better performance, we measured the processing time of each step in the PDP module. As shown in Figure 21, fetching subject attributes has the highest cost, which makes up 30 to 48% of the overall processing time. This results from the overhead of the SSL hand shaking between the PDP module and the AR. In real applications, mechanisms such as *keep-alive* connections and attribute value cache on the PDP side can be used to reduce this overhead and thus improve the overall performance of the system.

5.4.2 *Performance of VO Federation.* Based on the VO federation scheme in Section 5.3, we conduct a performance evaluation for the attribute sharing between ARs. The experimental system comprises three Linux servers (Pentium4 1GHz and 1G bytes of memory) in a closed 100-Base TX network. The two of them service AR that comprises OpenLDAP service as a directory service and an LDAPS gateway written in Java. The remaining Linux server is dedicated as an RP that requests attribute values from the local AR.

To investigate the impact of the cooperation between the local and remote ARs, we record the RP's processing times for making 10,000 queries in the cases of obtaining attributes from the local AR, from the remote AR, and from both. As shown in Figure 22, the access cost of the remote AR is over twice

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

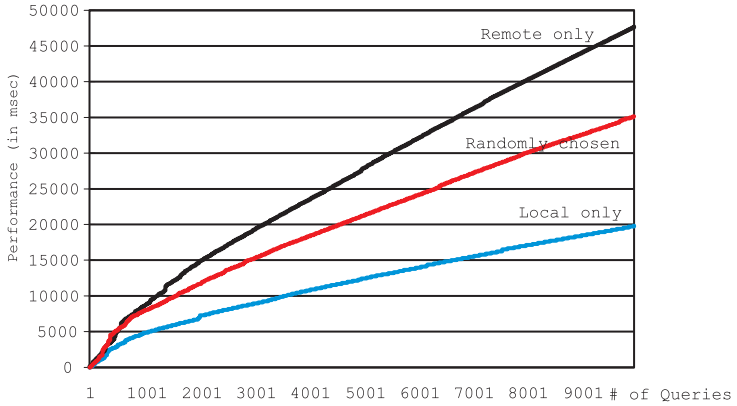


Fig. 22. Performance of attribute sharing between two ARs. The ratio of the random choice is 1:1 for the local and remote ARs.

as that of gaining access to the local AR. This is because of the extra cost of processing referrals. In the aforementioned steps for attribute sharing, steps 3 to 5 are performed only when the RP requests an attribute value on a remote AR. These steps induce another directory search that takes approximately 2 milliseconds, as long as step 2, and the additional communication between the ARs that takes 0.4 milliseconds in average.

Note that the cooperative processing time does not depend on the number of AR because the above steps are triggered by a referral to a single AR, no matter how many ARs are associated. Hence, if the average time through a directory search and a network communication is the same as the above results, we can estimate that the cost of attribute sharing with ARs is less than 4.8 milliseconds. Because the estimated time is 5 to 10% of the RP's performance (see Table I), the cost due to attribute sharing between ARs is acceptable.

6. RELATED WORK

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

The Community Authorization Service (CAS) [Pearlman et al. 2002] 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, consider a temporary group involving some users collaborating using their mobile devices, where the authorization policy is based on the location of the platforms (e.g., only users in the same room can gain access to the shared resources). Because there is no centralized point, CAS cannot solve this problem. Also, CAS lacks flexibility to support a new RP that has not established

a trust relationship with CAS or an existing RP to change its policy regarding its shared resources.

Instead of centralized authorization, the Virtual Organization Membership Service (VOMS) [Alfieri et al. 2005] describes an approach in which each RP has a set of local policies. To gain access to a shared resource, the 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 a centralized server to each RP's local site, VOMS can solve the scalability problem with the grid-mapfile and the flexibility of CAS, but it cannot support collaborations without a well-established infrastructure because it still requires a (globally) centralized attribute authority. Further, because an attribute in VOMS includes only 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 can gain permissions only from a VO administrator.

PRIMA [Lorch et al. 2003] is a privilege management system that supports ad hoc collaboration and permission delegation. To submit a request to an RP, a user provides a set of attributes that define the privileges of the user, such as file access permissions, user quota, network access, and the likes. 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 preissued 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 preassigned privileges, such as context-aware attributes [Covington et al. 2006]. 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 [Thompson et al. 2003] is a distributed policy management system, where a set of stakeholders defines 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 to the mutable attribute acquisition in our approach model, while the mutability of conditions is not supported in Akenti. Also, because it is extensively dependent on public key infrastructure (PKI), Akenti cannot support ad hoc collaborations without preestablished infrastructure.

