domRBAC: An Access Control Model for Modern Collaborative Systems

Antonios Gouglidis*, Ioannis Mavridis

Department of Applied Informatics, University of Macedonia, 156 Egnatia Str., 54006, Thessaloniki, Greece.

Abstract

Modern collaborative systems such as the Grid computing paradigm are capable of providing resource sharing between users and platforms. These collaborations need to be done in a transparent way among the participants of a virtual organization (VO). A VO may consist of hundreds of users and heterogeneous resources. In order to have a successful collaboration, a list of vital importance requirements should be fulfilled, viz. collaboration among domains, to ensure a secure environment during a collaboration, the ability to enforce usage constraints upon resources, and to manage the security policies in an easy and efficient way. In this article, we propose an enhanced role based access control model entitled domRBAC for collaborative applications, which is based on the ANSI INCITS 359-2004 access control model. The domRBAC is capable of differentiating the security policies that need to be enforced in each domain and to support collaboration under secure inter-operation. Cardinality constraints along with context information are incorporated to provide the ability of applying simple usage management of resources for the first time in a role-based access control model. Furthermore, secure inter-operation is assured among collaborating domains during role assignment automatically and in real-time. Yet, domRBAC, as an RBAC approach, intrinsically inherits all of its virtues such as ease of management, and separation of duty relationships with the latter also being supported in multiple domains. As a proof of concept, we implement a simulator based on

*Corresponding author. University of Macedonia, Thessaloniki, Greece.

Email addresses: agougl@uom.gr (Antonios Gouglidis), mavridis@uom.gr (Ioannis Mavridis)

URL: http://users.uom.gr/~agougl/ (Antonios Gouglidis)

the definitions of our proposed access control model and conduct experimental studies to demonstrate the feasibility and performance of our approach.

Keywords: Access control, Cross-domain authorization, Grid computing, RBAC, Resource usage management, Secure inter-operation

1. Introduction

Modern collaborative systems are becoming the *de facto* platform for the implementation of applications. These applications may vary in nature and are capable of solving different types of problem sets posed from either the scientific community or the business sector. Nevertheless, in most cases, the need for excessive processing power and large storage space is required. In collaborative systems, as the Grid, the notion of users, resources, and of virtual organizations (VOs) play an important role. To this effect, we explicitly set the following definitions, mainly based on (Benantar, 2005; Chakrabarti, 2007; Ferraiolo et al., 2003; Foster & Tuecke, 2005; Sandhu & Samarati, 1994). A user in a Grid environment can be a set of user identifiers or a set of invoked services that can perform on request one or more operations on a set of resources. A resource in a Grid environment can be any sharable hardware or software asset in a domain and upon which an operation can be performed. Lastly, a domain can be defined as a protected computing environment, consisted of users and resources under an access control policy. The collaboration that can be established among domains leads to the formation of a virtual organization. Yet, as in all types of computing systems, the role of access control in the Grid is to control and limit the actions or operations that are performed by a user on a set of resources. In brief, it enforces the access control policy of the system, and at the same time it prevents the access policy from subversion. Access control in the literature is also referred to as access authorization or simply authorization. A Grid access control policy can be defined as a Grid security requirement that specifies how a user may access a specific resource and when. Such a policy can be enforced in a Grid system through an access control mechanism. The latter is responsible for granting or denying a user access upon a resource. Therefore, an access control model can be defined as an abstract container of a collection of access control mechanism implementations, which is capable of preserving support for the reasoning of the system policies through a conceptual framework.

In this article we propose the domRBAC model, which is an access control model designed to enforce access control under secure inter-operation in collaborative systems, as the Grid computing paradigm. The proposed model is based on the domRBAC model defined in (Gouglidis & Mavridis, 2011). Specifically, the domRBAC access control model is redefined in this paper regarding the hierarchical domRBAC and role inheritance management. The updated version of domRBAC is making use of more efficient algorithms and data structures in the process of identifying a violation. Yet, it defines a new function for preventing privilege escalation, which can be caused by a new inter-domain role assignment. Moreover, since the definition of domRBAC is based on the ANSI INCITS 359-2004 standard (ANSI, 2004), it also supports all the components of the RBAC model, namely the core RBAC, hierarchical RBAC, static separation of duty relations, and dynamic separation of duty relations. The domRBAC access control model is defined in such way to be both secure and efficient in order to cope with the requirements posed by modern collaborative computing systems. Such functionality includes along with the foregoing, support for multiple domains and the capability of applying simple usage management policies upon resources for the first time in a role based approach. By the term of usage management we refer to the management of the usage of resources across and within domains (Jamkhedkar et al., 2010).

The remainder of this paper is organized as follows: Section 2 provides information about related research and presents our motivation. Section 3 provides sufficient details regarding the formal definition of domRBAC, and section 4 provides implementation aspects concerning the former definitions. Section 5 presents a prototype simulator that we have implemented according to our access control model definitions and details some experimental results. In section 6 we compare the proposed access control model with existing solutions, where applicable. Finally, section 7 summarizes this article.

2. Related Work and Motivation

In collaborative systems like the Grid computing paradigm, various access control models are implemented into mechanisms in order to preserve support for the reasoning of the system's policies. The most mainstream access control models are based on two models; viz. role-based access control (RBAC) and attribute-based access control (ABAC), which is mainly introduced to overcome a number of RBACs shortcomings (Yuan & Tong, 2005). However, the latter type of access control models lack in most cases of a proper formal definition, apart from the usage control model ($UCON_{ABC}$) (Sandhu & Park, 2003), which encompasses traditional access control, trust management and digital rights management for the protection of digital resources.

The RBAC access control model has received considerable attention from researchers, mainly due to its abstraction and generalization. It is abstract because it includes only properties that are relevant to security, and it is general since it supports various designs that can all be interpreted as valid ones. More of RBACs virtues are the support of a significant number of principles, namely the least privilege, separation of administrative functions and separation of duties (Sandhu et al., 1996). Furthermore, RBAC is capable of supporting constraints, as the static and dynamic separation of duty relationships, and last but not least it can enforce role based administration. However, RBAC lacks decentralized management of policies and it cannot support any type of usage management.

In order for RBAC to overcome some of its limitations regarding collaboration, a number of solutions were proposed that had to do with secure inter-operation, which is capable of tackling this issue. Research in (Shafiq et al., 2005) proposed an integer programming (IP)-based approach for optimal resolution of the examined conflicts. A policy integration framework is used for the merging of the individual RBAC policies into a global policy. However, this approach is not dynamic since the global policy is not a result of an incremental composition of the inter-domain policies. In (Chen & Crampton, 2007) an inter-domain role-mapping approach based on the least privilege principle is suggested. Yet, the applied greedy algorithm may not compute optimal solutions, and from a security perspective may fail to find a safe solution. Research in (Shehab et al., 2005) presents a protocol for secure inter-operation, which is based on the idea of access paths and access path's constraints. Nonetheless, the protocol does not check for violations during an inter-domain role assignment. Rather, it assumes that inter-domain role mappings already exist. In (Zhang & Parashar, 2004) the DRBAC is presented as a dynamic context-aware access control model for Grid applications. However, the management of inter-domain policies is not tackled. Nevertheless, resource usage management, to the best of our knowledge, is completely absent from existing RBAC-based models.

Attribute based access control (ABAC) has lately gained a lot of attention due to the development of Internet based distributed systems. However, in contrast to RBAC, attribute based access control, as already stated, has not been standardized yet. The only exception is the $UCON_{ABC}$ model. In such access control approaches, access decisions are provided on resources based on the requesters owned attributes. The advantage of these approaches is that it is possible to provide access to users in a collaborative environment without the need for them to be known by the resource *a priori*. Thus, they natively support distributed access control and collaboration among domains. $UCON_{ABC}$ is the only one that is capable of enforcing usage control. Nonetheless, functionalities such as administration and delegation are still absent. Another ABAC approach that gained considerable attention is the eXtensible Access Control Markup Language (XACML) that is an OASIS standard (OASIS, 2011). In XACML, the access control policies are specified in an XML-based language and exchanged among systems over the Web. A basic characteristic of XACML is its ability to support access control in a interoperable and flexible way. However, there are open world scenarios that XACML is unable to support. A number of XACML's shortcomings along with a number of enhancements to be made to the standard are presented in (Ardagna et al., 2011).

In Grid systems, the existence of various access control models, inevitably led to the implementation of different Grid authorization mechanisms. Additionally, each mechanism tried to further implement features not intrinsically supported by the implemented model (e.g. support of inter-domain collaborations, quality of service and so on). Representative authorization mechanisms in Grid systems are the Community Authorization Service (CAS) (Pearlman et al., 2002), the Virtual Organization Membership Service (VOMS) (Alfieri et al., 2004), Akenti (Thompson et al., 2003), PERMIS (Chadwick et al., 2003; Chadwick, 2005), and Usage Based Authorization (Zhang et al., 2006). Regarding mobile Grid systems, there are various architectures that have been proposed to provide solutions, as the virtual cluster approach in (Phan et al., 2002), the mobile OGSI.NET (Chu & Humphrey, 2004) and the Akogrimo project (Racz et al., 2007). Yet, the proposed authorization mechanisms are complementary to existing Grid authorization services, as the A4C infrastructure in Akogrimo.

In collaborative systems as the Grid, we identify a list of non-functional access control requirements that must be fulfilled. These are, as identified in (Gouglidis & Mavridis, 2012), the support of interoperability among participating domains, the existence of a secure collaborative environment, the support of resource usage management and ease of policy management. After an analysis of existing access control approaches, namely the RBAC and ABAC, it is concluded that the aforementioned cannot fully support all the requirements that are posed by modern collaborative systems. More specifically, it is realized that neither of them can tackle the difficulties risen by the defined Grid access control requirements flawlessly. Yet, the implementation of the access control approaches into a Grid authorization mechanism has the same level of applicability in Grid applications. This means that also the mechanisms cannot handle well the defined requirements of modern collaborative systems. This is mainly a result of applying general purpose access control models that are not specifically designed to tackle the requirements of such systems. Hence, motivated by the absence of an access control model able to fulfill the requirements of modern collaborative systems, as the Grid, we further proceed with the definition of a new access control model.

3. The domRBAC Access Control Model

We define our proposed access control model as an enhancement of the ANSI INCITS 359-2004 (ANSI, 2004). Although the ANSI standard cannot originally support the extra functionality required by modern collaborative systems, it provides a solid background for a new role based access control model. This section discusses the domRBAC model in a systematic manner, by providing all the required modifications and additions in the formal definitions of its base model.

