Access Control Requirements Engineering, Modeling and Verification in Multi-Domain Grid and Cloud Computing Systems

Antonios Gouglidis

Department of Applied Informatics

University of Macedonia

A dissertation submitted for the degree of

Doctor of Philosophy

November 2013

This Dissertation Advisory Committee (DAC) consists of the following members (listed alphabetically):

Professor Athanasios Manitsaris
(Dept. of Applied Informatics, University of Macedonia, Greece.)
Associate Professor Ioannis Mavridis (PhD Supervisor)
(Dept. of Applied Informatics, University of Macedonia, Greece.)
Assistant Professor Panayotis Fouliras
(Dept. of Applied Informatics, University of Macedonia, Greece.)

This Dissertation Defence Committee (DDC) consists of the following members (listed alphabetically):

Associate Professor Alexandros Chatzigeorgiou
(Dept. of Applied Informatics, University of Macedonia, Greece.)
Assistant Professor Christos Georgiadis
(Dept. of Applied Informatics, University of Macedonia, Greece.)
Associate Professor George Rahonis
(Dept. of Mathematics, Aristotle University of Thessaloniki, Greece.)
Associate Professor Ioannis Mavridis (PhD Supervisor)
(Dept. of Applied Informatics, University of Macedonia, Greece.)
Associate Professor Ioannis Stamelos
(Dept. of Informatics, Aristotle University of Thessaloniki, Greece.)
Assistant Professor Panagiotis Katsaros
(Dept. of Informatics, Aristotle University of Thessaloniki, Greece.)
Assistant Professor Panayotis Fouliras Assistant Professor
(Dept. of Applied Informatics, University of Macedonia, Greece.)

I would like to dedicate this dissertation to my wife Eleni and sons Dimitrios and Ioannis.

Acknowledgements

I would like to use the occasion of this dissertation to thank all the people who helped me in reaching this important goal.

First of all, I would like to sincerely thank Associate Professor Ioannis Mavridis, my advisor, not only for introducing me to scientific research and for all the opportunities he gave me during our collaboration, but also for the guidance and advice. His leadership, support, attention to details, hard work, and scholarship have set an example I hope to match some day.

I would like to thank Professor Athanasios Manitsaris and Assistant Professor Panayotis Fouliras at the Department of Applied Informatics of the University of Macedonia, for their time and dedication in reviewing my work and for their valuable feedback.

I would like to thank Dr. Vincent C. Hu at NIST for providing valuable information regarding ACPT and its underlying model checking technique, and for his collaboration and feedback on the proposed model checking technique.

I would like to thank Anna Fitsiou for her time in assisting me in proof-reading portions of my work.

I would like to thank my colleague and friend Christos Grompanopoulos at the Department of Applied Informatics of the University of Macedonia, for his collaboration and assistance over these years.

I would like to acknowledge the Research Committee of the University of Macedonia in Greece for partially funding my research.

Last, but not least, my family deserves all my gratitude for giving me the opportunity to undertake this work in the first place, constantly supporting me over the years, and for always being there.

Abstract

Recent advances in sciences and business models required the invention of new and innovative types of systems in order for them to be used as a development and deployment platform for applications. Examples of such systems are the Grid and Cloud computing paradigms. Both of them are evolutionary distributed and collaborative systems, which have currently become the de facto platforms for the development and deployment of various types of applications. Despite the different nature of these two types of systems, several requirements and principles remain the same in both of them. Security is an essential principle and it is required to be maintained during any collaboration among participants. Despite the benefits of existing security solutions there are few proposals that addressed the problem of how to maintain security among domains where each implement its own access control (AC) policy. Moreover, the majority of existing solutions are static in nature and not suitable for the examined systems.

In this dissertation, the notions of AC requirements engineering, AC modeling and verification of security properties are fully integrated within a common systems engineering methodology. In summary, the contribution of this dissertation is multifold: we initially describe a systems engineering methodology for the development of AC systems; we describe our proposed steps; then we define an AC model; and lastly we define a verification technique for the verification of security properties. Specifically, looking towards a holistic approach on the definition of AC requirements, we propose a four-layer conceptual categorization for the identification of security requirements and an evaluation framework. In a comparative review of the examined AC

models and mechanisms using the conceptual categorization, their pros and cons are exposed. Apart from the mapping of the AC area in Grid and Cloud systems, the given comparison renders valuable information for further enhancement of current approaches.

Moreover, we define an enhanced Role-Based Access Control (RBAC) model entitled domRBAC for collaborative systems, which is based on the ANSI INCITS 359-2004 AC 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 RBAC model. Furthermore, secure inter-operation is assured among collaborating domains during inter-domain role assignments, gradually and automatically. Yet, domRBAC, as an RBAC approach, intrinsically inherits all of its virtues such as ease of management, and Separation of Duty (SoD) with the latter also being supported in multiple domains. As a proof of concept, we implemented a simulator based on the definitions of our proposed AC model and conducted with experimental studies to demonstrate the feasibility and performance of our approach.

Lastly, we provide a formal definition of secure inter-operation properties in temporal logic, which can be verified using model checking techniques. The proposed technique consists of a generic one, and thus, can be used in any RBAC model to verify indirectly the correctness of the secure inter-operation functions that implement the global security policy. As a proof of concept, we provide examples that illustrate the enforcement of the defined secure inter-operation properties, which have to be verified in RBAC policies, and a performance analysis of the proposed technique.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Access control (AC) in modern distributed systems has become even more challenging since they are complex systems and require the collaboration among domains. AC is an essential process in all systems. The role of an AC system is to control and limit the actions or operations in the system, which are performed by a user on a set of resources. Nevertheless, an AC system is considered of three abstractions of control, namely AC policies, AC models, and AC mechanisms (Ferraiolo et al., 2003, chap. 2). Using the former abstractions of control, an AC system is responsible for the enforcement of an AC policy in it, and at the same time it is responsible for preventing the access policy from subversion. A policy can be defined as a high-level requirement that specifies how a user may access a specific resource and when. AC policies can be enforced in a system through an AC mechanism that is responsible for permitting or denying a user access upon a resource. An AC model can be defined as an abstract container of a collection of AC mechanism implementations, which are capable of preserving support for the reasoning of the system policies through a conceptual framework. Consequently, the AC model is capable of bridging the existing abstraction gap between the mechanism and policy in a system (Capitani di Vimercati et al., 2007; Sandhu and Samarati, 1994). This dissertation is concerned with requirements engineering, modeling, and verification of AC models for modern collaborative computing systems. In the remainder of the chapter, we give the motivation of our dissertation and its outline.

1.1 Motivation

In recent years, Grid and Cloud computing paradigms have become the focal point of science and enterprise computer environments. AC in Grid and Cloud computing systems is an active research area given the challenges and complex applications.

The Grid is an emergent technology that can be defined as a system able to share resources and provide problem solving in a coordinated manner within dynamic, multi-institutional virtual organizations (Foster et al., 2001). This definition depends mostly on the sharing of resources and the collaboration of individual users or groups within the same or among different virtual organizations, in a service-oriented approach. The Grid's unique characteristics, such as its highly distributed nature and the heterogeneity of its resources, require the revision of a number of security concepts. Trust, authentication, authorization and AC are some of the security concepts that is required to be revised in Grid systems.

The Cloud is a fairly new and emergent technology and its definition is a topic for discussion in several research papers (Foster et al., 2008). Nevertheless, Cloud computing is defined in (Mather et al., 2009, chap. 2) using five attributes viz. multitenancy, massive scalability, elasticity, pay as you go and self-provisioning of resources. These attributes successfully imprint the distinctive characteristics of the Cloud and differentiate it from similar technologies, as the Grid computing paradigm. Specifically, multitenancy refers to the business model implemented by the Cloud, where a single shared resource can be used from multiple users. Massive scalability refers to the potential of the system to scale (i.e., increase or decrease) in resources. The on-demand and rapid increment or decrement of computing resources is translated as elasticity of the Cloud. Thus, more storage space or bandwidth can be allocated when required and vice versa. Pay as you go is the process of paying for the resources that are used. Lastly, the users are provided with the ability to self-provision resources, namely storage space, processing power, network resources and so on. An additional characteristic defined in (Peter and Timothy, 2011) by the National Institute of Standards and Technology (NIST) is Broad Network Access, which states that available capabilities can be accessed using standard mechanisms over the network, and promote

their use by heterogeneous clients. Regarding Cloud's service model, it is based on the SPI framework as stated in (Mather et al., 2009, chap. 2) and (Peter and Timothy, 2011). SPI stands for Software-as-a-service (SaaS), Platform-as-a-service (PaaS) and Infrastructure-as-a-service (IaaS). Specifically, SaaS provides the software that is used under a business model, namely the usage-based pricing. PaaS offers the platform for the development of the applications, and lastly, IaaS handles the provision of the required hardware, software and equipment, in order to deliver a resource usage-based pricing model.

Moreover, the aforementioned service models are provided under three deployment models namely public, private and hybrid Cloud (Mather et al., 2009, chap. 2). The public Cloud is able to provision resources over the Internet and are accessible via a Web application. A third-party operates as the host and performs all the required operations (e.g., management, security). The private Cloud provides the same functionality as the public deployment model within internal and private networks. This model requires the acquisition of the appropriate hardware and software. The hybrid model refers to the combination of the public and private deployment models. Usually, the latter model is used to keep sensitive data in the private network and deploy non-core applications to the public. An additional service model proposed by NIST (Peter and Timothy, 2011) is the community Cloud, which refers to infrastructure exclusively used by a specific community of consumers from organizations that have shared concerns.

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

The RBAC model has received considerable attention from researchers for its capabilities. However, it lacks decentralized management of policies and 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. Further information regarding administration in multi-domain environments is provided in Subsection 2.2.3. Further, 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 because of the development of Internet based distributed systems. However, in contrast to RBAC, ABAC, as already stated, has not been standardized, yet. A first effort in standardizing ABAC has been lately initiated by NIST (NIST, 2013). An example of ABAC is considered the $UCON_{ABC}$ model. Generally in ABAC approaches, access decisions are provided on resources based on the requester's 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 AC and collaboration among domains. $UCON_{ABC}$ is known to be among the first attempts in AC 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 AC 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 AC in an 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 AC 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., 2003), Akenti (Thompson et al., 2003), PERMIS (Chadwick, 2005; Chadwick et al., 2003), 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 and 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. Lastly, it is also noteworthy that Cloud AC mechanisms, at the time of writing this dissertation, are not standardized, yet.

1.2 Objectives

Security and interoperability are issues of great concern for the adoption of Grid and Cloud computing (Foster et al., 2008). Security affect different aspects of modern system transactions among which AC represents the most critical. An important step towards the secure collaboration among participants either in Grid or Cloud systems is the definition of AC models that, apart from preserving the security in a domain, provide also a solution for a secure inter-operation among participants. Although a considerable amount of work has been done in the field of AC for collaborative systems (see Chapter 2), AC models that are available to date are not able to fully support the unique characteristics of modern collaborative systems (see Chapter 4). This situation reflects the fact that in recent years a variety of AC systems were proposed as ad-hoc solutions. In this dissertation, we focus on the definition of an AC model regulating secure inter-operation and access to resources in Grid and Cloud systems. The main objective of our work can be summarized as follows.

In the context of a systems engineering methodology; firstly, we define a requirements engineering approach that will help us in the definition of AC requirements specifically for the examined systems. Secondly, we define an AC model that will integrate the required functionality regarding security for Grid and Cloud systems, and check its efficiency in large-scale systems. Finally, the last objective is to provide a formal method to verify the correctness of the defined AC model.

1.3 Research areas

The contributions of this dissertation are centred on the following topics: definition of a requirements engineering approach, the modeling of an AC model for use in modern collaborative systems, and a verification technique to verify the correctness of the defined model against security properties. All the aforementioned are placed in the context of a systems engineering methodology for an end-to-end development of AC models, as originally presented in (Gouglidis and Mavridis, 2013). In the remainder of this section, we present the contributions in more details.

1.3.1 AC requirements engineering

The first contribution of this dissertation is the proposal of a requirements engineering approach for modern collaborative systems. The original contribution of our work can be summarized as follows (Gouglidis and Mavridis, 2009, 2010).

A layered requirements engineering approach. We define a new requirements engineering approach for the definition of security requirements. Current approaches can be characterized as generic and renders the process of security requirements identification a difficult task. To facilitate the process of identifying and defining AC requirements, we define a conceptual categorization that provides a layered approach for the identification of requirements in different layers concerned with security. The latter is able to capture all the distinctive characteristics regarding AC in modern computing environments and results in the definition of advanced AC models that are suitable for Grid and Cloud environments.

An evaluation framework for AC models and mechanisms. In the context of AC requirements identification and definition, a methodology is needed for the evaluation of existing AC models and mechanisms to identify potential required amendments. A contribution of our work is that the defined conceptual categorization can operate also as an evaluation framework for AC models and mechanisms. Models and mechanisms can be examined to ascertain whether they are suitable for use in any system, or if any augmentations are required to be applied.

1.3.2 AC modeling

An important contribution of this dissertation is the definition of an AC model for Grid and Cloud systems, which is able to gradually enforce AC among different domains, maintain secure collaboration, and provide basic usage management of resources (Gouglidis and Mavridis, 2011, 2012a). The original contribution of our work can be summarized as follows.

Maintenance of secure inter-operation. Modern collaborative systems are capable of providing resource sharing between users and platforms. These collaborations need to be done in a transparent way among its participants. To have a successful collaboration, of a vital importance requirement is to maintain secure inter-operation among the participants. Our defined model is able to provide secure inter-operation gradually, as required by highly dynamic and collaborative environments. Such functionality is provided by the defined domRBAC model automatically.

Usage control. A new characteristic of modern collaborative systems is that they require to provide usage control on their shared resources. Apart from ensuing a secure collaborative environment, our defined AC is able to provide basic usage control management features for shared resources. Compared with other approaches, ours is the first that formally defines AC management in RBAC models through a combination of cardinality constraints and context information.

An efficient AC model. Modern collaborative environments require the existence of an efficient AC model since both the number of users or domains and shared resources can scale up to hundreds or thousands. Our proposed AC model was defined using efficient structures and algorithms that resulted in a highly efficient AC model. An implemented simulator, which operates as an AC mechanism, verifies the efficiency of the model through a series of performance metrics.

1.3.3 Verification of security properties

The third important contribution we present in this dissertation is the definition of a verification approach using model checking techniques for the verification of security properties (e.g., secure inter-operation properties) in RBAC models. This

research work, was the result of a collaboration with the Information Technology Laboratory at NIST, where initially was presented in (Hu et al., 2008). The original contribution of our work can be summarized as follows.

Verification of secure inter-operation via model checking. Although we defined an AC model that is able to maintain a secure collaborative environment, we had to assure its correctness. To check the correctness regarding secure inter-operation, we used formal methods (i.e., temporal logic and model checking). Our contribution is threefold: *i)* A first contribution is the definition for the first time of secure inter-operation properties in temporal logic. The security properties can be verified using model checking techniques, and furthermore, verify the correct implementation of AC models and security policies. *ii)* A second contribution is the introduction of RBAC reasoning regarding role hierarchies in NIST's model checking technique, which was absent. *iii)* The latter technique is proposed as a management service/tool in modern collaborative systems, and thus, it is capable of verifying the a priori or a posteriori enforcement of RBAC policies. The usage of the technique as a management service/tool helps significantly in maintaining secure inter-operation since it is capable of verifying not only the correctness of the applied model, but also its implementation.

1.4 Structure of the dissertation

This chapter has discussed the motivation and the main objectives of our work and outlined the major contributions of this dissertation. The remaining chapters are structured as follow.

Chapter 2 discusses the state of the art of AC related to the objectives of the dissertation. Specifically, it presents information regarding AC in distributed and collaborative systems and elaborates on existing AC models and mechanisms in Grid and Cloud systems.

Chapter 3 presents the proposed systems engineering methodology.

Chapter 4 describes the proposed conceptual categorization for requirements engineering. The conceptual categorization is applied on the examined models and mechanisms to identify new security requirements regarding AC.

Chapter 5 illustrates our AC model. The model is a RBAC model based on the ANSI INCITS 359-2004. A formal definition of the model and an architecture for the evaluation and enforcement of the AC policies is provided.

Chapter 6 presents the proposed model checking technique as a mean to verify the correctness of the proposed model.

Chapter 7 summarizes the contributions of this dissertation and outlines future work.

Chapter 2

Background

2.1 Introduction

This chapter discusses AC in distributed and collaborative systems, i.e., in Grid and Cloud computing systems, and further presents the problem of supporting multi-domain administration in the aforementioned systems. Specifically, we review AC models and mechanism implementations, which can be applied in modern distributed systems that require collaboration among participants. The remainder of the chapter is organized as follows. Section 2.2 presents AC in Grid and Cloud computing systems. Section 2.3 refers to existing AC models for modern collaborative systems. Section 2.4 discusses standard mechanisms for implementing AC and surveys existing mechanisms, and section 2.5 refers to AC enforcement. Finally, Section 2.6 concludes the chapter.

2.2 AC in distributed and collaborative systems

In the following, we provide information regarding AC in Grid and Cloud computing systems, respectively.

2.2.1 Grid computing

As mentioned in the definition of the Grid, terms such as users, resources and services play an important role. To this effect, we explicitly set the following defi-

nitions, which are mainly based on (Benantar, 2005; Chakrabarti, 2007a; Ferraiolo et al., 2003; Foster and Tuecke, 2005; Sandhu and Samarati, 1994).

A service is an implementation of well defined functions that are able to interact with other functions. The service oriented architecture (SOA), in particular, is composed of a set of services that can be realized by technologies such as the web services.

A domain can be defined as a protected computer environment, consisting of users and resources under an AC policy. The collaboration which can be established among domains leads to the formation of a virtual organization.

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. Furthermore, we identify two types of users. These are the resource requester and the resource provider. The former type of user acts like a resource access or usage requester, and the latter type of user acts like a provider of its own shareable resources. All users are restricted by the policies enforced in their participating domains and virtual organization.

A resource in a Grid environment can be any shareable hardware or software asset in a domain and upon which an operation can be performed.

The access control's role is to control and limit the actions or operations in the Grid system that are performed by a user on a set of resources. In brief, it enforces the AC policy of the system, while simultaneously it prevents the access policy from subversion. AC in the literature is also referred to as access authorization or simply authorization.

A Grid AC 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 AC mechanism. The latter is responsible for granting or denying a user access upon a resource.

Finally, an AC model can be defined as an abstract container of a collection of AC mechanism implementations capable of preserving support for the reasoning of system policies through a conceptual framework. The AC model bridges the existing abstraction gap between the mechanism and the policy in a system.

2.2.2 Cloud computing

In Cloud systems, the main objective of AC is to grant authorized users the right to use a service, and at the same time to prevent access to non-authorized users. Similarly to the Grid paradigm, a Cloud AC policy can be defined as a Cloud security requirement that specifies how a user may access a specific resource and when. Such a policy can be enforced in a Cloud system through an AC mechanism, which is enforced by a Cloud Service Provider (CSP). The latter is responsible for granting or denying a user access upon a service. Therefore, AC in Cloud systems is similar to the Grid. The main difference is mostly the subject of the service delivery model that is applied on the Cloud system (SaaS, PaaS, IaaS) (Mather et al., 2009, chap. 2). Hence, in the SaaS delivery model, the CSP is responsible for managing all aspects of the network, server and application infrastructure. In the PaaS delivery model, the customer is responsible for AC to the applications deployed in the PaaS platform. Lastly, in the IaaS delivery model, the customers are entirely responsible for managing all aspects of AC. In general, AC in the Cloud is not standardized across CSPs, and user AC to Cloud resources is generally weak because of coarse user access management (Mather et al., 2009, chap. 6).

2.2.3 Multi-domain administration

Modern computing systems spans several administrative domains and therefore are facing challenging issues. Of a vital importance issue in multi-domain environments is the support of multi-domain administration, which can ensure secure inter-operation across its constituent domains (Bhatti et al., 2005). In Grid and Cloud computing systems, resource access policies can include diverse authorization requirements expressed as authorization constraints. While desirable in access control, constraints can lead to conflicts in the overall policy in a multi-domain environment. Thus, the administration of enterprise-wide access control poses several challenges that range from authorization management of users and resources within individual domains to conflict resolution among heterogeneous access control policies of multiple domains (Bhatti et al., 2005). Resolving the aforementioned issues will eventually lead to the enforcement of secure interop-

eration within the enterprise.

Therefore, there has been a growing interest in administration models built on RBAC and related schemes. Approaches in RBAC administration as in ARBAC99 (Sandhu and Munawer, 1999), in ARBAC02 (Oh and Sandhu, 2002) and the proposed scoped administration model proposed in (Crampton and Loizou, 2002) do not address the issues related to policy administration in a multi-domain environment. 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. The approach has been implemented in X-GTRBAC in (Bhatti et al., 2005). In (Chen and Crampton, 2007) an inter-domain role-mapping approach based on the least privilege principle is suggested. Least privilege is the administrative practice of selectively assigning permission to users such that the user is given no more permission that is necessary to perform his or her job function (Ferraiolo et al., 2003, chap. 1). Yet, the greedy algorithm applied 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 and Parashar, 2004) the DRBAC is presented as a dynamic context-aware AC model for Grid applications. However, the management of inter-domain policies is not tackled.

2.3 AC models

During the last decades various AC policies have been introduced, namely the Mandatory Access Control policies (MAC), the Discretionary Access Control policies (DAC) and the Role Based Access Control policies (RBAC). Each one of them serves specific security requirements in different working environments. As mentioned in the definition of the AC policy, a number of AC models are required and were developed in order for the policies to be represented by formal methods.

Research on the MAC, DAC and RBAC has proven that an AC model, which can express the RBAC policies is also capable of enforcing both MAC and DAC policies (Ferraiolo et al., 2003, chap. 6). It is noteworthy that an attempt started along with the advancement of RBAC for the design of a series of ABAC models. The ABAC model was mainly introduced to overcome a number of RBAC's shortcomings (Yuan and Tong, 2005) and has also been proven capable of enforcing MAC, DAC and RBAC policies (Park and Sandhu, 2004). For the reasons mentioned above, we will present the standard for the RBAC (ANSI, 2004), and Usage Control (Park and Sandhu, 2004; Sandhu and Park, 2003; Zhang et al., 2008) in the rest of this section. Both RBAC's and UCON's characteristics are able to tackle the complexity posed by modern collaborative systems at a satisfactory level.

2.3.1 Role based access control (RBAC)

The RBAC model has received considerable attentions from researchers for its capabilities of abstraction and generalization; abstraction, because it includes only properties that are relevant to security, and generalization because many designs could be considered valid interpretations of the model (Ferraiolo et al., 2003). In addition, RBAC supports various AC principles, such as Least Privilege, and Separation of Duties (SoD)/Administrations (Sandhu et al., 1996). The RBAC model consists of four components with different functionalities. The components are Core RBAC, Hierarchical RBAC, Static Separation of Duty (SSD), and Dynamic Separation of Duty (DSD).

The core RBAC model has five static elements, as depicted in Figure 2.1: users, roles, and permissions, which contain operations and objects. The relations between the elements are straightforward; roles are assigned to users and permissions are assigned to roles, and the mapping of relations between them are many-to-many (i.e., one user can be assigned to many roles) and many users can be assigned to one role. The same applies to the role to permission assignment. However, negative permissions are not supported in RBAC. This indirect assignment of users to permissions greatly enhances the administration in RBAC, and revocation of assignments can be done easily. Moreover, we distinguish design

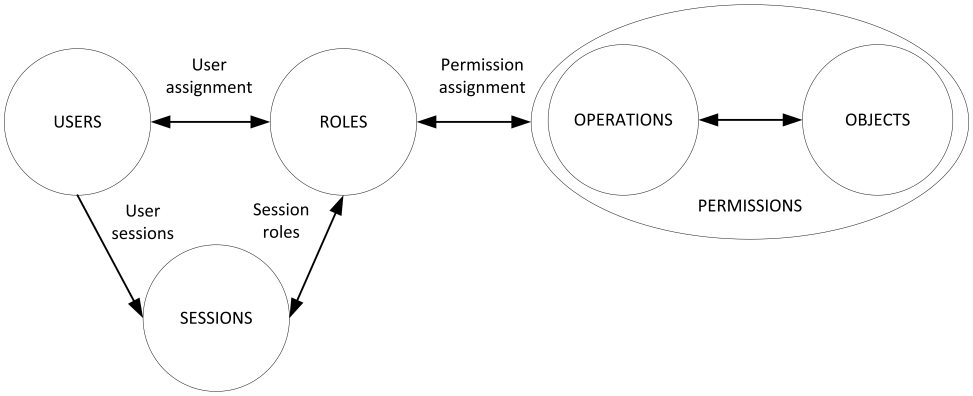

Figure 2.1: The core RBAC model (Ferraiolo et al., 2003).

and run-time phases in RBAC implementations; system administrators define assignments between the elements in the design phase, and the model enforces the assignments in the run-time phase.

The run-time phase is achieved by the concept of the session. This unique (among other group-based AC mechanisms) feature allows a set of users' roles to be activated. This means a user could be assigned to various roles during the design phase, but do not need to be always or simultaneously activated (by the principle of least privilege). However, the capability of sessions has been questioned with a suggestion of replacement for them (Li et al., 2007).

The Hierarchical RBAC enhances administration flexibility through the capability of permission (operations to objects) inheritance; permissions (assigned to a role) can be inherited to another role through hierarchical relation assignments without reassigning the same permissions to the inherited role. For instance, let's assume two roles r_1 and r_2 and two permission sets $PRMS_1 = (p_1, p_2)$ and $PRMS_2 = (p_3, p_4)$, which are initially assigned to roles r_1 and r_2, respectively. Role r_1 inherits role r_2 means all permissions of r_2 are also available to r_1. The inherited permission can be expressed by the union of $PRMS_1$ and $PRMS_2$. The immediate inheritance relation is denoted by the $\rightarrow$, for example, $r_1 \rightarrow r_2$. User membership refers to the assignment of users to roles in a hierarchy, and thus, users are authorized to have all the permission assigned to roles either directly or via inheritance. The Hierarchical RBAC supports general and limited role hierarchies. General hierarchies are composed of partial order sets of common

inheritance relations. And, in more restrictive environments, limited hierarchies require the existence of either a single immediate ascendant or descendant role in the hierarchy. Mathematically, hierarchy is a partial order defining seniority relations between roles, whereby senior roles acquire the permissions from their juniors and junior roles acquire users from their seniors (ANSI, 2004).

Another virtue of RBAC is to constrain authorization with SSD and DSD relationships to prevent the Conflict Of Interest (COI), which is common from business requirements. SSD handles the enforcement of static COI policies. For example, let r_1 and r_2 be two conflicting roles, and user u_1 assigned to role r_1. RBAC prohibits the assignment of user u_1 to role r_2 by enforcing an SSD constraint between roles r_1 and r_2 since the two roles are COI. The constraints are defined and restricted in the design phase. In the presence of role hierarchies, the SSD constrains are enforced in the same way for all the directly assigned and inherited roles. DSD relationships handle COI policies in the context of a session, where a user is activated with a set of assigned roles when logged into the system. As SSD, DSD constraints were specified in the design phase. However, they are enforced during the run-time of authoring process through activated sessions, and thus, preventing the simultaneous activation of two or more conflicting roles.

