Access Control Mechanism for Authorizing Grid Resources: A Distributed and Fine-Grained Approach

A thesis submitted during 2012 to the University of Hyderabad in partial fulfillment of a PhD degree in Department of Computer and Information Sciences, School of Mathematics and Computer and Information Sciences

by

Mustafa Kaiiali

Department of Computer and Information Sciences

School of Mathematics and Computer & Information Sciences

University of Hyderabad

(P.O.) Central University, Gachibowli

Hyderabad - 500 046

Andhra Pradesh

India

May 2012

To

The soul of my spiritual teacher ***Basel Al-Jaser****...*

My beloved daughter ***Tumouh****...*

My Precious ***Grandfather****...*

The soul of my ***Grandmother*** *may God bless her...*

My dear ***Parents, Brothers, & Sisters****...*

CERTIFICATE

This is to certify that the thesis entitled "**Access Control Mechanism for Authorizing Grid Resources: A Distributed and Fine-Grained Approach**" submitted by **Mr. Mustafa Kaiiali** bearing Reg. No **08MCPC19** in partial fulfillment of the requirements for the award of **Doctor of Philosophy in Computer Science** is a bona fide work carried out by him under our supervision and guidance. The thesis has not been submitted previously in part or in full to this or any other University or Institution for the award of any degree or diploma.

Prof. C. R. Rao
(Supervisor)
DCIS, UoHyd.

Dr. Rajeev Wankar
(Supervisor)
DCIS, UoHyd.

Head of the Department
Prof. P. N. Girija
Department of Computer
&
Information Sciences,
University of Hyderabad,
Hyderabad.

Dean of the School
Prof. T. Amaranath
School of Mathematics
&
Computer/Information Sciences
University of Hyderabad,
Hyderabad.

Declaration

I, **Mustafa Kaiiali** hereby declare that this thesis entitled "**Access Control Mechanism for Authorizing Grid Resources: A Distributed and Fine-Grained Approach**" submitted by me under the guidance and supervision of **Prof. C. R. Rao** and **Dr. Rajeev Wankar** is a bona fide research work. I also declare that it has not been submitted previously in part or in full to this University or any other University or Institution for the award of any degree or diploma.

Date:

Mustafa Kaiiali

Reg. No. 08MCPC19

Acknowledgments

Initially I would like to acknowledge the tireless help of my supervisors, **Prof. C.R. Rao** and **Dr. Rajeev Wankar**. Their invaluable guidance enabled me to accomplish my degree successfully in time. They have always allowed me complete freedom to define and explore my own directions in research. Their vast experience, & profound knowledge have been a constant source for me throughout my work.

I am also so grateful to the respected DRC members **Prof. Arun Agarwal** and **Prof. Hrushikesha Mohanty**, for their valuable suggestions and motivations. I would like to express my thanks to the Head of Department **Prof. P.N. Girija** and to the Dean of the School of Mathematics and Computer/Information Sciences **Prof. T. Amaranath** for their cooperation.

My special thanks to **Amer Sallam**, **Rafah Mohammed**, **Mr. P.S.V.S. Sai Prasad** and other research scholars in our department.

Again I would like to specially thank **Prof. Arun Agarwal** who allowed me to use his lab for experimentations.

Mustafa Kaiiali

Contents

List of Figures

List of Tables

List of Algorithms

List of Symbols

1 CHCM ... Concurrent HCM.

2 DHCM ... Distributed HCM.

3 GAG ... Grid Authorization Graph.

4 GAS ... Grid Authorization Simulator.

5 HCM ... Hierarchical Clustering Mechanism.

6 HDCM ... Hamming Distance Caching Mechanism.

7 PAAC ... Posteriori Analysis of Authorization Complexity.

8 PCM ... Primitive Clustering Mechanism.

9 RSPCM ... Rough Set based PCM.

10 TCM ... Temporal Caching Mechanism.

11 UARG ... User's Authorized Resource Group.

List of Publications

Most of the work presented in the thesis has been published or is under review for publication in referred International Journals and Conferences:

1. ***"Enhancing the Hierarchical Clustering Mechanism of Storing Resources' Security Policies in a Grid Authorization System"*** in the proceeding of the 6th International Conference on Distributed Computing and Internet Technology (ICDCIT), vol. 5966 of LNCS, pp. 134-139, Springer, 2010.

2. ***"New Efficient Tree-Building Algorithms for Creating HCM Decision Tree in a Grid Authorization System"*** in the proceeding of the 2nd international conference on Network Applications Protocols and Services (NETAPPS), pp. 1-6, Malaysia, Sept. 2010.

3. ***"A Rough Set based PCM for authorizing grid resources"*** in the proceeding of the IEEE 10th International Conference on Intelligent Systems Design and Applications (ISDA), Cairo, 29th Dec 2010, pp(s):391 - 396.

4. ***"Aspects in Grid Authorization"*** in the Proceeding of the International Conference on Nanoscience, Engineering & Advanced Computing (ICNEAC-2011).

5. ***"Concurrent HCM for Authorizing Grid Resources"*** in the proceeding of the 8th International Conference on Distributed Computing and Internet Technology (ICDCIT), vol. 7154 of LNCS, pp. 255-256, Springer, 2012.