3.1. Elements

The domRBAC model consists of the following six basic elements: users, roles, sessions, operations, objects, and containers. A basic difference between the ANSI INCITS 359-2004 and domRBAC is that the latter can support access control among participating domains. A domain can be defined as a protected computing environment, consisted of users and resources/objects under an access control policy. Such a functionality is of vital importance since it governs inter-operations among domains. Figure 1 illustrates the proposed access control model. Henceforth, we use the terms object and resource interchangeably.

Sessions, objects and operations are three concepts that are commonly used in access control. The latter two form a new element of permissions. A permission or a privilege is an approval to perform an operation on one or more RBAC protected objects. In domRBAC, the aforementioned elements provide the same functionality in their familiar sense. As in all role-based models, sessions are dynamic elements. They are used as intermediary entities between the users and roles elements. The user element usually depicts a physical person who interfaces with a computer system. User elements, in role-based models, are assigned to role elements and vice-versa. Sessions, in role-based models, are used to enforce dynamic security policies to computing systems. Each user can be associated with many sessions, and each session may have a combination of many active roles. In regard to objects, they are used to represent an entity in a computing system. Control of access to objects can be coarse-grained or fine-grained, depending on the computing system. For instance, the sharing of files and exhaustible system resources can be considered an example of coarse-grained access control. On the contrary, the granting of access in a database on the level of record or field is an example of fine-grained access control. Yet, in domRBAC, an object can be associated with many container elements. The container element is explained in detail later in this section. Lastly, the element of operations provides a set of allowed operations on objects. Both operations and objects are system dependent. This means that different types of operations applies to different objects.

Roles in domRBAC are enriched with the notion of domains, and are expressed in pairs of domains and roles. For the naming of the roles, we use the *DomainRole* notation. Thus, the *Domain* prefix indicates the role's domain name, and the *Role* suffix indicates the name of the role. A formal definition is given later in definition 1.ii. The naming notation is used only for the element of roles. Nonetheless, when assigning users or permissions to roles, it is understood that the former two are also bounded by the role's domain name. Through the role's naming convention, the domRBAC model can distinguish the security policies enforced among the autonomous domains.

The container is an abstract element that incorporates additional decision factors employed by the access decision function. The container can handle both the environment and usage level information. The environment attributes are used to set time constraints, spatial information and so on and so forth. Yet, the usage level attributes can limit the usage of shared resources. The information specified in the container element is based on (Neumann & Strembeck, 2003). Thus, a container attribute can represent a certain property of the environment or usage levels. A container function provides a mechanism to obtain the current value of a specific container attribute. Lastly, a container condition is a predicate that compares the current value of a container attribute either with a predefined constant, or another container attribute of the same domain. A significant enhancement of domRBAC, when compared to the ANSI INCITS 359-2004, is that the element of container can support resource usage policies.

Moreover, domRBAC can support additional constraints, namely static and dynamic role cardinality constraints, which can be applied to the process of role assignment and role activation, respectively. This means that the number of roles that can be assigned to and/or activated by the users of a system can be managed. The constraint of role cardinality is introduced to fulfill both requirements posed by the system administrators as well as resource owners. Administrators can use static role cardinality to limit the assignment of critical roles with users. Furthermore, dynamic role cardinality can be used for setting quality of service rules. Resource owners can manage the usage of their resources by limiting the number of users that utilizes them. Thus, it is feasible to create license agreements between users and resource owners. This leads users to receive high quality services in a computing system.

Furthermore, the domRBAC model supports the identification of inter-domain violations, in an automated way. The inter-domain violations are caused due to new immediate inter-domain role inheritance relations. The supported violations are: cyclic inheritance, privilege escalation, violation of static separation of duty relations in a domain, and violation of dynamic separation of duty relations in a domain. The domRBAC model is designed to preserve the security principle among collaborators. Nevertheless, the autonomy of a domain may be willing to be compromised (Shafiq et al., 2005). In the rest of this section, all formal definitions are given in the Z formal description language (ISO/IEC-13568, 2002), as it also happens in the ANSI INCITS 359-2004 standard.

3.2. Definitions

3.2.1. Definition 1. Core domRBAC.

The formal definition of core domRBAC model, based on (ANSI, 2004), is extended as follows:

i. USERS, ROLES, OPS, OBS, CNTRS, stands for users, roles, operations, objects and containers, respectively.
ii. $d_{domain}r_{role} \in$ ROLES is a role expressed in a DomainRole format, where Domain denotes a domain name and Role denotes a role name. For example, if a role r_m belongs to a domain d_i, we write $d_i r_m$.

iii. UA $\subseteq$ USERS $\times$ ROLES, a many-to-many mapping user-to-role assignment relation.

iv. assigned_users(d_ir_m:ROLES) $\rightarrow 2^{USERS}$, the mapping of role d_ir_m onto a set of users.
Formal definition: assigned_users(d_ir_m) = {u $\in$ USERS | (u,d_ir_m) $\in$ UA}.

v. PRMS = $2^{(OPS \times OBS)}$, the set of permissions.

vi. PA $\subseteq$ PRMS $\times$ ROLES, a many-to-many mapping permission-to-role assignment relation.

vii. assigned_permissions(d_ir_m:ROLES) $\rightarrow 2^{PRMS}$, the mapping of role d_ir_m onto a set of permissions.
Formal definition: assigned_permissions(d_ir_m)={p $\in$ PRMS | (p,d_ir_m) $\in$ PA}.

viii. CA $\subseteq$ CNTRS $\times$ OBS, a many-to-many mapping container-to-object assignment relation.

ix. assigned_containers(o: OBS) $\rightarrow 2^{CNTRS}$, the mapping of object o onto a set of containers.
Formal definition: assigned_containers(o)={c $\in$ CNTRS | (c,o) $\in$ CA}.

x. Op(p: PRMS) $\rightarrow \{op \subseteq OPS\}$, the permission to operation mapping, which gives the set of operations associated with permission p.

xi. Ob(p: PRMS) $\rightarrow \{ob \subseteq OBS\}$, the permission to object mapping, which gives the set of objects associated with permission p.

xii. SESSIONS = the set of sessions.

xiii. session_user(s: SESSIONS) $\rightarrow$ USERS, the mapping of session s onto a corresponding user.

xiv. session_roles(s: SESSIONS) $\rightarrow 2^{ROLES}$, the mapping of session s onto a set of roles.
Formal definition: session_roles(s) $\subseteq$ {d_ir_m $\in$ ROLES | (session_user(s),d_ir_m) $\in$ UA}.

xv. avail_session_perms(s: SESSIONS) $\rightarrow 2^{PRMS}$, the permissions available to a user in a session = $\bigcup_{d_ir_m \in session\_roles(s)} assigned\_permissions(d_ir_m)$.

3.2.2. Definition 2. Hierarchical domRBAC.

The hierarchical domRBAC is defined to cope with inter-domain role inheritance relations. Henceforth, we use i and j to refer to domains, where $i = j$ if we refer to an intra-domain relation, and $i \overset{\pm}{=} j$ if we refer to inter-domain relations ($intra - domain \subseteq inter - domain$).

i. $RH \subseteq ROLES \times ROLES$ is a partial order on $ROLES$ called the inheritance relation, written as $\geq$, where $d_i r_m \geq d_j r_n$ only if all permissions of $d_j r_n$ are also permissions of $d_i r_m$, and all users of $d_i r_m$ are also users of $d_j r_n$. Formal definition: $d_i r_m \geq d_j r_n \Rightarrow$ $authorized\_permissions(d_j r_n) \subseteq authorized\_permissions(d_i r_m) \wedge$ $authorized\_users_{(i,j)}(d_i r_m) \subseteq authorized\_users_{(i,j)}(d_j r_n)$.

ii. $authorized\_users_{(i,j)}(d_i r_m : ROLES) \rightarrow 2^{USERS}$, the mapping of role $d_i r_m$ onto a set of users in the presence of a role hierarchy. Formal definition: $authorized\_users_{(i,j)}(d_i r_m) = \{u \in USERS | d_j r_n \geq d_i r_m \wedge (u, d_j r_n) \in UA\}$.

iii. $authorized\_permissions_{(i,j)}(d_i r_m : ROLES) \rightarrow 2^{PRMS}$, the mapping of role $d_i r_m$ onto a set of permissions in the presence of a role hierarchy. Formal definition: $authorized\_permissions_{(i,j)}(d_i r_m) = \{p \in PRMS | d_i r_m \geq d_j r_n \wedge (p, d_j r_n) \in PA\}$.

Definition 2.iii, in domRBAC, is based on the corrected formal definition of *authorized_permissions*, as this is identified in (Li et al., 2007).

3.2.3. Definition 3. Constrained domRBAC.

Separation of duty (SoD) is a fundamental security principle that is supported in domRBAC. SoD serves as a requirement for critical operations, which are divided among two or more people, so that no single individual can compromise security. SoD methods are categorized into two broad categories, namely that of static and dynamic. Static SoD (SSD) are constraints that are placed on roles at the time roles are assigned to users. SSD are further defined in the presence of a hierarchy, where it works in the same way as in the latter case except that when enforcing the constraints both inherited roles and directly assigned roles are considered. Dynamic SoD (DSD) are constraints that are invoked when users are using the system to activate already assigned roles (Ferraiolo et al., 2003).