Lastly, one of its greatest virtues is the role based administration of RBAC. RBAC administration is divided into user and administrator spaces. The former includes user roles and the latter administrative roles with permissions and operations, respectively. Once again, the principle of least privileged is maintained. Various RBAC administration models were proposed in (Crampton and Loizou, 2002), (Ferraiolo et al., 2003), (Oh and Sandhu, 2002), (Sandhu et al., 1999), with different approaches in role based administration.

2.3.2 Usage control (UCON)

ABAC has lately gained a lot of attention due to the development of Internet based distributed systems. However, in contrast to RBAC, ABAC has not been standardized yet. The latter type of AC models can provide access decisions on resources based on the requester's owned attributes. The advantage of this approach is that it is possible to provide access to users in a collaborative en-

vironment without the need for them to be known by the resource a priori. In this section, we will present, in brief, the $UCON_{ABC}$ model (Park and Sandhu, 2004) as a representative ABAC model, which is based on a modern conceptual framework. The UCON conceptual framework encompasses traditional AC, trust management and digital rights management for the protection of digital resources. Nonetheless, functionalities such as administration and delegation are still absent.

UCON has introduced a number of novelties compared to both RBAC and other ABAC models, like its support for mutable attributes and continuity of access decision. Research has also been done regarding its usage in collaborative systems (Zhang et al., 2008). Figure 2.2 illustrates the $UCON_{ABC}$ model, which consists of eight components, viz. subjects, subject attributes, objects, object attributes, rights, authorizations, obligations and conditions. The notion of subjects and objects as well as the association with their attributes is straightforward. A subject can be an entity in a system and its definition, as well as its representation, is given by a number of properties or capabilities in the associated subject's attributes. For instance, role hierarchies similar to RBAC can be formed through the use of subject attributes. In regard to objects, they also represent a set of entities in a system. Each object can be associated with object attributes. Subjects can hold rights on objects. Through these rights, a subject can be granted access or usage of an object. This type of attributes can serve, for example, in the classification of the associated objects, by representing classes, security labels and so on and so forth. It is worth mentioning that both subject and object attributes can be mutable. This means that the values of the attributes can be modified as a result of an access. When an attribute is characterized as immutable, its value can be modified only by an administrative action and not by its user activity.

So far, a presentation of the most common components of the $UCON_{ABC}$ model was given. However, its novelties in AC are accrued mostly from the rest of its components. The component of rights represents a number of privileges that can be held and exercised from a subject to an object. In a similar way to RBAC's roles, the UCON conceptual framework supports hierarchies among rights. Note that rights are not set a priori, but they are determined during

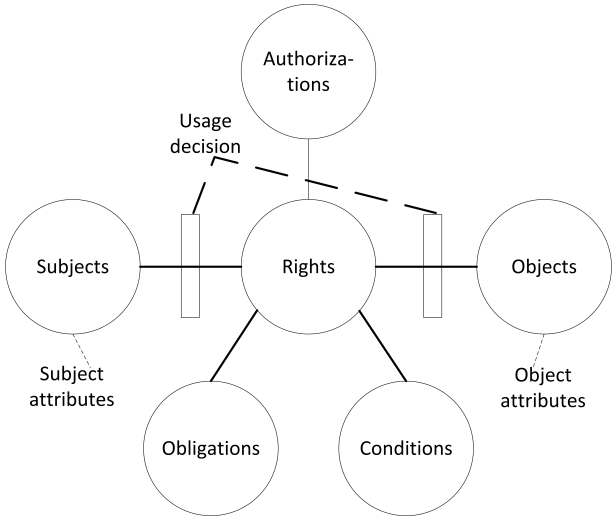

Figure 2.2: The $UCON_{ABC}$ model (Park and Sandhu, 2004).

the access. The access decision is given from a usage function by considering the following factors of subject and object attributes, authorizations, obligations and conditions. Authorizations in UCON are functional predicates, whose evaluation is used for taking decisions, namely if access to a subject is granted to an object. In a same manner with the usage function, the evaluation of authorizations is based on subject and object attributes, requested rights and a set of authorization rules. Authorizations can be characterized as pre-authorizations or ongoing-authorizations. The *pre* prefix refers timely before the requested right and the *ongoing* prefix during the time span of access.

Furthermore, obligations in UCON are used to capture the requirements that must be met from a subject requesting the usage of an object. They are also expressed as functional predicates and, as already mentioned, they are used in the evaluation of access both in the usage function as well as with authorizations. Obligations are also divided into pre-obligations and ongoing-obligations. The former is used usually for the retrieval of history information and the latter to check whether the requested requirement is fulfilled during the time span of access. Finally, conditions in UCON are used to capture factors that are accrued from the environment of the system. The semantic difference between conditions and other variables, namely authorization and obligation, is that the former cannot

be mutable since there is no direct semantic association with subjects.

2.4 AC mechanism implementations

As mentioned above, the terms of authorization and AC are used interchangeably. Nonetheless, the former definition is most commonly used in Grid systems. In this section, we will further analyse some of the AC mechanisms implemented in existing Grid middle-ware. A clustering of a number of implemented authorization infrastructures by the capabilities they support is provided in (Schlager et al., 2006). The AC architecture used in the majority of them is based on an attribute based approach. The main components in this architecture are the attribute authority (AA), the policy enforcement point (PEP), the policy decision point (PDP) and the policy authority (PA). This architecture is based on the AC framework recommended in (ITU-T, 1995). In X.812 the policy enforcement and decision point are referred to as Access Control Enforcement functions (AEF) and Access Control Decision function (ADF), respectively. The attribute authority is responsible for the generation and management of the subject, object and environment attributes. It is also responsible for the association of attributes with their owning elements as well as the provision and discovery of the attributes. The policy enforcement point requests and enforces access decisions coming from the policy decision point, which have to do with subject to object authorizations. The policy decision point is responsible for evaluating the system's policies and for decision taking. The decision for the granting or denial of access is passed to the policy enforcement point. Lastly, the policy authority is responsible for the creation and management of the authorization policies.

Furthermore, Grid authorization systems are also characterized by how the authorization of a user to a resource is achieved (Chakrabarti, 2007a). There are two different models used in the currently implemented Grid authorization systems. These are the push and the pull models. Most systems support either the former or the latter model. However, there are Grid authorization systems that support both of them. In the push model, a certificate generator usually creates certificates based on the user's credentials. Each one of the certificates is pushed on an access controller so as to grant or deny access to the resource,

based on the validity of the certificate. On the contrary, when the pull model is used by the authorization system, a minimum number of user credentials is provided to the access controller. In turn, it is the controller's responsibility to check the validity of the user based on the policies of the system. The push model is considered more scalable than the pull model. Nonetheless, the push model lacks usability, something in which the pull model is better since users do not have to obtain the certificate from the certificate generator. Moreover, the responsibility of granting access to a user is passed to the access controller.

Finally, Grid authorization systems can be categorized as virtual organization level and resource level systems (Chakrabarti, 2007a). The former refers to systems where a centralized authorization system handles the provision of credentials to the users, in order for them to access the resources. In opposition to the virtual organization level, systems that allow users to access the resources based on the credentials presented by the users are characterized as resource level ones. It is worth mentioning that as noted in (Chakrabarti, 2007a) the virtual organization and the resource level authorization systems cope with different aspects of the grid authorization. The first category of systems provides a consolidated authorization service for the virtual organization and the second category of systems implement the decision to authorize resource access. As a consequence, they complement each other and can provide a holistic authorization solution if combined.

2.4.1 Authorization Service (CAS)

The community authorization service (CAS) (Pearlman et al., 2002) is a virtual organization level authorization service developed by the Globus team. Its main objective is to cope with the flexibility, scalability and policy hierarchy issues, which primarily exist in Grid's security infrastructure (GSI) and GridMap, since the latter provides only a one-to-one mapping between global user names and local ones. CAS is capable of allowing the resource owners to grant access on portions of their resources to the virtual organization by letting the community determine who can use this allocation. CAS manages to overcome the limitations existing in GridMap by introducing a CAS server that operates as a trusted intermediary

between the users of the virtual organization and the resources. The CAS server is capable of managing all the policies that control the access to the resources of a community. It contains information about the users, resources, certificate attributes, servers as well as policy statements. According to CAS, a user has to contact the CAS server at any request to access a resource in a community. This requires from the user to be authenticated by providing the user's own proxy credential. The identity and rights that the user holds in virtual organizations are established by using its local database. In turn, the server issues a signed policy assertion with the user's identity and rights in the target virtual organization. The policy assertion is then embedded in a new proxy certificate generated by the CAS client. The new proxy certificate is used on the resource of the virtual organization to authenticate the user and to grant access to the resource based on the embedded policy assertion. The certificates used in CAS are X.509 extensions. The proxy credentials that authenticate the user on the CAS server have much longer span of life that the proxy certificates.

2.4.2 Virtual Organization Membership Service (VOMS)

The virtual organization membership service (VOMS) (Alfieri et al., 2003) is also a virtual organization level authorization service developed for the European Data Grid (EDG) that solves the same problems as CAS does in EDG. The VOMS system operates as a front-end on top of a database and it consists of four components, viz. the user server, user client, administration server and administration client. The user server receives requests from a client and returns information regarding the user. The user client contacts the server by presenting the certificate of a user or proxy to the latter and receives a list of groups, roles and capabilities of the user. The administration server is responsible for accepting the client's request and updating the database. Lastly, the administration client is used by the administrators of the virtual organization for administrative issues like the addition of new users, the creation of new groups and so on and so forth. According to VOMS, a bidirectional authentication of the server and the client occurs. During the authentication process, a safe communication channel is instantiated between them. In turn, the client can send a request to the server.

When the server receives the request from the client, the request is checked for its integrity and if no problem exists, the server sends a pseudo-certificate to the user. The client also checks the pseudo-certificate for its integrity. The user can now create a proxy certificate based on the received pseudo-certificate and present it to the resources to gain access on them. A user in VOMS is allowed to be a member of many virtual organizations and also to receive credentials from multiple VOMS systems.

2.4.3 GridMap

GridMap is the simplest and most widely used resource level authorization service. It is rather static and lacks scalability. GridMap is implemented as a file, which holds a list of authenticated distinguished names of the Grid users and their mapping with the equivalent account names of the local users. The policies that describe the access restrictions are kept in each local resource. AC is also left to local systems, so when a user requests access to a resource, the decision to grant or deny the access permission is based on the information present in the local AC mechanism and the local GridMap file.

2.4.4 Akenti

Akenti (Thompson et al., 2003) is a resource level authorization system that was created to cope with environments that consist of highly distributed resources and their use by multiple stakeholders. A stakeholder is defined as someone who controls access on a resource. Akenti consists of a resource gateway that operates as a policy enforcement point and of resources, which are accessed via the resource gateway. It makes use of X.509 certificates for the authentication of the users who request access to a resource. The communication between the user and resource gateway is accomplished through secure SSL/TLS channels. When a user requests access to a resource, access is determined by the combined policy on the resource. These policies can be created by different and unrelated stakeholders and are expressed with signed certificates. The resource gateway can ask from the Akenti server the privileges that a user has on a resource. The Akenti server operates as a policy decision point. In turn, the server retrieves

all the relevant certificates, checks their validity and sends a response back to the resource gateway. The latter enforces the operation indicated by the policy decision point. This architecture gives Akenti the ability to restrict access to resources based on predefined AC policies, without requiring the existence of a central administrative authority.

2.4.5 Privilege and Role Management Infrastructure Standards Validation Project (PERMIS)

PERMIS is a role based X.509 privilege management infrastructure and resource level authorization system (Chadwick, 2005; Chadwick et al., 2003) that supports the hierarchical RBAC model. The main components that constitute PERMIS are the PERMIS authorization enforcement point, the authorization decision point, the authorization policy and the privilege allocator. The first two components are responsible for the user authentication and decision making, respectively. The authorization decision point can retrieve policies and attribute certificates from LDAP servers and base its decision on the retrieved information. The descriptions of the policies are specified by the authorization policy. The content of the policies specifies who has access on which resource and under what conditions. The privilege allocator is responsible for the allocation of privileges to the users. The privileges are attribute certificates that include role to user associations. Additionally, a delegation issuing service provides the users with the ability to delegate a subset of their privileges to another user of their domain. When a user requests use of a resource, the authorization enforcement point authenticates the user. In turn, the enforcement point passes the user's distinguished name to the decision point. The latter retrieves information relevant to the user from an LDAP server. After performing the validation of the policies, the roles that are embedded in the attribute certificates are transferred as an object to the user. The user is authenticated in every attempt to access a resource. This results in the transfer of the object, which keeps the roles of the user embedded, from the enforcement to the decision point, so as to grant or deny access.

2.4.6 Usage based authorization framework

An attempt to apply a usage based authorization framework in Grid systems is presented in (Zhang et al., 2006). Subject and object attributes are used for the definition of usage control policies, and conditions provide context based authorization for the support of ad-hoc collaborations. Continuity of decision and mutable attributes are also supported. Yet, obligations are not supported. In the current state, the management of attributes is centralized. Nonetheless, in case of a distributed attribute repository, a lot of complexity is added since the system must keep all the multiple copies of the attributes consistent. The main components of the framework's architecture include a policy decision point and a policy enforcement point. The attributes and the identity certificates of users can be stored in attribute and identity authorities, respectively. When access is requested, the decision point makes the control decision based on the collected attributes and is enforced by the enforcement point. A notable feature is its support of a hybrid model that uses both the pull and push models to cope with the different types of attributes. Immutable attributes in the usage based authorization framework are pushed to the policy decision point by the requesting subject. On the contrary, when it comes to immutable attributes, they are pulled from the attribute repositories.

2.4.7 Cloud AC mechanism implementations

Regarding Cloud AC mechanisms, at the time of writing this dissertation, are not standardized. Most of CSPs are enterprises (i.e., Google, Amazon, SUN) and provide specific AC features. Furthermore, collaboration among Clouds can be cumbersome since there is no standard API across CSPs, and thus, makes very difficult to manage AC among participants. Lastly, CSPs lack of a support for granular privilege AC based on roles, which will lead to support of principles of least privileged and SoD (Mather et al., 2009, chap. 6).

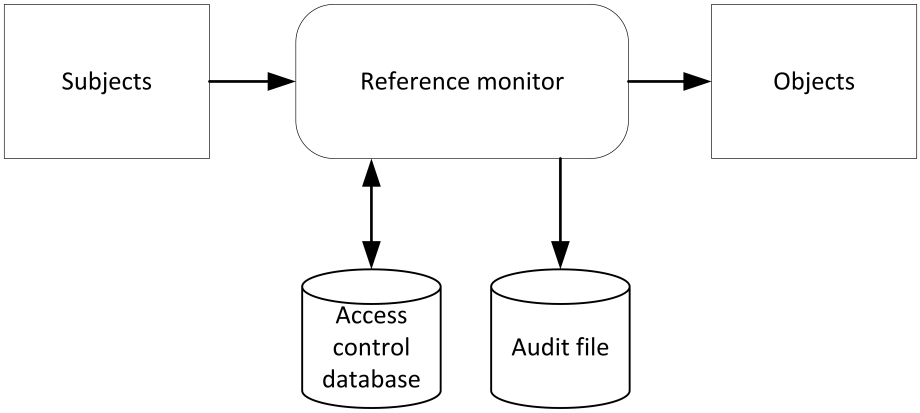

Figure 2.3: The reference monitor (Ferraiolo et al., 2003, chap. 2).

2.5 AC enforcement

In this section, a brief presentation of the reference monitor concept is given. This is mainly done because the application of the reference monitor concept is known to achieve high assurance AC mechanisms. Furthermore, it provides guidelines for the design and implementation of secure computer systems (Ferraiolo et al., 2003, chap. 2).

The process of AC in any computer system guarantees that any access to the resources of the system conforms to its AC policy. The application of the abstract concept of the reference monitor is capable of providing the requirements that are posed from the AC process. As it can be also seen in Figure 2.3, the reference monitor operates as an access mediator between the subject's access requests and the system's objects. The accesses comply with the system's security policy. The reference monitor can be informed for the security policy of the computer system from an AC database. Moreover, all the security relevant transactions are kept into an audit file for security and traceability reasons.

The architecture of the reference monitor is the result of the application of three key implementation principles. These principles are the completeness, isolation and verifiability. Completeness requires from the reference monitor to invoke all the subject's references to an object and also to constitute it impossible to bypass it. The isolation principle ensures that the reference monitor must be tamper-proof. This means that it must be impossible for an attacker to pene-

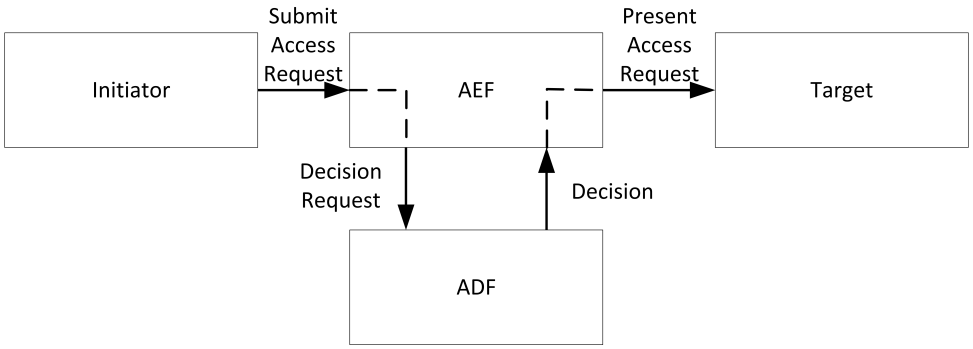

Figure 2.4: Fundamental AC functions in X.812 (ITU-T, 1995).

trate the reference monitor in a malicious way. Lastly, the verifiability principle appertains to the checking and validation of the system's security design through the use of software and system engineering techniques.

Nonetheless, the aforementioned reference monitor principles seem to be insufficient, especially in enterprise environments. This is mostly because the main objective of the reference monitor is the enforcement of each system's policy. Yet, it does not interfere with the articulation of a system's security policies. Thus, the principles of flexibility, manageability and scalability are introduced. The first principle ensures that the AC policy of an enterprise can be enforced by the existing security system. The next refers to the ease of policy management and the latter requires the security system to cope with the fluctuations in the number of the participating users and resources in a computer system.

The concept of reference monitor in open systems has been standardized with the X.812 AC framework (ITU-T, 1995). In brief, the main functions in X.812 are the Access Control Decision Function (ADF) and the Access Control Enforcement Function (AEF). The former component is responsible for the making of AC decisions. The decisions are made based on information applied by the AC policy rules, the context in which the access request is made, and the Access Control Decision Information (ADI). ADI is a portion in the Access Control Information (ACI) function, which includes any information used for AC purposes, including contextual information. Lastly, the AEF is responsible for the enforcement of the decision taken from the ADF. Figure 2.4 illustrates the fundamental AC functions in X.812.

2.6 Chapter summary

In this chapter, we introduced AC in distributed and collaborative systems. Specifically, we clearly defined AC in the context of both Grid and Cloud computing systems. We further identified the problem of multi-domain administration is such systems. A comprehensive presentation of prominent AC model was performed, as well as a survey of existing AC mechanisms for Grid and Cloud systems. Lastly, we presented the standard in enforcing AC. All the aforementioned information should be taken into consideration when building a new AC model or system for collaborative systems.

Chapter 3

Methodology for AC systems development

3.1 Introduction

AC systems are highly complex and of great significance. Therefore, their explicit definition, design and development are mandatory for the production of AC systems that correspond to their initial requirements. To manage the increased complexity of AC systems, systems engineering (SE) processes are applied. Specifically, in this chapter we recall the SE and requirements engineering (RE) processes based on (Kotonya and Sommerville, 1998; Siddiqi and Shekaran, 1996; Sommerville, 2001; Stevens, 1998), and provide further information regarding the process of verification.

The remainder of the chapter is organized as follows. Section 3.2 refers to SE and RE, and the verification process. Section 3.3 refers to the proposed methodology and, lastly, section 3.4 concludes this chapter.

3.2 Systems engineering

SE is concerned with the problems of specifying, designing and integrating large-scale socio-technical systems. Socio-technical systems are complex, organizational systems which include people, organizational processes and, sometimes, other

more specialized hardware systems (Sommerville, 2010, chap. 10).

A significant distinction between a system and some other collection of things is that a system has emergent properties, i.e., properties that only become apparent when the system is put together. Some emergent properties are predictable and the goal of the system design is to ensure that the system has these properties. Other emergent properties are planned, but not predictable, and other are unplanned, i.e., the designers didn't consider them, and therefore, can often cause problems. The structure of the system is the way in which the system components are put together and interconnected. Thus, if a system is put together in a wrong way, then it won't have the expected emergent properties (Sommerville, 2010, chap. 10).

In this dissertation, we are mostly concerned with the security properties in an AC system. The security of a system is a system property that reflects the system's ability to protect itself from accidental or deliberate attack. Security is an essential prerequisite for safety, reliability, and availability. Safety is concerned with ensuring that the system cannot cause damage irrespective of whether or not it conforms to its specification. Reliability is concerned with conformance to a given specification and delivery of service. Lastly, availability refers to the ability of a system, at a point in time, to be operational and able to deliver the requested services (Sommerville, 2010, chap. 14).

SE consists of a number of processes. A process consists of a structured set of activities that is intended to achieve some outcome. Thus, a SE process is a set of engineering activities whose goal is to produce a system. A simple model of a SE process could suggest the model to be linear with a smooth transition from one phase of the process to another. However, in practise it's never as simple as that since different sub-systems are inevitably at different stages of development, problem might arise which affect some, but not all parts of the system, and so on. An example of a system's life-cycle includes the processes of requirements engineering, architectural design, component development, and the integration and verification of what has been built (Stevens, 1998, chap. 2).

Following, we provide information regarding the process of requirements engineering and verification.

3.2.1 Requirements engineering

This section is intended as an overview of requirements engineering, which is being increasingly recognized as the most critical phase of the systems development process (Siddiqi and Shekaran, 1996). If the requirements for a system are not right, there will inevitable be problems after the system is delivered. There are immense variations in what is generally understood as a requirement. The term *requirement* might be used to refer to statements that are clearly at totally different levels of detail. These variations in the statement of requirements arise because of the different ways requirements are used in different organizations. Therefore, a statement of a requirement can range from a high-level abstract statement of a service to be provided or of a system constraint to a detailed mathematical functional specification of a system component (Kotonya and Sommerville, 1998, chap. 2).

Requirements can be identified into four types, as follows.

- Functional requirements, where define some functionality to be provided by the system (i.e., what the system will do).
- Non-functional requirements, where define some operational constraints on the behaviour of the system (i.e., how the system should behave).
- Design requirements, where define constraints on the system design or implementation.
- Process requirements, where define constraints on the system development process.

Although it is convenient to classify requirements into different classes, there is not really a strict distinction between these. Specifically, non-functional requirements at one level of abstraction are translated into more detailed functional requirements.

Regarding requirements there is a need for specifications at different levels of detail. The principal reason for this is that these different specifications are designed for different purposes and readers. Requirements definition or specifications can be classified into three types, as follows.

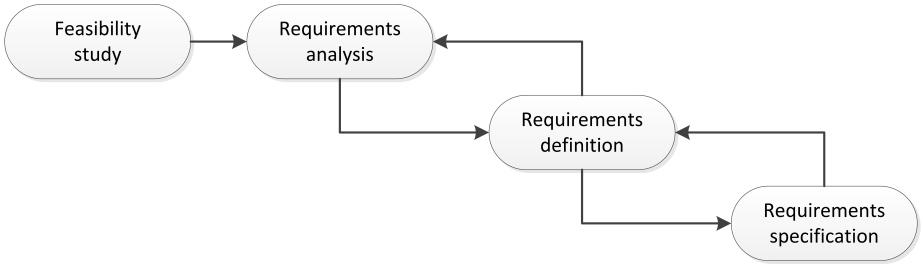

Figure 3.1: Requirements engineering process (Sommerville, 2001, chap. 1).

- User requirements definition.
- System requirements specification.
- Software or hardware specifications.

User requirements definition has to be understandable by potential end-users of the system and their managers as well as the developers of the system. The system requirements specification is a more detailed document that is usually part of the contract for the system. Lastly, software or hardware specifications include a detailed description that can serve as a basis for a design or implementation.

Figure 3.1 illustrates the activities of the requirements engineering process. Despite the fact that there is a clear division between process activities, in practice, the process is iterative with the activities interleaved and with very blurred boundaries between them. Specifically, analysis definitions and specification may be seen as a single activity. In more detail, the feasibility study activity is clearly related to the feasibility study activity that takes place during the conceptual design. Requirement analysis is the process of finding out the requirements and requirements definition are intended to be a high-level description of the system requirements. Lastly, system requirements specification define the requirements in detail as a basis for the contract for the system procurement and for the system developers.

3.2.2 Verification

The principal methods for the verification of complex systems can be grouped under four types. These are testing, simulation, deductive verification, and model

checking (Heljanko, 2006). Testing is performed on the system itself. However, testing of distributed systems is not always a cost effective process since it can be performed when an implementation of the system is available. Furthermore, it can only prove the existence of bugs, but not their absence. Simulation-based approaches ensure that a finite number of user-defined system trajectories meet the desired specification. However, simulation suffers from completeness as it is impossible or impractical to test all system trajectories. Furthermore, simulation-based testing is semi-automatic since the user must provide a large number of test cases. Deductive verification is based on manual mathematical proof of correctness of a model of a system. It is a very highly cost process and, furthermore, requires highly skilled personnel. Model checking verifies a system's model against defined properties; it is not vulnerable to the like-hood that an error is exposed. This contrasts with testing and simulation that are aimed at tracing the most probable defects. Additionally, it provides diagnostic information in case a property is invalidated, which is very useful for debugging purposes. In principle, model checking is an automated process and its use requires neither a high degree of user interaction nor complex test data. Furthermore, it does not require the development of custom tools for verifying a system, which can be a time-consuming and error-prone process. On the contrary, verification via model checking can be applied using existing model checkers. Therefore, model checking can serve as a technique to detect non-conformance between the AC system and its specifications (e.g., secure inter-operation) as efficiently as possible. It is also noteworthy that when using model checking it is feasible to perform a security analysis of a system. Security analysis generalizes safety analysis since with security analysis we can study not only safety, but also several other interesting properties (e.g., mutual-exclusion) (Li and Tripunitara, 2006).

3.3 The proposed methodology

In this section, we provide information regarding our proposed methodology for the development of AC systems. The methodology is independent of the applied development model of a system since the stages of requirements engineering and verification exist in most of them (e.g., sequential, spiral and so on). Figure

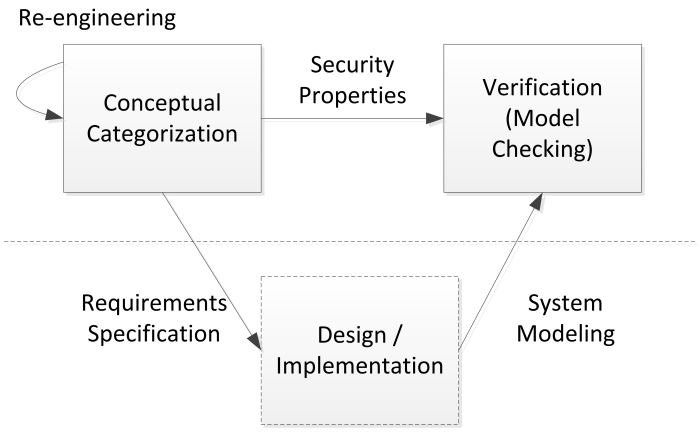

Figure 3.2: System development process.

