Available online at www.sciencedirect.com
SCIENCE DIRECT®
Journal of Network and Computer Applications ▮ (▮▮▮▮) ▮▮▮–▮▮▮
ELSEVIER
Journal of NETWORK and COMPUTER APPLICATIONS
www.elsevier.com/locate/jnca

Towards secure information sharing using role-based delegation ☆

Gail-Joon Ahn[a,*], Badrinath Mohan[a], Seng-Phil Hong[b]

[a]*University of North Carolina at Charlotte, Charlotte, NC, USA*
[b]*Information and Communications University, Taejon, Republic of Korea*

Abstract

As computing becomes more pervasive, information sharing occurs in broad, highly dynamic network-based environments. Such pervasive computing environments pose a difficult challenge in formally accessing the resources. The digital information generally represents sensitive and confidential information that organizations must protect and allow only authorized personnel to access and manipulate them. As organizations implement information strategies that call for sharing access to resources in the networked environment, mechanisms must be provided to protect the resources from adversaries. In this paper, we seek to address the issue of how to advocate selective information sharing while minimizing the risks of unauthorized access. We integrate a role-based delegation framework to propose a system architecture. We also demonstrate the feasibility of our framework through a proof-of-concept implementation.

Keywords: Information sharing; Role-based; Delegation

☆Portions of this paper appeared in preliminary form under the title Secure Information Sharing Using Role-based Delegation in Proceedings of IEEE International Conference on Information Technology: Coding & Computing (ITCC), Las Vegas, NV, April 5–7, 2004.

*Corresponding author. Tel.: +1 704 687 3783; fax: +1 704 687 4893.
E-mail addresses: gahn@uncc.edu (G.-J. Ahn), bmohan@uncc.edu (B. Mohan), philhong@icu.ac.kr (S.-P. Hong).

1084-8045/$ - see front matter © 2005 Elsevier Ltd. All rights reserved.
doi:10.1016/j.jnca.2005.08.004

1. Introduction

Several organizations have transited from their old and disparate business models based on ink and paper to a new, consolidated ones based on digital information on the Internet. The Internet is uniquely and strategically positioned to address the needs of a growing segment of population in a very cost-effective way. It provides tremendous connectivity and immense information sharing capability which the organizations can use for their competitive advantage. However, balancing the competing goals of collaboration and security is difficult because interaction in collaborative systems is targeted towards making people, information, and resources available to all who need it, whereas information security seeks to ensure the integrity of these elements while providing it only to those with proper authorization. Furthermore, as computing becomes more pervasive, information sharing occurs in broad, highly dynamic network-based environments. Such pervasive computing environments pose a difficult challenge in formally accessing the resources (The Pittsburgh Pebbles PDA Project, 2004).

Digital information generally represents sensitive and confidential information that organizations must protect and allow only authorized personnel to access and manipulate them. As organizations implement information strategies that call for sharing access to resources in the networked environment, mechanisms must be provided to protect the resources from adversaries. We seek to address the issue of how to advocate selective information sharing in pervasive computing environments while minimizing the risks of unauthorized access. We integrate a role-based delegation framework (Zhang et al., 2003) to propose a system architecture. We also demonstrate the feasibility of our framework through a proof-of-concept implementation.

The rest of this paper is organized as follows: in Section 2, we discuss role-based delegation including details of system architecture. Section 3 overviews other research and related technologies. Section 4 describes implementation details. Section 5 concludes this paper.

2. Role-based delegation

Ahn et al. (2003) have recently identified the following issues in collaborative environments. First, selective information sharing is necessary. We are dealing with friends, not adversaries, and should provide relevant information expeditiously. Second, the information may be shared across organizational boundaries. Medical records may be exchanged between collaborative hospitals for shared patient; researchers may reside in different healthcare organizations. Because sharing a resource across organizational boundaries often means authorizing a server to give access to a third party, it implies enabling resource servers to reason about previously unknown third parties. This requirement contrasts with many conventional systems, wherein a server only needs to reason about the set of users known inside a given organization. Third, it is impossible to fully predicate what data

should be shared, when and to whom. And another thing is that a mechanism must be provided for revoking the sharing when it is no longer needed. All these factors have to be considered in order to formulate the mechanism for information sharing among collaborating organizations.

In order to deal with the aforementioned issues, our work, called FRDIS (A Framework of Role-based Delegation for Information Sharing), leverages the existing models (Sandhu et al., 1996; Zhang et al., 2003). To illustrate each functional component in our model, we use the role hierarchy example illustrated in Fig. 1 and Table 1.

To simplify the discussion of delegation, we assume a user cannot be delegated to a role if the user is already a member of that role. For example, project leader Deloris with role PL1 cannot be delegated to the role PO1 or PC1 since he has already been an implicit member of these roles.

2.1. Role delegation

We first define a new relation called delegation relation (DLGT). It includes three elements: original user assignments UAO, delegated user assignment UAD, and constraints. The motivation behind this relation is to address the relationships among different components involved in a delegation. In a user-to-user delegation, there are four components: a delegating user, a delegating role, a delegated user, and a delegated role. For example, (*Deloris*, PL1, *Cathy*, PL1) means *Deloris* acting in role PL1 delegates role PL1 to *Cathy*. The delegation relation supports role hierarchies: a user who is authorized to delegate a role r can also delegate a role r' that is junior to r. For example, (*Deloris*, PL1, *Lewis*, PC1) means *Deloris* acting in role PL1 delegates a junior role PC1 to *Lewis*. A delegation relation is one-to-many relationship on user assignments. It consists of original user delegation (ODLGT)

Fig. 1. Role hierarchy and membership.