Apart from the support of static and dynamic separation of duty constraints in each domain, domRBAC supports static and dynamic role cardinality constraints. Static role cardinality constraints can restrict the number of users assigned to a role, to a maximum number. Moreover, dynamic role cardinality constraints can restrict the number of users that activate a role, to a maximum number in all concurrent sessions. In the following, we redefine SSD and DSD in the presence of domains, and we define static and dynamic role cardinality.

i. **Static Separation of duty (SSD):** SSD $\subseteq (2^{ROLES} \times N)$ is a collection of pairs (d_irs,n) in SSD, where each d_irs is a role set in a domain d_i, t a subset of roles in d_irs, and n is a natural number ≥2, with the property that no user of domain d_i is assigned to n or more roles from the set d_irs in each (d_irs,n) ∈ SSD.
Formal definition:
$\forall (d_irs, n) \in SSD, \forall t \subseteq d_irs : |t| \geq n$
$\Rightarrow \bigcap_{d_ir_m \in t} assigned\_users(d_ir_m) = \emptyset.$

ii. **SSD in the presence of a hierarchy:** In the presence of a role hierarchy SSD is redefined based on authorized users rather than assigned users as follows:
Formal definition:
$\forall (d_irs, n) \in SSD, i = j, \forall t \subseteq d_irs : |t| \geq n$
$\Rightarrow \bigcap_{d_ir_m \in t} authorized\_users_{(i,j)}(d_ir_m) = \emptyset.$

iii. **Dynamic Separation of Duty (DSD):** DSD $\subseteq (2^{ROLES} \times N)$ is a collection of pairs (d_irs,n) in DSD, where each d_irs is a role set and n a natural number ≥2, with the property that no subject may activate n or more roles from the set d_irs in each dsd ∈ DSD.
Formal definition:
$\forall d_irs \in 2^{ROLES}, n \in \mathbb{N}, (d_irs, n) \in DSD \Rightarrow n \geq 2.|d_irs| \geq n$, and
$\forall s \in SESSIONS, \forall d_irs \in 2^{ROLES},$
$\forall role\_subset \in 2^{ROLES}, \forall n \in \mathbb{N}, (d_irs, n) \in DSD,$
$role\_subset \subseteq d_irs, role\_subset \subseteq session\_roles(s)$
$\Rightarrow |role\_subset| < n.$

iv. **Static role cardinality (SRC):** If static role cardinality constraint is required for any role d_ir_m, then d_ir_m cannot be assigned to more than a maximum number of users.
SRC $\subseteq (ROLES \times N)$ is a collection of pairs (d_ir_m, n) in static role cardinality, where d_ir_m is a role r_m in a domain d_i and n is a natural number ≥ 0, with the property that the number of users assigned with role d_ir_m cannot exceed the number n in each (d_ir_m, n) ∈ SRC.
Formal definition:
$d_ir_m \in ROLES, n \in \mathbb{N}, n \geq 0,$
$\forall (d_ir_m, n) \in SRC \Rightarrow |assigned\_users(d_ir_m)| \leq n.$

v. **SRC in the presence of a hierarchy:** In the presence of a role hierarchy static role cardinality constraint is redefined based on authorized users rather than assigned users as follows:

$d_ir_m \in ROLES, i \stackrel{\pm}{=} j, n \in \mathbb{N}, n \geq 0,$
$\forall (d_ir_m, n) \in SRC \Rightarrow |authorized\_users_{(i,j)}(d_ir_m)| \leq n.$

vi. **Dynamic role cardinality constraint (DRC):** If dynamic role cardinality is required for any role d_ir_m, then d_ir_m cannot be activated for more than a maximum number of authorized users in all concurrent sessions of a system.
DRC $\subseteq$ ($ROLES \times N$) is a collection of pairs (d_ir_m, n) in dynamic role cardinality, where d_ir_m is a role r_m and n is a natural number ≥ 0, with the property that the number of concurrent role activations by users authorized for role d_ir_m cannot exceed the number n.
Formal definition:
$d_ir_m \in ROLES, n \in \mathbb{N}, n \geq 0,$
$\forall s \in$ SESSIONS, (d_ir_m,n) $\in$ DRC $\Rightarrow \sum |d_ir_m \cap session\_roles(s)| \leq n.$

After defining both the container element and the DRC constraint, we elaborate on the supported types of resource usage policies. The first type is via the container element, by declaring the required attribute value, function and condition of the container. However, this type of resource usage policy is unable to provide quality of service to consumers since each container element restricts the usage of a resource on per role activation. A second type of resource usage policy with quality of service capabilities is provided via the combination of the container element and DRC constraint. This type of resource usage policy enforcement restricts the usage of a resource on all concurrent role activations.

3.2.4. Definition 4. Role Inheritance Management.

The domRBAC model aims at providing a comprehensive solution to secure inter-operation based on the principles of security, autonomy and containment (Ravi Sandhu, 2008). In order to establish a secure inter-operation among the participating domains, domRBAC provides two new administrative commands for managing inter-domain role inheritance relations. The administrative commands can be used by the administrator of each domain, according to the interoperability requirements of each system. Their objective is to check for a number of violations before committing an inter-domain role inheritance relation. Thus, based on the definitions 4.i, 4.ii, 4.iii and 4.iv, we introduce the *InterdomainPolicyViolation* function for the checking of inter-domain violations due to the inter-domain role inheritance relations, and two new inter-domain administrative commands *AddInterdomainInheri-*

tance and *DeleteInterdomainInheritance* for establishing and discarding immediate inter-domain inheritance relationships, respectively. Intra-domain management not listed below is handled the same as in the ANSI INCITS 359-2004 standard.

i. **Intra-domain violation of role assignment:** As stated in (Shafiq et al., 2005) an inter-domain policy causes a violation of role assignment constraint of domain d_i if it is allowed to a user u of domain d_i to access a local role d_ir_m even though u is not directly assigned to d_ir_m or any of the roles that are senior to d_ir_m in the role hierarchy of domain d_i. We identify role assignment violations by checking for cyclic inheritance in the inter-domain role hierarchy graph. Role assignment violations can occur due to the addition of a new immediate inter-domain inheritance relationship $d_ir_{m_{asc}} \gg d_jr_{n_{desc}}$ between existing roles $d_ir_{m_{asc}}$, $d_jr_{n_{desc}}$, where $d_ir_{m_{asc}}$ is a role ascendant of $d_jr_{n_{desc}}$.

ii. **Privilege escalation:** Apart from cycle inheritance, we identify another case that may lead to privilege escalation due to a new inter-domain role assignment. The applied methodology ensures that the principle of security is preserved during the collaboration at the cost of reducing inter-operation. The security principle ensures that if an access is not permitted within an individual domain, it must not be permitted under secure inter-operation.

iii. **Intra-domain violation of SSD relationships:** An inter-domain policy causes an intra-domain violation of SSD relationships of domain d_i if it is allowed to a user u of domain d_i to be assigned to any two conflicting roles d_ir_m and d_ir_n of domain d_i. We identify violations of SSD relationships, using the following properties (Ferraiolo et al., 2003):
Property 1: If there are two roles d_ir_m and d_jr_n that are mutually exclusive, then neither one should inherit the other, either directly or indirectly.
Property 2: If there are two roles d_ir_m and d_jr_n that are mutually exclusive, then there can be no third role that inherits both of them.

iv. **Intra-domain violation of DSD relationships:** An inter-domain policy causes an intra-domain violation of DSD relationships of domain d_i if it is allowed to a user u of domain d_i to activate any two conflicting roles d_ir_m and d_ir_n of domain d_i. We identify violations of DSD relationships similarly to definition 4.iii due to the following property (Ferraiolo et al., 2003):

Property 3: If SSD holds, then DSD is maintained. Thus, properties 1 and 2 must be guaranteed.

v. **InterdomainPolicyViolation:** This function checks if any of the aforementioned violations occur during the creation of a new inter-domain role assignment. Hence, it returns *true* if a violation occurs and *false* otherwise.

vi. **AddInterdomainInheritance:** This command establishes a new immediate inter-domain inheritance relationship $d_i r_{m_{asc}} \gg d_j r_{n_{desc}}$ between existing roles $d_i r_{m_{asc}}$, $d_j r_{n_{desc}}$. The command is valid if and only if $d_i r_{m_{asc}}$ and $d_j r_{n_{desc}}$ are members of the $ROLES$ dataset, $d_i r_{m_{asc}}$ is not an immediate ascendant of $d_j r_{n_{desc}}$, and violations of role assignment and of SSD and DSD relationships do not occur.
Formal definition:
$AddInterdomainInheritance(d_i r_{m_{asc}}, d_j r_{n_{desc}} : NAME) \lhd$
$d_i r_{m_{asc}}, d_j r_{n_{desc}} \in ROLES;$
$InterdomainPolicyViolation(d_j r_{n_{desc}}) = false;$
$\neg(d_i r_{m_{asc}} \gg d_j r_{n_{desc}}); \neg(d_j r_{n_{desc}} \geq d_i r_{m_{asc}})$
$\geq' = \geq \cup \{dr, dq : ROLES | dr \geq d_i r_{m_{asc}} \wedge d_j r_{n_{desc}} \geq dq \bullet dr \mapsto dq\} \rhd$

vii. **DeleteInterdomainInheritance:** This command deletes an existing immediate inter-domain inheritance relationship $d_i r_{m_{asc}} \gg d_j r_{n_{desc}}$. The command is valid if and only if the roles $d_i r_{m_{asc}}$ and $d_j r_{n_{desc}}$ are members of the $ROLES$ dataset, and $d_i r_{m_{asc}}$ is an immediate ascendant of $d_j r_{n_{desc}}$. The new inter-domain inheritance relation is computed as the reflexive-transitive closure of the immediate inheritance relation resulted after deleting the relationship $d_i r_{m_{asc}} \gg d_j r_{n_{desc}}$.
Formal definition:
$DeleteInterdomainInheritance(d_i r_{m_{asc}}, d_j r_{n_{desc}} : NAME) \lhd$
$d_i r_{m_{asc}}, d_j r_{n_{desc}} \in ROLES; (d_i r_{m_{asc}} \gg d_j r_{n_{desc}})$
$\geq' = (\gg \ \{d_i r_{m_{asc}} \mapsto d_j r_{n_{desc}}\})^* \rhd$

4. Implementation Aspects

In this section a series of implementation aspects of the model are discussed. Our approach utilizes algorithms derived from the theory of graphs. This is done since firstly graphs help in the visualization of inter-domain role inheritance relations, secondly adjacency lists make it easy to find subgraphs and finally adjacency queries are fast. Knowing that role hierarchies are represented as sparse graphs, we choose to use linked lists for their representation instead of matrices since the former is more efficient. A list of implementation aspects follow in the rest of this section.

i. $G = (V, E)$ is the inter-domain role hierarchy directed graph, which consists of a finite, nonempty set of role vertices $V \subseteq ROLES$ and a set of edges E. Each edge is an ordered pair $(d_i r_m, d_j r_n)$, $i \overset{\pm}{=} j$ of role vertices that indicates the following relation: $d_i r_m \geq d_j r_n$.

ii. A path in a G graph is a sequence of edges $(d_i r_1, d_i r_2)$, $(d_i r_2, d_i r_3)$, $\ldots$, $(d_i r_{n-1}, d_i r_n)$. This path is from role vertex $d_i r_1$ to role vertex $d_i r_n$ and has length n-1. The path represents not immediate inheritance relation between role vertex $d_i r_1$ and $d_i r_n$.

iii. An adjacency list representation for graph $G = (V, E)$ is an array L of $|V|$ lists, one for each role vertex in V. For each role vertex $d_i r_m$, there is a pointer $L_{d_i r_m}$ to a linked list containing all the role vertices that are adjacent to $d_i r_m$. A linked list is terminated by a nil pointer. Henceforth, we refer to the adjacency list as A_G. We also set $A_G[d_i r_m, d_j r_n] = 1$ if there is an edge from role vertex $d_i r_m$ to role vertex $d_j r_n$, and $A_G[d_i r_m, d_j r_n] = 0$ otherwise. Despite the fact that the latter notation is mostly used in matrices, we choose to use it also in linked lists for simplicity and readability reasons.

iv. The transitive closure of a graph $G = (V, E)$ is a graph $G^* = (V, E^*)$ such that E^* contains an edge (u, v) if and only if G contains a path (of at least one edge) from u to v. The algorithm used to implement the transitive closure is based on the detection of strong components (Nuutila, 1995; Purdom, 1970), having a worst case time complexity of $O(|V||E|)$. Henceforth, we refer to the transitive closure list of a directed graph $G = (V, E)$ with adjacency list A_G as T_G. Furthermore, we set $T_G[d_i r_m, d_j r_n] = 1$ if there is a path from $d_i r_m$ to $d_j r_n$ of length 1 or more, and 0 otherwise.