3.2 illustrates the proposed methodology in a system development process that consists of the stages of requirements engineering, systems design and verification. We propose for the stage of requirements engineering to apply the Conceptual Categorization (CC) (Gouglidis and Mavridis, 2010) and during verification to apply model checking techniques. CC is capable of tailoring the requirements engineering through a re-engineering process. Furthermore, CC operates as an input for the verification stage by defining security requirements. The security requirements are transformed into security properties in temporal logic (e.g., LTL, CTL). The set of defined security properties can be verified on the system's transition system (TS) using model checking techniques.

In our proposed methodology, we are mostly concerned in performing security requirements engineering and verification of the system to be developed. Therefore, we are not concerned with other stages of the development process, as the design and implementation of the system. However, we depict in Figure 3.2 their interaction with the analysed stages. It is noteworthy to mention that the proposed stages can be used transparently in any development model without breaking it because of system's engineering modular approach.

Fundamentally, security requirements can be identified during a requirement engineering stage, transformed into properties (e.g., using temporal logic) and be verified on the transition states of an AC system (Hu et al., 2011). In this dissertation, we elaborate on the verification of properties in RBAC systems using

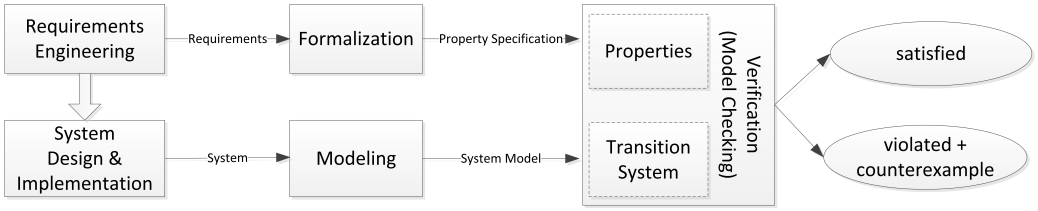

Figure 3.3: System development and verification process.

formal methods. Specifically, through requirements engineering, we provide a set of formally defined security requirements as properties to a model checking mechanism to verify the conformance to the formal RBAC model. Figure 3.3 illustrates a schematic view of the methodology. Verification is a critical process well separated from the previous stages of requirements engineering, systems design, and implementation. Verification is used in the comparison of the initial conceptual system based on defined requirements to the computer representation that implements that conception, and concerned with building the system right. Specifically, it must ensure that the system does what it should, only the way it should, and does not do what is should not do (Stevens, 1998).

For the verification of security properties, we are interested in applying a technique that is able to support the verification of properties in RBAC policies. Additionally, exportation of the verified AC policies in the eXtensible Access Control Markup Language (XACML) is highly desired since it is becoming the de facto language for the description of policy rules in modern collaborative systems, as the Grid and Cloud computing paradigms (Gouglidis and Mavridis, 2012a). Furthermore, it should be easy to express security properties regarding secure inter-operation and to successfully verify them against multiple RBAC policies that can be composed to a global security policy. Consequently, the initial security requirements (i.e., secure inter-operation) of the conceptual AC system will be verified in the implemented AC system.

Several papers have examined the automated verification of AC models and generic policies, and a number of techniques have been proposed to verify them (Hansen and Oleshchuk, 2005), (Hu et al., 2008), (Sun et al., 2011), (Jayaraman et al., 2011), (Fisler et al., 2005), (Hu and Ahn, 2008), (Hughes and Bultan, 2008), (Jha et al., 2008). The great number of different techniques is mostly

the result of the need for more expressive power or better performance. Several of them use a verification tool as back end. Such tools are for instance, Alloy (Alloy), a declarative language with support of first order logic and relational calculus, NuSMV (NuSMV), a symbolic model checker that verifies temporal logic properties in a finite state system, the SPIN model checker (SPIN) and so on and so forth. However, there are cases where AC policies are defined as ordering relations, which are further translated to Boolean satisfiability problems and applied to SAT solvers (Hughes and Bultan, 2008). A SAT solver is a program that takes a formula in conjunctive normal form (CNF) and returns an assignment, or says none exists. These techniques can serve as a foundation for the verification of specifications of a system. A specification of a system can be defined as *"what the system is supposed to do"* (Lamport, 2002).

We choose to apply the technique proposed in (Hu et al., 2008), which focuses on the verification of generic properties for AC models. The technique is able to cope with various types of AC models including static, dynamic, and historical. It also supports the generation of test cases to check the conformance between models and policy rules through combinatorial test array (NIST, 2012), and optionally generate the verified AC policies in eXtensible Access Control Markup Language (XACML) version 2.0 or 3.0. We adopt the finite state machine to describe the transitions of the authorization states, and the usage of static constraints so to adequately cover the verification of secure inter-operation properties in RBAC. The technique is to verify specified AC properties against AC models using a black-box model checking method (Hu et al., 2011). An implementation, Access Control Policy Tool (ACPT) (Hwang et al., 2010), is developed by NIST Computer Security Division in corporation of North Carolina State University.

ACPT provides graphical user interface (GUI) templates for composing AC policies and properties. Checking for conformance of AC properties and models is through the SMV (Symbolic Model Verification) model checker. In addition, ACPT provides a complete test suite generated by NIST's combinatorial testing tool ACTS (NIST, 2012) and an XACML policy output for the verified model. Through these four major functions, ACPT performs syntactic and semantic verifications as well as the interfacing for composing and combining AC policies. ACPT assures the efficiency of specified AC policies, and detects policy faults

that leak or prohibit legitimate access privileges. Currently, ACPT provides model templates for three major AC models: static Attribute-Based AC, Multi-Leveled Security, and stated Work-Flow, and partially implements the methods described in (Hu et al., 2011). Despite providing all the adequate functionality for the verification of AC policies, the function of RBAC reasoning regarding role hierarchies is absent. Nevertheless, we applied this model checking technique for its capabilities of defining and verifying basic RBAC rule statements and property propositions.

3.4 Chapter summary

In this chapter, we discussed the importance of the systems engineering process, and further elaborated on the requirements engineering and verification processes. Since complex AC systems require the usage of SE processes for their development, we described our proposed processes in the context of a SE engineering methodology for the development and verification of AC systems. The requirements engineering stage is based on CC (see Chapter 4) and the verification stage on a sound and mathematical underpinning technique (i.e., model checking) (see Chapter 6). The proposed methodology can be applied on any existing development process since it does not break the development model.

Chapter 4

AC requirements engineering approach

This chapter presents our proposed Conceptual Categorization (CC) for Grid and Cloud systems, which is capable of enhancing the existing requirements engineering process since it facilitates the identification of requirements. Additionally, CC can be used as a comparative tool for AC models and mechanisms.

4.1 Introduction

The Grid and Cloud are two promising computing technologies. However, contemporary implementations are characterised by an intrinsic complexity due to lack of standards, ad-hoc implementations and use of approaches which are not specifically designed for these environments. AC is such an example. Security system designers need to define AC approaches that can cope with the complexity of these environments. Requirements engineering can be used as a process in their development. However, an approach that incorporates the characteristics of these systems is non-existent. Therefore, we identify the need for a holistic approach in AC requirements definition that will enhance and facilitate the process of their identification, and that will result in new AC models for Grid and Cloud computing systems.

The remainder of this chapter is structured as follows. Section 4.2 presents

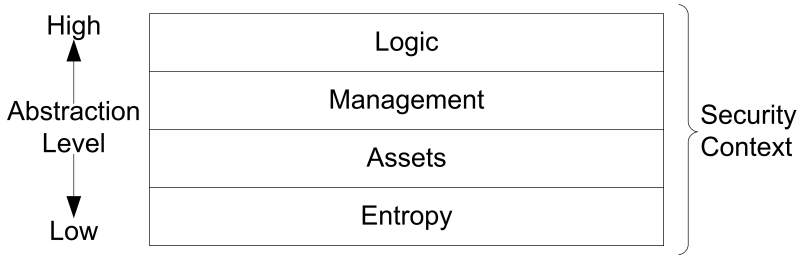

Figure 4.1: CC layers.

the proposed CC for Grid and Cloud systems. A motivating scenario is presented in Section 4.3, to demonstrate the identification of AC requirements based on the proposed categorization and assess their implementation in relation to contemporary approaches. Section 4.4 uses the CC to compare the AC models and mechanisms examined in Chapter 2. Finally, the chapter is concluded in Section 4.5.

4.2 The proposed conceptual categorization

Current Grid systems have been categorized and classified in the existing literature based on different criteria, either qualitative or quantitative. Most of these categorizations are quite vague, in regard to the limits of each category (Alexander Kipp, 2008). This makes the definition of AC requirements a cumbersome process. Moreover, despite the use of generic systems engineering processes, security engineers lack an auxiliary abstract approach able to enhance and facilitate the definition of AC requirements. As a solution, a conceptual four-layer categorization that is capable of defining and evaluating security requirements is proposed.

As depicted in Figure 4.1, the proposed CC is based on four abstraction layers: entropy layer, assets layer, management layer and logic layer. The differentiation from generic security engineering approaches is that, in our case, factors that affect the security of the systems are mainly considered in their categorization. Briefly, the CC identifies and groups security requirements into discrete layers of different abstraction level. The abstraction level refers to the ability of a layer to identify requirements in different breadth and depth. The entropy layer

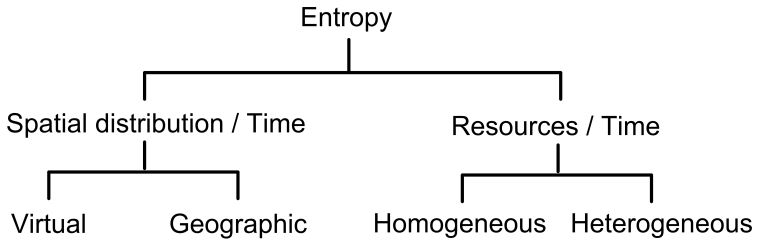

Figure 4.2: Entropy layer classification.

identifies requirements from the dispersion of the objects in a system and the assets layer from the type of shared objects within the boundaries of the entropy layer. The next layer defines requirements from policy management and the logic layer incorporates requirements that are not handled by the former layers.

4.2.1 Entropy layer

Entropy is a layer capable of capturing the abstract characteristics of a system stemmed from its distribution. The term entropy refers to the virtual and geographic distribution of a system in association with the factor of time. Current classifications of Grid systems are static and based mostly on the geographic distribution of their resources (Gridipedia, 2009) or on their size (Kurdi et al., 2008). The entropy layer uses existing Grid distribution characteristics and the incorporated time factor in order to identify changes in the number of participating objects as well as alterations of them over time. Figure 4.2 depicts the entropy layer classification.

In order to illustrate the flexibility of this layer regarding the capture of the distribution characteristics of a system, we provide the examples of a volunteer desktop Grid project named SETI@home (SETI@home, 2009) and of a science Grid project named EGEE (EGEE, 2009). Data from (BOINC, 2009) and (Gridmap, 2009) are depicted in Figures 4.3 and 4.4, respectively. The entropy lines represent the fluctuations in number of the involving objects, in relation to the spatial distribution over time. Issues like the authentication of the distributed objects can be examined under the entropy layer.

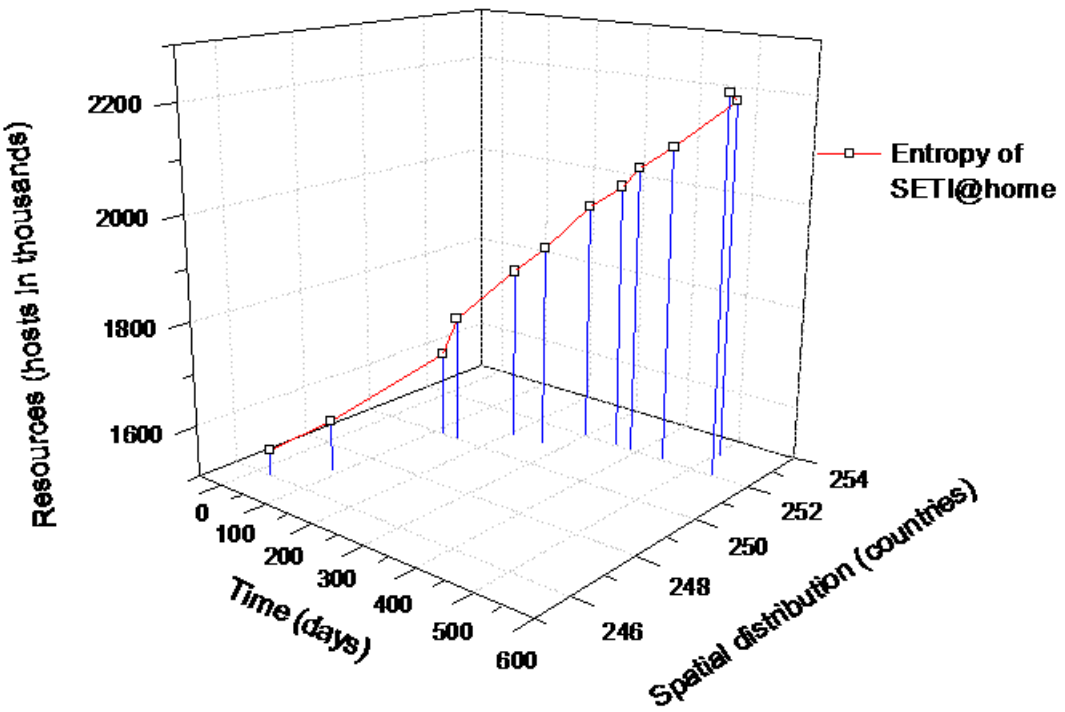

Figure 4.3: Entropy of the SETI@home.

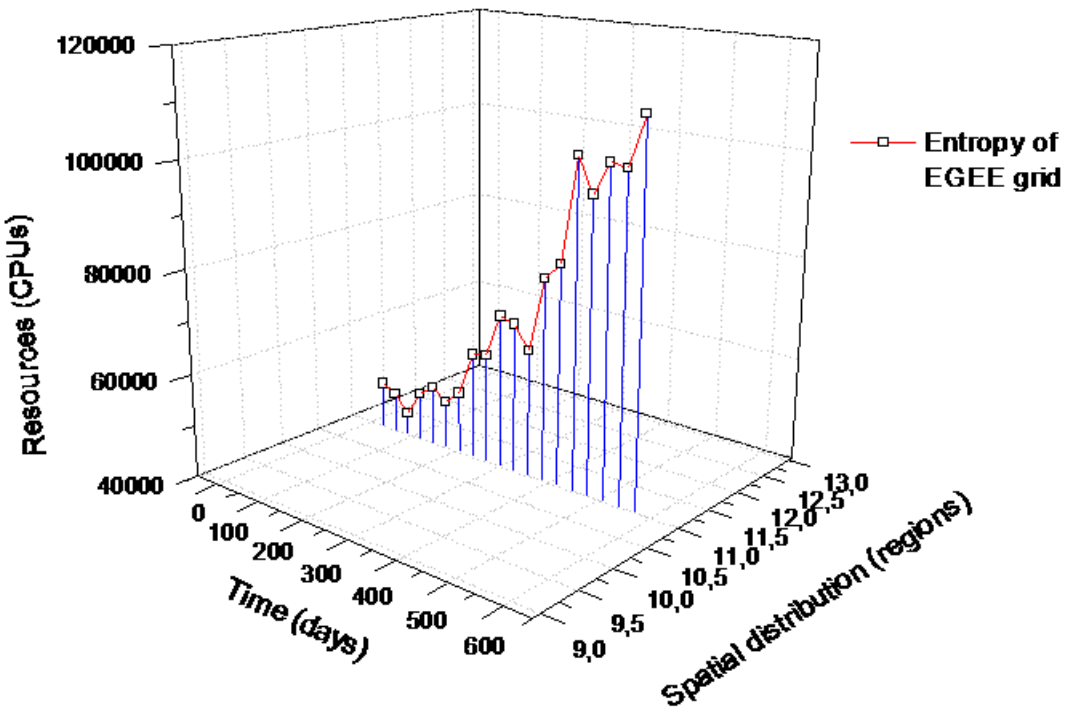

Figure 4.4: Entropy of the EGEE Grid infrastructure.

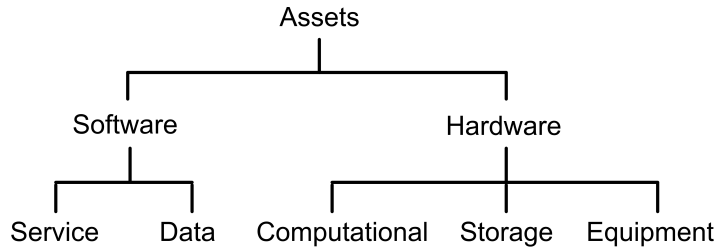

Figure 4.5: Assets layer classification.

4.2.2 Assets layer

The assets layer, as illustrated in Figure 4.5, is used to wrap all the assets in a system. As an asset, we define any shareable object in a system. In a Grid or Cloud system, an asset can either be of software or hardware type. The proposed classification in the assets layer is partially based on the existing literature (Green, 2002; Krauter et al., 2002; Kurdi et al., 2008). Under the software class, we further divide assets into two subclasses, these of service and data. Services have been used in the grid due to the adoption of service oriented architecture. The provision of fine-grained assets such as data is vital in distributed and collaborative systems. The requirement of sharing information at data-record-level in a database management system among a number of users is an illustrative example (Broadfoot, 2003). Similarly, we divide the hardware class into three distinct subclasses, those of computational, storage and equipment. Examples of computational assets are the usage of CPU or RAM of a system. Concerning the storage assets we refer to the usage of raw storage space for the saving of data. Last but not least, an equipment is an asset that is usually used as an input or output device within a grid system.

4.2.3 Management layer

The management layer is used to address the need for capturing the security issues stemmed from the management of policies among the objects in a system as well as from trust relationships. Figure 4.6 illustrates the proposed classification. The distribution level of a system, as defined in the entropy layer, affects the management of its policies. Usually, science Grids with a high-level

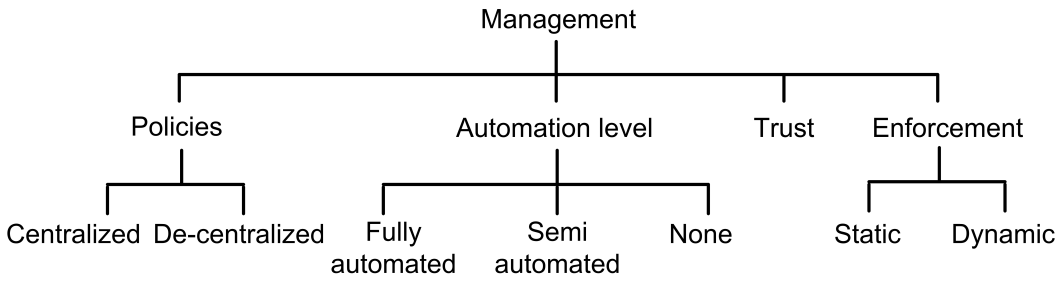

Figure 4.6: Management layer classification.

of distribution require de-centralized management and vice-versa. Peer-to-peer networks are an example of de-centralized management, too. On the contrary, enterprise applications using Cloud computing technologies require centralized management. The enforcement of several management operations is another factor that needs to be further classified. Here, we identify two classification levels that of static and dynamic enforcement. By static we refer to operations that can take place before and after the execution of a number of actions performed on an object by a subject. The dissimilarity between static and dynamic enforcement of operations is that, in the latter, the policy enforcement can also take place during the execution of an operation. The automation level pertains exclusively to the intervention of an administrator to the management routines. Fully automation means that management is done by the system itself (Kephart, 2005). Semi automated systems are those that are partially managed by the system itself and the administrators. However, cases still exist where management automation is completely absent. Such systems are solely administered by humans. Operations, such as problem identification, conflict resolution and revocation of privileges should be considered under the management layer. Finally, trust management must be taken into consideration in the process of security engineering. The life-cycle of trust includes the creation, negotiation and management of it (Chakrabarti, 2007b) and is considered an important part of security.

4.2.4 Logic layer

The main concern of the logic layer is the application models and the type of their execution in a system. Based on the definition of Grid and Cloud computing

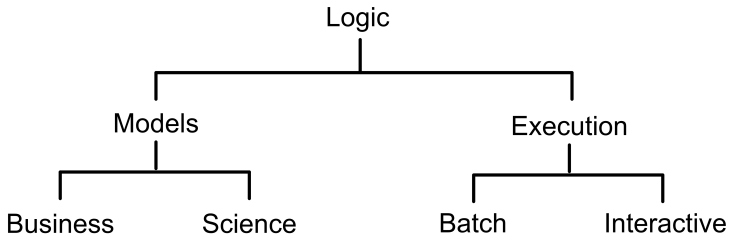

Figure 4.7: Logic layer classification.

systems and requirements identified in the existing literature (Alexander Kipp, 2008; Altmann and Veit, 2007), the classification of the logic layer as depicted in Figure 4.7 is suggested. The logic layer is split into two classes. The models class helps in the identification of security requirements that can rise from the nature of the application being executed in the Grid and Cloud environment. We propose a further classification of it into business and science applications. However, in both subclasses similar requirements exist. Usually the support of collaborations, work-flows and co-operations fall under science projects. In addition, technologies such as Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS) and Software-as-a-Service (SaaS) are enterprise examples, which are usually met in Cloud computing systems (Foster et al., 2008). Furthermore, a classification of the execution mode of a Grid or Cloud application into batch and interactive can be made. Science projects usually require a batch-based execution of applications to provide results through the computation of data. In contrast, most business applications require an interactive environment to tackle the highly dynamic enterprise environment.

4.2.5 Re-engineering in CC

In addition to the simple, sequential development process, the CC approach can be used for tailoring the process of requirements engineering. This is required since in practise new requirements always emerge that inevitably leads to implementation changes. CC can be used for re-engineering existing systems. Therefore, it can be applied on old systems, which usually have a considerable amount of intelligence and experience encapsulated within them. Figure 4.8 depicts the basic steps that help to re-engineer existing systems using an evolutionary life-

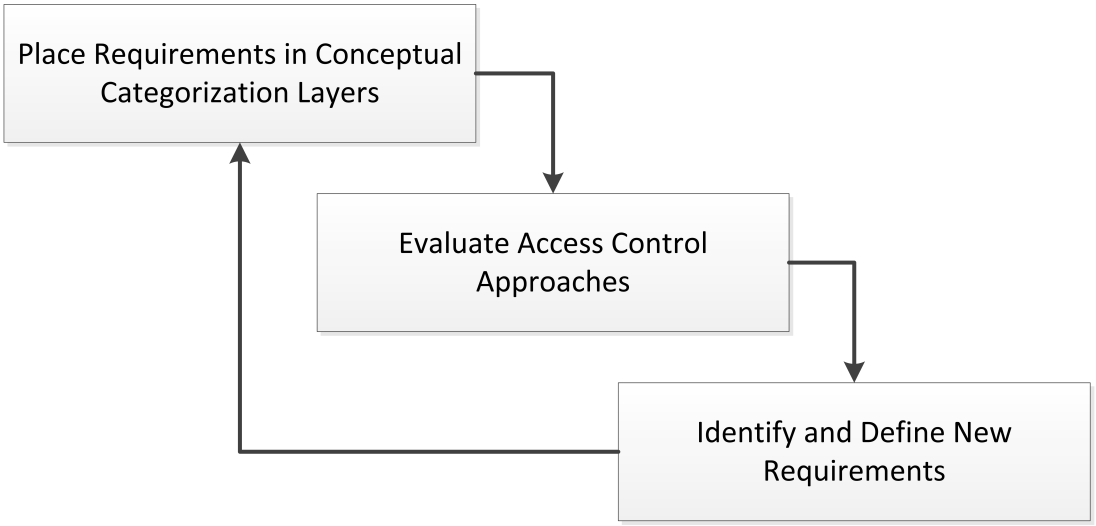

Figure 4.8: Re-engineering in CC.

cycle. The process can be seen as a spiral where cycling through the different processes leads to the desired outcome. Specifically, the evolutionary life-cycle includes the placement of requirements in CC layers. In turn, an evaluation of the examined system is performed. This second process, checks the compliance of the already defined requirements. If expectations from the examined systems are low, then new requirements are identified and defined, which when applied will potentially lead to a new system. New requirements are placed in CC layers and the process is repeated. This evolutionary life-cycle helps for faults to be found more quickly and provides the opportunity to include updated technology, and at the same time it facilitates the whole process of delivering a fully functional system (Stevens, 1998).

4.3 Identifying AC requirements

A generic Grid AC scenario, enriched with some of Cloud computing characteristics, follows. We demonstrate the process of identifying AC requirements for the scenario by applying our proposed CC. The operational environment is illustrated in Figure 4.9. The Virtual Organization (VO) is composed of individually admin-

istered domains, which can dynamically join in or quit the collaboration. Users from the participating domains can request on demand usage of Grid services. More precisely, the VO is composed of companies A and B, represented respectively by domains A and B. An Application Service Provider (ASP) is a corporate organization that can share a number of pay-per-use services. A complementary entity provides a computational computing infrastructure (CCI). Users Alice from company A and Bob from company B require collaborating and producing a statistical analysis on a subset of their data. Figure 4.10 illustrates the information flow between the involving entities, on VO level. Users can request capabilities from their local domain, collaborate with other users, manage their data and request on-demand use of services. Services can be administered and also permitted to use segments of users' data via a delegation mechanism. In turn, a service can submit data segments to the CCI. Services can be provided as composite services, and thus, requiring automatically re-delegation mechanisms among the involving services. The system may prompt users for parameters completion during an operation, whose life-span can vary, depending on the complexity of the computations. At the CCI, the resource owner can alter any AC policy for any resource and user at run-time. For instance, let's assume a policy that permits the execution of the statistical analysis application at the CCI for both Alice and Bob. However, prior to the statistical analysis completion, the resource owner restricts Bob's access with a new policy permitting him to use CPU cycles only when CCI is idle, and thus, leading to a delay of his computations, until the completion of Alice's job.

Entropy requirements: The virtual distribution level of the system is low since there is only one formatted VO. On the other hand, the geographic distribution level that depends on the number of the participating domains can be high, which additionally entails heterogeneity issues. In order for the AC system to limit access to participating objects, it must be able to successfully authenticate them since domains may make use of different authentication protocols. Furthermore, since the VO formation is not static, the AC system must continually observe all kinds of modifications. As far as this scenario is concerned, UCON can cope with the complexity of the entropy layer. This is due to the support of attribute repositories that can be dispersed across the domains. The use of

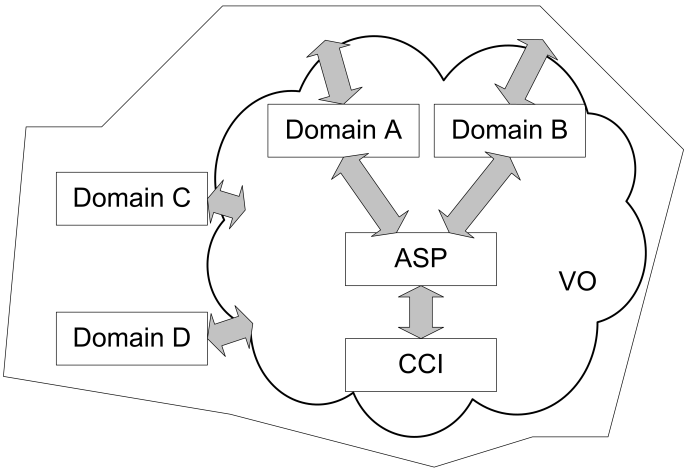

Figure 4.9: Operational environment.