Table 1
Role membership

ROLES	DIR	PL1	PL2	PO1	PO2
USERS	John	Deloris	Cathy	Michael David	Mark Lewis

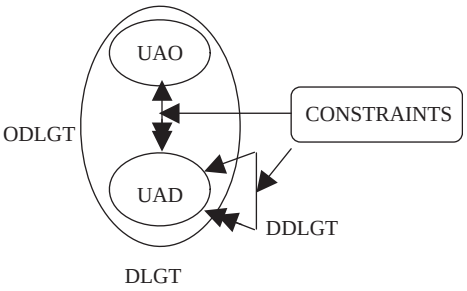

Fig. 2. Delegation relation.

and delegated user delegation (DDLGT). Fig. 2 illustrates components and their relations in FRDIS. We assume each delegation relation may have a duration constraint associated with it. If the duration is not explicitly specified, we consider the delegation as permanent unless another user revokes it. The function Duration returns the assigned duration-restriction constraint of a delegated user assignment. If there is no assigned duration, it returns a maximum value.

In some cases, we may need to define whether or not each delegation can be further delegated and for how many times, or up to the maximum delegation depth. We introduce two types of delegation: single-step delegation and multi-step delegation. Single-step delegation does not allow the delegated role to be further delegated; multi-step delegation allows multiple delegations until it reaches the maximum delegation depth. The maximum delegation depth is a natural number defined to impose restriction on the delegation. Single-step delegation is a special case of multi-step delegation with maximum delegation depth equal to one.

Also, we have an additional concept, delegation path (DP) that is an ordered list of user assignment relations generated through multi-step delegation. A delegation path always starts from an original user assignment. We use the following notation to represent a delegation path.

$uao_0 \rightarrow uad_1 \rightarrow uad_i \rightarrow uad_n$.

Delegation paths starting with the same original user assignment can further construct a delegation tree. A delegation tree (DT) expresses the delegation paths in a hierarchical structure. Each node in the tree refers to a user assignment and each edge to a delegation relation. The layer of a user assignment in the tree is referred as the delegation depth. The function Prior maps one delegated user assignment to the

delegating user assignment; function Path returns the path of a delegated user assignment; and function Depth returns the depth of the delegation path.

FRDIS has the following components and theses components are formalized from the above discussions.

- U is a set of users.
- R is disjoint sets of roles and administrative roles, respectively.
- $UAO \subseteq U \times R$ is an original user to role assignment relation.
- $UAD \subseteq U \times R$ is a delegated user to role assignment relation.
- $UA = UAO \cup UAD$
- $DLGT \subseteq UA \times UA$ is one to many delegation relation. A delegation relation can be represented by $(u, r, u', r') \in DLGT$, which means the delegating user u with role r delegated role r' to user u'.
- $ODLGT \subseteq UAO \times UAD$ is an original user delegation relation.
- $DDLGT \subseteq UAD \times UAD$ is a delegated user delegation relation.
- $DLGT = ODLGT \cup DDLGT$.
- T is a set of duration-restricted constraint.
- $DP \subseteq UA \times UA$ represents a delegation path.
- $DT \subseteq UA \times UA$ represents a delegation tree.
- *Path*: $UA \rightarrow DP$ is a function that maps a UA to a delegation path.
- *Prior*: $UA \rightarrow UA$ is a function that maps a user assignment to another subsequent user assignment that forms a delegation relation.

Constraints are an important aspect of RBAC and can lay out higher-level organizational policies. In theory, the effects of constraints can be achieved by establishing procedures and sedulous actions of security administrators (Ferraiolo et al., 1999). Specification of policies for the authorization needs in the context of role management has been also studied by Lupu and Sloman (1997). In FRDIS, the constraints are enforced by a set of integrity rules that provide management and regulators with the confidence that critical security policies are uniformly and consistently enforced. In the framework, when a user delegates a role, all context constraints that are assigned to the user and anchored to the delegated role are delegated as well.

In order to specify and enforce constraints in role-based delegation authorizations, we first overview some identified constraints in role-based systems (Ahn and Sandhu, 2000; Bertino et al., 1999a, b; Sandhu et al., 1996):

(1) *Static separation of duty* (*SSOD*)/*Incompatible roles assignment*: This constraint states that no common user should be assigned to conflicting roles. A frequently used example is a user cannot be a purchase manager while at the same time being an account payable manager for the same organization. We denote a set of incompatible role assignments as *IRA*.

(2) *Incompatible users*: This constraint states that two conflicting users cannot be assigned to the same role. For example, it might be a company's policy that members from same subdivision should not be assigned to the same steering committee. We denote a set of incompatible users as *IU*.

(3) *Incompatible permissions*: This constraint states that conflicting permissions cannot be assigned to the same role. We denote a set of incompatible permissions as *IP*.

(4) *Cardinality constraints*: This constraint states that a role can have a maximum number of members or a user may belong to a maximum number of roles. For example, there may be only one person in the role of CEO in an organization. As stated in Sandhu et al. (1996), the role cardinality is difficult to implement since the system may not know exactly how many users are still "alive"—some may leave without notifying security officers. We denote the cardinality of x as $cardi(x)$, where x is a role term or a user term. It is futile to enumerate all role-based constraints, as there are too many possibilities and variants (Ahn and Sandhu, 2000). In the subsequent sections, we show that the enhanced rule-based language is expressive enough to specify a wide range of constraints.