In turn, we provide the algorithms that implement definitions 4.i to 4.v.

v. **Intra-domain violation of role assignment:** The algorithm for detecting cycles is given in Figure 2. In short, we iterate onto every vertex $d_i r$ of the transitive closure list, and for each adjacent vertex $d_j r$ to vertex $d_i r$ we check if vertex $d_i r = d_j r$. In such case, $T_G[d_i r, d_j r] = 1$ since there is a path from $d_i r$ to $d_j r$ of length one or more.

vi. **Privilege escalation:** We describe the algorithm that implements the current function using an example. Assume that we have two domains d_1 and d_2, as shown in Figure 3. In domain d_1 we have role d_1r_a that inherits role d_1r_b. Likewise, in domain d_2 we have role d_2r_c that inherits roles d_2r_d and d_2r_e. Let users u_1 and u_2 be assigned to roles d_2r_d and d_2r_e, respectively. At first, we commit an inter-domain role assignment between roles d_2r_d and d_1r_a (d_2r_d inherits d_1r_a). The latter role assignment does not raise any problem to the security nor the autonomy principles. Now, if we try to make a new inter-domain role assignment between d_1r_b and d_2r_e (d_1r_b inherits d_2r_e), user u_1 of domain d_2 can then activate role d_2r_e, even though it was not assigned with him/her at the beginning. Through this process, a user can get more privileges in his/her parent domain. Thus, in domRBAC, we identify such cases and we reject the inter-domain role assignments. In order to identify this kind of privilege escalation cases we work as follows. Assume that we want to make the aforementioned inter-domain role assignment between roles d_1r_b and d_2r_e (d_1r_b inherits d_2r_e). We gradually check for each role in the target domain d_2 if there is a role d_2r_x that may lead its assigned users to gain more privileges in their parent domain. To identify such cases we assume that the initial inter-domain role assignment can be applied and we check for each role d_2r_x, which is in equal or lower depth when compared with role d_2r_e, if $T_G[d_2r_e, d_2r_x] \neq T_{G_{d_2}}[d_2r_e, d_2r_x]$ or $T_G[d_2r_x, d_2r_e] \neq T_{G_{d_2}}[d_2r_x, d_2r_e]$, where $T_{G_{d_2}}$ is the transitive closure list of domain d_2 with all inter-domain role assignments excluded. If yes, then we raise an error and we discard the inter-domain role assignment. Otherwise, we commit the inter-domain role assignment. The algorithm for preserving the security principle is given in Figure 4. For simplicity reasons, we describe the algorithm using the T_G, T_{G_i} notation that was previously defined, instead of describing analytically the iterations onto the list data structures.

vii. **Intra-domain violation of SSD relationships:** The algorithm for detecting intra-domain violations of SSD relationships is given in Figure 5. In more detail, we check for each pair of SSD relationship, if the new role assignment raises an error. In order to do so, we iterate onto the transitive closure list and we firstly check for each SSD pair if *Property 1* is being violated. If yes, a violation has been occurred and the function returns *true*. Otherwise, we continue to check for each SSD pair if *Property 2* is being violated. If yes, a violation has been occurred and the function returns *true*. Otherwise, no violation has been occurred and the function returns *false*. Specifically, *Property 1* is maintained in lines 2-11, and *Property 2* in lines 12-19.

viii. **Intra-domain violation of DSD relationships:** The algorithm for detecting intra-domain violations of DSD relationships is given in Figure 6.

ix. **InterdomainPolicyViolation:** Figure 7 presents the implementation of the function with function parameter $d_i r_{n_{desc}}$ with the latter being a role that can be inherited by any role $d_i r_{n_{asc}}$. It is worthy to mention that the function parameter is only required by the *sp_violation* function. All the other functions check for violations in all domains.

The following example illustrated in Figure 8 shows how the aforementioned functions are used to identify any of the supported violations. Let's assume a multi-domain access control policy that allows collaboration between domain d_1 and domain d_2. Domain d_1 has the following roles: d_1r_a, d_1r_b, d_1r_c, d_1r_d and d_1r_e. Role d_1r_a inherits all permissions of d_1r_b which further inherits d_1r_e. Role d_1r_c inherits all permissions of d_1r_d which further inherits d_1r_e. An SSD relation is specified for d_1r_b and d_1r_c meaning that these roles cannot be assigned to the same user simultaneously. Domain d_2 has the following roles: d_2r_f and d_2r_g. Role d_2r_f inherits all permissions of d_2r_g. The multi-domain access control policy defines the following inter-domain inheritance relationships between domains d_1 and d_2, which are applied in the following chronological order.

(a) Role d_1r_b inherits role d_2r_g.

(b) Role d_2r_g inherits role d_1r_c.

The inter-domain role relationship described in (a) does not raise any of the discussed violations. Regarding the inter-domain role relationship in (b) we work as follows:

Step 1. Assuming that the inter-domain role relationship in (b) can be applied, the required adjacency and transitive closure list representations are constructed, as follows:

$$A_G = \begin{cases} Vertex & Linked\,list \\ d_1r_a: & d_1r_b \rightarrow nil \\ d_1r_b: & d_1r_e \rightarrow d_2r_g \rightarrow nil \\ d_1r_c: & d_1r_d \rightarrow nil \\ d_1r_d: & d_1r_e \rightarrow nil \\ d_1r_e: & nil \\ d_2r_f: & d_2r_g \rightarrow nil \\ d_2r_g: & d_1r_c \rightarrow nil \end{cases} \qquad A_{G_{d_1}} = \begin{cases} Vertex & Linked\,list \\ d_1r_a: & d_1r_b \rightarrow nil \\ d_1r_b: & d_1r_e \rightarrow nil \\ d_1r_c: & d_1r_d \rightarrow nil \\ d_1r_d: & d_1r_e \rightarrow nil \\ d_1r_e: & nil \end{cases}$$

$$T_G = \begin{cases} Vertex & Linked\,list \\ d_1r_a: & d_1r_b \rightarrow d_1r_c \rightarrow d_1r_d \rightarrow d_1r_e \rightarrow d_2r_g \rightarrow nil \\ d_1r_b: & d_1r_c \rightarrow d_1r_d \rightarrow d_1r_e \rightarrow d_2r_g \rightarrow nil \\ d_1r_c: & d_1r_d \rightarrow d_1r_e \rightarrow nil \\ d_1r_d: & d_1r_e \rightarrow nil \\ d_1r_e: & nil \\ d_2r_f: & d_2r_g \rightarrow d_1r_c \rightarrow d_1r_d \rightarrow d_1r_e \rightarrow nil \\ d_2r_g: & d_1r_c \rightarrow d_1r_d \rightarrow d_1r_e \rightarrow nil \end{cases}$$

$$T_{G_{d_1}} = \begin{cases} Vertex & Linked\,list \\ d_1r_a: & d_1r_b \rightarrow d_1r_e \rightarrow nil \\ d_1r_b: & d_1r_e \rightarrow nil \\ d_1r_c: & d_1r_d \rightarrow d_1r_e \rightarrow nil \\ d_1r_d: & d_1r_e \rightarrow nil \\ d_1r_e: & nil \end{cases}$$

Step 2. The $InterdomainPolicyViolation$ function is executed using the d_1r_c as function parameter. In turn, the rest functions are executed, namely the *ci_violation*(), *sp_violation*(d_1r_c), *ssd_violation*(), and *dsd_violation*(). The only existing SSD relationship pair is that of (d_1r_b, d_1r_c). The computations are performed based on the predefined algorithms. As resulted, two different types of violations are identified. The first is identified by the *sp_violation*(d_1r_c) function call. This happens because during the collaboration, role d_1r_a inherits role d_1r_c. However, the latter inheritance relationship did not exist in the initial domain d_1, thus, it may lead to privilege escalation. A second violation is determined by the *ssd_violation*() function, since it identifies that the inter-domain relationship allows to role d_1r_b to access the permissions of role d_1r_c through d_2r_g. This is not permissible, since the former two roles are mutually exclusive. The identification of the two violations will discard the inter-domain inheritance relationship, assumed in the hypothesis of step 1.

Concerning the implementation of the domRBAC model, a number of performance issues had to be solved. Such issues were related to the need for continually re-creating new adjacency and transitive closure lists. Hence, when the re-creation of the adjacency list is required, we update (insert or delete) only the parts of the structure that needs to change, and not the whole adjacency list. In a similar way to the update operation of the adjacency list, there is a need to update the transitive closure, when we add a new edge to a graph, or remove an existing one. To improve the construction time of the transitive closure list, we do not re-create it from scratch. Instead the only new paths we add are the ones which use the new edge (u, v). These paths will be of the form $a \rightsquigarrow u \rightarrow v \rightsquigarrow b$, where $\rightsquigarrow$ is used as a logical connective and interpreted as *leads to*. We can find all these new paths by looking in the old transitive closure for the vertices $a \in A$ which had paths to u and for vertices $b \in B$ which v had paths to. The new edges in the transitive closure will then be (a, b), where $a \in A \bigcup u$ and $b \in B \bigcup v$. If we represent the transitive closure graph G with an adjacency matrix, we can very easily find the sets A and B. A will include all the vertices which have a one (1) in the column u and B will be all the vertices which have a one (1) in the row v. Since the number of vertices in each set A or B is bounded by V, the total number of edges needed to be added is $O(V^2)$ (Carlstrom, 2004). However, since information is stored in lists, the equivalent total number of edges needed to be added becomes significantly lower.

5. Simulation and Experimental Study

This section introduces the overall architecture of the simulator, which implements the part that is responsible for the management of policies since the majority of domRBAC's enhancements are heavily depended on the management of inter-domain policies in real-time. The goal of the simulator is to produce a series of experimental results, so as to check the applicability and efficiency of the domRBAC during collaboration.