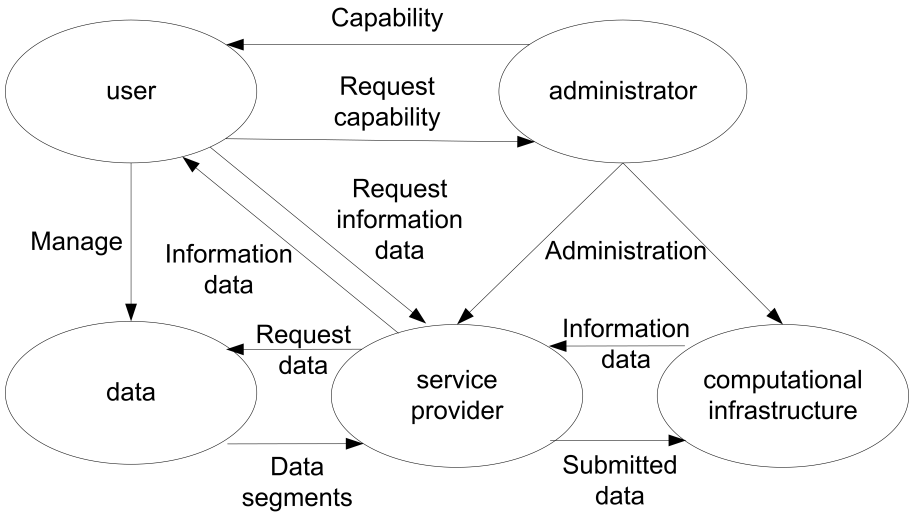

Figure 4.10: Flow of information in a VO.

attributes also overcomes the heterogeneity issues. UCON is flexible enough to deal with the dynamic changes in the number of participants during the collaboration. On the contrary, RBAC handles better centralized architectures where participants are known a priori. Therefore, RBAC appears to be inappropriate for the current scenario and layer.

Assets requirements: AC must be enforced on different types of assets. The scenario considers fine-grained AC on data since it requires sending for computation only segments of users' data. The ASP provides a number of services and the CCI a number of hardware resources. AC for service and hardware levels can be characterized as coarse-grained, since the scenario describes only permission, denial and restriction of access upon them. Thus, the AC model must be able to enforce fine-grained AC on data and coarse-grained on services and hardware resources, respectively. UCON can handle fine-grained AC because of attributes. RBAC is rather coarse-grained compared to the former approach when it comes to assets definition. Assets, in RBAC, are grouped under roles and in order to become more granular, the assignments must be split into more. However, the use of context variables in known RBAC variations (Tolone et al., 2005) overcomes such limitations. Once again, the UCON approach is preferred since it supports natively fine-grained AC, and because it is easier to modify in order to support course-grained AC than for RBAC to support fine-grained AC.

Management requirements: In this scenario, a number of services uses segments of users' data and submits them at the CCI. This requires a delegation mechanism. Thus, the AC system must be able to support delegation of access rights from grid users to the ASP and CCI. A security issue consists the revocation of a delegated right. We assume that delegated rights must be revoked after the completion of a job or on demand by the user. The former requirement demands from the AC system an automation level and the latter to apply changes dynamically. Furthermore, trust relationships must exist between the involving parties. In another use case, a user from an unknown domain may request to use a service. The AC system must be in position to decide whether to deny or provide limited access to the user. Policy conflict resolution must also be examined when composite services exist. This is required due to the inheritance of authorization rights amongst services. Delegation of rights and trust relationships are

supported by both AC approaches. Policy conflict resolution can be cumbersome for UCON, and easier for RBAC. In this case, a sensible choice would be the selection of RBAC, since it supports improved administrative capabilities compared to UCON. Revocation of user assignments, hierarchies and temporal constraints are some of RBAC's virtues making it superior in comparison to UCON.

Logic requirements: During Bob's collaboration with Alice, his access at the CCI has been restricted by the resource owner. This requires an AC system that must support dynamic collaborations. Occurring interactions between the user and the application require from the AC system to support them as well. More requirements are the support of stateful sessions due to long lived transactions and decomposition of composed services. UCON is the only approach capable of supporting interactive environments via continuity of decisions and mutable attributes. Moreover, the use of obligations can handle well a number of business requirements. However, topics like service decomposition are left intact from all AC approaches.

4.4 Comparison of AC models and mechanisms

In this section, the AC models and architectures described in Chapter 2 are compared. The comparison is attempted with respect to the CC with a view to specify a number of deficiencies in the examined models and architectures. The criteria used throughout the comparison are based on the requirements that were defined and the evaluation is based on the level of fulfilment of the requirements by the AC models and architectures, respectively.

4.4.1 Comparing the AC models

Table 4.1 illustrates the evaluation of the RBAC and $UCON_{ABC}$ models with respect to the entropy, assets, management and logic layers in the CC.

Concerning the entropy layer, the requirements that were defined, demand both the support of AC among different domains and the dynamic joining of new ones. The proposed standard RBAC model, as already seen, handles better centralized architectures and is rather weak in inter-domain collaborations. Such

functionality is absent from the standard model. However, research in (Shafiq et al., 2005) has proven that RBAC can also be applied in multi-domain environments where distributed multiple organizations inter-operate. Yet, RBAC requires that all user domains must be known a priori, in order to access an object. On the contrary, the $UCON_{ABC}$ model, due to its support of attributes, can cope better with highly distributed environments. Furthermore, one of UCON's features 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.

Table 4.1: Comparisons between the different AC models.

Access Control Models	Entropy	Assets	Management	Logic
RBAC	Low / Medium	Low Medium	Medium / High	Medium
$UCON_{ABC}$	High	Medium	Low	Medium

In regard to the layer of assets, we mentioned that fine-grained access to resources should be supported. Additionally it should support obligations from the side of the resource provider. RBAC usually provides more course-grained AC to resources in contrast to $UCON_{ABC}$. Research has also been done in RBAC to extend it and to support finer-grained AC through the use of context (Tolone et al., 2005). Obligations are supported in $UCON_{ABC}$, but not in the notion demanded by the requirements. The notion of obligations is completely absent in RBAC.

RBAC supports improved administrative capabilities on the level of a domain in comparison to $UCON_{ABC}$. In more detail, RBAC can also provide management in a role-based fashion (Ferraiolo et al., 2003, chap. 8). However, a number of issues arise when it comes to inter-domain management of policies, and solutions are provided in existing literature (Shafiq et al., 2005). In contrast to the RBAC, $UCON_{ABC}$ lacks administration.

Finally, the fulfilment of requirements in the logic layer is fairly the same in both AC models. Nonetheless, RBAC supports the principles of SoD and least privilege more efficiently.

4.4.2 Comparing the AC mechanisms

Table 4.2 depicts the evaluation of the AC mechanisms with respect to the entropy, assets, management and logic layers in the CC, while Table 4.3 illustrates a summary of the comparison. Besides the specified requirements, in our evaluation, we consider a list of extra parameters as stated in (Chakrabarti, 2007a). This is due to the adoption of an attributed based approach with strong resemblance by the authorization systems, and thus, making their evaluation more difficult.

The parameters of interoperability, user and mechanism scalability were taken into account in the layer of entropy. Besides the GridMap authorization system, the rest of them handle interoperability well. This is mainly due to the support of standard protocols, namely the SAML and XACML. The support of attributes helps in the fulfilment of the requirements we have defined for the entropy layer. User scalability is affected by two factors. These are the authorization model in use and the type of policy management. Usually systems that support a push based model and a centralized management of policies are less complex. In overall, GridMap exhibits the worst performance in the entropy layer, while CAS, VOMS, PERMIS and Usage based authorization the best.

Regarding the evaluation of the authorization systems for the layer of assets, we examined their ability to permit multiple users to control access on the same resource. As depicted in Table 4.3, VOMS and PERMIS are able to support multiple stakeholders on a resource. In regard to the parameter of obligations, only the Usage based authorization system supports it. Yet, obligations are from the side of the user and not from the resource provider.

The evaluation of the management of policies is based on multiple parameters, namely the administrative overhead, revocation of attributes, decentralized management, ease of management and automation. As we already mentioned, ABAC approaches lack management. Nevertheless, they provide support of decentralized management and require low administrative overhead in most implementations. Automation of procedures is absent or weakly supported. Lastly, revocation of privileges is present mostly in resource level solutions, and encounter problems in the rest of them.

Table 4.2: Comparisons among the different AC mechanisms.

Mechanisms															
	Entropy			**Assets**		**CC Layers**						**Management**			**Logic**
CAS	+	+	+	-	-	-	+	-	O	-	-	O	O	-	
VOMS	+	+	+	+	-	-	+	-	O	-	-	O	O	O	
GridMap	-	O	-	-	-	O	-	+	O	-	+	O	O	-	
Akenti	O	O	-	+	-	+	+	+	O	O	+	O	O	-	
PERMIS	+	+	+	-	-	+	+	+	O	O	+	O	O	+	
Usage based authorization	+	+	+	-	O	+	+	-	-	O	+	O	O	+	
	Interoperability	User scalability	Mechanism scalability	Multiple stakeholders	Obligations	Revocation	Administrative overhead	Decentralized management	Ease of management	Automation	Usability	Autonomy	Security	Containment	

+: Parameter is supported, -: Parameter is not supported, O: Partial/weak support of parameter.

The principles defined as requirements in the logic layer, in conjunction with the usability of the system, serves as evaluation parameters for the last layer. The principles of autonomy and security are fairly supported by all the examined systems. Nonetheless, the principle of containment is present in PERMIS and Usage based authorization, due to the support of RBAC. Lastly, the usability of a system is affected by either the push or pull model in use.

Table 4.3: Summary of the comparisons among the different AC mechanisms.

Access Control Mechanisms	Entropy	Assets	Management	Logic
CAS	High	Low	Low	Low
VOMS	High	Medium	Low	Low
GridMap	Medium	Low	Low	Low / Medium
Akenti	Medium / High	Medium	Medium	Low / Medium
PERMIS	High	Low	Medium	Medium
Usage based authorization	High	Medium	Low Medium	Medium

4.5 Chapter summary

Classic requirements engineering processes have been used in the definition of AC requirements for Grid and Cloud computing systems. In many cases, this led to the adoption of existent or modified AC approaches. Furthermore, contemporary implementations seem to be inadequate in fulfilling the new security requirements set by these systems. Stemmed from the need to design new AC approaches and contemplating an auxiliary holistic approach in defining security requirements, we recommended a four-layer CC for Grid and Cloud systems. Its layered scheme is able to enhance and facilitate the process of defining AC requirements. We further use the proposed CC as a foundation in defining AC requirements, and thus, resulting in new AC models for Grid and Cloud systems. A first comparison of the RBAC with the $UCON_{ABC}$ model, has shown that neither of them can tackle the difficulties stemmed from the defined Grid AC requirements flawlessly. Based on the results of the foregoing comparison, it was expected for the Grid

authorization mechanisms to have the same level of applicability in Grid environments. Indeed, the hypothesis has proven right, indicating that the examined mechanisms cannot handle well the defined requirements and parameters in all the layers of the CC. Based on the results stemmed from our research, we believe that the design and implementation of proper AC models for distributed and collaborative systems is needed. Current AC models are not specifically designed to tackle the requirements of Grid and Cloud systems. Lastly, we illustrated how to identify a list of core requirements by applying the CC, and how to use it as a comparison tool.

Chapter 5

domRBAC: The proposed access control model

This chapter presents our proposed RBAC model entitled domRBAC and designed to be applied in Grid and Cloud systems. The model allows the evaluation and enforcement of AC under secure inter-operation in distributed and collaborative systems on run-time and is capable of applying simple usage management policies upon resources for the first time in a role based approach.

5.1 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 this chapter, we propose the domRBAC model, which is an AC model designed to enforce AC under secure inter-operation in distributed and collaborative systems, as the Grid and Cloud computing paradigms.

The proposed model is based on the ANSI INCITS 359-2004 standard (ANSI, 2004). Thus, it supports all the components of the RBAC model, namely the Core RBAC, Hierarchical RBAC, SSD relations, and DSD relations. The domRBAC

model is defined in such way to be both secure and efficient in order to cope with the requirements posed by modern 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).

In the examined systems, we identify a list of functional AC requirements that must be fulfilled based on the CC presented in Chapter 2. These are, as identified, 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 AC 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/Cloud AC requirements flawlessly. Yet, the implementation of the AC approaches into a Grid/Cloud 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 AC models that are not specifically designed to tackle the requirements of such systems. Hence, motivated by the absence of an AC model able to fulfil the requirements of modern collaborative systems, we further proceed with the definition of a new AC model.

The remainder of this chapter is organized as follows: Section 5.2 provides sufficient details regarding the formal definition of domRBAC. Section 5.3 provides implementation aspects concerning the former definitions. Section 5.4 presents a prototype simulator that we have implemented according to our AC model definitions and details some experimental results. Section 5.5 we compare the proposed AC model with existing solutions, where applicable. Finally, Section 5.6 summarizes this chapter.

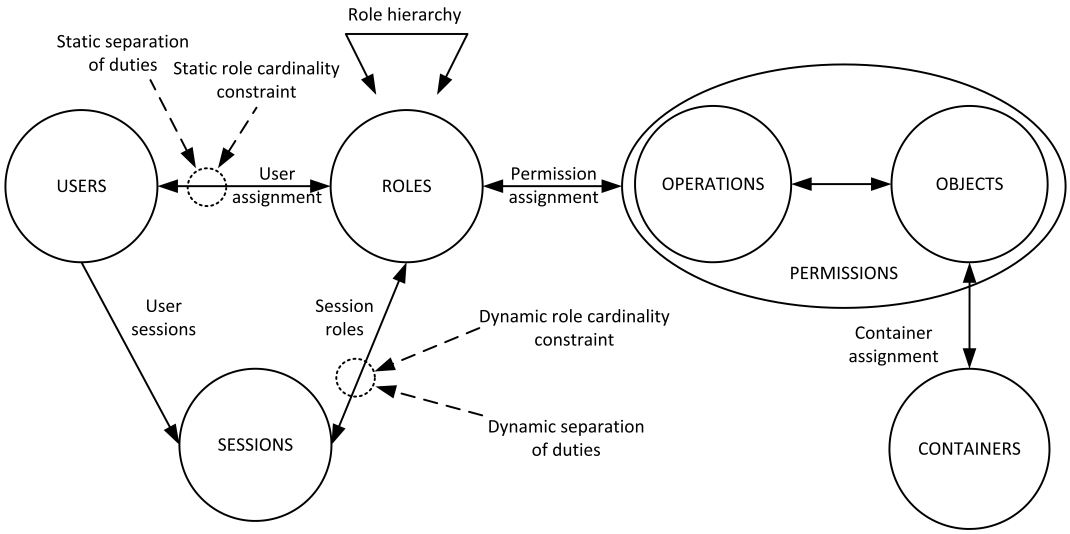

Figure 5.1: The domRBAC model.

5.2 The domRBAC model

We define our proposed AC 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 distributed and collaborative systems, it provides a solid background for a new RBAC 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.

5.2.1 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 AC among participating domains. A domain can be defined as a protected computing environment, consisted of users and resources/objects under an AC policy. Such a functionality is of vital importance since it governs inter-operations among domains. Figure 5.1 illustrates the proposed AC model. Henceforth, we use the terms object and resource interchangeably.

Sessions, objects and operations are three concepts that are commonly used

in AC. 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 representing 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 AC. On the contrary, the granting of access in a database on the level of record or field is an example of fine-grained AC. 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. Operations and objects are dependent on the system. 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 and 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 uses 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 SSD relations in a domain, and violation of DSD 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.

5.2.2 Definitions

5.2.2.1 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 set of user-to-role assignment relation mapping.

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

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

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

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

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

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) $\subseteq \{d_i r_m \in$ ROLES | (session_user(s),$d_i r_m$) $\in$ UA$\}$.

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

5.2.2.2 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 $i \stackrel{+}{=} j$ 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).

5.2.2.3 Definition 3. Constrained domRBAC.

SoD is a fundamental security principle that is supported in domRBAC. SoD serves as a requirement for critical operations, which are divided between two or among 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, chap. 5).

Apart from the support of SSD and DSD 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 ($d_i rs$,n) in SSD, where each $d_i rs$ is a role set in a domain d_i, t a subset of roles in $d_i rs$, and n is a natural number $\geq$2, with the property that no user of domain d_i is assigned to n or more roles from the set $d_i rs$ in each ($d_i rs$,n) $\in$ SSD.
Formal definition:

$\forall(d_i rs, n) \in SSD, \forall t \subseteq d_i rs : |t| \geq n$
$\Rightarrow \bigcap_{d_i r_m \in t} assigned\_users(d_i r_m) = \emptyset.$

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:
$\forall(d_i rs, n) \in SSD, i = j, \forall t \subseteq d_i rs : |t| \geq n$
$\Rightarrow \bigcap_{d_i r_m \in t} authorized\_users_{(i,j)}(d_i r_m) = \emptyset.$

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

iv. **Static role cardinality (SRC):** If static role cardinality constraint is required for any role $d_i r_m$, then $d_i r_m$ cannot be assigned to more than a maximum number of users.

SRC $\subseteq (ROLES \times N)$ is a collection of pairs ($d_i r_m$, n) in static role cardinality, where $d_i r_m$ is a role r_m in a domain d_i and n is a natural number $\geq$ 0, with the property that the number of users assigned with role $d_i r_m$ cannot exceed the number n in each ($d_i r_m$, n) $\in$ SRC.
Formal definition:
$d_i r_m \in ROLES, n \in \mathbb{N}, n \geq 0$,
$\forall(d_i r_m, n) \in SRC \Rightarrow |assigned\_users(d_i r_m)| \leq n.$

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:

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

vi. **Dynamic role cardinality constraint (DRC):** If dynamic role cardinality is required for any role $d_i r_m$, then $d_i r_m$ 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 ($d_i r_m$, n) in dynamic role cardinality, where $d_i r_m$ 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 $d_i r_m$ cannot exceed the number n.

Formal definition:
$d_i r_m \in ROLES, n \in \mathbb{N}, n \geq 0,$
$\forall s \in$ SESSIONS, ($d_i r_m$,n) $\in$ DRC $\Rightarrow \sum |d_i r_m \cap session\_roles(s)| \leq n.$

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.

5.2.2.4 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 *INTERDOMAIN_POLICY_VIOLATION* function for the checking of inter-domain violations due to the inter-domain role inheritance relations, and two new inter-domain administrative commands *AddInterdomainInheritance* 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 $d_i r_m$ even though u is not directly assigned to $d_i r_m$ or any of the roles that are senior to $d_i r_m$ 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 $d_i r_{m_{asc}} \gg d_j r_{n_{desc}}$ between existing roles $d_i r_{m_{asc}}$, $d_j r_{n_{desc}}$, where $d_i r_{m_{asc}}$ is a role ascendant of $d_j r_{n_{desc}}$.

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 $d_i r_m$ and $d_i r_n$ of domain d_i. We identify violations of SSD relationships, using the following properties (Ferraiolo et al., 2003, chap. 5):

   Property 1: If there are two roles $d_i r_m$ and $d_j r_n$ that are mutually exclusive, then neither one should inherit the other, either directly or indirectly.

Property 2: If there are two roles $d_i r_m$ and $d_j r_n$ 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 $d_i r_m$ and $d_i r_n$ of domain d_i. We identify violations of DSD relationships similarly to definition 4.iii due to the following property (Ferraiolo et al., 2003, chap. 5):

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

v. **INTERDOMAIN_POLICY_VIOLATION:** 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:
$AddInterdomainInheritance(d_i r_{m_{asc}}, d_j r_{n_{desc}} : NAME) \lhd$
$d_i r_{m_{asc}}, d_j r_{n_{desc}} \in ROLES;$
$INTERDOMAIN\_POLICY\_VIOLATION(d_j r_{n_{desc}}) = false;$
$\neg(d_i r_{m_{asc}} \gg d_j r_{n_{desc}}); \neg(d_j r_{n_{desc}} \geq d_i r_{m_{asc}})$
$\geq' = \geq \cup \{dr, dq : ROLES | dr \geq d_i r_{m_{asc}} \wedge d_j r_{n_{desc}} \geq dq \bullet dr \mapsto dq\} \rhd$

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:
$DeleteInterdomainInheritance(d_i r_{m_{asc}}, d_j r_{n_{desc}} : NAME) \lhd$
$d_i r_{m_{asc}}, d_j r_{n_{desc}} \in ROLES; (d_i r_{m_{asc}} \gg d_j r_{n_{desc}})$
$\geq' = (\gg \setminus \{d_i r_{m_{asc}} \mapsto d_j r_{n_{desc}}\})^* \rhd$

5.3 Implementation aspects

In this section, a series of implementation aspects of the model are discussed. Our approach uses 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 sub-graphs 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 $(d_i r_m, d_j r_n)$, $i \stackrel{+}{=} j$ 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 $(d_i r_1, d_i r_2)$, $(d_i r_2, d_i r_3)$, ..., $(d_i r_{n-1}, d_i r_n)$. 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

```
    Algorithm implemented by the CI_VIOLATION function.
1:  function CI_VIOLATION() : boolean
2:      for all vertex d_i r_m ∈ T_G
3:          for all adjacent vertex d_i r_n
4:              if (d_i r_m = d_i r_n)
5:                  return true
6:      return false
7:  end function
```

Figure 5.2: Identification of cycles in role assignment

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 5.2. 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 5.3. In domain d_1 we have role $d_1 r_a$ that inherits role $d_1 r_b$. Likewise, in domain d_2 we have role $d_2 r_c$ that inherits roles $d_2 r_d$

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 5.4. 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 5.5. 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

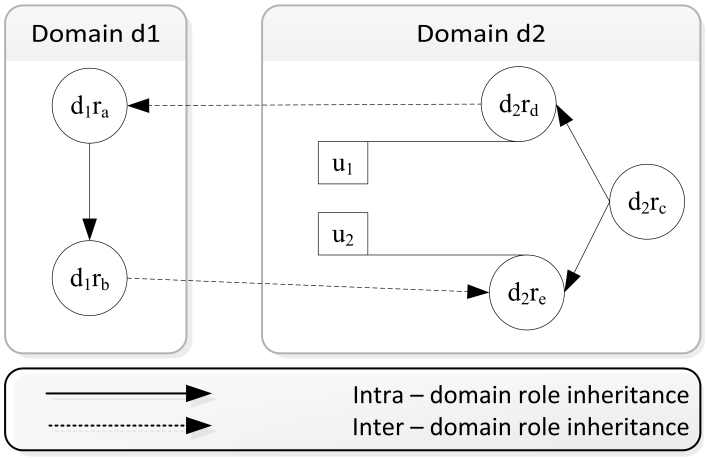

Figure 5.3: Privilege escalation example.

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 5.6.

ix. **INTERDOMAIN_POLICY_VIOLATION:** Figure 5.7 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 5.3 shows how the aforementioned functions are used to identify any of the supported violations. Let's assume a multi-domain AC 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

Algorithm SP_VIOLATION function.

```
1:  function SP_VIOLATION(d_i r_j) : boolean
2:      for all vertex d_i r ∈ domain i where d_i r ≥ d_i r_j
3:          if (T_G[d_i r_j, d_i r] ≠ T_{G_i}[d_i r_j, d_i r]) or
4:              (T_G[d_i r, d_i r_j] ≠ T_{G_i}[d_i r, d_i r_j])
5:          return true
6:      return false
7:  end function
```

Figure 5.4: Assurance of the security principle

Algorithm SSD_VIOLATION function.

```
1:  function SSD_VIOLATION() : boolean
2:      for all SSD pair (d_i r_m, d_i r_n)
3:          for all vertex dr ∈ T_G
4:              if (d_r = d_i r_m)
5:                  for all adjacent vertex d'r
6:                      if (d'r = d_i r_n)then
7:                          return true
8:              if (d_r = d_i r_n)
9:                  for all adjacent vertex d'r
10:                     if (d'r = d_i r_m)then
11:                         return true
12:     for all SSD pair (d_i r_m, d_i r_n)
13:         for all vertex dr ∈ T_G
14:             foundSSDRole = 0
15:             for all adjacent vertex d'r
16:                 if (d'r = d_i r_m) or (d'r = d_i r_n)
17:                     foundSSDRole++
18:                 if foundSSDRole = 2 then
19:                     return true
20:     return false
21: end function
```

Figure 5.5: Intra-domain violation of SSD relationships

```
     Algorithm DSD_VIOLATION function.
1:   function DSD_VIOLATION() : boolean
2:      for all DSD pair (d_i r_m, d_i r_n)
3:         for all vertex dr ∈ T_G
4:            if (d_r = d_i r_m)
5:               for all adjacent vertex d'r
6:                  if (d'r = d_i r_n)then
7:                     return true
8:            if (d_r = d_i r_n)
9:               for all adjacent vertex d'r
10:                 if (d'r = d_i r_m)then
11:                    return true
12:     for all DSD pair (d_i r_m, d_i r_n)
13:        for all vertex dr ∈ T_G
14:           foundDSDRole = 0
15:           for all adjacent vertex d'r
16:              if (d'r = d_i r_m) or (d'r = d_i r_n)
17:                 foundDSDRole++
18:              if foundDSDRole = 2 then
19:                 return true
20:     return false
21:  end function
```

Figure 5.6: Intra-domain violation of DSD relationships

```
     Algorithm INTERDOMAIN_POLICY_VIOLATION function.
1:   function
       INTERDOMAIN_POLICY_VIOLATION(d_i r_{n_desc}) : boolean
2:     return ci_violation() or
3:            SP_VIOLATION(d_i r_{n_desc}) or
4:            SSD_VIOLATION() or
5:            DSD_VIOLATION()
6:   end function
```

Figure 5.7: Inter-domain policy violation function

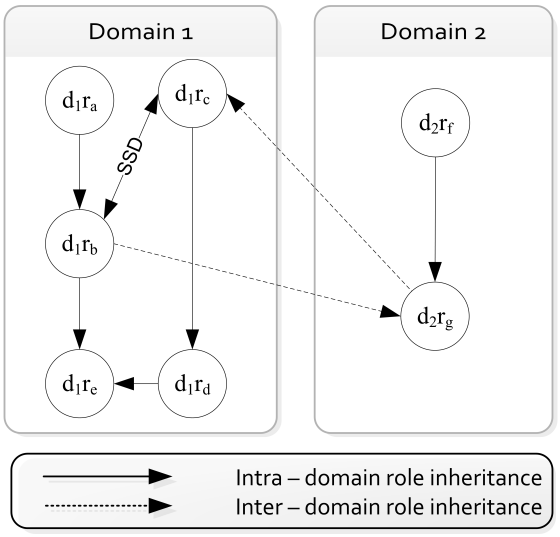

Figure 5.8: A multi-domain AC policy defining interoperation between d1 and d2.

inherits all permissions of d_2r_g. The multi-domain AC 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:

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

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

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

Step 2. The $INTERDOMAIN\_POLICY\_VIOLATION$ 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 collab-

oration, role d_1r_a inherits role d_1r_c. However, the latter inheritance relationship did not exist in the initial domain d_1, and 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.