2.2. Role revocation

Several different semantics are possible for user revocation. Hagstrom et al. (2001) categorized revocations into three dimensions in the context of owner-based approach: global and local (propagation), strong and weak (dominance), and deletion or negative (resilience). Barka and Sandhu (2000b) further identified user grant-dependent and grant-independent revocation (grant-dependency). Since negative authorization is not considered in FRDIS, we articulate user revocation in the following dimensions: grant-dependency, propagation, and dominance.

Grant-dependency refers to the legitimacy of a user who can revoke a delegated role. Grant-dependent revocation means only the delegating user can revoke the delegated user from the delegated role membership. Grant-independent revocation means any original user of the delegating role can revoke the user from the delegated role.

Dominance refers to the effect of a revocation on implicit/explicit role memberships of a user. A strong revocation of a user from a role requires that the user be removed not only from the explicit membership but also from the implicit memberships of the delegated role. A weak revocation only removes the user from the delegated role (explicit membership) and leaves other roles intact. Strong revocation is theoretically equivalent to a series of weak revocations. To perform strong revocation, the implied weak revocations are authorized based on revocation policies. However, a strong revocation may have no effect if any upward weak revocation in the role hierarchy fails (Sandhu et al., 1999).

Propagation refers to the extent of the revocation to other delegated users. A cascading revocation directly revokes a delegated user assignment in a delegation relation and also indirectly revokes a set of subsequent propagated user assignments. A non-cascading revocation only revokes a delegated user assignment.

Our preliminary study shows grant-dependent revocation for brevity. Suppose the revocation in Fig. 3 is weak non-cascading, for *John* to revoke *Cathy* from role PL1, it is important to note that only *Cathy*'s membership of role PL1 is changed; other role memberships of *Cathy* and all the delegated user assignments propagated by

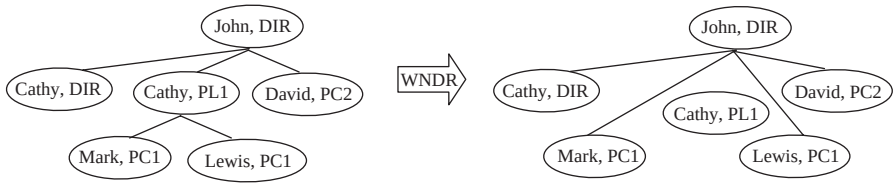

Fig. 3. Weak non-cascading revocation.

Cathy are still valid. If the revoked node is not a leaf node, non-cascading revocation may leave a "hole" in the delegation tree. A solution might be the revoking user takes over the delegating user's responsibility. In this example, *John* takes over the delegating user's responsibility from *Cathy*, and changes all delegation relations: $(Cathy, PL1, u, r) \in DLGT$ to $(John, DIR, u, r) \in DLGT$. In this case, *John* takes over *Cathy*'s delegating responsibility for *Mark* and *Lewis*.

2.3. Rule-based policy specification language

FRDIS defines policies that allow regular users to delegate their roles. It also specifies the policies regarding which delegated roles can be revoked. A rule-based language is adopted to specify and enforce these policies. It is a declarative language in which binds logic with rules. The advantage is that it is entirely declarative so it is easier for security administrator to define policies.

A *rule* takes the form:

$H \leftarrow F1\&F2\& \ldots \&Fn$

where H, $F1, F2, \ldots, Fn$ are Boolean functions.

There are three sets of rules in the framework: basic authorization rules specify organizational delegation and revocation policies; authorization derivation rules enforce these policies in collaborative information systems; and integrity rules specify and enforce role-based constraints.

For example, a user–user delegation authorization rule forms as follows:

$can\_delegate(r, cr, n) \leftarrow,$

where r, cr, and n are elements of roles, prerequisite conditions, and maximum delegation depths, respectively.

This is the basic user-to-user delegation authorization rule. It means that a member of the role r (or a member of any role that is senior to r) can assign a user whose current membership satisfies prerequisite condition cr to role r (or a role that is junior to r) without exceeding the maximum delegation depth n.

A user delegation request is further authorized by the *user–user delegation authorization derivation rule* that takes the form

$der\_can\_delegate(u, r, u', r', dlg\_opt) \leftarrow$
$\quad can\_delegate(r'', cr, n)\&$
$\quad active(u, r, s)\&$
$\quad delegatable(u, r)\&$

$$senior(r, r'')\&$$
$$in(u', cr)\&$$
$$junior(r', r'')\&$$
$$in(depth(u, r), n),$$

where u and u' are elements of users; r, r', and r'' are elements of roles; cr and s are elements of prerequisite condition and sessions, respectively; *dlg_opt* is a Boolean term, if it is true, then further delegation is allowed. This argument is used as Boolean control of delegation propagation.

This rule means that a user u with a membership of a role r senior to r'' activated in session s can delegate a user u' whose current role membership satisfies prerequisite condition cr to role r' (r' is junior to role r'') without exceeding the maximum delegation depth n. Similar rules are also defined for role-based revocations and are applied to specify constraints.