RBAC-based approaches have been proposed for secure interoperation in multidomain environments [Joshi et al. 2004; Shafiq et al. 2005], where each domain deploys RBAC policies and a set of global access control policies is composed to control shared resources accesses. Very recently, a framework for secure collaboration between domains has been proposed in Shehab and colleagues [2005], where each domain uses RBAC and policies are locally enforced by individual domains in a mediator-free manner. The advantage of our framework is that we leverage the flexibility and expressive power of UCON

model for general access control policies, which can be regarded as an abstract model beyond individual policy models in different domains.

Context-aware authorizations have been studied by several researchers. In Covington and colleagues [2001], security-relevant contextual information is captured by environment roles, an extension to RBAC. A context-aware access control model based on RBAC is presented in Zhang and Parashar [2003] for pervasive Grid applications, where a context agent collects environmental information and dynamically enforces user-role and permission-role. In Tolone and colleagues [2005], 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.

7. CONCLUSION AND FUTURE WORK

An authorization framework is proposed in this paper for collaborative computing systems following the PEI 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 architecture in our prototype includes only authorizations, conditions, and preobligations of UCON; we plan to explore ongoing obligations as future work.

REFERENCES

ALFIERI, R., CECCHINIB, R., CIASCHINIC, V., DELL'AGNELLOD, L., FROHNERE, A., LORENTEYF, K., AND SPATAROG, F. 2005. From gridmap-file to voms: Managing authorization in a grid environment. *Future Gener. Comput. Syst. 21.*

BECK, K. 1999. *Extreme Programming Explained: Embrace Change*. Addison-Wesley.

BELL, D. E. AND LAPADULA, L. J. 1975. Secure computer systems: Mathematical foundations and model. Tech. rep., Mitre Corp., Bedford, MA.

BERTINO, E., CRISPO, B., JOSHI, J., DU, W. K., AND SANDHU, R. S. 2004. Panel: Security for grid-based computing systems issues and challenges. In *Proceedings of the 9th ACM Symposium on Access Control Models and Technologies*, 125.

COHEN, B. 2003. Incentives build robustness in bittorrent. In *Proceedings of the 1st Workshop on Economics of Peer-to-Peer Systems*. http://www.bittorrent.com/bittorrentecon.pdf.

COVINGTON, M. J., LONG, W., SRINIVASAN, S., DEY, A. K., AHAMAD, M., AND ABOWD, G. D. 2001. Securing context-aware applications using environment roles. In *Proceedings of the 9th ACM Symposium on Access Control Models and Technologies (SACMAT '01)*. Chantilly, VA. 10–20.

COVINGTON, M. J., SASTRY, M. R., AND MANOHAR, D. J. 2006. Attribute-based authentication model for dynamic mobile environments. In *Proceedings of the 3rd International Conference on Security in Pervasive Computing (SPC'06)*. Lecture Notes in Computer Science, Springer.

DB4OBJECT. http://www.db4o.com/.

DENNING, D. E. 1976. A lattice model of secure information flow. *Comm. ACM 19,* 5 (May).

FOSTER, I., KESSEKAN, C., TSUDIK, G., AND TUECKEL, S. 1998. A security architecture for computational grids. In *Proceedings of ACM Conference on Computer and Communications Security*.

FOSTER, I., KESSELMAN, C., AND TUECKE, S. 2001. The anatomy of the grid: Enabling scalable virtual organization. *Int. J. Supercomput. Appl. 15,* 3.

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

JOSHI, J., BHATTI, R., BERTINO, E., AND GHAFOOR, A. 2004. Access control language for multidomain environments. *IEEE Intern. Comput.*, 40–50.

LORCH, M., ADAMS, D. B., KAFURA, D., KONENI, M. S. R., RATHI, A., AND SHAH, S. 2003. The prima system for privildge management, authorization and enforcement in grid environments. In *Proceedings of the 4th International Workshop on Grid Computing*.

MOD_DAV. a DAV module for Apache, http://www.webdav.org/mod_dav/.