5.1. The domRBAC Simulator

The domRBAC simulator is capable of checking security policies and to decide either to commit or reject a policy. Hence, it serves as an Access Control Decision Function (ADF), which together with the Access Control Enforcement Function (AEF) implements the concept of reference monitor (Ferraiolo et al., 2003). An ADF is responsible for the making of access control decisions. The decisions are made based on information applied by the access control policy rules, the context in which the access request is made, and the Access Control Decision Information (ADI) (ITU, 1995). The ADI is a portion in the Access Control Information (ACI) function, which includes any information used for access control purposes, including contextual information. Lastly, the AEF is responsible for the enforcement of the decision taken from the ADF. The concept of reference monitor in open systems has been standardized with the X.812 access control framework (ITU, 1995). The core simulator is implemented in the Debian Linux 2.6.32-5-686 platform using the C++ 4.4.5 programming language and making use of the BOOST 1.42.0 C++ library (Boost, 2011). The graphical user interface is built using the Qt 4.6.3 (Nokia, 2011).

Figure 9 illustrates the overall architecture of the domRBAC simulator. The simulator is capable of reading access control policies from two different types of input files. The first file type is in the XML (W3C, 2011) and the second in the DOT language (Graphviz, 2011). DOT is a plain text graph description language, which can describe in a simple way graphs that both humans and computer programs can use. The XML file is currently using a custom syntax opposed to that of XACML RBAC (OASIS, 2011), for simplicity reasons. Due to the modular design of the simulator, this can be changed in the future. The XML file can be validated along with an XSD file (W3C, 2011) to an external validation engine, in order to check the correctness of the XML file. The content of the XSD file is presented in Appendix A. The XML policies are loaded in the core of the simulator using the SAX streaming API (XML-DEV, 2011), since policies can get too big for the available memory. There is also support for the DOM tree-based API (W3C, 2005), only for the part of the simulator that handles the viewing of the available policies. Both parsers are implemented using the equivalent libraries provided by the Qt. However, since the writing of policies can be cumbersome, the domRBAC simulator is capable of loading files that are described in the DOT language. In order to produce random and of different size input data, we used the NetworkX python package (NetworkX, 2011) for the creation of DOT files. Furthermore, the visualization of security policies is possible through the Graphviz library (Graphviz, 2010). Using a configuration file, where a list of parameters can be defined, the core simulator can start the simulation by creating random role assignments. The latter is performed by the random policy generator, which is responsible for the creation of additional policies on top of the imported access control policies. During the simulation a number of role assignments, SSD and DSD relationships are randomly requested, based on the information described in the configuration file. However, after each request the simulator automatically checks if any type of violation is being raised. If not, it commits the requested role assignment, and otherwise, it rejects the role assignment. The main interface of the simulator is depicted in Figure 10. On the left side of the screen, the loaded access control policies are available in tree-view, and on the right side the access control policies, the adjacency and transitive closure lists are being visualized, each one on a different tab. On the top of the screen there is a menu with all the available options (e.g. check for violations, run simulation and so on), and on the bottom a console logs all the actions of the domRBAC simulator.

Since domRBAC depends heavily on both adjacency and transitive closure lists, we use state-of-the-art functions provided by the BOOST C++ library, in order to construct and compute them. In more detail, we use the *adjacency_list* class that implements a generalized adjacency list graph structure. This is a two-dimensional structure, where each element of the first dimension represents a vertex, and each of the vertices contains a one-dimensional structure that is its edge list. Regarding the computation of the transitive closure we make use of the *transitive_closure()* function, which transforms the input graph g into the transitive closure graph tc. Concerning the update operations in both lists, we update only the required information, as described in section 4.

5.2. Performance Evaluation

As an access control management decision is dynamically determined by checking if any violation occurs during a cross-domain role assignment, the performance of the system should be considered. Using the domRBAC simulator, a list of performance metrics are captured during various stress tests. The metrics are mostly considered with time values and memory consumption. The technical characteristics of the test platform were: 1.6GHz Intel Mobile Pentium processor, 786MB of RAM, and using the Debian Linux

2.6.32-5-686 platform. The simulator was compiled using the *-O3* optimization parameter, in order to increase performance (Free Software Foundation, 2008). The test cases were created using the *gnc_graph* function that returns a growing network with copying (GNC) directed graph, which is built by adding nodes one at a time with a link to one previously added node, chosen uniformly at random, and to all of that nodes successors (NetworkX, 2011). In turn, the simulator loads the role hierarchies of each domain and randomly performs role assignment operations, as well as static and dynamic separation of duty relationships. After each new request of role assignment, the simulator checks if the role assignment will be committed or discarded based on the inter-domain policy violation function and logs the transaction. The latter can be repeated continuously and in real-time, in order to collect performance data.

For the performance evaluation of the proposed access control model, we performed a series of simulations on different data sets. More specifically, we used test cases of 50, 100, 150 and 200 domains containing 100 roles each, called as group 'A' of data, and of 5, 10, 15 and 20 domains containing 1000 roles each, called as group 'B' data. Thus, we created collaborative domains that contained 5000, 10000, 15000 and 20000 roles, respectively. This was done in order to analyze the behavior of the model. For instance, a collaborative system that consists of 5000 roles, in the above-mentioned data sets, can be the result of a combination of 50 domains containing each 100 roles or of 5 domains containing each 1000 roles. Table 1 summarizes the performance of the ADF's memory consumption along with decision time values and a number of violations. This information is the result of a simulation of 5000 random role assignment, SSD and DSD requests. It can be seen in group's 'A' data set that the simulation leads to higher numbers of inter-domain role assignments and to lower numbers of SSD and DSD relationships, in contrast to group's 'B' data set. This behavior is logical and expected to be seen since the low number of roles in a domain have a negative effect regarding the creation of new SSD and DSD relationships. Furthermore, the low number of roles combined with a high number of domains act in favor of creating more inter-domain role assignments. Yet, such a behavior results to a slightly higher memory consumption of approximately 3% on average in the adjacency lists, which is increased to the level of 17% on average in the transitive closure lists. Moreover, the high number of violations depicted in Table 1 is the result of domRBAC's design decisions since domRBAC tries to preserve the security privilege instead of gaining inter-operation. In general, it is not a feasible task to create a global multi-domain policy where inter-operation is allowed among domains without violating the security or the autonomy of a participating domain (Shafiq et al., 2005). However, the violation of a domain's security in not permissible. Nevertheless, the autonomy of a domain may be willing to be compromised. For the overall autonomy loss (OAL) and overall interoperability level (OIL) of the collaborative system we use similar equations, as in (Shafiq et al., 2005). Hence, for calculating the autonomy loss and interoperability level we use the equations 1 and 2, respectively. In 1 the OAL is calculated by subtracting the total number of committed intra-role assignments from the total number of requested intra-role assignments during the simulation period, and dividing the difference by the latter to get the value of OAL. Similarly to OAL, in 2 the OIL is calculated by subtracting the total number of violations occurred due to inter-role assignments from the total number of requested inter-role assignments during the simulation period, and dividing the difference by the latter to get the value of OIL. Using the aforementioned expressions the autonomy loss and inter-operation level reached at 6% and 7.5% in group's 'A' data set and 2% and 8% in group's 'B' data set, respectively.

$$OAL = \frac{(Intra\,role\,assignments) - (Committed\,intra\,role\,assignments)}{(Intra\,role\,assignments)} \tag{1}$$

$$OIL = \frac{(Inter\,role\,assignments) - (Inter\,violations)}{(Inter\,role\,assignments)} \tag{2}$$

Additionally, we present a performance evaluation concerning the time required to compute the data structures used in domRBAC and the time that is required for the ADF to check and identify potential violations. Appendix B.1 and Appendix B.2 include the descriptive statistical measures in detail, viz. mean, median, mode, maximum and standard deviation values, of groups 'A' and 'B', respectively. Figures 11 and 12 illustrate the mean and maximum time values of calculations, respectively. Maximum decision time, in both cases, is calculated as the sum of the adjacency list creation, transitive closure list creation and of a maximum value of cycle inheritance, privilege escalation, SSD and DSD violation. The creation of the transitive closure list presupposes the creation of the adjacency list. Hence, it is required to add both time values. We further add the maximum time value among violations since violation algorithms can be executed concurrently. In case a violation is determined by a thread that implements a function that identifies a violation, the former sends a signal to the other threads to terminate calculations, and it further returns a *true* value to indicate the determination of a violation. The maximum decision time regarding mean values ranges approximately from 2 to 50 milliseconds, and from 120 to 1142 milliseconds in worst cases (maximum values). As shown, the ADF performs better when the collaboration is the result of group's 'A' data set opposed to that of group's 'B' data set. Furthermore, based on the results, it is concluded that the ADF performs with higher values during the identification of SSD and DSD violations in group 'A' data, instead of that in 'B'. In addition, standard deviation shows that dispersion of time values is higher in group 'B' data set. This means that data points are spread out over a large range of values. It is noteworthy that the mode statistical measure in all cases is between 0 to 2 milliseconds and the mean value equal to 0 milliseconds. This is mostly due to the identification of a large number of violations. Regarding the median and maximum time values we can once again observe that are lower in group's 'A' data set. Figure 13 shows in a logarithmic scale a comparison of the mean and maximum decision time values, where both lines show a similar behavior, with the maximum values being 1.6% and 18.5% higher in average opposed to the equivalent mean values, in group 'A' and 'B' data set, respectively.

Based on the analysis of the collected data, the results show that the performance is acceptable for the Grid computing paradigm, as well as for general collaboration requirements.

6. Discussion

Despite the existence of a large number of proposed and implemented access control models, there are only a few, to the best of our knowledge that provide information relevant to their performance. Moreover, performance comparison among access control models seems to be a difficult task since different performance metrics are provided, if any, in each one of them. Hence, only a partial comparison of the results can be made.

In (Zhang et al., 2008) a performance analysis of a usage-based security framework for collaborative systems, is presented. The test case included an analysis of a file sharing prototype system. The performance analysis have shown that ADF's performance for updating the code of a software module varied between 2304 to 15958 milliseconds, depending on the experiment's input data set. Knowing that there cannot be a direct comparison between the framework in (Zhang et al., 2008) and domRBAC simulator, we examine the processing time in order to check if existing time delays of our ADF are within acceptable time limits. Hence, in regard to domRBAC's ADF we identified a latency of 1142 milliseconds in worst cases and thus it can be concluded that such delays are within acceptable limits since the latency is significantly lower.