The fundamental element of the specification language is a set of functions. We categorize these functions into three groups:

- a set of specification functions, expressing information of role-based delegation components;
- a set of authorization functions, describing an authorization information or decision. We further divide them into a set of basic authorization functions (BAP), a set of derived authorization functions (DAP), and a set of negative authorization functions (NAP); and
- a set of utility functions, providing supportive functionalities, e.g. comparison and aggregation.

Some functions and their semantics are listed in Tables 2–4. We use UT, RT, PT, ST, UAT, PAT, DLGTT, DPT, CRT, IRAT, IUT, IRRT, BT, DT, and NT to indicate set of users, roles, permissions, sessions, user assignments, permission assignments, delegations, delegation paths, prerequisite conditions, incompatible role assignments, incompatible users, incompatible roles, booleans, durations, and

Table 2
Utility functions

Predicate	Arity	Argument	Return	Meaning
all_other	2	Set of XT, XT	Set of XT	*all_other*$(X, x) = X - \{x\}$
equals	2	XT, XT	BT	If *equals*(x, y) is true, then $x = y$
in	2	XT, set of XT	BT	If *in*(x, y) is true, then $x \in y$
not	1	BT	BT	*not*(true) = false and *not*(false) = true
one_element	1	Set of XT	XT	*one_element*(X) returns one element in set X
satisfy	2	UT, CR	BT	If *satisfy*(u, cr) is true, then user u satisfies cr

Table 3
Specification functions

Predicate	Arity	Arg	Return	Meaning
cardi	1	RT	NT	*cardi*(r) returns current number of users in role r
delegatable	1	UAT	BT	If *delegatable*(*ua*) is true, then *ua* can further delegate
maxcardi	1	RT	NT	*maxcardi*(r) returns the maximum cardinality allowed for role r

Table 4
Authorization functions

Predicate	Arity	Arg	Type	Meaning
override	2	Rule head, BT	BAP	*override*(H, b) states conflict resolution policy for $H \leftarrow B$
cannot_assign	2	UT, RT	NAP	*cannot_assign*(u, r) means user u cannot be assigned role r
cannot_assignp	2	RT, PT	NAP	*cannot_assignp*(r, p) means permission p cannot be assigned to role r
cannot_activate	3	UT, RT, ST	NAP	*cannot_activate*(u, r, s) means u cannot activate r in sessions

natural numbers, respectively.We borrow the notion of two non-deterministic functions from RCL2000 (Ahn and Sandhu, 2000): *one_element* and *all_other* (originally as OE and AO). These functions are introduced to replace explicit quantifiers, thus keep the language simple and intuitive. The *one_element*(X) function allows us to get one element x_i from set X. Multiple occurrences of *one_element*(X) in a single rule statement select the same element x_i. With *all_other*(X, x_i) we can get a subset of X by taking out one element x_i.

2.3.1. Constraints specification

In order to represent role-based privilege management constraints, we define rules that are extremely suited for constraints specification as well as enforcement. We articulate several constraints and specify them using a rule-based language introduced in Zhang et al. (2003).

A *static separation of duty* (*SSOD*): *incompatible roles assignment constraint* states that no common user can be assigned to conflicting roles in the incompatible role set $ira = \{r_1, r_2, \ldots\}$. This constraint can be represented as

$$\begin{aligned} &cannot\_assign(u, r) \leftarrow \\ &\quad senior(r, one\_element(ira))\& \\ &\quad member\_of(u, one\_element(all\_other(ira, one\_element(ira)))), \end{aligned}$$

where $u \in U$, $r \in R$, and $ira \in IRA$.

The rule says if r equals one element of a set of the incompatible role assignments *ira*, and a user u is already member of another role other than r in the incompatible role set, then u cannot be assigned role r.

An *incompatible users constraint* states that two conflicting users in the incompatible user set $iu = \{u1, u2, \ldots\}$ cannot be assigned to the same role. This constraint can be represented as

$$cannot\_assign(u, r) \leftarrow equals(u', one\_element(all\_other(iu, u)))\& member\_of(u', r).$$

An *incompatible permissions constraint* states that two conflicting permissions in the incompatible user set $ip = \{p1, p2, \ldots\}$ cannot be assigned to the same role. This constraint can be represented as

$$cannot\_assignp(r, p) \leftarrow equals(p', one\_element(all\_other(ip, p)))\& in(p', permissions\_role(r)).$$

A *role cardinality constraint* states that a role can have a maximum number N of user members. This constraint can be represented as

$$cannot\_assign(u, r) \leftarrow greater\_than(cardi(r), maxcardi(r) - 1).$$

A *user cardinality constraint* states that a user can be member of a maximum number N of roles. This constraint can be represented as

$$cannot\_assign(u, r) \leftarrow greater\_than(cardi(u), maxcardi(u) - 1).$$

We have demonstrated how different constraints can be specified using rules.

2.4. Architectural framework

Our system is designed to provide access control and delegation in collaborative environments. We use the term *Hive* to describe its system architecture. In a bee hive, honeybees collect nectar and pollen and store them for other bees to use to make honey; in the *Hive*, any legitimate users can access resources at anytime and anywhere. To do so, access control services should be provided to users who have access privileges that can be originally assigned by a security officer or can be delegated by other legitimate user(s). The notions described in *Hive* are designed to be utilized within an administrative-directed delegation management architecture.

An overview of the preliminary architecture is shown in Fig. 4. It consists of a number of services and management agents together with the objects to be managed.