OASIS XACML. *Core Specification: eXtensible Access Control Markup Language (XACML)*. OASIS XACML.

OPENLDAP. http://www.openldap.org/.

OPENSSL. http://www.openssl.org/.

PARK, J. 2003. Usage control: A unified framework for next generation access control. Ph.D. thesis, George Mason University.

PARK, J. AND SANDHU, R. 2004. The UCON_{abc} usage control model. *ACM Trans. Inform. Syst. Secur. 7,* 1 (Feb), 128–174.

PARK, J., ZHANG, X., AND SANDHU, R. 2004. Attribute mutability in usage control. In *Proceedings of the Annual IFIP WG 11.3 Working Conference on Data and Applications Security*. Sitges, Catalonia, Spain, 15–29.

PARK, J. S. AND SANDHU, R. 2000. Binding identities and attributes using digitally signed certificates. In *Proceedings of the Annual Computer Security Applications Conference*. New Orleans, LA. 120–127.

PEARLMAN, L., WELCH, V., FOSTER, I., AND KESSELMAN, K. 2002. A community authorization service for group collaboration. In *Proceedings of IEEE Workshop on Policies for Distributed Systems and Networks*.

SAILER, R., JAEGER, T., ZHANG, X., AND VAN DOORN, L. 2004. Attestation-based policy enforcement for remote access. In *Proceedings of ACM Conference on Computer and Communication Security*. Washington, DC, USA, 308–317.

SANDHU, R. 1993. Lattice-based access control models. *IEEE Comput. 26,* 11 (Nov.).

SANDHU, R. 2000. Engineering authority and trust in cyberspace: The OM-AM and RBAC way. In *Proceedings of the 5th ACM Workshop on Role-based Access Control*. Berlin, Germany, 111–119.

SANDHU, R., RANGANATHAN, K., AND ZHANG, X. 2006. Secure information sharing enabled by trusted computing and PEI models. In *Proceedings of the ACM Symposium on Information, Computer, and Communication Security*. Taipei, Taiwan.

SHAFIQ, B., JOSHI, J., BERTINO, E., AND GHAFOOR, A. 2005. Secure interoperation in a multidomain environment employing RBAC poilcies. *IEEE Trans. Knowl. Data Eng. 17,* 11 (Nov), 1557–1577.

SHEHAB, M., BERTINO, E., AND GHAFOOR, A. 2005. Secure collaboration in mediator-free environments. In *Proceedings of the 12th ACM Conference on Computer and Communication Security*.

SUBVERSION. http://subversion.tigris.org/.

TCG MTM. 2006. Mobile trusted module specification, https://www.trustedcomputinggroup.org/specs/mobilephone/tcg-mobile-trusted-module-0.9.pdf.

TCG TPM. 2003. Main part 1 design principles specification version 1.2, https://www.trustedcomputinggroup.org.

THOMAS, R. AND SANDHU, R. 1997. Task-based authorization controls (TBAC): Models for active and enterprise-oriented authorization management. In *Proceedings of the 11th IFIP WG 11.3 Working Conference on Database and Application Security*. Published as Database Security XI: Status and Prospects. T. Y. Lin and X. Qian, Eds. North-Holland.

THOMPSON, M., ESSIARI, A., AND MUDUMBAI, S. 2003. Certificate-based authorization policy in a pki environment. *ACM Trans. Inform. Syst. Secur. 6,* 4.

TOLONE, W., AHN, G., AND PAI, T. 2005. Access control in collaborative systems. *ACM Comput. Surv. 37,* 1 (March).

WELCH, V., SIEBENLIST, F., FOSTER, I., BRESNAHAN, J., CZAJ, K., GAWOR, J., KESSELMAN, C., MEDER, S., PEARLMAN, L., AND TUECKE, S. 2003. Security for grid services. In *Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing*. Seattle, WA, 48–57.

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

ZHANG, G. AND PARASHAR, M. 2003. Dynamic context-aware access control for grid applications. In *Proceedings of the 4th International Workshop on Grid Computing*.

ZHANG, X., PARISI-PRESICCE, F., SANDHU, R., AND PARK, J. 2005. Formal model and policy specification of usage control. *ACM Trans. Inform. Syst. Secur. 8,* 4 (Nov), 351–387.

Received November 2006; accepted June 2007