In regard to secure inter-operation, in (Shafiq et al., 2005) there is an analysis that shows the trade-off between interoperability and autonomy for two collaborating domains. The domains consist of a maximum number of 20 roles each. Autonomy loss can be set at different levels, based on the requirements of the collaboration. Hence, autonomy loss varied from 39% to 52%, and interoperability level from 30% to 36%. However, this approach is not dynamic as in the proposed model. Instead, the merging of the individual RBAC policies into a global policy needs to be done from scratch, in order to gain the optimality criterion of maximizing inter-domain role accesses without exceeding the autonomy losses beyond an acceptable limit. Regarding the simulation results in domRBAC, the autonomy loss and inter-operation reached the level of 4% and 7.75% on average, respectively. It is noteworthy that in the simulation the data sets had significantly higher number of roles, compared to that in (Shafiq et al., 2005), and moreover that in the proposed access control model the global policy is a result of a continually changing environment. Autonomy or inter-operation thresholds are not supported in domRBAC.

Therefore, based on the comparative data, we verified the efficiency of the proposed access control model in regard to its performance and provided level of autonomy loss and inter-operation. In regard to security, this is preserved and assured due to the existence of the inter-domain policy violation function.

7. Conclusions

An access control model is proposed in this article for collaborative systems, as the Grid. To meet scalability, security and basic usage management requirements in access control, the proposed domRBAC model is used to support various security policies for collaborative environments. Our proposed access control model is capable of enforcing security policies among multiple domains, having each different security policies. Furthermore, domRBAC assures a secure collaborative environment by gradually and in real-time checking for violations, which can be caused by new inter-domain role assignments. Moreover, it provides basic resource usage management for the first time in a role based approach. An implemented simulator capable of enforcing multi-domain security policies demonstrated the feasibility of our access control model. A performance study shows that domRBAC can perform well even when implemented in low-end systems. Additionally, through a comparative review it is shown that the proposed access control model is able to perform better in many cases compared with existing implementations. Yet, due to design decisions that requires the maintenance of the security principle, the proposed access control model, results in relatively lower interoperability levels, yet acceptable, compared with existing solutions. In overall, compared to other similar approaches, the domRBAC model provides access control with adequate dynamics for computing systems, as the Grid, and also manages to fulfill critical requirements of modern collaborative environments.

Appendix A. The XSD file

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="DomainRole_Graph"> <xs:complexType>
<xs:sequence minOccurs="0" maxOccurs="unbounded">
<xs:element name="Organization"/>
<xs:element name="DomainRole" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence> </xs:complexType> </xs:element>
<xs:element name="Organization">
<xs:complexType> <xs:sequence>
<xs:element name="Org_Name" type="xs:string"/>
</xs:sequence> </xs:complexType> </xs:element>
<xs:element name="Org_Name"> <xs:complexType mixed="true"/>
</xs:element>
<xs:element name="DomainRole">
<xs:complexType> <xs:sequence>
<xs:element name="Name" type="xs:string"/>
<xs:sequence minOccurs="0" maxOccurs="unbounded">
<xs:element name="Inter_Parent_Role"
  type="xs:string" minOccurs="0" maxOccurs="1"/>
</xs:sequence>
```

```
<xs:sequence minOccurs="0" maxOccurs="unbounded">
<xs:element name="Inter_Child_Role"
  type="xs:string" minOccurs="0" maxOccurs="1"/>
</xs:sequence>
<xs:sequence minOccurs="0" maxOccurs="unbounded">
<xs:element name="Intra_Parent_Role"
  type="xs:string" minOccurs="0" maxOccurs="1"/>
</xs:sequence>
<xs:sequence minOccurs="0" maxOccurs="unbounded">
<xs:element name="Intra_Child_Role"
  type="xs:string" minOccurs="0" maxOccurs="1"/>
</xs:sequence>
<xs:sequence minOccurs="0" maxOccurs="unbounded">
<xs:element name="SSD_Role"
  type="xs:string" minOccurs="0" maxOccurs="1"/>
</xs:sequence>
<xs:sequence minOccurs="0" maxOccurs="unbounded">
<xs:element name="DSD_Role"
  type="xs:string" minOccurs="0" maxOccurs="1"/>
</xs:sequence>
<xs:element name="SR_Cardinality"
  type="xs:unsignedLong" minOccurs="0" maxOccurs="1"/>
<xs:element name="DR_Cardinality"
  type="xs:unsignedLong" minOccurs="0" maxOccurs="1"/>
</xs:sequence> </xs:complexType>
</xs:element>
<xs:element name="Name">
<xs:complexType mixed="true"/>
</xs:element>
<xs:element name="Inter_Parent_Role">
<xs:complexType mixed="true"/>
</xs:element>
<xs:element name="Inter_Child_Role">
<xs:complexType mixed="true"/>
</xs:element>
<xs:element name="Intra_Parent_Role">
<xs:complexType mixed="true"/>
</xs:element>
```

```
<xs:element name="Intra_Child_Role">
<xs:complexType mixed="true"/>
</xs:element>
<xs:element name="SSD_Role">
<xs:complexType mixed="true"/>
</xs:element>
<xs:element name="DSD_Role">
<xs:complexType mixed="true"/>
</xs:element>
<xs:element name="SR_Cardinality">
<xs:complexType mixed="true"/>
</xs:element>
<xs:element name="DR_Cardinality">
<xs:complexType mixed="true"/>
</xs:element>
</xs:schema>
```

Appendix B. Statistical measures

Appendix B.1. Group A data set

Appendix B.2. Group B data set

References

Alfieri, R., Cecchini, R., Ciaschini, V., Dell' Agnello, L., Frohner, A., Gianoli, A., Lorentey, K., & Spataro, F. (2004). Voms, an authorization system for virtual organizations. In *Grid Computing* (pp. 33–40). Springer.

ANSI (2004). Ansi incits 359-2004, role based access control.

Ardagna, C., De Capitani di Vimercati, S., Paraboschi, S., Pedrini, E., Samarati, P., & Verdicchio, M. (2011). Expressive and deployable access control in open web service applications. *Services Computing, IEEE Transactions on*, *4*, 96 –109.

Benantar, M. (2005). *Access Control Systems: Security, Identity Management and Trust Models*. Springer-Verlag New York, Inc.

Boost (2011). Boost c++ libraries.

Carlstrom, B. (2004). Design and analysis of algorithms, problem set no.6 solutions.

Chadwick, D. (2005). Authorisation in grid computing. *Information Security Technical Report*, *10*, 33–40.

Chadwick, D., Otenko, A., & Ball, E. (2003). Role-based access control with x. 509 attribute certificates. *Internet Computing, IEEE*, *7*, 62–69.

Chakrabarti, A. (2007). *Grid computing security*. Spinger-Verlag.

Chen, L., & Crampton, J. (2007). Inter-domain role mapping and least privilege. In *SACMAT '07: Proceedings of the 12th ACM symposium on Access control models and technologies* (pp. 157–162). New York, NY, USA: ACM.

Chu, D. C., & Humphrey, M. (2004). Mobile ogsi.net: Grid computing on mobile devices. *Grid Computing, IEEE/ACM International Workshop on*, *0*, 182–191.

Ferraiolo, D. F., Kuhn, D. R., & Chandramouli, R. (2003). *Role-Based Access Control*. Artech House, Inc.

Foster, I., & Tuecke, S. (2005). Describing the elephant: The different faces of it as service. *Queue*, *3*, 26–29.

Free Software Foundation, I. (2008). Using the gnu compiler collection.

Gouglidis, A., & Mavridis, I. (2011). Role-based secure inter-operation and resource usage management in mobile grid systems. In C. Ardagna, & J. Zhou (Eds.), *Information Security Theory and Practice. Security and Privacy of Mobile Devices in Wireless Communication* (pp. 38–53). Springer Berlin / Heidelberg volume 6633 of *Lecture Notes in Computer Science*.

Gouglidis, A., & Mavridis, I. (2012). Computational and data grids: Principles, applications and design. chapter Grid Access Control Models and Architectures. (pp. 217–234). IGI Global.

Graphviz (2010). Graphviz - graph visualization software.

Graphviz (2011). The dot language.

ISO/IEC-13568 (2002). Information technology z - formal specification notation - syntax, type system and semantics. International Standard.

ITU (1995). Information technology - open systems interconnection - security frameworks for open systems: Access control framework.

Jamkhedkar, P. A., Heileman, G. L., & Lamb, C. C. (2010). An interoperable usage management framework. In *Proceedings of the tenth annual ACM workshop on Digital rights management* DRM '10 (pp. 73–88). New York, NY, USA: ACM.

Li, N., Byun, J., & Bertino, E. (2007). A critique of the ansi standard on role-based access control. *Security Privacy, IEEE*, *5*, 41 –49.

NetworkX (2011). Networkx.

Neumann, G., & Strembeck, M. (2003). An approach to engineer and enforce context constraints in an rbac environment. In *SACMAT '03: Proceedings of the eighth ACM symposium on Access control models and technologies* (pp. 65–79). New York, NY, USA: ACM.

Nokia (2011). Qt - cross-platform application and ui framework.

Nuutila, E. (1995). *Efficient transitive closure computation in large digraphs*. Ph.D. thesis Acta Polytechnica Scandinavica, Helsinki University of Technology.

OASIS (2011). Oasis extensible access control markup language (xacml) tc.

Pearlman, L., Welch, V., Foster, I., Kesselman, C., & Tuecke, S. (2002). A community authorization service for group collaboration. In *Policies for Distributed Systems and Networks. Proceedings. Third International Workshop on* (pp. 50–59). IEEE.

Phan, T., Huang, L., & Dulan, C. (2002). Challenge: integrating mobile wireless devices into the computational grid. In *Proceedings of the 8th annual international conference on Mobile computing and networking* (p. 278). ACM.

Purdom, P. (1970). A transitive closure algorithm. *BIT Numerical Mathematics*, *10*, 76–94. 10.1007/BF01940892.

Racz, P., Burgos, J., Inacio, N., Morariu, C., Olmedo, V., Villagra, V., Aguiar, R., & Stiller, B. (2007). Mobility and qos support for a commercial mobile grid in akogrimo. In *Mobile and Wireless Communications Summit, 2007. 16th IST* (pp. 1 –5).

Ravi Sandhu, V. B. (2008). The ascaa principles for next-generation role-based access control. In *Proc. 3rd International Conference on Availability, Reliability and Security (ARES)* (pp. xxvii–xxxii). Barcelona, Spain volume 6.

Sandhu, R., & Park, J. (2003). Usage control: A vision for next generation access control. In *Computer Network Security* (pp. 17–31). Springer Berlin / Heidelberg volume 2776.

Sandhu, R. S., Coyne, E. J., Feinstein, H. L., & Youman, C. E. (1996). Role-based access control models. *IEEE Computer*, *29*, 38–47.

Sandhu, R. S., & Samarati, P. (1994). Access control: Principles and practice. *IEEE Communications Magazine*, *32*, 40–48.

Shafiq, B., Joshi, J. B. D., Bertino, E., & Ghafoor, A. (2005). Secure interoperation in a multidomain environment employing rbac policies. *IEEE Trans. on Knowl. and Data Eng.*, *17*, 1557–1577.

Shehab, M., Bertino, E., & Ghafoor, A. (2005). Serat: Secure role mapping technique for decentralized secure interoperability. In *SACMAT '05: Proceedings of the tenth ACM symposium on Access control models and technologies* (pp. 159–167). New York, NY, USA: ACM.

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

W3C (2005). Document object model (dom).

W3C (2011). Xml technology.

XML-DEV (2011). Simple api for xml.

Yuan, E., & Tong, J. (2005). Attributed based access control (abac) for web services. *Web Services, IEEE International Conference on*, *0*, 561–569.

Zhang, G., & Parashar, M. (2004). Dynamic context-aware access control for grid applications. In *Grid Computing, 2003. Proceedings. Fourth International Workshop on* (pp. 101–108). IEEE.

Zhang, X., Nakae, M., Covington, M., & Sandhu, R. (2006). A usage-based authorization framework for collaborative computing systems. In *Proceedings of the 11th ACM symposium on Access control models and technologies* (pp. 180–189). ACM.

Zhang, X., Nakae, M., Covington, M. J., & Sandhu, R. (2008). Toward a usage-based security framework for collaborative computing systems. *ACM Trans. Inf. Syst. Secur.*, *11*, 1–36.

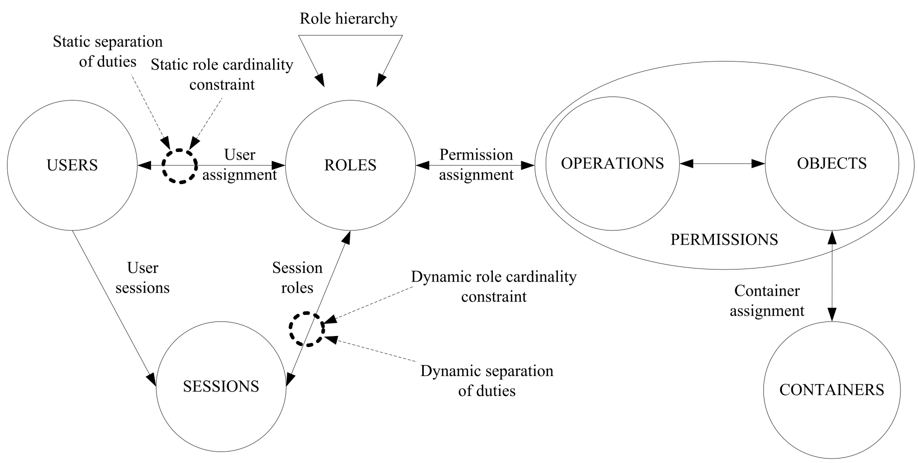

Figure 1: The domRBAC access control model.