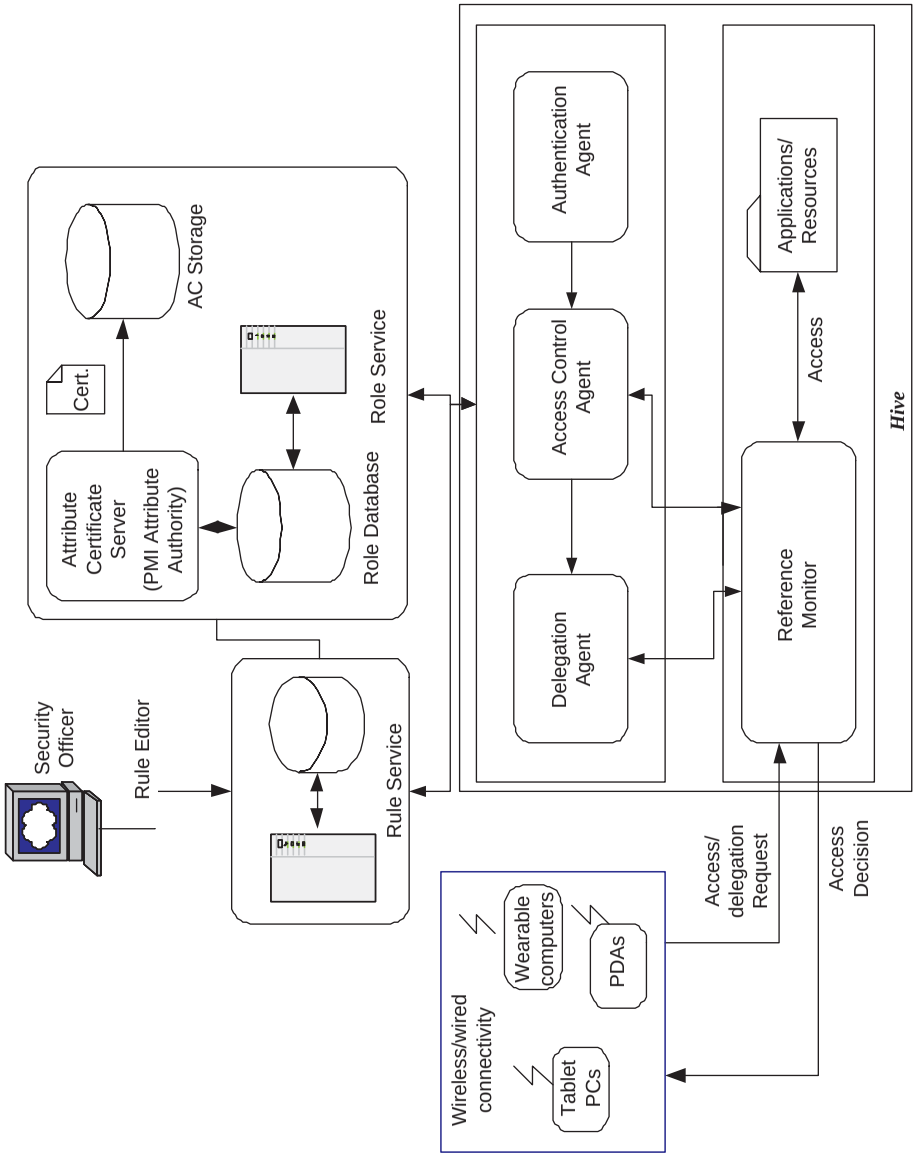

Fig. 4. FRDIS architecture: *Hive*.

The enforcement agents are based on a combination of roles and rules for specifying and interpreting policies. Since delegation and revocation services are only part of a security infrastructure, we choose a modular approach to our architecture that allows the delegation and revocation services to work with current and future authentication and access control services. The modularity enables future enhancements of our approach.

The role service is provided by a role server, which is an implementation of the RBAC and *Hive* components. A role server maintains RBAC database and provides user credentials, role memberships, associated permissions, and delegation relations of the system. The rule service is provided by a rule server, which manages delegation and revocation rules. These rules are always associated with a role, which specifies the role that can be delegated. They are implemented as authorization policies that authorize requests from users. The delegation agent is an administrative infrastructure, which authorizes delegation and revocation requests from users by applying derivation authorization rules and processes delegation and revocation transactions on behalf of users.

The implementation requirements related to the delegation framework are not only a delegation agent, but also authentication and access control agents. The authentication agent is used to authenticate users during their initial sign-on and supply them with an initial set of credentials. The reference monitor makes access control decisions based on information supplied by the access control agent. In large role-based system, there may be tens or hundreds of delegation and revocation rules. The rule editor is developed to simplify the management of these rules. As a portion of an integrated RBAC administration platform to manage various RBAC and *Hive* components, the rule editor is used to view, create, edit, and delete delegation and revocation rules.

3. Related works

Pervasive computing enables users to access services and information at anytime and anywhere. Under such computing environments, the information sharing tends to be very dynamic and often ad hoc. Hence, the traditional management approach is not appropriate to such environments because the workload on a security officer (or a small group of security officers) will be overwhelming. Since the very goal of our research is to enable users to access and selectively share resources in distributed systems, we assume that users can be trusted to exercise their discretions on resources: if Alice explicitly shares a resource with Bob, she trusts Bob to use the resource.