Lastly, we provide a case scenario where the enforcement of usage control upon resources is demonstrated. Figure 5.9 shows a simple policy in a domain d_1. Role d_1r_a is a role senior to d_1r_b. User Alice requires to share the CPU cycles of her mobile device. Since the CPU capabilities of the device are limited, she decides to share only 50% of her CPU cycles, and to provide to each consumer

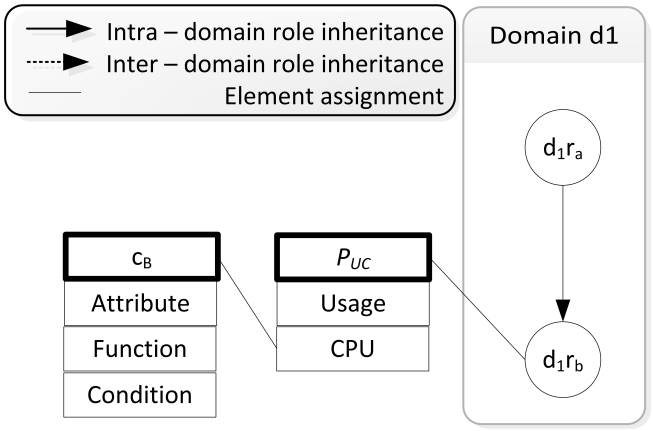

Figure 5.9: Resource usage management enforcement.

at most 5% of her shareable CPU cycles. In order to apply the aforementioned policy, role d_1r_b is assigned to permission $P_{UC} = (Usage, CPU)$. This means that a usage operation is assigned to a CPU object. A container c_B is assigned to object CPU. Container c_B has the following properties: a container attribute that defines the CPU usage value equal to 5%, a container function that returns the current CPU usage, and a container condition $\leq$. Moreover, a DRC constraint is applied to limit the number of active users to 10 ($DRC_{d_1r_b} = (d_1r_b, 10)$). The latter constraint ensures that the number of concurrent active users cannot exceed the 10 users. Thus, in conjunction with the container element it is ensured that the usage of CPU not exceed the 50%, and that each consumer receives at most 5% of CPU. If the DRC constraint was omitted, Alice would not be able to limit the usage of her resources, nor guarantee 5% of CPU usage to the consumers.

5.4 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 domRBAC during collaboration. Lastly, an evaluation of domRBAC is performed

using the CC.

5.4.1 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 ADF, which together with the Access AEF implements the concept of reference monitor (Ferraiolo et al., 2003, chap. 2). An ADF is responsible for the making of AC decisions. The decisions are made based on information applied by the AC policy rules, the context in which the access request is made, and the ADI (ITU-T, 1995). The ADI is a portion in the Access Control Information (ACI) function, which includes any information used for AC 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 AC framework (ITU-T, 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 build using the Qt 4.6.3 (Nokia, 2011).

Figure 5.10 illustrates the overall architecture of the domRBAC simulator. The simulator is capable of reading AC 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, 2012). 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 B. 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

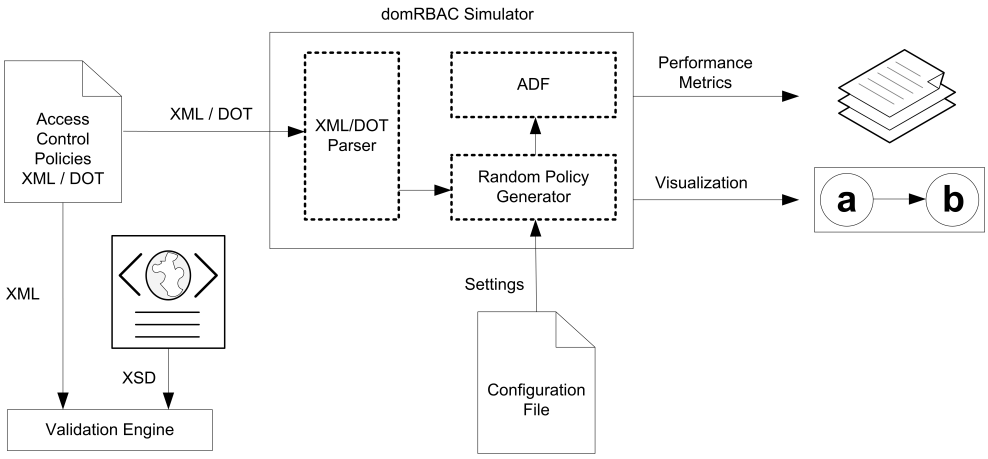

Figure 5.10: Overall architecture of the domRBAC simulator.

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, 2012) 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 AC 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 5.11. On the left side of the screen, the loaded AC policies are available in tree-view, and on the right side the AC 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.

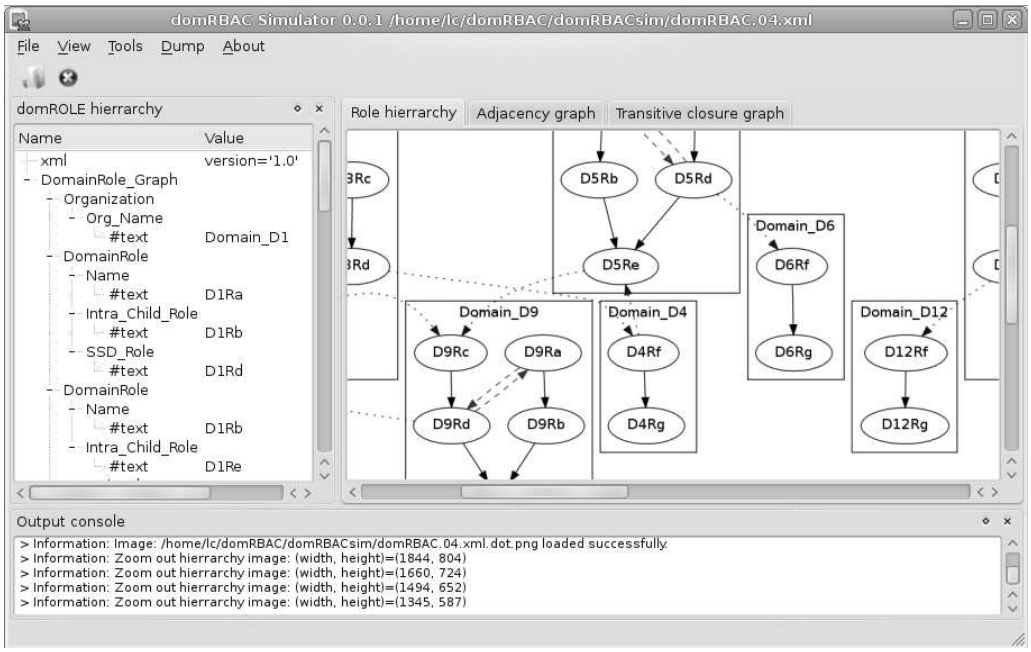

Figure 5.11: The main interface of the domRBAC simulator.

Since domRBAC depends heavily on 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.

5.4.2 Performance evaluation

Since an AC 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, 2012). In turn, the simulator loads the role hierarchies of each domain and randomly performs role assignment operations, as well as SSD and DSD 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 AC 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 5.1 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 5.1 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 equations 5.1 and 5.2, respectively. In 5.1 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 5.2 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.

$$OAL = \frac{(Intra\ role\ assignments) - (Committed\ intra\ role\ assignments)}{(Intra\ role\ assignments)} \tag{5.1}$$

$$OIL = \frac{(Inter\ role\ assignments) - (Inter\ violations)}{(Inter\ role\ assignments)} \tag{5.2}$$

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 A include the descriptive statistical measures in detail, viz. mean, median, mode,

Table 5.1: Performance evaluation of ADF's memory usage and identified violations (5000 requests)

Domains	Roles per domain	Role assign-ments	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

maximum and standard deviation values, of groups 'A' and 'B', respectively. Figures 5.12 and 5.13 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 5.14 shows in a logarithmic scale a comparison of the mean and maximum decision time values, where both lines show a similar behaviour, 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.

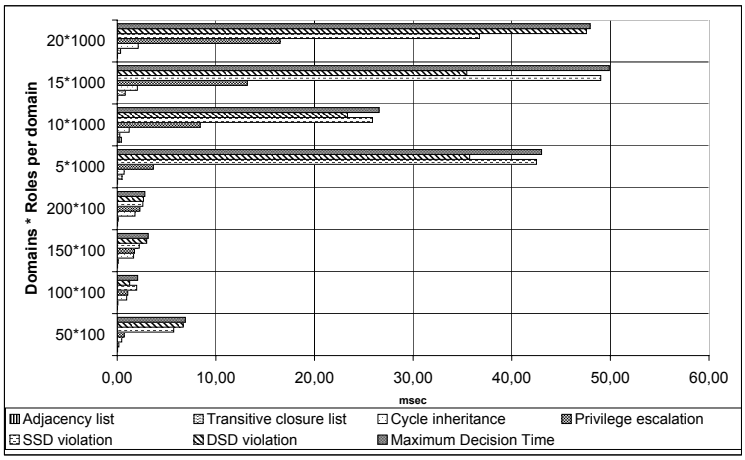

Figure 5.12: Time of computations - mean values.

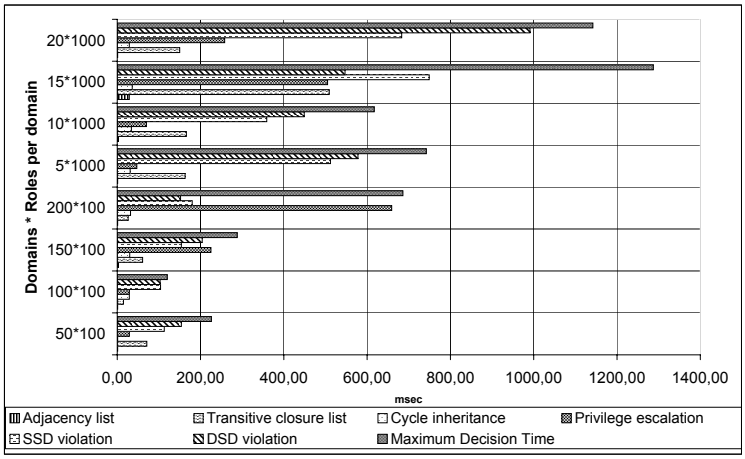

Figure 5.13: Time of computations - max values.

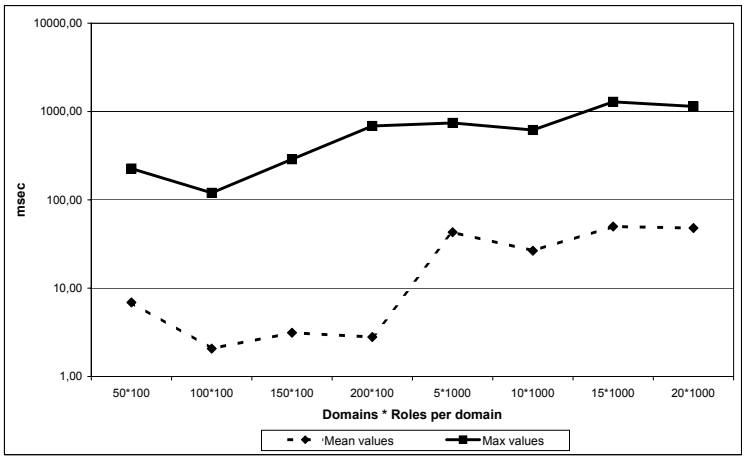

Figure 5.14: Comparison of mean and max values.

5.4.3 Evaluation using the CC.

After the definition and evaluation regarding the performance of domRBAC, we further provide an evaluation of it using the CC, provided in Chapter 4. Table 5.2 depicts the evaluation of domRBAC in the context of the four layers in the CC (cf. Table 4.1). Consequently, domRBAC is capable of strengthening RBAC approaches in all layers. Specifically, the features in domRBAC that strengthens RBAC approaches are: *i)* In the entropy layer, domRBAC is capable of supporting interoperability and has the ability to scale well. *ii)* In the layer of assets, we identify that domRBAC is able to support basic usage control and therefore usage restrictions can be introduced. Moreover, it is possible to define resources that are being shared by multiple stakeholders. *iii)* In the management layer, we saw that using a small administrative overhead it is able to automate the management of policies in an easy and efficient way. *vi)* Lastly, domRBAC strengthens RBAC in the layer of logic since it provides features such as autonomy and security.

Table 5.2: Evaluation of domRBAC using the CC.

Access Control Model	Entropy	Assets	Management	Logic
domRBAC	High	Medium	High	Medium / High

5.5 Discussion

Despite the existence of a large number of proposed and implemented AC models, there are only a few, to the best of our knowledge that provides information relevant to their performance. Moreover, performance comparison among AC 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 AC 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 AC 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.

5.6 Chapter summary

An AC model is proposed in this chapter for distributed and collaborative systems. To meet scalability, security and basic usage management requirements in AC, the proposed domRBAC model is used to support various security policies for collaborative environments. Our proposed AC 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 AC 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 AC model is able to perform better in many cases compared with existing implementations. Yet, due to design decisions that require the maintenance of the security principle, the proposed AC model, results in relatively lower interoperability levels, yet acceptable, compared with existing solutions. In overall, compared to other similar approaches, the domRBAC model provides AC with adequate dynamics for computing systems, and also manages to fulfil critical requirements of modern distributed and collaborative systems.

Chapter 6

Verification of secure inter-operation in multi-domain RBAC

This chapter presents our proposed model checking technique for the verification of secure inter-operation properties in multi-domain RBAC systems. The technique is the result of a collaboration with NIST's Computer Security Division. In particular, the proposed model checking technique is based on that in (Hu et al., 2008) and re-defined and implemented in the context of RBAC systems accordingly. Therefore, through the verification process, the correctness of an AC system and security policies can be verified against secure inter-operation properties.

6.1 Introduction

As stated in (Capitani di Vimercati et al., 2007), a system can be argued to be secure only if the model is secure and the mechanism correctly implements the model. However, despite the importance of the aforementioned statement, there is not, to the best of our knowledge, any formal definition of all the properties related to secure inter-operation in RBAC systems, in order for them to be verified using a model checking technique. In most cases, the verification is limited to the

verification of separation of duty constraints as in (Hansen and Oleshchuk, 2005), (Schaad and Moffett, 2002). Therefore, stemmed from the absence of related work, we define the secure inter-operation properties that should be verified in an AC system that implements a global role-based security policy in a multi-domain environment. A multi-domain environment consists of individual domains where each implement an intra-domain AC policy. During a collaboration, an inter-domain AC policy is being formed, which consists of the individual intra-domain and cross-domain policies. In particular, we assume a transition system (TS) for an RBAC model and we formally define the security properties of cyclic inheritance, privilege escalation, separation of duties (SoD) and autonomy in temporal logic. Secure inter-operation is maintained when all the aforementioned apply in an RBAC system (Shafiq et al., 2005).

The structure of the remainder of this chapter is as follows. In Section 6.2 a formal definition of the ANSI INCITS 359-2004 is recalled, along with our proposed modifications, the definition of its transition system and the properties related to secure inter-operation. Implementation aspects are discussed in Section 6.3 and proof of concept examples, on how to verify the defined properties, are provided in Section 6.4. The performance of the applied technique and a number of issues are discussed in Section 6.5. Finally, we conclude this chapter in Section 6.6.

6.2 Model checking secure inter-operation

In this section, we provide basic prerequisite information about secure inter-operation, the basis model checking technique applied, the core and hierarchical RBAC, modified accordingly, and an overview of the transition system that is defined in (Hu et al., 2011). Since the latter is targeted to generic AC models, we partially redefined it for RBAC model. Additionally, we provide the definition of a list of properties that are required to be verified during secure inter-operation to assure a consistent and conflict-free inter-operation policy. Specifically, to verify the security principle we define the security properties of cyclic inheritance, privilege escalation and separation of duties. Furthermore, a formal definition of the autonomy principle is provided as a property to be verified in a global security

policy.

6.2.1 Secure inter-operation

Secure inter-operation in collaborative systems is required for secure collaboration among participating parties such that the principles of autonomy and security can be guaranteed (Gong and Qian, 1996). The principle of autonomy states that if an access is permitted by an individual system, it must also be permitted under secure inter-operation. The principle of security states that if an access is denied by an individual system, it must also be denied under secure inter-operation. In a RBAC collaborative system, violations of secure inter-operation can be caused by adding inter-domain role inheritance relations. As stated in (Shafiq et al., 2005) these types of violations can be detected by checking for cyclic inheritance, privilege escalation, and violation of SoD relations in RBAC policies.

In the following subsection, we illustrate how to identify the aforementioned properties in a RBAC policy. Henceforth, to differentiate roles, users and permissions among domains, we use the *DomainRole* format in (Gouglidis and Mavridis, 2012a) whenever is needed to, where *Domain* denotes a domain name and *Role* denotes a role name. Thus, a role can be expressed as $d_{domain}r_{role}$, and if a role r_k belongs to a domain d_i, we write $d_i r_k$. The same applies for users and permissions. Further, an arrow $\rightarrow$ in Figures 6.1 - 6.3 denotes an immediate inheritance relation between two roles. For example, $r_1 \rightarrow r_2$ denotes that role r_1 inherits the permissions of role r_2.

6.2.1.1 Cyclic inheritance property.

In multi-domain RBAC systems, the cyclic inheritance refers to the problem that a user $d_i u_t$ assigned to the role $d_i r_k$ in domain d_i, is authorized for the permissions of another local role $d_i r_j$ such as $d_i r_j \gg d_i r_k$ (see Subsection 6.2.2 for definition of $\gg$), even though $d_i u_t$ is not directly assigned to $d_i r_j$ in the role hierarchy of domain d_i as shown in Figure 6.1.

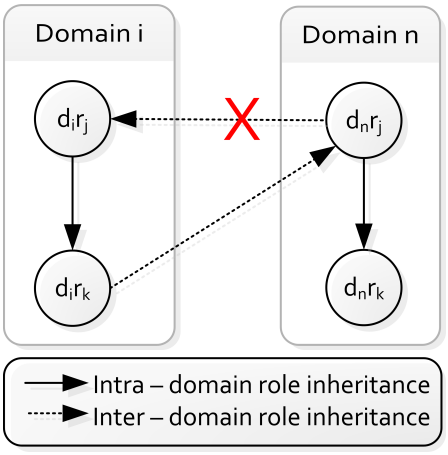

Figure 6.1: Cyclic inheritance.

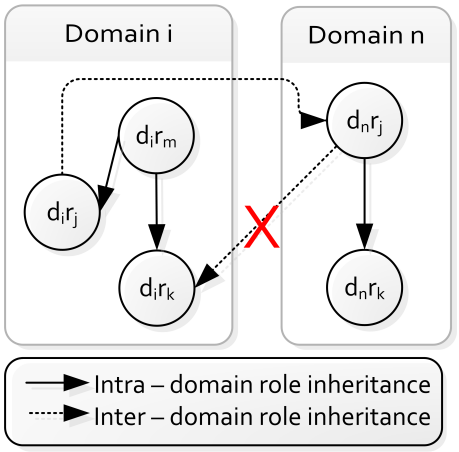

Figure 6.2: Privilege escalation.

6.2.1.2 Privilege escalation property.

Privilege escalation refers to the problem that a user $d_i u_t$ assigned to a role $d_i r_j$ in domain d_i, is authorized for the permissions of another local role $d_i r_k$ such as $\neg(d_i r_j \geq d_i r_k)$ (see Subsection 6.2.2 for definition of $\geq$), even though $d_i u_t$ is not directly assigned to role $d_i r_k$ in the role hierarchy of domain d_i (Figure 6.2).

6.2.1.3 Separation of duty property.

SoD requires two or more division between users, so that no single user can compromise security. SoD methods can be further categorized into SSD and

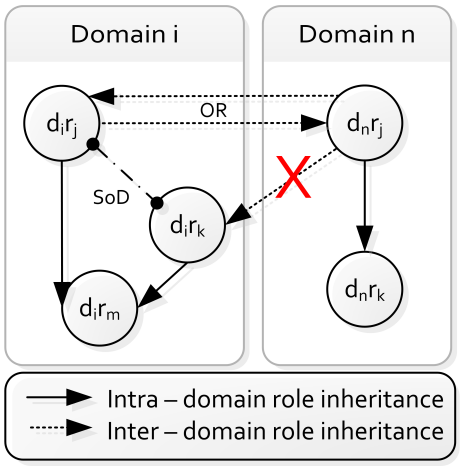

Figure 6.3: Separation of duty.

DSD. SSD are constraints that are placed on roles at the time roles are assigned to users. When implementing SSD in role hierarchy, both inherited and directly assigned roles need to be considered. In the same manner, DSD needs to check the role hierarchy when users activate already assigned roles (Ferraiolo et al., 2003).

Verification of the SSD property is based on the following properties (Ferraiolo et al., 2003):

Property 1. Roles r_k and r_m are mutually exclusive if neither one inherit the other directly or indirectly.

Property 2. If roles r_k and r_m are mutually exclusive then there is no other role inherits both of them.

Similar to SSD, DSD has the property (Ferraiolo et al., 2003):

Property 3. If SSD holds, then DSD is maintained.

Thus, properties 1 and 2 must be guaranteed.

6.2.1.4 Autonomy property.

In addition to the security principle, autonomy should also be preserved for secure inter-operation. Maintaining the autonomy of all collaborative domains is a key requirement of the policy for inter-operation. However, access of inter-operation may be significantly reduced or even not authorized at all if the autonomy of

individual domains is over addressed. Therefore, balancing autonomy and interoperability might be considered (Shafiq et al., 2005). In almost any collaborative environment, it is not permissible to violate any domain's security policy. However, some domains may be willing to compromise their autonomy for the sake of establishing more interoperability, provided that autonomy loss remains within acceptable limits. Specifically, when using a RBAC policy integration framework, a violation in the autonomy of a domain may occur because of induced SoD constraints, as described in (Shafiq et al., 2005). An induced SoD constraint is a SoD constraint between two intra-domain roles (e.g., d_1r_1 and d_1r_2) which do not conflict with each other in their original domain's RBAC policy. In a multi-domain system such a SoD constraint will deny concurrent access on roles d_1r_1 and d_1r_2, and thus, reducing the autonomy in the original domain. Nevertheless, the autonomy principle, when applicable, can be verified by checking if all the assigned and authorized permissions of a role d_ir_k in a domain d_i are preserved for inter-operation.

6.2.2 Model definitions

Each domain specifies its own policy in most collaborative systems today. Hence, we separate the specification of single domain AC policies (i.e., intra-domain administration) from multiple domains collaborative policies (i.e., inter-domain administration). Both specifications follow the ANSI INCITS 359-2004 definition of RBAC (ANSI, 2004). We also define review functions for intra-domain and inter-domain administration. The main components (ANSI, 2004) are defined below.

- USERS, ROLES, OPS, OBS, stands for users, roles, operations, and objects, respectively.
- UA $\subseteq$ USERS $\times$ ROLES, a many-to-many set of user-to-role assignment relation mapping.
- PRMS $= 2^{(OPS \times OBS)}$, the set of permissions.
- PA $\subseteq$ PRMS $\times$ ROLES, a many-to-many set of permission-to-role assignment relation mapping.

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

For intra-domain, we redefine the hierarchical relations and administrative review functions below.

- assigned users: $SU_{d_i}(d_i r_k : ROLES) \rightarrow 2^{USERS}$, the mapping of role $d_i r_k$ onto a set of users enrolled in domain d_i. Formal definition: $SU_{d_i}(d_i r_k) = \{d_i u_t \in USERS | (d_i u_t, d_i r_k) \in UA\}$.
- assigned permissions: $SP_{d_i}(d_i r_k : ROLES) \rightarrow 2^{PRMS}$, the mapping of role $d_i r_k$ onto a set of permissions defined in domain d_i. Formal definition: $SP_{d_i}(d_i r_k) = \{d_i p_w \in PRMS | (d_i p_w, d_i r_k) \in PA\}$.
- $RH_{d_i} \subseteq ROLES \times ROLES$ is a partial order set on $ROLES$ of inheritance relation in domain d_i, denoted as $\geq$, where $d_i r_k \geq d_i r_m$ only if all permissions of $d_i r_m$ are also permissions of $d_i r_k$, and all users of $d_i r_k$ are also users of $d_i r_m$. Formal definition: $d_i r_k \geq d_i r_m \Rightarrow UP_{d_i}(d_i r_m) \subseteq UP_{d_i}(d_i r_k) \wedge UU_{d_i}(d_i r_k) \subseteq UU_{d_i}(d_i r_m)$.
- authorized users: $UU_{d_i}(d_i r_k : ROLES) \rightarrow 2^{USERS}$, the mapping of role $d_i r_k$ onto a set of users enrolled in domain d_i in the presence of a role hierarchy defined in domain d_i. Formal definition: $UU_{d_i}(d_i r_k) = \{d_i u_t \in USERS | d_i r_m \geq d_i r_k, (d_i u_t, d_i r_m) \in UA\}$.
- authorized permissions: $UP_{d_i}(d_i r_k : ROLES) \rightarrow 2^{PRMS}$, the mapping of role $d_i r_k$ onto a set of permissions defined in domain d_i in the presence of a role hierarchy define in domain d_i. Formal definition: $UP_{d_i}(d_i r_k) = \{d_i p_w \in PRMS | d_i r_k \geq d_i r_m, (d_i p_w, d_i r_m) \in PA\}$.

For inter-domain, we extend the aforementioned hierarchy relations and administrative review functions below:

- $RH \subseteq ROLES \times ROLES$ is a partial order set on $ROLES$ of inheritance relation, denoted as $\geq$, where $d_i r_k \geq d_j r_m$ only if all permissions of $d_j r_m$ are also permissions of $d_i r_k$, and all users of $d_i r_k$ are also users of $d_j r_m$. Formal definition: $d_i r_k \geq d_j r_m \Rightarrow UP(d_j r_m) \subseteq UP(d_i r_k) \wedge UU(d_i r_k) \subseteq UU(d_j r_m)$.

- authorized users: $UU(d_i r_k : ROLES) \rightarrow 2^{USERS}$, the mapping of role $d_i r_k$ onto a set of users enrolled in any domain in the presence of a inter-domain role hierarchy. Formal definition: $UU(d_i r_k) = UU_{d_i}(d_i r_k) \cup \{d_j u_t \in USERS | d_j r_m \geq d_i r_k, (d_j u_t, d_j r_m) \in UA\}$.

- authorized permissions: $UP(d_i r_k : ROLES) \rightarrow 2^{PRMS}$, the mapping of role $d_i r_k$ onto a set of permissions defined in any domain in the presence of a inter-domain role hierarchy. Formal definition: $UP(d_i r_k) = UP_{d_i}(d_i r_k) \cup \{d_j p_w \in PRMS | d_i r_k \geq d_j r_m, (d_j p_w, d_j r_m) \in PA\}$.

The absence of relational operators in temporal logic, i.e., $\gg$ and $\geq$, led us to the definition of a series of appropriate predicates below.

- $IR(r_k, r_m)$ denotes the existence of an immediate (either inter or intra domain) inheritance relationship between the two roles. Formal definition: $IR(r_k, r_m) = true \Leftrightarrow r_k \gg r_m$. The operator $\gg$ means immediate inheritance relation as defined in (ANSI, 2004).

- $MR_{d_i}(d_i r_k, d_i r_m)$ denotes that there is an (immediate or not) inheritance relationship between the two roles in the role hierarchy defined in domain d_i. Formal definition: $MR_{d_i}(d_i r_k, d_i r_m) = true \Leftrightarrow d_i r_k \geq d_i r_m$.

- $RP(r_k, r_m)$ denotes that for two roles with an immediate inheritance relation $(r_k, r_m : r_k \gg r_m)$ the set of role's r_k assigned permissions is a subset of role's r_m authorized permissions. Formal definition: $RP(r_k, r_m) = true \Leftrightarrow IR(r_k, r_m) \wedge SP_{d_i}(r_k) \subseteq UP(r_m)$.