```
ci_violation() : boolean
    for each vertex d_i r ∈ T_G
        for each adjacent vertex d_j r to d_i r
            if (d_i r = d_j r)
                return true
    return false
```

Figure 2: Identification of cycles in role assignment

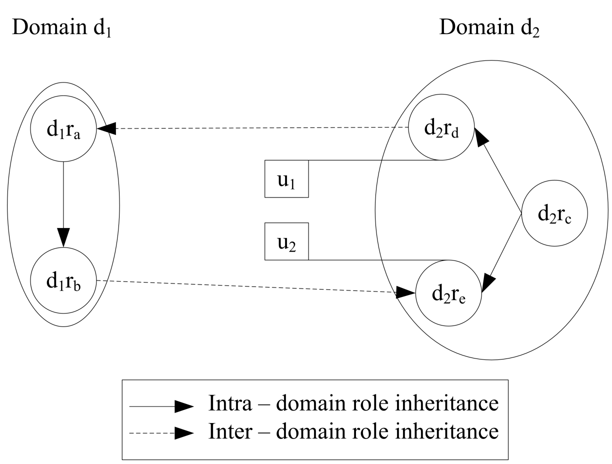

Figure 3: Privilege escalation example.

sp_violation$(d_i r_j)$: **boolean**
    for each vertex $d_i r \in domain\ i$ **where** $d_i r \geq d_i r_j$
        if $(T_G[d_i r_j, d_i r] \neq T_{G_i}[d_i r_j, d_i r])$ **or**
            $(T_G[d_i r, d_i r_j] \neq T_{G_i}[d_i r, d_i r_j])$
        return true
    return false

Figure 4: Assurance of the security principle

ssd_violation$()$: **boolean**
    for each $SSD\ pair\ (d_i r_m, d_i r_n)$
        for each vertex $dr \in T_G$
            if $(d_r = d_i r_m)$
                for each adjacent vertex $d'r$ to dr
                    if $(d'r = d_i r_n)$**then**
                        return true
            if $(d_r = d_i r_n)$
                for each adjacent vertex $d'r$ to dr
                    if $(d'r = d_i r_m)$**then**
                        return true
    for each $SSD\ pair\ (d_i r_m, d_i r_n)$
        for each vertex $dr \in T_G$
            $foundSSDRole = 0$
            for each adjacent vertex $d'r$ to dr
                if $(d'r = d_i r_m)$ or $(d'r = d_i r_n)$
                    $foundSSDRole{+}{+}$
                if $foundSSDRole = 2$ **then**
                    return true
    return false

Figure 5: Intra-domain violation of SSD relationships

1. **dsd_violation**() : **boolean**
2. **for each** $DSD\ pair$ (d_ir_m, d_ir_n)
3. **for each** vertex $dr \in T_G$
4. **if** $(d_r = d_ir_m)$
5. **for each** adjacent vertex $d'r$ to dr
6. **if** $(d'r = d_ir_n)$**then**
7. **return true**
8. **if** $(d_r = d_ir_n)$
9. **for each** adjacent vertex $d'r$ to dr
10. **if** $(d'r = d_ir_m)$**then**
11. **return true**
12. **for each** $DSD\ pair$ (d_ir_m, d_ir_n)
13. **for each** vertex $dr \in T_G$
14. $foundDSDRole = 0$
15. **for each** adjacent vertex $d'r$ to dr
16. **if** $(d'r = d_ir_m)$ or $(d'r = d_ir_n)$
17. $foundDSDRole++$
18. **if** $foundDSDRole = 2$ **then**
19. **return true**
20. **return false**

Figure 6: Intra-domain violation of DSD relationships

1. **InterdomainPolicyViolation**$(d_ir_{n_{desc}})$: **boolean**
2. **return ci_violation**() **or**
3. **sp_violation**$(d_ir_{n_{desc}})$ **or**
4. **ssd_violation**() **or**
5. **dsd_violation**()

Figure 7: Inter-domain policy violation function

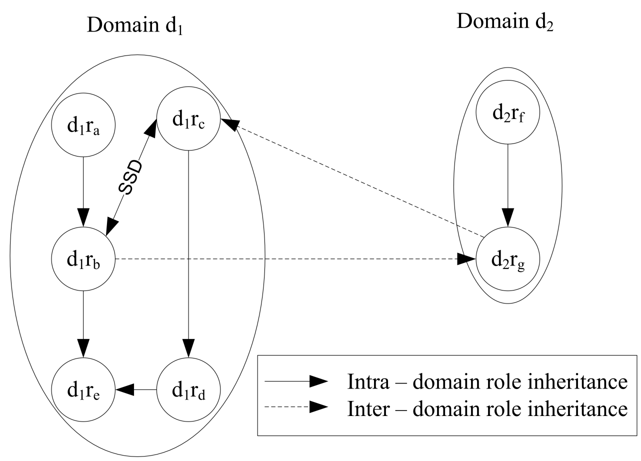

Figure 8: A multidomain access control policy defining interoperation between d_1 and d_2.

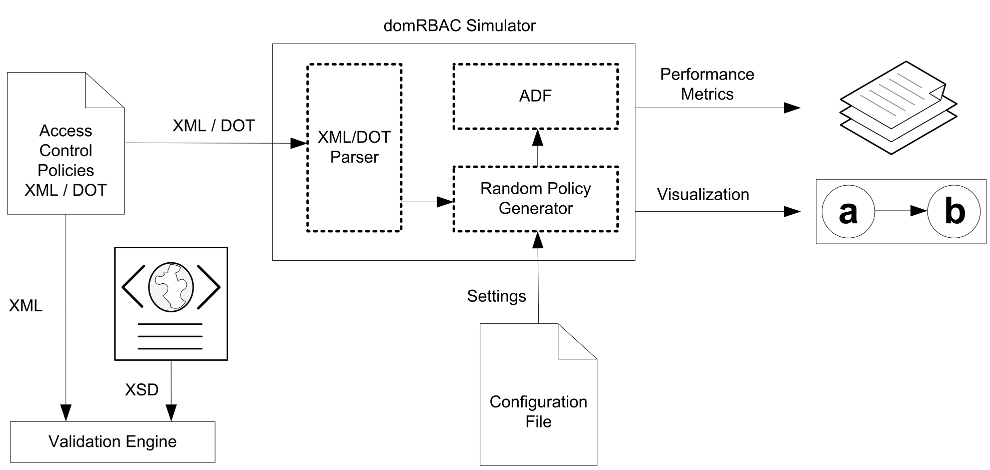

Figure 9: Overall architecture of the domRBAC simulator.

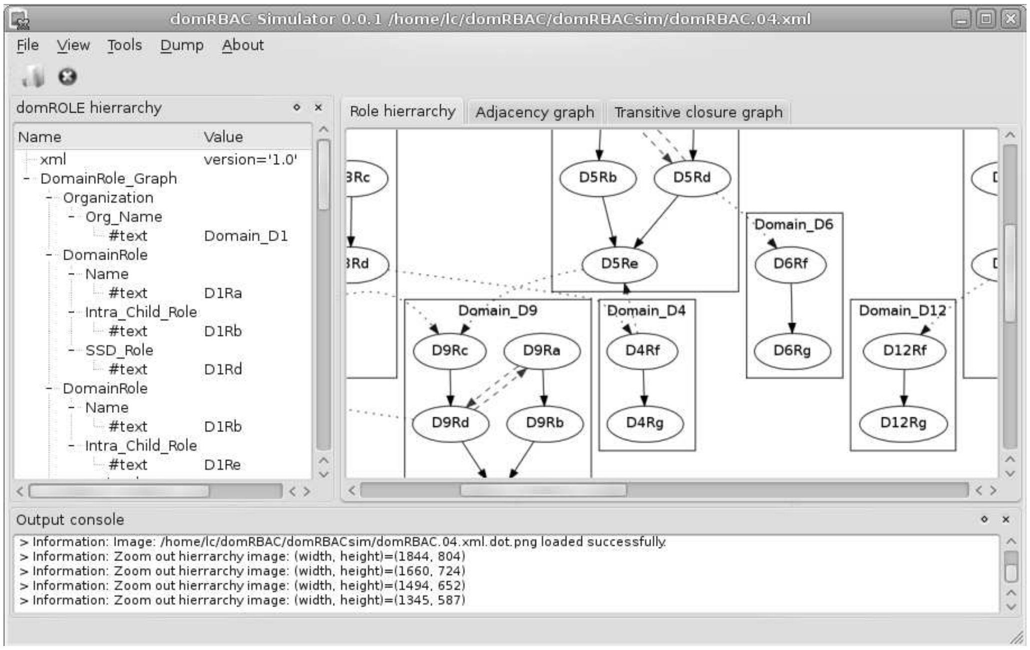

Figure 10: The main interface of the domRBAC simulator.

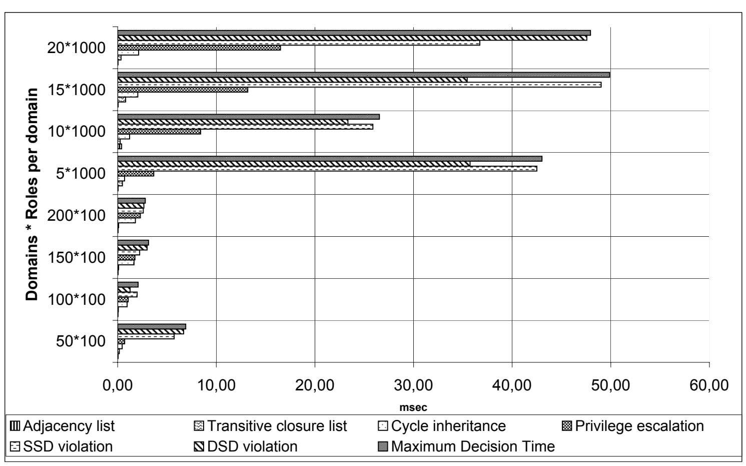

Figure 11: Time of computations - mean values.

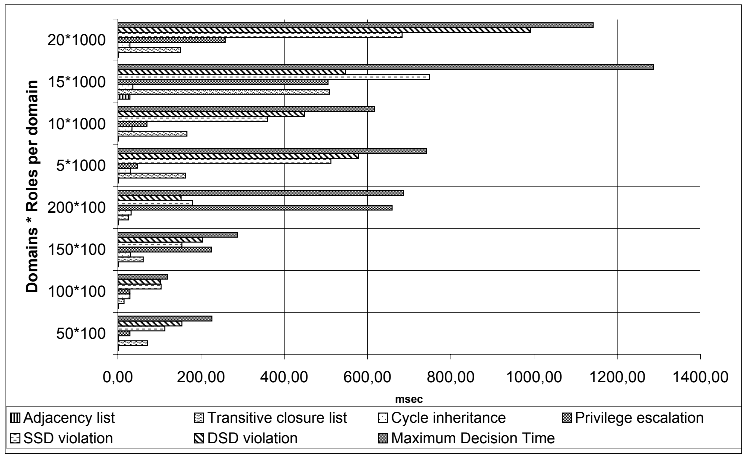

Figure 12: Time of computations - max values.

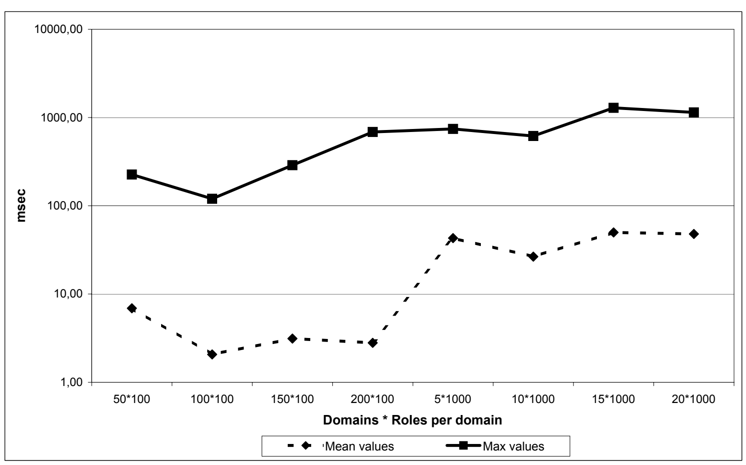

Figure 13: Comparison of mean and max values.

		Adjacency list (msec)	Transitive closure list (msec)	Cycle inheritance (msec)	Privilege Escalation (msec)	SSD Violation (msec)	DSD Violation (msec)
N	Valid	5000	5000	5000	5000	5000	5000
	Missing	0	0	0	0	0	0
Mean		,04	,17	,46	,71	5,73	6,69
Median		,00	,00	,00	1,00	,00	,00
Mode		0	0	0	0	0	0
Std. Deviation		,192	1,651	,499	1,095	18,604	22,323
Maximum		1	71	1	29	113	154

Figure 14: Statistics of experiment with 50 domains of 100 roles each.

		Adjacency list (msec)	Transitive closure list (msec)	Cycle inheritance (msec)	Privilege Escalation (msec)	SSD Violation (msec)	DSD Violation (msec)