We also consider enhancing the scalability of information sharing. Some of the projects that address this issue are UC Baltimore's eBiquity (Kagal et al., 2001), Pebbles project (Myers et al., 1998), UC Berkely's Ninja project (Goldberg et al., 1999; Gribble et al., 2001), Stanford's Interactive Workspace Project (Candea and Fox, 2000), and Matt Blaze's PolicyMaker (Blaze et al., 1996) with the notion of Role-Based Access Control (RBAC). We have found that delegation is one of

promising approaches to rectify the above-mentioned issue. There are many definitions and different types of delegation in the literature (Abadi et al., 1993; Barka and Sandhu, 2000a; Gasser and McDermott, 1990). In general, it is referred to as the process whereby one active entity in a system authorizes another entity to act on behalf of the former by transferring a set of rights. Through delegation, individual user is trusted and empowered to share resources to which they have access. Godo et al. (2003) proposed a multi-agent system to help in the revision of medical prescriptions containing antibiotics of restricted use. The proposed approach attaches an agent to each patient which is responsible of checking different medical aspects associated with his prescribed therapy. Although the monitoring system may allow the use of agents and roles in healthcare organizations, their approach does not address how secure information sharing can be occurred among distributed domains through an innovative access control mechanism.

3.1. Privilege management infrastructure

PMI is based on the ITU-T Recommendation X.509 (2001) of directory systems specification, which introduced PKI in its earlier version. Public-key certificates are used in PKI while attribute certificates are a central notion of PMI. Public-key certificates are signed and issued by certification authority (CA), while attribute certificates are signed and issued by attribute authority (AA). PMI is to develop an infrastructure for access control management based on attribute certificate framework (Shin et al., 2002). Attribute certificates bind attributes to an entity. The types of attributes that can be bound are role, group, clearance, audit identity, and so on. Attribute certificates have a separate structure from that of public key certificates.

PMI consists of four models: general model, control model, delegation model, and roles model. General and control models are required, whereas roles and delegation models are optional. The general model provides the basic entities which recur in other models. It consists of three foundation entities: the object, the privilege asserter, and the privilege verifier. The control model explains how access control is managed when privilege asserters request services on object. When the privilege asserter requests services by presenting his/her privileges, the privilege verifier makes access control decisions based upon the privilege presented, privilege policies, environmental variables, and object methods. The delegation model handles a situation when privilege delegation is necessary. It introduces two additional components: source of authority (SOA) and other AAs. When delegation is used, SOA assigns privilege to AAs, and AAs delegate privileges to an end-entity privilege asserter. Lastly, PMI roles model also introduces two additional components: role assignment and role specification. Role assignment is to associate privilege asserters with roles, and its binding information is contained in attribute certificate called role assignment attribute certificate. The latter is to associate roles with privileges, and it can be contained in attribute certificate called role-specification attribute certificate or locally configured at a privilege verifier's system.

4. Implementation details

Our implementation leverages *Hive* features and X.509 attribute certificate. We attempt to implement the proof-of-concept prototype implementation of *Hive* on privilege management infrastructure (PMI) (ITU-T Recommendation X.509, 2001). PMI provides certificate-based authorization with attribute certificates while public-key infrastructure (PKI) does certificate-based authentication with public-key certificates, so called identity certificates.

Three components are identified for managing attribute certificates: privilege asserter, privilege verifier, and PMI AA. Two different attribute certificates are employed: role assignment attribute certificate (RAAC) for assigning roles to a user and role specification attribute certificate (RSAC) to assign specific permissions to a role. Our implementation is divided into two components. The first component is to build APIs for both a role-based decision making engine and attribute certificates. Those APIs are the core building blocks for constructing an access control policy server and an attribute certificate server. The second component is to implement each entity integrating with APIs. Some of PMI modules in Shin et al. (2002) were utilized to construct the above-mentioned components.

We also developed an application working as an access control policy server. This application has been developed in C++. An engine for making access control decisions is a major component in this application. After receiving a valid RAAC and requested objects (with operation type) from the server, the engine extracts permissions from the RSAC and checks if the requested object (with operation type) is in the list of permissions. The programming library, called RBAC API, was developed to facilitate such procedures.

The current work is focused on the multimedia information sharing between a server in *Hive* and handheld devices. A user of a handheld device can communicate directly with the server and also send commands to the server application. The communication channel has been supported by Microsoft Windows Sockets (WinSock). The management of multimedia information is handled by Microsoft Windows Media Control Interface (MCI). We also used a library such as Victor Image processing library for the conversion between BMP files to JPEG files. In order to enhance the performance, our application is capable of zipping files in both the server and handheld devices developed by Visual Studio 6.0 (VC++ version 6.0) and Embedded Visual Tools 3.0 (EVC++ 3.0), respectively.

The snapshots in Figs. 5 and 6 illustrate how a client can share resources in a server in *Hive*. To support this feature, our implementation consists of several modules.[1]

Communication channel establishment: The communication link is established through the socket connection. It includes creation and listen modes. The following is a sample method to create a link.

```
Create (UINT nSocketPort = 0,
```

[1]These procedures are performed after the attributes are validated.