- $IB_{d_i}(d_i r_k, d_i r_m, r_n)$ denotes that for two roles $d_i r_k$ and $d_i r_m$ in domain d_i the set of role's r_n authorized permissions, regardless of the domain to which it belongs, includes the assigned permissions of both roles $d_i r_k$ and $d_i r_m$, where r_n is a role senior to roles $d_i r_k$ and $d_i r_m$. Formal definition:

$IB_{d_i}(d_i r_k, d_i r_m, r_n) = true \Leftrightarrow SP_{d_i}(d_i r_k) \cup SP_{d_i}(d_i r_m) \subseteq UP(r_n) \wedge r_n \geq d_i r_k \wedge r_n \geq d_i r_m$.

- $BA(d_i r_k)$ denotes that the mapping of role $d_i r_k$ onto the set of all its assigned and authorized permissions in domain d_i is a subset of all its permissions under the presence of an inter-domain hierarchy. Formal definition: $BA(d_i r_k) = true \Leftrightarrow UP_{d_i}(d_i r_k) \subseteq UP(d_i r_k)$.

6.2.3 Transition system

In this subsection, we define AC rule, property, and transition system for RBAC models. The definitions are modified from (Hu et al., 2011). We use Computation Tree Logic (CTL) for the specification of policy properties.

In CTL, prefixed path quantifiers assert arbitrary combinations of linear-time operators. For our purpose, we use universal path quantifier $\forall$ means *"for all paths"* and the linear temporal operators $\Box$ and $\Diamond$ means *"always"* and *"eventually"*, respectively. Furthermore, we use the temporal modalities $\forall\Box\Phi$ representing *invariantly* Φ, and $\forall\Diamond\Phi$ representing inevitably Φ, where Φ is a state formula.

Definition 1. An RBAC rule is a proposition of type "if c then d", where constraint c is a predicate expression on $(r, UP(r))$ for the permission decision d. Thus, RBAC policies consist a sequence of rules of the form $(r, UP(r))$ in the logic expression c.

Definition 2. An RBAC AC property p is a formula of type "$b \rightarrow d$", where the result of the access permission d depends on quantified predicate b on $(r, UP(r))$ mapping. The $\rightarrow$ means "imply".

Definition 3. A transition system TS is a tuple (S, Act, δ, i_0) where

- S is a set of states, $S = \{Permit, Deny\}$,
- Act is a set of actions,
  where $Act = \{(r_1, UP(r_1)), \ldots, (r_n, UP(r_n))\}$,
- δ is a transition relation where $\delta : S \times Act \rightarrow S$, and
- $i_0 \in S$ is the initial state.

The p in Definition 2 is expressed by the proposition $p : S \times Act^2 \rightarrow S$ of TS, which can be collectively translated in terms of logical formula such that $p = (s_i * (r_1, UP(r_1)) * \ldots * (r_n, UP(r_n))) \rightarrow d$, where $p \in P$ is a set of properties, and $*$ is a Boolean operator in CTL (Baier and Katoen, 2008).

The RBAC rule function as the transition relation δ in the TS. Thus, by representing RBAC AC property in temporal logic formula p, we can assert that model TS satisfies p by $TS \vDash \forall\Box(b \rightarrow \forall\Diamond d)$. Property $\forall\Box(b \rightarrow \forall\Diamond d)$ is a response pattern such that d responds to b globally (b is the cause and d is the effect) (SAnToS Laboraroty, 2012).

6.2.4 Specification of properties

In a collaborative RBAC system, violations of secure inter-operation may be caused by add-hoc inter-domain role inheritance. As stated in (Shafiq et al., 2005) and (Gouglidis and Mavridis, 2012a), such violations can be checked by detecting cyclic inheritance, privilege escalation, and violation of SoD relations in the system.

We provide formal representations of the aforementioned security properties using temporal logic below.

6.2.4.1 Cyclic inheritance property.

To detect a cyclic inheritance for a role $d_i r_k$, we check if the proposition $RP(d_i r_j, d_i r_k) \rightarrow \forall\Diamond Deny$ is satisfied invariantly in the TS, formally:

$$TS_{RBAC} \vDash \forall\Box(RP(d_i r_j, d_i r_k) \rightarrow \forall\Diamond Deny). \tag{6.1}$$

6.2.4.2 Privilege escalation property.

To detect privilege escalation for a role $d_i r_k$ against a role $d_i r_j$, we check if the proposition $(\neg MR_{d_i}(d_i r_j, d_i r_k) \wedge RP(d_i r_j, d_i r_k)) \rightarrow \forall\Diamond Deny$ is satisfied invari-

antly by the TS_{RBAC}, formally:

$$TS_{RBAC} \vDash \forall\Box((\neg MR_{d_i}(d_ir_j, d_ir_k) \wedge RP(d_ir_j, d_ir_k)) \rightarrow \forall\Diamond Deny). \tag{6.2}$$

6.2.4.3 Separation of duty property.

In general, we enforce SoD by role pairs (Ferraiolo et al., 2003). The minimum number of mutual exclusion role pairs needs to be checked for the $(d_irs, n) \in SSD$ constraint, where each d_irs is a role set and n is a number ≥ 2 such that no user is assigned to or authorized for n or more roles from the set d_irs in each $(d_irs, n) \in SSD$. This is equal to the binomial coefficient ${}_{|d_irs|}C_2 \equiv \begin{pmatrix} |d_irs| \\ 2 \end{pmatrix} \equiv \frac{|d_irs|!}{2!(|d_irs|-2)!}$.

SSD property verification rely on mutual exclusion of roles specified by role pairs (Ferraiolo et al., 2003). This implies that roles cannot have common assigned users for any role pair in the role set d_irs, formally:

$$TS_{RBAC} \vDash \forall\Box((d_ir_j \in d_irs_w \wedge d_ir_k \in d_irs_w \wedge (RP(d_ir_j, d_ir_k) \vee RP(d_ir_k, d_ir_j) \vee IB_{d_i}(d_ir_j, d_ir_k, r_m))) \rightarrow \forall\Diamond Deny). \tag{6.3}$$

6.2.4.4 Autonomy property.

The autonomy principle, when applicable, can be verified by checking if all the assigned and authorized permissions of a role d_ir_k in a domain d_i are preserved during inter-operation, formally:

$$TS_{RBAC} \vDash \forall\Box(BA(d_ir_k) \rightarrow \forall\Diamond Permit). \tag{6.4}$$

```
    Algorithm Rule and property creation algorithm
1:  procedure ITERATOR_SKELETON (T_G)
2:    for all vertex dr_i ∈ T_G
3:      for all adjacent vertex dr_j
4:        //Generate the required rule or property
5:  end procedure
```

Figure 6.4: Iterator skeleton function.

6.3 Implementation aspects

This section discusses aspects of the implemented technique. The technique described in (Hu et al., 2008) is unable to specify role hierarchies for RBAC policies because it is not geared for RBAC models. To specify role hierarchies, we propose a role-to-role mapping algorithm derived from the graph theory in terms of Definition 1. When defining a role hierarchy $r_k \geq r_m$, r_k and r_m are assigned to permissions $PRMS_k$ and $PRMS_m$, respectively. In turn, we generate additional rules according to the Definition 1, apart from the initial rules that map roles and their assigned permissions. For example, regarding the previously mentioned roles r_k and r_m, an additional rule is generated automatically to record the r_k's inheritance of permission $PRMS_m$. In this way, a reasoning for RBAC hierarchies is introduced, which depicts the role hierarchy $r_k \geq r_m$ (i.e., role r_k is a senior of role r_m). As role hierarchies are represented by sparse graphs, we use linked lists instead of matrices for implementing role hierarchies since, in the examined case, the former are more efficient regarding performance (see Subsection 5.3).

To specify role hierarchies according to the Definition 1, we compute the transitive closure list T_G from the adjacency list A_G of graph $G = (V, E)$ built from RBAC rules. The result is generated from iterating the vertices of the transitive closure list T_G. And, using iteration for the vertices to generate AC properties for verification as defined in Definition 2. The pseudo code in Figure 6.4 procedure illustrates how the iteration is generated.

Specifically, SAX stream API is employed as the parser to load a RBAC policy in XML (created by ACPT (Hwang et al., 2010)) and produce the adjacency list A_G. In turn, the T_G is computed from the A_G. Next, a new XML file is created by iterating the vertices of T_G using the iterator in Figure 6.4. The XML includes

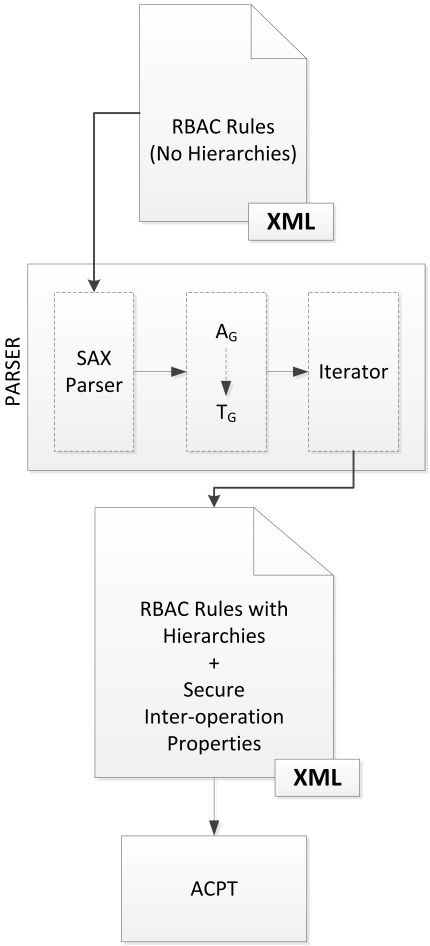

Figure 6.5: The proposed tool chain.

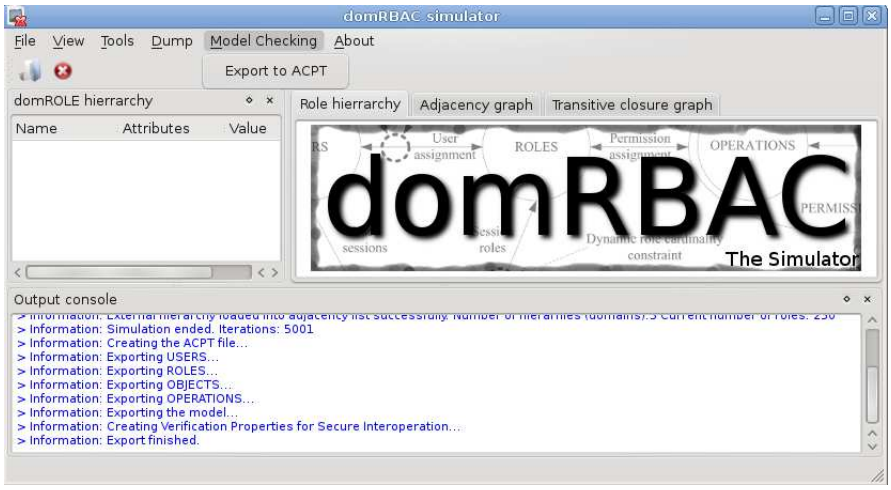

Figure 6.6: Integration of the proposed parser in the domRBAC simulator.

both the RBAC rules with hierarchies, and properties to be verified. Figure 6.5 illustrates the described tool chain for the process. Figure 6.6 depicts the proposed parser integrated in the domRBAC simulator.

6.4 Application examples

In this section, we provide a series of proof of concept examples for the verification of the properties defined in subsection 6.2.4. The examples were implemented in the NuSMV model checker (NuSMV). The relevant code was based on the NuSMV source code generated by ACPT (Hwang et al., 2010), and recoded accordingly in order to depict only the portions of code required by a RBAC model. The NuSMV code of all examples along with their outputs and counterexamples are provided in Appendix C.

6.4.1 Verification of cyclic inheritance and autonomy properties

In Figure 6.7 we assume a multi-domain AC policy that allows the collaboration between domains d_1 and d_2. Domain d_1 has the following roles: d_1r_a and d_1r_b where d_1r_a inherits all permissions of d_1r_b. In turn, domain d_2 has the following

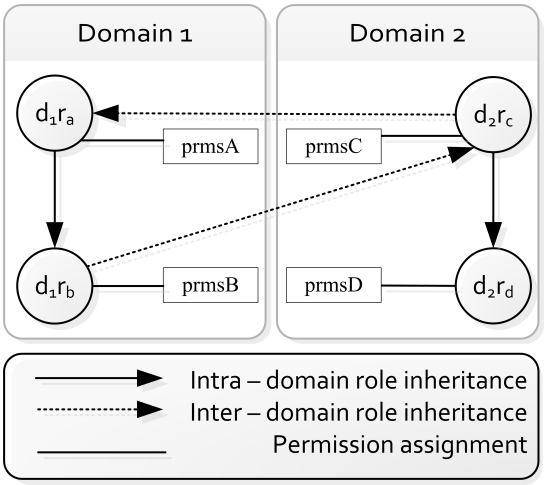

Figure 6.7: Cyclic inheritance and autonomy verification.

roles: d_2r_c and d_2r_d where d_2r_c inherits all permissions of d_2r_d. Furthermore, a collaboration is depicted between domains $d_1$and d_2 where it is instantiated by the addition of two inter-domain role assignments: d_1r_b inherits d_2r_c and d_2r_c inherits d_1r_a.

It is straightforward that the aforementioned inter-domain policy leads to a cyclic inheritance violation since it is permitted to role d_1r_b to access the permissions of its senior role d_1r_a, in domain d_1, through role d_2r_c in domain d_2. Therefore, also the verification of the satisfaction relation: $TS_{RBAC} \vDash \forall\Box(RP(d_1r_a, d_1r_b) \rightarrow \forall\Diamond Deny)$ in NuSMV is evaluated as *false*.

Furthermore, we proceed with the verification of the autonomy property in the aforementioned RBAC policy, as this is defined in subsection 6.2.4. At first, we appoint the authorized permissions for each role in each individual domain without taking into account any inter-operation between the two domains. Thus, in domain d_1, role d_1r_a is authorized for permissions $prmsA$ and $prmsB$, and role d_1r_b is authorized for permission $prmsB$. Additionally, in domain d_2, role d_2r_c is authorized for permissions $prmsC$ and $prmsD$, and role d_2r_d is authorized for permission $prmsD$. Therefore, in order to verify the autonomy property under secure inter-operation, we check the following satisfaction relations in NuSMV, which are evaluated as *true*, and thus, meaning that the autonomy principle is maintained in both domains.

- $TS_{RBAC} \vDash \forall\Box(BA(d_1r_a) \rightarrow \forall\Diamond Permit)$
- $TS_{RBAC} \vDash \forall\Box(BA(d_1r_b) \rightarrow \forall\Diamond Permit)$
- $TS_{RBAC} \vDash \forall\Box(BA(d_2r_c) \rightarrow \forall\Diamond Permit)$
- $TS_{RBAC} \vDash \forall\Box(BA(d_2r_d) \rightarrow \forall\Diamond Permit)$

6.4.2 Verification of privilege escalation and SSD properties

Let us assume a multi-domain AC policy, as depicted in Figure 6.8, that allows two domains to collaborate. 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 constraint $((d_1r_b, d_1r_c), 2)$ is applied, meaning that these roles cannot be assigned to the same user. In turn, 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. Additionally, we have the following two inter-domain inheritance relationships between domains d_1 and d_2, which are role d_1r_b inherits role d_2r_g and role d_2r_g inherits role d_1r_c.

In the latter global security policy, a privilege escalation violation can be identified since roles d_1r_a and d_1r_b are able to be authorized with the permissions of roles d_1r_c and d_1r_d, which is not permissible in domain d_1. As a proof of concept, all the following satisfaction relations are evaluated as *false* in NuSMV.

- $TS_{RBAC} \vDash \forall\Box((\neg MR_{d_1}(d_1r_a, d_1r_c) \wedge RP(d_1r_a, d_1r_c)) \rightarrow \forall\Diamond Deny)$
- $TS_{RBAC} \vDash \forall\Box((\neg MR_{d_1}(d_1r_a, d_1r_d) \wedge RP(d_1r_a, d_1r_d)) \rightarrow \forall\Diamond Deny)$
- $TS_{RBAC} \vDash \forall\Box((\neg MR_{d_1}(d_1r_b, d_1r_c) \wedge RP(d_1r_b, d_1r_c)) \rightarrow \forall\Diamond Deny)$
- $TS_{RBAC} \vDash \forall\Box((\neg MR_{d_1}(d_1r_b, d_1r_d) \wedge RP(d_1r_b, d_1r_d)) \rightarrow \forall\Diamond Deny)$

Likewise an SSD violation occurs since any user u assigned to role d_1r_b is also authorized for role d_1r_c, which is not permissible in domain d_1 due to the SSD constraint. Thus, the satisfaction relation $TS_{RBAC} \vDash \forall\Box((d_1r_b \in (d_1r_b, d_1r_c) \wedge d_1r_c \in (d_1r_b, d_1r_c) \wedge (RP(d_1r_b, d_1r_c) \vee RP(d_1r_c, d_1r_b))) \rightarrow \forall\Diamond Deny)$ is evaluated

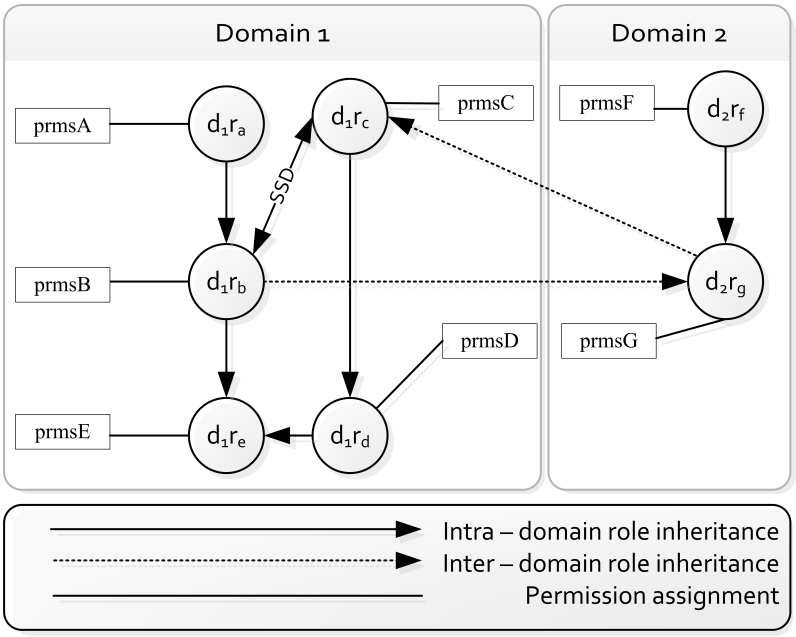

Figure 6.8: Privilege escalation and SSD verification.

in NuSMV as *false*. In the latter satisfaction relation, BI is omitted since there is not any role senior to both roles d_1r_b and d_1r_c.

6.5 Performance evaluation and discussion

In this section, we present a series of performance metrics regarding the evaluation of the proposed technique in small, medium and large-scale cloud systems. Furthermore, we recall performance metrics regarding the enforcement of secure inter-operation in domRBAC (Gouglidis and Mavridis, 2012a) to compare the performance overhead between the proposed technique and embedded secure inter-operation approaches. Specifically, Subsection 6.5.1 provides information regarding the simulation of AC policies and requests and system configuration. Subsection 6.5.2 presents the evaluation of the proposed technique as a management service/tool. Subsection 6.5.3 demonstrates performance metrics regarding domRBAC. Subsection 6.5.4 presents a discussion on the results.

6.5.1 Specifications

To evaluate performance, we first generated AC requests from cloud users using the NetworkX python package (NetworkX, 2012), which operated as input data for the proposed parser. Cloud users' AC requests and corresponding access privileges according to AC polices were created by the *gnc_graph* function, which returns a growing network with copying (GNC) directed graph built by adding nodes linked to previous ones (one at a time) (Krapivsky and Redner, 2005), (NetworkX, 2012).

Next step, we simulated role assignments for a number of cloud hosted domains using the domRBAC simulator (Gouglidis and Mavridis, 2012a). We assumed the existence of 5, 10, 15 and 20 cloud hosted domains containing 50 roles each. In turn, we simulated collaborations among cloud hosted domains, which resulted in an aggregation of 250, 500, 750 and 1000 roles per collaborative environment, respectively. Therefore, we have managed to perform simulation of various size systems (i.e., small to large-scale systems) (Schaad et al., 2001).

Our evaluation of the simulated cloud system is running on the Microsoft Windows 2003 Server Enterprise Edition operating system with service pack 2 running on 3.0 GHz Intel Pentium processor, with 2 GB of RAM.

6.5.2 Secure inter-operation via verification

In this section, we provide a number of quantitative results from the technique used for the verification of the proposed secure inter-operation properties. Table 6.1 summarizes the information generated from both the implemented parser and NuSMV. Specifically, RBAC policies are indicated by the number of edges of T_G. The verification time for specifications is significantly bigger for large scale systems, as the time was increased by both the number of reachable states from binary decision diagram's (BDD) and specifications. The number of BDD's reachable states increase when the number of RBAC policies increases. However, the specifications can be parallel verified. Plus, the examined simulated data were evaluated by the NuSMV symbolic model checker in both normal and optimized mode. Normal mode does not include any additional command line parameters, while optimised mode includes three parameters to improve the per-

Table 6.1: Summary of the evaluated data.

#	Roles	T_G Edges	# of specifications	Reachable states	Execution time (min.)
1.	5*50	1031	12405	$2^{29.2366}$	< 1
2.	10*50	1784	27267	$2^{32.2279}$	176
3.	15*50	3179	52143	$2^{33.9799}$	1472
4.	20*50	3421	76964	$2^{35.2236}$	5820

formance (NuSMV).

Table 6.2 summarizes the aforementioned performance measurements for test cases number 2, 3 and 4 in both normal and optimized mode. Test case number 1 is omitted because its single process execution time takes less than a minute. Figures 6.9, 6.10 and 6.11 illustrates the time required for each of the nine processes in both normal and optimized mode, for test cases 2, 3 and 4, respectively. To calculate the speed improvement for property verification from running nine processes in parallel versus one single process, we used the following formula:

$$Reduction\_time(\%) = (1 - \frac{maxT}{Single\_process\_time}) * 100 \tag{6.5}$$

where $maxT$ is the maximum time value of a set of elements $T = (t_{p_i})_{i=1}^{N}$, where N is the number of parallel processes and t_{p_i} is the execution time of a process p_i. Table 6.2 shows the time required for one process to verify all the specifications, where $minT$ is time value of a set of elements $T = (t_{p_i})_{i=1}^{N}$, and $\sum_{i=1}^{N} t_{p_i}$ is the total time required for all parallel processes to finish when executed sequentially.

The results conclude that parallel processing significantly improves property verification performance as in the example cases, which evenly distribute specifications to nine processes and reduced time by 86% in normal and 77% in optimized mode on the average when compared to the time from single process. Further, from Figures 6.9, 6.10 and 6.11, we conclude that time for NuSMV in normal mode fluctuated greater than in optimized mode. Thus, performance in optimized mode is more steady and predictable than in normal mode. The results also show that the following applies to small and medium systems: $maxT_{normal} < maxT_{optimized}$ and $minT_{normal} < minT_{optimized}$. However, this will significantly change in large scale systems where $maxT_{normal} \gg maxT_{optimized}$ and

Table 6.2: Summary of the performance measurements using 9 processes (Normal versus Optimized mode).

#	Single process time (min.)	$\sum_{i=1}^{N} t_{p_i}$ (min.)	maxT (min.)	minT (min.)	Reduction time
1 N	<1	N/A	N/A	N/A	N/A
1 O	<1	N/A	N/A	N/A	N/A
2 N	176	163	21	15	88%
2 O	134	205	23	21	82%
3 N	1472	1521	208	130	86%
3 O	744	1622	222	163	70%
4 N	5820	5865	901	357	84%
4 O	1506	2241	283	196	81%

N: Normal mode
O: Optimized mode

$minT_{normal} \gg minT_{optimized}$. Lastly, in all cases, we observed that $\sum_{i=1}^{N} t_{p_i} \neq Single\_process\_time$, which means that the sum of all times required for verifying a number of specifications using sequentially processes differs from the time required to verify the same number of specifications using a single process.

In summary, the parallel property verification seems to significantly improve the efficiency for the proposed technique in term of the time required for a complete property verification for large scale systems. Therefore, the proposed technique can be used to verify the correctness of AC systems as a management service/tool. And, despite the limitations of model checking (i.e., state-space explosion issue), it is an effective technique to expose potential design and implementation errors (Baier and Katoen, 2008).

6.5.3 Secure inter-operation in domRBAC

Using the same example data set for the domRBAC simulator, we performed a series of tests as summarized in Table 6.3. The maximum execution time is the sum of six metrics, which are required for creating/updating the A_G and T_G and detecting cycle inheritance, privilege escalation or SSD/DSD violations. As shown in Table 6.3, domRBAC take less than one msec for the test cases 1 and 2, and at maximum one msec for the test cases 3 and 4.

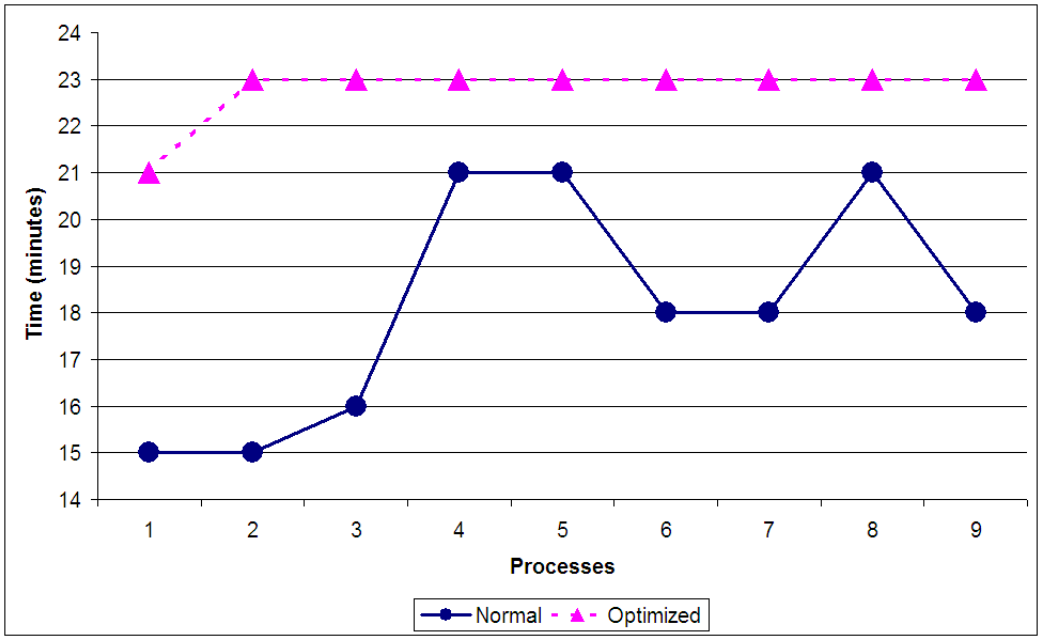

Figure 6.9: Parallel verification of specifications for test case #2 (approximately 3029 specifications per process).