N	Valid	5000	5000	5000	5000	5000	5000
	Missing	0	0	0	0	0	0
Mean		,03	,07	,96	1,07	1,97	1,26
Median		,00	,00	1,00	1,00	,00	,00
Mode		0	0	1	1	0	0
Std. Deviation		,183	,471	,638	1,064	11,077	7,691
Maximum		1	15	29	29	104	103

Figure 15: Statistics of experiment with 100 domains of 100 roles each.

		Adjacency list (msec)	Transitive closure list (msec)	Cycle inheritance (msec)	Privilege Escalation (msec)	SSD Violation (msec)	DSD Violation (msec)
N	Valid	5000	5000	5000	5000	5000	5000
	Missing	0	0	0	0	0	0
Mean		,04	,10	1,66	1,75	2,23	2,99
Median		,00	,00	2,00	1,00	,00	,00
Mode		0	0	2	1	0	0
Std. Deviation		,197	1,014	1,126	3,713	13,247	17,458
Maximum		2	61	30	225	154	204

Figure 16: Statistics of experiment with 150 domains of 100 roles each.

		Adjacency list (msec)	Transitive closure list (msec)	Cycle inheritance (msec)	Privilege Escalation (msec)	SSD Violation (msec)	DSD Violation (msec)
N	Valid	5000	5000	5000	5000	5000	5000
	Missing	0	0	0	0	0	0
Mean		,04	,10	1,80	2,30	2,60	2,66
Median		,00	,00	2,00	2,00	,00	,00
Mode		0	0	2	1	0	0
Std. Deviation		,194	,574	1,169	9,542	16,562	16,312
Maximum		1	26	32	659	180	152

Figure 17: Statistics of experiment with 200 domains of 100 roles each.

		Adjacency list (msec)	Transitive closure list (msec)	Cycle inheritance (msec)	Privilege Escalation (msec)	SSD Violation	DSD Violation (msec)
N	Valid	5000	5000	5000	5000	5000	5000
	Missing	0	0	0	0	0	0
Mean		,04	,48	,70	3,66	42,50	35,73
Median		,00	,00	1,00	2,00	,00	,00
Mode		0	0	1	0	0	0
Std. Deviation		,205	4,430	,695	4,639	95,400	86,945
Maximum		1	163	31	47	512	578

Figure 18: Statistics of experiment with 5 domains of 1000 roles each.

		Adjacency list (msec)	Transitive closure list (msec)	Cycle inheritance (msec)	Privilege Escalation (msec)	SSD Violation (msec)	DSD Violation (msec)
N	Valid	5000	5000	5000	5000	5000	5000
	Missing	0	0	0	0	0	0
Mean		,04	,26	1,21	8,41	25,88	23,35
Median		,00	,00	1,00	6,00	,00	,00
Mode		0	0	1	0	0	0
Std. Deviation		,192	2,476	,671	8,518	64,975	61,731
Maximum		2	166	34	70	359	449

Figure 19: Statistics of experiment with 10 domains of 1000 roles each.

		Adjacency list (msec)	Transitive closure list (msec)	Cycle inheritance (msec)	Privilege Escalation (msec)	SSD Violation (msec)	DSD Violation (msec)
N	Valid	5000	5000	5000	5000	5000	5000
	Missing	0	0	0	0	0	0
Mean		,05	,81	2,04	13,19	49,02	35,44
Median		,00	,00	2,00	8,00	,00	,00
Mode		0	0	2	0	0	0
Std. Deviation		,612	13,506	1,101	16,996	128,596	101,218
Maximum		29	509	36	505	749	547

Figure 20: Statistics of experiment with 15 domains of 1000 roles each.

		Adjacency list (msec)	Transitive closure list (msec)	Cycle inheritance (msec)	Privilege Escalation (msec)	SSD Violation (msec)	DSD Violation (msec)
N	Valid	5000	5000	5000	5000	5000	5000
	Missing	0	0	0	0	0	0
Mean		,03	,34	2,13	16,51	36,72	47,57
Median		,00	,00	2,00	11,00	,00	,00
Mode		0	0	2	0	0	0
Std. Deviation		,175	2,718	,787	16,468	114,469	146,528
Maximum		1	150	29	258	683	991

Figure 21: Statistics of experiment with 20 domains of 1000 roles each.

Table 1: Performance evaluation of ADF's memory usage and identified violations (5000 requests)

Domains	Roles per domain	Role assignments	SSD	DSD	Inter-domain	Adj. list (KB)	Trans. closure list (KB)	Role viol.	SSD viol.	DSD viol.
50	100	5362	88	111	266	10362	39788	4565	8	12
100	100	10029	44	34	52	20029	62120	4865	2	1
150	100	14979	39	45	52	29979	90448	4870	1	0
200	100	19925	38	35	64	39925	114294	4875	0	0
5	1000	5970	251	244	77	10970	48662	3947	50	25
10	1000	10553	108	110	53	20553	65166	4413	12	12
15	1000	15456	147	117	22	30456	100485	4503	15	8
20	1000	20379	129	152	20	40379	164882	4591	3	6