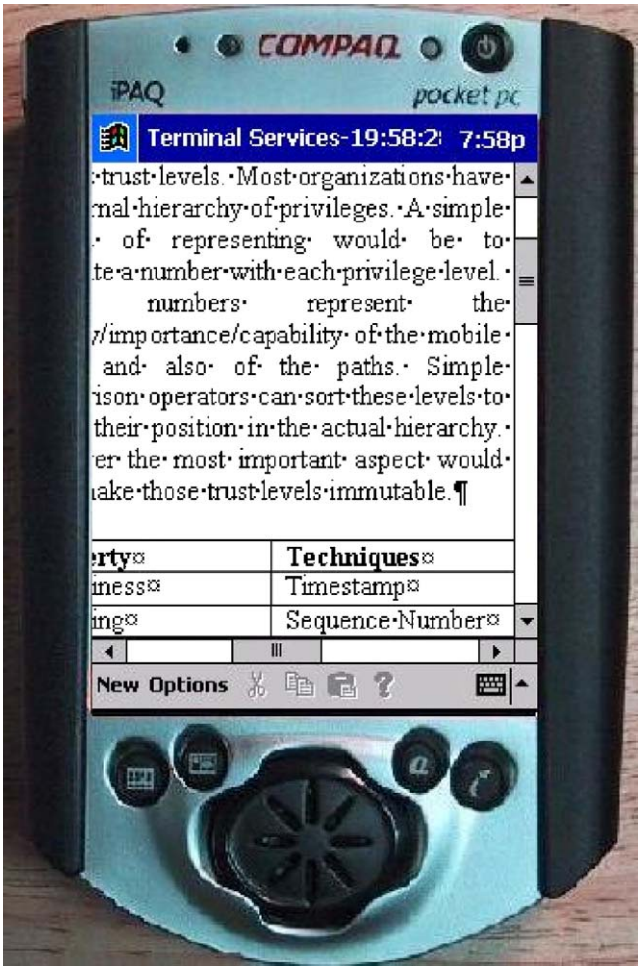

Fig. 5. Accessing information via a handheld device with delegated privilege.

```
  int nSocketType = SOCK_STREAM,
  LPCTSTR lpszSocketAddress = NULL).
```

Controlling messages and commands: Both parties need to exchange messages and a client needs to send commands to a server. The following snippet is an example of this procedure.

```
Int SendTo (const void* lpBuf,
  int nBufLen,
  UINT nHostPort,
  LPCTSTR lpszHostAddress = NULL,
  int nFlags = 0);
```

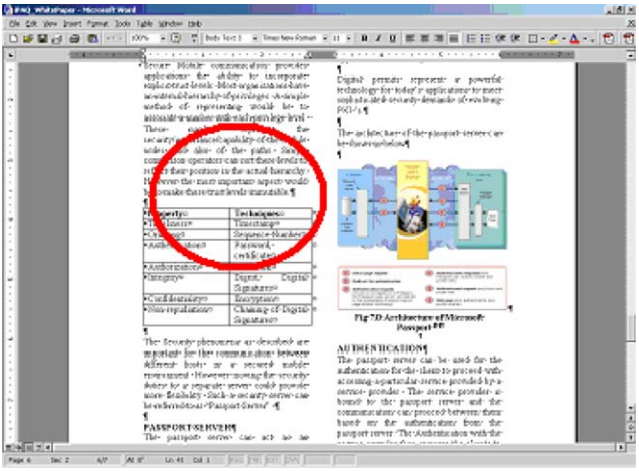

Fig. 6. Sharing information in a server.

```
Int ReceiveFrom (void* lpBuf,
  int nBufLen,
  CString& rSocketAddress,
  UINT& rSocketPort,
  int nFlags = 0);
```

```
If (nCode = =MESSAGE_RBDOWN) {
SetCursorPos (nSrcX, nSrcY);
mouse_event (MOUSEEVENTF_RIGHTDOWN,0,0,0,0);
mouse_event(MOUSEEVENTF_RIGHTUP,0,0,0,0); }
// the mouse_event API //
// for a typical right click //
```

Information sharing between both parties: Once the communication is set up both parties are ready to share information. The video information or screen information is obtained through Windows GDI by creating a temporary Graphics object. The image is copied in the form of a BMP file.[2] A snippet of code is as follows.

```
hbm = ::CreateDIBSection(
  hdc,&bmi,DIB_RGB_COLORS,
  (void**)&imagedata,0,0);
CFile fo(
    Cstring (".\\temp.bmp"),
    CFile::modeCreate|CFile::modeWrite);
Fo.Write(imagedata, bmi.bmiHeader.biSizeImage).
```

[2]The sound can also be captured. The capture of sound is implemented by using Windows MCI calls.

5. Conclusion

Sharing information and resources in collaborative environments entails addressing several requirements not raised by traditional single-user environments in part due to the unpredictability of users and the unexpected manners in which users and applications interact in collaborative sessions. In this paper, we have discussed issues of information sharing introducing an architecture, *Hive*. We also attempted to utilize an existing delegation framework and attribute certificates in PMI. In addition, we demonstrated the feasibility of our architecture through a proof-of-concept implementation. Currently, we are investigating how this technology can be applied to K-12 education environment supporting some of features in The Pittsburgh Pebbles PDA Project, 2004.

Acknowledgements

The work of Gail-J. Ahn and Badrinath Mohan was supported, in part, by funds provided by National Science Foundation (NSF-IIS-0242393 and NSF-CNS-0130799) and Department of Energy Early Career Principal Investigator Award (DE-FG02-03ER25565).

References

Abadi M, Burrows M, Lampson B, Plotkin G. A calculus for access control in distributed systems. ACM Trans Programming Languages Systems 1993;15(4):706–34.