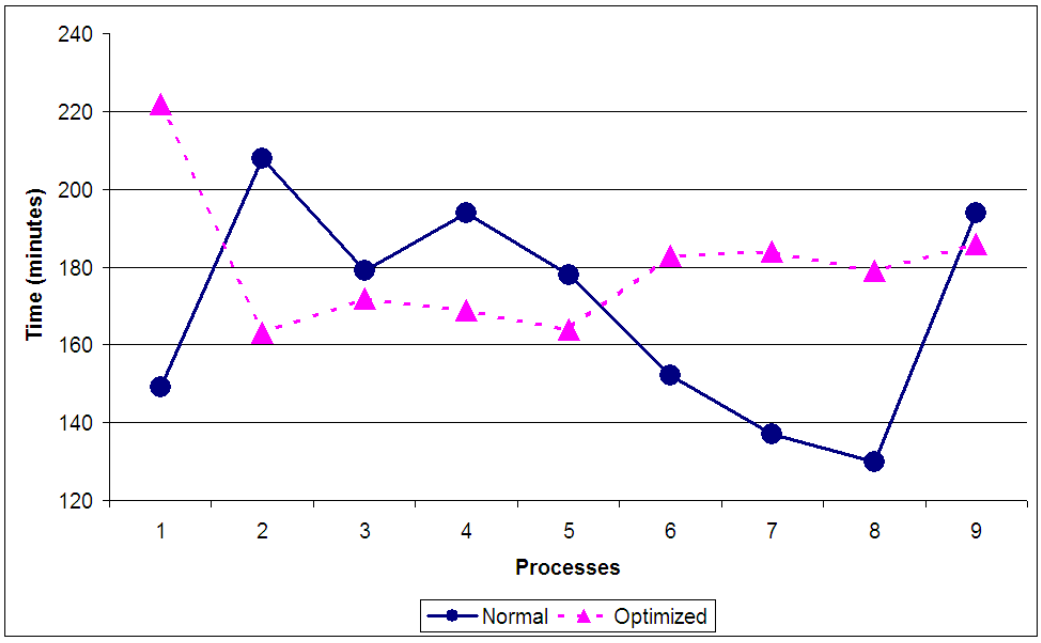

Figure 6.10: Parallel verification of specifications for test case #3 (approximately 5793 specifications per process).

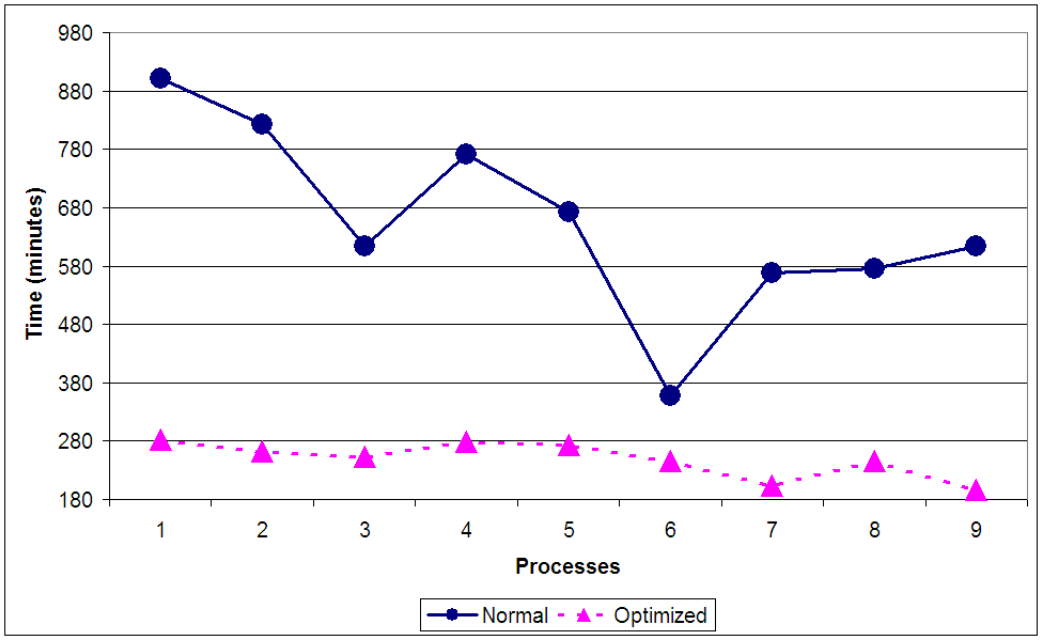

Figure 6.11: Parallel verification of specifications for test case #4 (approximately 8551 specifications per process).

Table 6.3: Summary of the evaluated data in the domRBAC simulator.

#	Roles	T_G Edges	Maximum Execution Time (msec)
1.	5*50	1031	< 1
2.	10*50	1784	< 1
3.	15*50	3179	≤ 1
4.	20*50	3421	≤ 1

6.5.4 Discussion

From Tables 6.1, 6.2 and 6.3, it is shown that secure inter-operation aware AC models (e.g. domRBAC) are able to perform faster than verification techniques. However, an AC model cannot assure or verify its correctness. The proposed verification technique fulfils the latter requirement using formal methods, and thus, it is able to assure the correctness of both the AC model and security policies. Moreover, it can be used as a valuable administrative service/tool for both the a priori and a posteriori enforcement of RBAC policies, and thus, ensuing a highly secured collaborative environment.

In addition, the proposed technique can benefit of automated testing with ACTS. The latter is integrated in ACPT as presented in Section 3.3. ACTS provides pairwise testing, which has become a popular approach to software quality assurance since it often provides effective error detection at low cost. Automating test generation via ACTS can provide much more thorough testing than is possible with most conventional methods. Moreover, testing has a sound empirical basis in the observation that software failures have been shown to be caused by the interaction of relatively few variables and stronger assurance for critical software can be provided by testing all variable interactions to an appropriate strength (Kuhn and Kacker, 2010). As a proof of concept, we provide a list of indicative results in Table 6.4. Specifically, we illustrate for each of our examined case the number of test cases generated by ACTS and their execution time.

6.6 Chapter summary

In this chapter, we demonstrated the security policy verification in multi-domain cloud systems. We applied a partially redefined ANSI INCITS 359-2004 to express both intra-domain and inter-domain administrations. We defined cyclic inheritance, privilege escalation, and SoD constraints properties in temporal logic. The verification of these security properties is vital because they assure the correctness of the enforced policy for secure inter-operation of cloud systems. We further verified the properties in a RBAC implementation. We modified the model checking technique in (Hu et al., 2011) in the context of RBAC models. We proposed

Table 6.4: Automated combinatorial with ACTS.

#	Number of Test cases	Execution Time (sec)
1.	62500	0.297
2.	250000	1.11
3.	562500	1.64
4.	1000000	2.843

Test Generation Profile:

Degree of interaction: 2
Mode: scratch
Algorithm: ipog
Progress Info: off
Debug mode: off
Verify Coverage: off

System Under Test:

Name: Fireeye Input
Number of Params: 3

a parser to tackle RBAC reasoning and handle role hierarchies. To the best of our knowledge, there is no equivalent secure inter-operation verification technique for RBAC that is applied to cloud systems. The efficiency of our technique was evaluated via a number of simulations. The results showed that our approach is feasible for large scale systems by independent and parallel processing. We conclude that our proposed technique as a management service/tool for system administration allows verifying either the a priori or a posteriori enforcement of RBAC policies correctness. Also, the integration of automating test generation via ACTS strengthens our technique with the capability to perform software assurance. To sum, the aforementioned renders the proposed technique an efficient approach to maintain secure inter-operation in multi-domain cloud systems.

Chapter 7

Conclusions

In this dissertation, we have investigated AC in the context of modern computing systems, as the Grid and Cloud. In particular, introductory information and related work are presented in Chapters 1- 2. Chapter 3 presented our SE methodology, and Chapter 4 was concerned with CC, a RE approach for the identification of security requirements and the evaluation of AC models and mechanisms. Chapter 5 was concerned with the modeling of a modern AC model whereas Chapter 6 was concerned with the verification of security properties using model checking. In this last chapter, we shortly summarize the contribution of this dissertation and we outline some topics left for future work.

7.1 Summary of the contributions

The contributions of this dissertation are multi-fold.

A methodology for the development and verification of AC models. We proposed in (Gouglidis and Mavridis, 2013) a SE methodology for the development and verification of AC systems, i.e., apply the CC during the requirements engineering stage and model checking during the verification stage. Therefore, we resulted in the development of AC systems that correspond to their initial requirements. The proposed SE methodology is independent of the applied development model of a system since the stages of requirements engineering and verification exist in most of them, and thus, the aforementioned stages can be

used transparently in any development model without breaking it.

Requirements engineering. We introduced a requirements engineering process entitled CC for the identification of requirements in modern computing environments and evaluation of existing AC models and mechanisms (Gouglidis and Mavridis, 2009, 2010). The CC is a layered approach for the definition of requirements. Additionally, it can be used as an evaluation tool. Its application resulted in the identification of vital importance AC requirements that should be fulfilled by modern AC approaches 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. Moreover, the application of the CC as an evaluation tool on existing AC models and mechanisms led to the identification of their pros and cons (Gouglidis and Mavridis, 2012b).

Access control. Based on the identified requirements, we proposed an enhanced RBAC model entitled domRBAC for collaborative applications, which is based on the ANSI INCITS 359-2004 AC model (Gouglidis and Mavridis, 2012a). 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 RBAC model. Furthermore, secure inter-operation is gradually assured among collaborating domains during role assignment in an automatic way. Yet, domRBAC, as an RBAC approach, intrinsically inherits all of its virtues such as ease of management, and SoD relationships with the latter also being supported in multiple domains. Through the implementation of a simulator based on the definitions of our proposed AC model and we conducted experimental studies, which have demonstrated domRBAC's high efficiency and performance.

Verification of security properties. To check the correctness of our proposed AC model, we provided a formal definition of secure inter-operation properties in temporal logic for RBAC policies, which can be verified using model checking techniques. The proposed technique consists of a generic one since it is based on NIST's (National Institute of Standards and Technology) generic model

checking technique and has been enriched with RBAC reasoning (Gouglidis et al., 2013a,b). Currently, it can be used in any RBAC model to verify indirectly the correctness of the secure inter-operation functions that implement the global security policy. Through a number of examples, we illustrated both the a priori and a posteriori enforcement of the defined secure inter-operation properties, which have to be verified in RBAC policies. A performance analysis of the proposed technique was also performed.

7.2 Future work

The research described in this dissertation can be extended along several directions.

Usage control. Our AC model is able to provide basic management regarding usage control. The latter has been demonstrated in Section 5.3 by using cardinality constraints along with context information. Since our approach is based on RBAC, it is able to enforce constraints only during an active session. In our model, we assume that during a session constraints cannot be modified, and thus, usage control is concerned in the context of active sessions and furthermore, they are not mutable. Therefore, our model in its current form cannot handle with continuity of decisions. Such cases are handled more efficiently in usage control models (i.e., $UCON_{ABC}$).

Verification of properties. The verification of security properties was implemented using the NuSMV symbolic model checker, as proposed by NIST. Through several evaluations of the technique, we have evaluated its performance. However, further improvements regarding its efficiency can be applied. Complimentary techniques could be used for its improvement, as combinatorial testing. Nevertheless, such a solution might not be an optimum since it doesn't verify all possible cases, but only a subset of them. A solution might be to migrate the technique in a relational model checker (i.e., Alloy) since conducted research results have shown that might perform better (Frappier et al., 2010).

Automated conversion of security requirements into security properties. In this dissertation, we managed to identify a number of security requirements that must be maintained to assure secure inter-operation. Security

requirements were described in native language, i.e., not using a formal language for their description. Therefore, the definition of security properties in temporal logic was performed manually. In future work, it could be possible to describe the initial security requirements using a formal language and automatically convert them into temporal logic formulas.

7.3 Closing remarks

This work has appeared in varying forms of journals and conference papers (see appendix D). In particular, our CC is described in (Gouglidis and Mavridis, 2009, 2010), and its usage as an evaluation tool in (Gouglidis and Mavridis, 2012b). Our proposed RBAC model is illustrated in (Gouglidis and Mavridis, 2011, 2012a). The SE methodology is proposed in (Gouglidis and Mavridis, 2013), and lastly, our approach to the verification of security properties is modelled in (Gouglidis et al., 2013a,b).

Appendix A

Statistical data

		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 A.1: 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 A.2: 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 A.3: 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 A.4: 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 A.5: 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 A.6: 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 A.7: 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 A.8: Statistics of experiment with 20 domains of 1000 roles each.

Appendix B

XSD schema

```
<?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 C

NuSMV code, outputs and counterexamples

Example of cyclic inheritance and autonomy properties as depicted in subsection 6.4.1

```
MODULE main
VAR
 ROLES : { dummy ,D1Ra ,D1Rb ,D2Rc ,D2Rd };
 OBJECTS : { dummy ,objA ,objB ,objC ,objD };
 OPERATIONS : { dummy ,read };
 RBAC_InterDomain : RBAC_InterDomain ( ROLES,
  OBJECTS, OPERATIONS);
ASSIGN
 next (ROLES) := ROLES ;
 next (OBJECTS) := OBJECTS ;
 next (OPERATIONS) := OPERATIONS ;
MODULE RBAC_InterDomain(ROLES, OBJECTS, OPERATIONS)
VAR
 decision : {Permit, Deny};
ASSIGN
 init (decision) := Deny ;
 next (decision) := case
```

```
  ROLES = D1Ra & OBJECTS = objA & OPERATIONS = read : Permit ;
  ROLES = D1Ra & OBJECTS = objB & OPERATIONS = read : Permit ;
  ROLES = D1Ra & OBJECTS = objC & OPERATIONS = read : Permit ;
  ROLES = D1Rb & OBJECTS = objB & OPERATIONS = read : Permit ;
  ROLES = D1Rb & OBJECTS = objC & OPERATIONS = read : Permit ;
  ROLES = D1Rb & OBJECTS = objA & OPERATIONS = read : Permit ;
  ROLES = D2Rc & OBJECTS = objC & OPERATIONS = read : Permit ;
  ROLES = D2Rc & OBJECTS = objD & OPERATIONS = read : Permit ;
  ROLES = D2Rc & OBJECTS = objA & OPERATIONS = read : Permit ;
  ROLES = D2Rc & OBJECTS = objB & OPERATIONS = read : Permit ;
  ROLES = D2Rd & OBJECTS = objD & OPERATIONS = read : Permit ;
  ROLES = D1Ra & OBJECTS = objD & OPERATIONS = read : Permit ;
  ROLES = D1Rb & OBJECTS = objD & OPERATIONS = read : Permit ;
  1 : Deny;
  esac;
-- Cyclic inheritance property
SPEC AG ((ROLES = D1Rb) & (OBJECTS = objA) &
 (OPERATIONS = read) -> AF  decision =  Deny)
-- Autonomy properties
SPEC AG ((ROLES = D1Ra) & (OBJECTS = objA) &
(OPERATIONS = read) -> AF  decision =  Permit)
SPEC AG ((ROLES = D1Ra) & (OBJECTS = objB) &
(OPERATIONS = read) -> AF  decision =  Permit)
SPEC AG ((ROLES = D1Rb) & (OBJECTS = objB) &
(OPERATIONS = read) -> AF  decision =  Permit)
SPEC AG ((ROLES = D2Rc) & (OBJECTS = objC) &
(OPERATIONS = read) -> AF  decision =  Permit)
SPEC AG ((ROLES = D2Rc) & (OBJECTS = objD) &
(OPERATIONS = read) -> AF  decision =  Permit)
SPEC AG ((ROLES = D2Rd) & (OBJECTS = objD) &
(OPERATIONS = read) -> AF  decision =  Permit)
*** This is NuSMV 2.4.3 (compiled on Tue May 22
14:08:54 UTC 2007)...
```

```
-- specification AG (((ROLES = D1Rb & OBJECTS = objA) &
OPERATIONS = read) -> AF decision = Deny) IN
RBAC_IntraDomain is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
-> State: 1.1 <-
  ROLES = D1Rb
  OBJECTS = objA
  OPERATIONS = read
  RBAC_InterDomain.decision = Deny
-> Input: 1.2 <-
-- Loop starts here
-> State: 1.2 <-
  RBAC_InterDomain.decision = Permit
-> Input: 1.3 <-
-> State: 1.3 <-
-- specification AG (((ROLES = D1Ra & OBJECTS = objA) &
OPERATIONS = read) -> AF decision = Permit)
IN RBAC_IntraDomain is true
-- specification AG (((ROLES = D1Ra & OBJECTS = objB) &
OPERATIONS = read) -> AF decision = Permit)
IN RBAC_IntraDomain is true
-- specification AG (((ROLES = D1Rb & OBJECTS = objB) &
OPERATIONS = read) ->
 AF decision = Permit) IN RBAC_IntraDomain is true
-- specification AG (((ROLES = D2Rc & OBJECTS = objC) &
OPERATIONS = read) -> AF decision = Permit)
IN RBAC_IntraDomain is true
-- specification AG (((ROLES = D2Rc & OBJECTS = objD) &
OPERATIONS = read) -> AF decision = Permit)
IN RBAC_IntraDomain is true
-- specification AG (((ROLES = D2Rd & OBJECTS = objD) &
```

```
OPERATIONS = read) -> AF decision = Permit)
IN RBAC_IntraDomain is true
```

Example of privilege escalation and SSD properties as depicted in subsection 6.4.2. Due to their identical nature only one of the four privilege escalation specifications is being verified in the following NuSMV source code.

```
MODULE main
VAR
 ROLES : {dummy, D1Ra, D1Rb, D1Rc, D1Rd, D1Re,
  D2Rf, D2Rg};
 OBJECT : {dummy, ObjA, ObjB, ObjC, ObjD,
  ObjE, ObjF, ObjG};
 OPERATION : {dummy, read};
 RBAC_InterDomain : RBAC_InterDomain(ROLES, OBJECT,
  OPERATION);
ASSIGN
 next (ROLES) := ROLES ;
 next (OBJECT) := OBJECT ;
MODULE RBAC_InterDomain(ROLES,OBJECT,OPERATION)
VAR
 decision : {Permit, Deny};
ASSIGN
 init (decision) := Deny ;
 next (decision) := case
  ROLES = D1Ra & OBJECT = ObjA & OPERATION = read : Permit ;
  ROLES = D1Ra & OBJECT = ObjB & OPERATION = read : Permit ;
  ROLES = D1Ra & OBJECT = ObjC & OPERATION = read : Permit ;
  ROLES = D1Ra & OBJECT = ObjD & OPERATION = read : Permit ;
  ROLES = D1Ra & OBJECT = ObjE & OPERATION = read : Permit ;
  ROLES = D1Ra & OBJECT = ObjG & OPERATION = read : Permit ;
  ROLES = D1Rb & OBJECT = ObjB & OPERATION = read : Permit ;
  ROLES = D1Rb & OBJECT = ObjE & OPERATION = read : Permit ;
  ROLES = D1Rb & OBJECT = ObjD & OPERATION = read : Permit ;
```

```
  ROLES = D1Rb & OBJECT = ObjC & OPERATION = read : Permit ;
  ROLES = D1Rb & OBJECT = ObjG & OPERATION = read : Permit ;
  ROLES = D1Re & OBJECT = ObjE & OPERATION = read : Permit ;
  ROLES = D1Rd & OBJECT = ObjD & OPERATION = read : Permit ;
  ROLES = D1Rd & OBJECT = ObjE & OPERATION = read : Permit ;
  ROLES = D1Rc & OBJECT = ObjC & OPERATION = read : Permit ;
  ROLES = D1Rc & OBJECT = ObjD & OPERATION = read : Permit ;
  ROLES = D1Rc & OBJECT = ObjE & OPERATION = read : Permit ;
  ROLES = D2Rf & OBJECT = ObjF & OPERATION = read : Permit ;
  ROLES = D2Rf & OBJECT = ObjG & OPERATION = read : Permit ;
  ROLES = D2Rf & OBJECT = ObjC & OPERATION = read : Permit ;
  ROLES = D2Rf & OBJECT = ObjD & OPERATION = read : Permit ;
  ROLES = D2Rf & OBJECT = ObjE & OPERATION = read : Permit ;
  ROLES = D2Rg & OBJECT = ObjG & OPERATION = read : Permit ;
  ROLES = D2Rg & OBJECT = ObjC & OPERATION = read : Permit ;
  ROLES = D2Rg & OBJECT = ObjD & OPERATION = read : Permit ;
  ROLES = D2Rg & OBJECT = ObjE & OPERATION = read : Permit ;
  1 : Deny;
  esac;
-- SSD verification for any user and SSD(D1Rb, D1Rc)
SPEC AG ( ((ROLES = D1Rb) & (OBJECT = ObjC) &
  (OPERATION = read)) | ((ROLES = D1Rc) &
  (OBJECT = ObjB) & (OPERATION = read)) ->
   AF  decision =  Deny)
-- Privilege escalation between D1Ra and
SPEC AG ( (ROLES = D1Ra) & (OBJECT = ObjC) &
(OPERATION = read)  -> AF  decision =  Deny)
*** This is NuSMV 2.4.3 (compiled on Tue May
22 14:08:54 UTC 2007)...
-- specification AG (((((ROLES = D1Rb) & OBJECT = ObjC) &
  OPERATION = read) | (((ROLES = D1Rc) & OBJECT = ObjB) &
  OPERATION = read)) -> AF decision = Deny)
  IN RBAC_InterDomain is false
```

```
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
-> State: 1.1 <-
 ROLES = D1Rb
 OBJECT = ObjC
 OPERATION = read
 RBAC_InterDomain.decision = Deny
-> Input: 1.2 <-
-- Loop starts here
-> State: 1.2 <-
 RBAC_InterDomain.decision = Permit
-> Input: 1.3 <-
-> State: 1.3 <-
-- specification AG (((ROLES = D1Ra & OBJECT = ObjC) &
OPERATION = read) -> AF decision = Deny)
IN RBAC_InterDomain is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
-> State: 2.1 <-
 ROLES = D1Ra
 OBJECT = ObjC
 OPERATION = read
 RBAC_InterDomain.decision = Deny
-> Input: 2.2 <-
-- Loop starts here
-> State: 2.2 <-
 RBAC_InterDomain.decision = Permit
-> Input: 2.3 <-
-> State: 2.3 <-
```

Appendix D

Publications

International journals

i. **Security policy verification for multi-domains in cloud systems**
(co-authors: Ioannis Mavridis, Vincent C. Hu)
International Journal of Information Security, Springer, 2013.
Impact Factor (2012): 0.480
Abstract: *The cloud is a modern computing paradigm with the ability to support a business model by providing multi-tenancy, scalability, elasticity, pay as you go and self-provisioning of resources by using broad network access. Yet, cloud systems are mostly bounded to single domains, and collaboration among different cloud systems is an active area of research. Over time, such collaboration schemas are becoming of vital importance since they allow companies to diversify their services on multiple cloud systems to increase both uptime and usage of services. The existence of an efficient management process for the enforcement of security policies among the participating cloud systems would facilitate the adoption of multi-domain cloud systems. An important issue in collaborative environments is secure interoperation. Stemmed from the absence of relevant work in the area of cloud computing, we define a model checking technique that can be used as a management service/tool for the verification of multi-domain cloud policies. Our*

proposal is based on NIST's (National Institute of Standards and Technology) generic model checking technique and has been enriched with RBAC reasoning. Current approaches, in Grid systems, are capable of verifying and detect only conflicts and redundancies between two policies. However, the latter cannot overcome the risk of privileged user access in multi-domain cloud systems. In this paper, we provide the formal definition of the proposed technique and security properties that have to be verified in multi-domain cloud systems. Furthermore, an evaluation of the technique through a series of performance tests is provided.

ii. **domRBAC: An Access Control Model for Modern Collaborative Systems**

(co-authors: Ioannis Mavridis)

Journal of Computers & Security, Elsevier, 2012.

Impact Factor (2012): 1.158

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

Refereed papers in proceedings of international conferences and workshops

i. Verification of Secure Inter-operation Properties in Multi-domain RBAC Systems
(co-authors: Ioannis Mavridis, Vincent C. Hu)
IEEE Trustworthy Computing Workshop, Gaithersburg, MD, USA, 2013.
Abstract: *The increased complexity of modern access control (AC) systems stems partly from the need to support diverse and multiple administrative domains. Systems engineering is a key technology to manage this complexity since it is capable of assuring that an operational system will adhere to the initial conceptual design and defined requirements. Specifically, the verification stage of an AC system should be based on techniques that have a sound and mathematical underpinning. Working on this assumption, model checking techniques are applied for the verification of predefined system properties, and thus, conducting a security analysis of a system. In this paper, we propose the utilization of automated and error-free model checking techniques for the verification of security properties in multi-domain AC systems. Therefore, we propose a formal definition in temporal logic of four AC system properties regarding secure inter-operation with Role-Based Access Control (RBAC) policies in order to be verified by using model checking. For this purpose, we demonstrate the implementation of a tool chain for expressing RBAC security policies, reasoning on role hierarchies and properly feeding the model checking process. The proposed approach can be applied in any RBAC model to efficiently detect non-conformance between an AC system and its security specifications. As a proof of concept, we provide*

examples illustrating the verification of the defined secure inter-operation properties in multi-domain RBAC policies.

ii. A Methodology for the Development and Verification of Access Control Systems in Cloud Computing
(co-authors: Ioannis Mavridis)
12th IFIP Conference on e-Business, e-Services, e-Society, IFIP I3E 2013.
Abstract: *Cloud computing is an emergent technology that has generated significant interest in the marketplace and is forecasted for high growth. Moreover, Cloud computing has a great impact on different type of users from individual consumers and businesses to small and medium size (SMBs) and enterprise businesses. Although there are many benefits to adopting Cloud computing, there are significant barriers to adoption, viz. security and privacy. In this paper, we focus on carefully planning security aspects regarding access control of Cloud computing solutions before implementing them and, furthermore, on ensuring they satisfy particular organizational security requirements. Specifically, we propose a methodology for the development of access control systems. The methodology is capable of utilizing existing security requirements engineering approaches for the definition and evaluation of access control models, and verification of access control systems against organizational security requirements using techniques that are based on formal methods. A proof of concept example is provided that demonstrates the application of the proposed methodology on Cloud computing systems.*

iii. **Role-based Secure Inter-operation and Resource Usage Management in Mobile Grid Systems**
(co-authors: Ioannis Mavridis)
Workshop in Information Security Theory and Practice, WISTP'11, 2011.
Abstract: *Dynamic inter-domain collaborations and resource sharing comprise two key characteristics of mobile Grid systems. However, inter-domain collaborations have proven to be vulnerable to conflicts that can lead to privilege escalation. These conflicts are detectable in inter-operation policies, and occur due to cross-domain role relationships. In addition, resource*

sharing requires to be enhanced with resource usage management in virtual organizations where mobile nodes act as resource providers. In this case the enforcement of resource usage policies and quality of service policies are required to be supported due to the limited capabilities of the devices. Yet, the ANSI INCITS 359-2004 standard RBAC model provides neither any policy conflict resolution mechanism among domains, nor any resource usage management functionality. In this paper, we propose the domRBAC model for access control in mobile Grid systems at a low administrative overhead. The domRBAC is defined as an extension of the standardized RBAC by incorporating additional functionality to cope with requirements posed by the aforementioned systems. As a result, domRBAC facilitates collaborations among domains under secure inter-operation, and provides support for resource usage management in the context of multi-domain computing environments, where mobile nodes operate as first-class entities.

iv. **On the Definition of Access Control Requirements for Grid and Cloud Computing Systems**
(co-authors: Ioannis Mavridis)
Third International ICST Conference on Networks for Grid Applications, GridNets 2009.
Abstract: *The emergence of grid and cloud computing systems has introduced new security concepts, so it requires new access control approaches. Traditional systems engineering processes can be enriched with helper approaches that can facilitate the definition of access control requirements in such complex environments. Looking towards a holistic approach on the definition of access control requirements, we propose a four-layer conceptual categorization. In addition, an example is given so that to demonstrate the utilization of the proposed categorization in a grid scenario for defining access control requirements, and evaluate their fulfilment vis-a-vis contemporary employed access control approaches.*

v. **A Foundation for Defining Security Requirements in Grid Computing**
(co-authors: Ioannis Mavridis)

13th Panhellenic Conference on Informatics, IEEE PCI 2009.

Abstract: *Despite the wide adoption by the scientific community, grid technologies have not been given the appropriate attention by enterprises. This is merely due to the lack of enough studying and defining security requirements of grid computing systems. More specifically, access control in grid systems has been addressed with the same models for collaborative systems based on distributed computing across multiple administrative domains. However, existing solutions are not based on a foundation for a holistic approach in grid access control. This paper aims to provide an adequate approach in this direction. Additionally, a comparative review of current access control models is provided in the context of our proposed four-layer conceptual grid categorization.*

Chapters in books

i. **Grid access control models and architectures**

(co-authors: Ioannis Mavridis)

Computational and Data Grids:Principles, Designs, and Applications, IGI Global, 2011.

Abstract: *In recent years, grid computing has become the focal point of science and enterprise computer environments. Access control in grid computing systems is an active research area given the challenges and complex applications. First, a number of concepts and terminology related to the area of grid access control are provided. Next, an analysis of the Role Based Access Control (RBAC) and Usage Control ABC ($UCON_{ABC}$) models is given, due to their adoption from the grid computing systems. Additionally, a presentation of well known grid access control architectures illustrates how the theoretical access control models are implemented into mechanisms. In a comparative review of the examined access control models and mechanisms, their pros and cons are exposed. Apart from the mapping of the access control area in grid computer systems, the given comparison renders valuable information for further advancement of current approaches.*

Presentations

i. **Towards new access control models for Cloud computing systems**
(co-authors: Ioannis Mavridis)
Kaspersky Lab - IT Security for the Next Generation, Conference for Young Professionals, European Cup 2011, University of Applied Sciences, Erfurt, Germany, January 28th-30th, 2011.
Abstract: *Cloud computing is a composition of existing technologies viz. virtualization technology, disk storage, processors and so on, which gained considerable attention mostly from the enterprise. Cloud security is an active research area, due to the newly introduced SPI service model and the different deployment models that require the revision of several security concepts. Specifically, in this paper we give a brief presentation of Cloud computing and the terminology of access control concepts used in the Cloud. Additionally, we elaborate on the identification of access control's distinctive characteristics in the aforementioned systems. We use a conceptual categorization, which is a systems engineering methodology, in order to identify a series of characteristics for access control in the Cloud computing paradigm. Furthermore, we present a comparative review of two prominent access control models for the Cloud, namely the Role-based Access Control model (RBAC) and the Usage Control model (UCONABC). We anticipate this initiative to help for the definition of concrete access control requirements and the design and implementation of new access control models, in order to accelerate the adoption of Cloud technologies.*

Project

The project has been funded by the Research Committee of the University of Macedonia in Greece.

Title: "Research Grant Program in support of Basic Research (5th framework)"

Project: "Development of a new improved role-based model for controlling access to Web services in Grid/Cloud computing environments"

Scientific coordinator: Associate Professor Ioannis Mavridis

Research collaborator: Antonios Gouglidis

Deliverables:

i. **Technical Guide to domRBAC Simulator**

ii. **domRBAC the Simulator version 0.1**

References

Lutz Schubert R.P. Horst Schwichtenberg C.T. Karanastasis E. Alexander Kipp, S.W. A new approach for classifying grids. Technical report, BEinGRID, 2008. 38, 44

Roberto Alfieri, Roberto Cecchini, Vincenzo Ciaschini, Luca dell'Agnello, Frohner, Alberto Gianoli, Karoly Lorentey, and Fabio Spataro. Voms, an authorization system for virtual organizations. In *European Across Grids Conference*, pages 33–40, 2003. 4, 21

Alloy. A language and tool for relational models, http://alloy.mit.edu/alloy/. URL `http://alloy.mit.edu/alloy/`. 35

Jörn Altmann and Daniel Veit, editors. *Grid Economics and Business Models, 4th International Workshop, GECON 2007, Rennes, France, August 28, 2007, Proceedings*, volume 4685 of *Lecture Notes in Computer Science*, 2007. Springer. ISBN 978-3-540-74428-3. 44

ANSI. ANSI INCITS 359-2004, role based access control, 2004. 14, 16, 55, 57, 60, 93, 95

C.A. Ardagna, S. De Capitani di Vimercati, S. Paraboschi, E. Pedrini, P. Samarati, and M. Verdicchio. Expressive and deployable access control in open web service applications. *Services Computing, IEEE Transactions on*, 4(2):96 –109, april-june 2011. ISSN 1939-1374. doi: 10.1109/TSC.2010.29. 4

Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. The MIT Press, 2008. ISBN 026202649X. 97, 107

Messaoud Benantar. *Access Control Systems: Security, Identity Management and Trust Models*. Springer-Verlag New York, Inc., 2005. 11

Rafae Bhatti, Basit Shafiq, Elisa Bertino, Arif Ghafoor, and James BD Joshi. X-gtrbac admin: A decentralized administration model for enterprise-wide access control. *ACM Transactions on Information and System Security (TISSEC)*, 8 (4):388–423, 2005. 12, 13

BOINC. Boinc all projects statistics - distributed computing statistics, 2009. URL `http://www.allprojectstats.com`. 39

Boost. Boost c++ libraries, http://www.boost.org/, 2011. URL `http://www.boost.org/`. 77

Martin A.P. Broadfoot, P.J. A critical survey of grid security requirements and technologies. Technical report, Oxford University Computing Laboratory, 2003. 42

Sabrina Capitani di Vimercati, Sara Foresti, and Pierangela Samarati. Authorization and access control. In Milan Petkovic and Willem Jonker, editors, *Security, Privacy, and Trust in Modern Data Management*, Data-Centric Systems and Applications, pages 39–53. Springer Berlin Heidelberg, 2007. ISBN 978-3-540-69861-6. URL `http://dx.doi.org/10.1007/978-3-540-69861-6_4`. 1, 88

Brian Carlstrom. Design and analysis of algorithms, problem set no.6 solutions, 2004. URL `http://carlstrom.com/stanford/cs161/ps6sol.pdf`. 75

David Chadwick. Authorisation in grid computing. *Information Security Technical Report*, 10(1):33–40, 2005. 4, 23

David W. Chadwick, Alexander Otenko, and Edward Ball. Role-based access control with x.509 attribute certificates. *IEEE Internet Computing*, 7(2):62–69, 2003. IEEE Computer Society. 4, 23

Anirban Chakrabarti. *Grid Computing Security, Grid Authorization Systems*, volume 6. Springer Berlin Heidelberg, 2007a. 11, 19, 20, 51

Anirban Chakrabarti. Managing trust in the grid. In *Grid Computing Security*, pages 215–246. Springer Berlin Heidelberg, 2007b. ISBN 978-3-540-44492-3. doi: 10.1007/978-3-540-44493-0_10. URL `http://dx.doi.org/10.1007/978-3-540-44493-0_10`. 43

Liang Chen and Jason Crampton. Inter-domain role mapping and least privilege. In *SACMAT '07: Proceedings of the 12th ACM symposium on Access control models and technologies*, pages 157–162, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-745-2. doi: http://doi.acm.org/10.1145/1266840.1266866. 13

David C. Chu and Marty Humphrey. Mobile ogsi.net: Grid computing on mobile devices. *Grid Computing, IEEE/ACM International Workshop on*, 0:182–191, 2004. ISSN 1550-5510. doi: http://doi.ieeecomputersociety.org/10.1109/GRID.2004.44. 5

Jason Crampton and George Loizou. Administrative scope and role hierarchy operations. In *In Proceedings of Seventh ACM Symposium on Access Control Models and Technologies (SACMAT 2002)*, pages 145–154, 2002. 13, 16

EGEE. Enabling grids for e-science, 2009. URL `http://eu-egee.org`. 39

David F. Ferraiolo, D. Richard Kuhn, and Ramaswamy Chandramouli. *Role-Based Access Control*. Artech House, Inc., 2003. ix, 1, 11, 13, 14, 15, 16, 25, 50, 62, 65, 66, 77, 92, 98

Kathi Fisler, Shriram Krishnamurthi, Leo A. Meyerovich, and Michael Carl Tschantz. Verification and change-impact analysis of access-control policies. In *Proceedings of the 27th international conference on Software engineering*, ICSE '05, pages 196–205, New York, NY, USA, 2005. ACM. ISBN 1-58113-963-2. doi: 10.1145/1062455.1062502. URL `http://doi.acm.org/10.1145/1062455.1062502`. 34

I. Foster, Zhao Yong, I. Raicu, and S. Lu. Cloud computing and grid computing 360-degree compared. In *Grid Computing Environments Workshop, 2008. GCE '08*, pages 1–10, 2008. 2, 5, 44

Ian Foster and Steven Tuecke. Describing the elephant: The different faces of it as service. *Queue*, 3(6):26–29, 2005. 11

Ian Foster, Carl Kesselman, and Steven Tuecke. The anatomy of the grid - enabling scalable virtual organizations. *International Journal of Supercomputer Applications*, 15:2001, 2001. 2

Marc Frappier, Benoît Fraikin, Romain Chossart, Raphaël Chane-Yack-Fa, and Mohammed Ouenzar. Comparison of model checking tools for information systems. In *Proceedings of the 12th international conference on Formal engineering methods and software engineering*, ICFEM'10, pages 581–596, Berlin, Heidelberg, 2010. Springer-Verlag. ISBN 3-642-16900-7, 978-3-642-16900-7. URL `http://dl.acm.org/citation.cfm?id=1939864.1939911`. 115

Inc Free Software Foundation. Using the gnu compiler collection., 2008. URL `http://gcc.gnu.org/onlinedocs/gcc-4.4.5/gcc/`. 80

Li Gong and Xiaolei Qian. Computational issues in secure interoperation, 1996. 90

Antonios Gouglidis and Ioannis Mavridis. A foundation for defining security requirements in grid computing. In *Proceedings of the 2009 13th Panhellenic Conference on Informatics*, PCI '09, pages 180–184, Washington, DC, USA, 2009. IEEE Computer Society. ISBN 978-0-7695-3788-7. 6, 114, 116

Antonios Gouglidis and Ioannis Mavridis. On the definition of access control requirements for grid and cloud computing systems. In *Networks for Grid Applications*, volume 25 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 19–26. Springer Berlin Heidelberg, 2010. ISBN 978-3-642-11733-6. 6, 33, 114, 116

Antonios Gouglidis and Ioannis Mavridis. Role-based secure inter-operation and resource usage management in mobile grid systems. In *Proceedings of the 5th IFIP WG 11.2 international conference on Information security theory and practice: security and privacy of mobile devices in wireless communication*, WISTP'11, pages 38–53, Berlin, Heidelberg, 2011. Springer-Verlag. ISBN

978-3-642-21039-6. URL http://dl.acm.org/citation.cfm?id=2017824.2017829. 7, 116

Antonios Gouglidis and Ioannis Mavridis. domRBAC: An access control model for modern collaborative systems. *Computers & Security*, 31(4):540 – 556, 2012a. ISSN 0167-4048. doi: 10.1016/j.cose.2012.01.010. URL http://www.sciencedirect.com/science/article/pii/S0167404812000144. 7, 34, 90, 97, 104, 105, 114, 116

Antonios Gouglidis and Ioannis Mavridis. Grid access control models and architectures. In *Computational and Data Grids: Principles, Applications and Design*, pages 217–234. IGI Global, 2012b. 114, 116

Antonios Gouglidis and Ioannis Mavridis. A methodology for the development and verification of access control systems in cloud computing. In *Collaborative, Trusted and Privacy-Aware e/m-Services*, pages 88–99. Springer Berlin Heidelberg, 2013. 6, 113, 116

Antonios Gouglidis, Ioannis Mavridis, and Vincent C. Hu. Verification of secure inter-operation properties in multi-domain rbac systems. In *International Workshop on Trustworthy Computing (TC 2013), co-located at the SERE 2013, Washington D.C. USA*. IEEE, 2013a. 115, 116

Antonios Gouglidis, Ioannis Mavridis, and VincentC. Hu. Security policy verification for multi-domains in cloud systems. *International Journal of Information Security*, pages 1–15, 2013b. ISSN 1615-5262. doi: 10.1007/s10207-013-0205-x. URL http://dx.doi.org/10.1007/s10207-013-0205-x. 115, 116

Graphviz. Graphviz - graph visualization software, 2010. URL http://www.graphviz.org/. 78

Graphviz. The dot language, 2012. URL http://www.graphviz.org/content/dot-language. 77

D Green. Grid technology. the future of the internet? the future of it?, 2002. URL https://ludit.kuleuven.be/nieuws/pdf/grid.pdf. 42

Gridipedia. Types of grid, 2009. URL http://www.gridipedia.eu/types-of-grids.html. 39

Gridmap. Gridmap visualizing the "state" of the grid, 2009. URL http://gridmap.cern.ch/gm. 39

Frode Hansen and Vladimir Oleshchuk. Conformance checking of RBAC policy and its implementation. In Robert Deng, Feng Bao, HweeHwa Pang, and Jianying Zhou, editors, *Information Security Practice and Experience*, volume 3439 of *Lecture Notes in Computer Science*, pages 144–155. Springer Berlin / Heidelberg, 2005. ISBN 978-3-540-25584-0. URL http://dx.doi.org/10.1007/978-3-540-31979-5_13. 34, 89

Keijo Heljanko. Model checking based software verification, 2006. URL http://iplu.vtt.fi/digitalo/modelchecking.pdf. 32

Hongxin Hu and GailJoon Ahn. Enabling verification and conformance testing for access control model. In *Proceedings of the 13th ACM symposium on Access control models and technologies*, SACMAT '08, pages 195–204, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-129-3. doi: 10.1145/1377836.1377867. URL http://doi.acm.org/10.1145/1377836.1377867. 34

Vincent C. Hu, D. Richard Kuhn, and Tao Xie. Property verification for generic access control models. In *Proceedings of the 2008 IEEE/IFIP International Conference on Embedded and Ubiquitous Computing - Volume 02*, EUC '08, pages 243–250, Washington, DC, USA, 2008. IEEE Computer Society. ISBN 978-0-7695-3492-3. doi: 10.1109/EUC.2008.22. URL http://dx.doi.org/10.1109/EUC.2008.22. 8, 34, 35, 88, 99

Vincent C. Hu, D. Richard Kuhn, Tao Xie, and JeeHyun Hwang. Model checking for verification of mandatory access control models and properties. *International Journal of Software Engineering and Knowledge Engineering*, 21(1): 103–127, 2011. 33, 35, 36, 89, 96, 110

Graham Hughes and Tevfik Bultan. Automated verification of access control policies using a SAT solver. *Int. J. Softw. Tools Technol. Transf.*, 10(6):503–

520, October 2008. ISSN 1433-2779. doi: 10.1007/s10009-008-0087-9. URL http://dx.doi.org/10.1007/s10009-008-0087-9. 34, 35

JeeHyun Hwang, Tao Xie, Vincent Hu, and Mine Altunay. ACPT: A tool for modeling and verifying access control policies. In *Proceedings of the 2010 IEEE International Symposium on Policies for Distributed Systems and Networks*, POLICY '10, pages 40–43, Washington, DC, USA, 2010. IEEE Computer Society. ISBN 978-0-7695-4238-6. doi: 10.1109/POLICY.2010.22. URL http://dx.doi.org/10.1109/POLICY.2010.22. 35, 99, 101

ISO/IEC-13568. Information technology z - formal specification notation - syntax, type system and semantics, 2002. International Standard. 59

ITU-T. X.812 recommendation, 1995. ix, 19, 26, 77

Pramod A. Jamkhedkar, Gregory L. Heileman, and Chris C. Lamb. An interoperable usage management framework. In *Proceedings of the tenth annual ACM workshop on Digital rights management*, DRM '10, pages 73–88, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0091-9. doi: http://doi.acm.org/10.1145/1866870.1866885. URL http://doi.acm.org/10.1145/1866870.1866885. 56

Karthick Jayaraman, Vijay Ganesh, Mahesh Tripunitara, Martin Rinard, and Steve Chapin. Automatic error finding in access-control policies. In *Proceedings of the 18th ACM conference on Computer and communications security*, CCS '11, pages 163–174, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0948-6. doi: 10.1145/2046707.2046727. URL http://doi.acm.org/10.1145/2046707.2046727. 34

Somesh Jha, Ninghui Li, Mahesh Tripunitara, Qihua Wang, and William Winsborough. Towards formal verification of role-based access control policies. *IEEE Transactions on Dependable and Secure Computing*, 5:242–255, 2008. ISSN 1545-5971. doi: http://doi.ieeecomputersociety.org/10.1109/TDSC.2007.70225. 34

Jeffrey O. Kephart. Research challenges of autonomic computing. In *Proceedings of the 27th international conference on Software engineering*, ICSE '05, pages

15–22, New York, NY, USA, 2005. ACM. ISBN 1-58113-963-2. doi: 10.1145/1062455.1062464. URL http://doi.acm.org/10.1145/1062455.1062464. 43

Gerald Kotonya and Ian Sommerville. *Requirements Engineering - Processes and Techniques*. John Wiley & Sons, 1998. URL http://www.comp.lancs.ac.uk/computing/resources/re/. 28, 30

PL Krapivsky and S. Redner. Network growth by copying. *Physical Review E*, 71(3):036118, 2005. 105

Klaus Krauter, Rajkumar Buyya, and Muthucumaru Maheswaran. A taxonomy and survey of grid resource management systems for distributed computing. *Softw. Pract. Exper.*, 32(2):135–164, February 2002. ISSN 0038-0644. doi: 10.1002/spe.432. URL http://dx.doi.org/10.1002/spe.432. 42

D. Richard Kuhn and Dr. Raghu Kacker. Automated combinatorial test methods - beyond pairwise testing. 2010. 110

Heba Kurdi, Maozhen Li, and Hamed Al-Raweshidy. A classification of emerging and traditional grid systems. *IEEE Distributed Systems Online*, 9(3):1–, March 2008. ISSN 1541-4922. doi: 10.1109/MDSO.2008.8. URL http://dx.doi.org/10.1109/MDSO.2008.8. 39, 42

Leslie Lamport. *Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers*. Addison-Wesley Professional, 1st edition, 2002. 35

N. Li and M.V. Tripunitara. Security analysis in role-based access control. *ACM Transactions on Information and System Security (TISSEC)*, 9(4):391–420, 2006. 32

Ninghui Li, Ji-Won Byun, and Elisa Bertino. A critique of the ANSI standard on role-based access control. *IEEE Security and Privacy*, 5(6):41–49, 2007. 15, 62

T. Mather, S. Kumaraswamy, and S. Latif. *Cloud Security and Privacy: An Enterprise Perspective on Risks and Compliance*. Oreilly & Associates Inc, 2009. 2, 3, 12, 24

NetworkX. Networkx, http://networkx.lanl.gov/, 2012. URL `http://networkx.lanl.gov/`. 78, 80, 105

Gustaf Neumann and Mark Strembeck. 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*, pages 65–79, New York, NY, USA, 2003. ACM. ISBN 1-58113-681-1. doi: http://doi.acm.org/10.1145/775412.775421. 59

NIST. Combinatorial and pairwise testing, http://csrc.nist.gov/groups/sns/acts/, 2012. URL `http://csrc.nist.gov/groups/SNS/acts/`. 35

NIST. Guide to attribute based access control (abac) definition and considerations, 2013. URL `http://csrc.nist.gov/publications/drafts/800-162/sp800_162_draft.pdf`. 4

Nokia. Qt - cross-platform application and ui framework, 2011. URL `http://qt.nokia.com/`. 77

NuSMV. A new symbolic model checker, http://nusmv.fbk.eu/. URL `http://nusmv.fbk.eu/`. 35, 101, 106

Esko Nuutila. *Efficient transitive closure computation in large digraphs*. PhD thesis, Acta Polytechnica Scandinavica, Helsinki University of Technology, 1995. 68

OASIS. Oasis extensible access control markup language (xacml) tc, 2011. URL `http://www.oasis-open.org/`. 4, 77

Sejong Oh and Ravi Sandhu. A model for role administration using organization structure. In *Proceedings of the seventh ACM symposium on Access control models and technologies*, pages 155–162. ACM, 2002. 13, 16

Jaehong Park and Ravi Sandhu. The ucon abc usage control model. *ACM Trans. Inf. Syst. Secur.*, 7(1):128–174, 2004. ix, 14, 17, 18

L. Pearlman, V. Welch, I. Foster, C. Kesselman, and S. Tuecke. A community authorization service for group collaboration, 2002. 4, 20

Mell Peter and Grance Timothy. The NIST definition of cloud computing, September 2011. URL `http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf`. 2, 3

T. Phan, L. Huang, and C. Dulan. Challenge: integrating mobile wireless devices into the computational grid. In *Proceedings of the 8th annual international conference on Mobile computing and networking*, page 278. ACM, 2002. ISBN 158113486X. 5

Paul Purdom. A transitive closure algorithm. *BIT Numerical Mathematics*, 10:76–94, 1970. ISSN 0006-3835. URL `http://dx.doi.org/10.1007/BF01940892`. 10.1007/BF01940892. 68

P. Racz, J.E. Burgos, N. Inacio, C. Morariu, V. Olmedo, V. Villagra, R.L. Aguiar, and B. Stiller. Mobility and qos support for a commercial mobile grid in akogrimo. In *Mobile and Wireless Communications Summit, 2007. 16th IST*, pages 1 –5, 2007. doi: 10.1109/ISTMWC.2007.4299247. 5

Venkata Bhamidipati Ravi Sandhu. The ascaa principles for next-generation role-based access control. In *Proc. 3rd International Conference on Availability, Reliability and Security (ARES)*, volume 6, pages xxvii–xxxii, Barcelona, Spain, 2008. 64

Ravi Sandhu and Qamar Munawer. The arbac99 model for administration of roles. In *Computer Security Applications Conference, 1999.(ACSAC'99) Proceedings. 15th Annual*, pages 229–238. IEEE, 1999. 13

Ravi Sandhu and Jaehong Park. Usage control: A vision for next generation access control. In *Computer Network Security*, volume 2776, pages 17–31. Springer Berlin / Heidelberg, 2003. 3, 14

Ravi Sandhu, Venkata Bhamidipati, and Qamar Munawer. The arbac97 model for role-based administration of roles. *ACM Trans. Inf. Syst. Secur.*, 2(1): 105–135, 1999. 16

Ravi S. Sandhu and Pierangela Samarati. Access control: Principles and practice. *IEEE Communications Magazine*, 32:40–48, 1994. 1, 11

Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996. 14

SAnToS Laboraroty. Spec patterns, responce property pattern, http://patterns.projects.cis.ksu.edu/, 2012. URL `http://patterns.projects.cis.ksu.edu/documentation/patterns/response.shtml`. 97

Andreas Schaad and Jonathan D. Moffett. A lightweight approach to specification and analysis of role-based access control extensions. In *Proceedings of the seventh ACM symposium on Access control models and technologies*, SACMAT '02, pages 13–22, New York, NY, USA, 2002. ACM. ISBN 1-58113-496-7. doi: 10.1145/507711.507714. URL `http://doi.acm.org/10.1145/507711.507714`. 89

Andreas Schaad, Jonathan Moffett, and Jeremy Jacob. The role-based access control system of a european bank: a case study and discussion. In *Proceedings of the sixth ACM symposium on Access control models and technologies*, pages 3–9. ACM, 2001. 105

Christian Schlager, Manuel Sojer, Bjorn Muschall, and Gunther Pernul. Attribute-based authentication and authorisation infrastructures for e-commerce providers. In *E-Commerce and Web Technologies*, volume 4082, pages 132–141. Springer Berlin / Heidelberg, 2006. 19

SETI@home, 2009. URL `http://setiathome.ssl.berkeley.edu`. 39

Basit Shafiq, James B. D. Joshi, Elisa Bertino, and Arif Ghafoor. Secure interoperation in a multidomain environment employing RBAC policies. *IEEE Trans. on Knowl. and Data Eng.*, 17(11):1557–1577, 2005. ISSN 1041-4347. doi: http://dx.doi.org/10.1109/TKDE.2005.185. 13, 49, 50, 59, 65, 81, 86, 87, 89, 90, 93, 97

Mohamed Shehab, Elisa Bertino, and Arif Ghafoor. SERAT: Secure role mapping technique for decentralized secure interoperability. In *SACMAT '05: Proceed-*

ings of the tenth ACM symposium on Access control models and technologies, pages 159–167, New York, NY, USA, 2005. ACM. ISBN 1-59593-045-0. doi: http://doi.acm.org/10.1145/1063979.1064007. 13

Jawed I. A. Siddiqi and M. Chandra Shekaran. Requirements engineering: The emerging wisdom. *IEEE Software*, 13(2):15–19, 1996. 28, 30

Ian Sommerville. *Software engineering (6th ed.)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001. ISBN 0-201-39815-X. ix, 28, 31

Ian Sommerville. *Software Engineering*. Addison-Wesley, 2010. 29

SPIN. The SPIN model checker, http://spinroot.com/spin/. URL `http://spinroot.com/spin/`. 35

R. Stevens. *Systems Engineering: Coping With Complexity*. Prentice Hall, 1998. ISBN 9780130950857. URL `http://books.google.gr/books?id=PPBp2RwMFWwC`. 28, 29, 34, 45

Wuliang Sun, Robert France, and Indrakshi Ray. Rigorous analysis of uml access control policy models. In *Policies for Distributed Systems and Networks (POLICY), 2011 IEEE International Symposium on*, pages 9–16. IEEE, 2011. 34

Mary R. Thompson, Abdelilah Essiari, and Srilekha Mudumbai. Certificate-based authorization policy in a pki environment. *ACM Trans. Inf. Syst. Secur.*, 6(4): 566–588, 2003. 4, 22

William Tolone, Gail-Joon Ahn, Tanusree Pai, and Seng-Phil Hong. Access control in collaborative systems. *ACM Comput. Surv.*, 37(1):29–41, 2005. 48, 50

W3C. Document object model (dom), 2005. URL `http://www.w3.org/DOM/`. 77

W3C. Xml technology, 2011. URL `http://www.w3.org/standards/xml/`. 77

XML-DEV. Simple api for xml, 2011. URL `http://www.xml.org/xml-dev`. 77

Eric Yuan and Jin Tong. Attributed based access control (abac) for web services, 2005. 3, 14

G. Zhang and M. Parashar. Dynamic context-aware access control for grid applications. In *Grid Computing, 2003. Proceedings. Fourth International Workshop on*, pages 101–108. IEEE, 2004. ISBN 076952026X. 13

Xinwen Zhang, Masayuki Nakae, Michael J. Covington, and Ravi Sandhu. A usage-based authorization framework for collaborative computing systems, 2006. 5, 24

Xinwen Zhang, Masayuki Nakae, Michael J. Covington, and Ravi Sandhu. Toward a usage-based security framework for collaborative computing systems. *ACM Trans. Inf. Syst. Secur.*, 11(1):1–36, 2008. 14, 17, 86