Ahn G-J, Sandhu R. Role-based authorization constraints specification. ACM Trans Inf System Security 2003;3(4):207–26.

Ahn G-J, Zhang L, Shin D, Chu B. Authorization management for role-based collaboration. In: IEEE international conference on system, man and cybernetic (SMC2003); Washington, DC, October 2003. p. 4128–214.

Barka E, Sandhu R. A role-based delegation model and some extensions. In: Proceedings of the 16th annual computer security application conference; Sheraton, New Orleans. December 11–15, 2000a.

Barka E, Sandhu R. Framework for role-based delegation model. In: Proceedings of the 23rd national information systems security conference; Baltimore, MD, October 16–19, 2000b. p. 101–14.

Bertino E, Ferrari E, Atluri V. The specification and enforcement of authorization constraints in workflow management systems. ACM Trans Inf System Security 1999a;2(1):65–104.

Bertino E, Jajodia S, Samarati P. A flexible authorization mechanism for relational data management systems. ACM Trans Inf Systems 1999b;17(2):101–40.

Blaze M, Feigenbaum J, Lacy J. Decentralized trust management. In: Proceedings of the 1996 IEEE symposium on security and privacy; May 1996. p. 164–73.

Candea G, Fox A. Using dynamic mediation to integrate COTS entities in a ubiquitous computing environment. In: Proceedings of the second international symposium on handheld and ubiquitous computing; 2000.

Ferraiolo DF, Barkley JF, Kuhn DR. A role based access control model and reference implementation within a corporate intranet. ACM Trans Inf System Security 1999;2(1).

Gasser M, McDermott E. An architecture for practical delegation in a distributed system. In: Proceedings of the IEEE computer society symposium on research in security and privacy; Oakland, CA, May 7–9, 1990.

Godo L, Puyol-Gruart J, Sabater J, Torra V, Barrufet P, Fàbregas X. A multi-agent system approach for monitoring the prescription of restricted use of antibiotics. Artif Intell Med 2003;27(3):259–82.

Goldberg I, Gribble SD, Wagner D, Brewer EA. The Ninja jukebox. In: Proceedings of the second USENIX symposium on internet technologies and systems (USITS-99); 1999.

Gribble, SD, et al. The Ninja architecture for robust Internet-scale systems and services. Comput Networks 2001.

Hagstrom A, Jajodia S, Presicce FP, Wijesekera D. Revocations—a classification. In: Proceedings of the 14th IEEE computer security foundations workshop; Nova Scotia, Canada, June 2001. p. 44–58.

Kagal L, Finin T, Joshi A. Trust-based security in pervasive computing environments. IEEE Comput 2001. p. 2–5.

Lupu E, Sloman M. A policy based role object model. In: Proceedings of the first IEEE international enterprise distributed object computing workshop (EDOC'97); 1997.

Myers BA, Stiel H, Gargiulo R. Collaboration using multiple PDAs connected to a PC. In: Proceedings of the CSCW'98: ACM conference on computer-supported cooperative work; Seattle, WA, November 14–18, 1998. p. 285–294.

The Pittsburgh Pebbles PDA Project. Available at www.cs.cmu.edu/~pebbles; 2004

Sandhu RS, Coyne EJ, Feinstein HL, Youman CE. Role-based access control models. IEEE Comput 1996;29(2):38–47.

Sandhu R, Bhamidipati V, Munawer Q. The ARBAC97 model for role-based administration of roles. ACM Trans Inf System Security 1999;2(1).

Shin D, Ahn G-J, Cho S. Role-based EAM using X.509 attribute certificate. In: Proceedings of the sixteenth annual IFIP WG 11.3 working conference on data and application security; King's College, University of Cambridge, UK, July 29–31, 2002. p. 285–94.

Zhang L, Ahn G-J, Chu B. A rule-based framework for role-based delegation and revocation. ACM Trans Inf System Security 2003;6(3):404–41.

ITU-T Recommendation X.509. Information technology: open systems interconnection—the directory: public-key and attribute certificate frameworks, ISO/IEC 9594-8, 2001.

Gail-Joon Ahn is an Associate Professor of Software and Information Systems Department at University of North Carolina at Charlotte and a coordinator of Laboratory of Information Integration, Security and Privacy (LIISP) which has been designated as a National Center of Academic Excellence in Information Assurance Education by National Security Agency and Department of Homeland Security. Ahn received PhD and MS degrees from George Mason University, Fairfax, Virginia, and BS degree in Computer Science from SoongSil University, Seoul, Korea. His principal research and teaching interests are in information and systems security. His research foci include vulnerability and risk management, access control, and security architecture for distributed systems. Ahn is currently an information director of ACM Special Interest Group on Security, Audit and Control (SIGSAC) and is a recipient of Department of Energy Early Career Principal Investigator Award.

Badrinath Mohan received his MS degree from University of North Carolina at Charlotte (UNC Charlotte) and was an active research assistant of LIISP at UNC Charlotte. His research interests include access control, identity management and embedded systems.

Seung-Phil Hong received his PhD degree from Information and Communications University, Taejon, Korea. Also, he received BS and MS degrees from Indiana State University and Ball State University, respectively. He worked for the Research and Development Center in LG- CNS Co, Ltd. from 1997–2004, and was actively involved in research in information security. His research interests include access control, security architecture, PKI, and e-buisness security.