6. ***"Grid Authorization Graph"*** is communicated to FGCS journal.

7. ***"Aspects in Grid Authorization"*** is invited as a chapter for the book entitled ***"Emerging Trends in Cloud Computing and Distributed Systems"*** published by iConcept Press.

Abstract

The heterogeneity, massiveness and dynamism of grid environments complicate and delay the authorization process. This brings out the need for a fast and scalable fine-grained access control (FGAC) mechanism to cater well to grid requirements.

Every resource in a grid has its own security policy, which may be identical or quite similar to other security policies of some other resources. This leads to huge repetitions in checking security rules while finding the user's authorized resource group.

This fact motivates the idea of the ability to cluster the resources which have similar security policies in a hierarchical manner based on their shared security rules. So the authorization system can use a hierarchical decision tree to find the user's authorized resource group more quickly. We have introduced this idea in the Hierarchical Clustering Mechanism (HCM).

HCM outperforms the existing security policy storing mechanisms as the Brute Force Approach and the Primitive Clustering Mechanism (PCM). However, Building the decision tree of HCM out of the security policies of the individual resources is not a trivial process. An algorithm which keenly chooses the root security rule of the tree and the root security rule of each sub-tree is required.

Moreover, HCM is slightly an expensive process in terms of computations required to create the decision tree and memory consumption. So it may cause a bottleneck

for small systems. A Rough Set based PCM (RSPCM) is proposed for that. RSPCM is more simple compared to HCM. On the other hand, it is more efficient and less redundant compared to the normal PCM.

Furthermore, the stability of HCM against the dynamic changes in the grid and the suitability of HCM for several kinds of grid has to be analyzed. Several tools like the Temporal Caching Mechanism, the Hamming Distance Caching Mechanism and the Incremental Updation of the decision tree have been integrated in HCM for better scalability. Also, the concurrent version of HCM has been introduced in order to make HCM more scalable against the number of grid users.

HCM demands a centralized authorization database where all the resources' security policies have been kept in order to build the decision tree. This is difficult to achieve in a heterogeneous, multi-domain environment like the grid. In our work, we have shown how HCM can be used in Cross-Domain authorization mode by introducing three different model each suits a specific scenario.

HCM is not totally free of redundancy. Moreover, HCM cannot easily describe the OR-based security policies. A Grid Authorization Graph (GAG) is proposed with several tools embedded to overcome all the limitations of HCM and entirely eliminates the redundancy in checking security rules. GAG is considered as our proposed solution in this thesis to the Grid Authorization problem.

Finally, HCM/GAG should fit into the current grid authorization architecture. For that embedding HCM/GAG in GT4 authorization framework is illustrated in this thesis with various components to be added or modified. All experiments are done on the Grid Authorization Simulator (GAS) which is a C# simulator developed in our Labs.

Chapter 1

Introduction

Grid systems have attracted the researchers in the last decade. Grid authorization system is one of the most studied topics. Several researchers proposed many different facets of authorization. The majority of the Grid Authorization research was concentrated on access control mechanisms, security policy languages or authorization sequences. Authorization process has inherent latency which has not been noticed by the researchers (to the best of our knowledge).

Although grid resources works in cooperative manner, the access control mechanisms analyze their security policies in an isolated way that leads to huge redundancy in the authorization process. Akenti [2, 3] system addresses this problem; still it is at a primitive level. This motivated us to investigate this issue in depth.

This chapter introduces Grid computing, Grid security and different issues in Grid security with emphasis on Grid Authorization and Access Controls. An overview of all contributions and the layout of the thesis is presented in this chapter.

1.1 Grid Computing, Concept and Definition

Normal computers are no longer enough to solve the world's recent challenging problems. Computationally intensive grand challenge applications, such as weather forecasting and earthquake analysis can never be performed on classical computers. Moreover, using supercomputers is not an economical solution for many organizations. They are expensive; hard to maintain; and consume a lot of power. In Present days, many organizations find the solution in some form of distributed computing.

Distributed computing is a field of computer science that deals with distributed systems. A distributed system is "a collection of independent computers that appears to its users as a single coherent system" (Andrew S. Tanenbaum)[4]. The spread of high-speed broadband networks serves to make the distributed computing more available to organizations than supercomputers.

Grid computing is considered to be a new form of distributed computing. The concept of Grid in Computing was first introduced by Leonard Kleinrock in 1969 when he wrote: "We will probably see the spread of computer utilities, which, like present electric and telephone utilities, will service individual homes and offices across the country".

In 1998, Ian Foster and Carl Kesselman [5] provided an initial definition in their book "*The Grid: Blueprint for a New Computing Infrastructure*". A **Computational Grid** is a hardware and software infrastructure that provides dependable, consistent, pervasive and inexpensive access to high-end computational capabilities.

In 2001, Foster, Kesselman and Tuecke [6] refined their definition of a Grid to "Coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations".

Foster [7] later produced a 3-points checklist that could be used to understand exactly what can be identified as a Grid system. Grid is a system that:

- **Coordinates resources that are not subject to centralized control**: Grid resources are owned by different companies or under the control of different administrative units. So, they do not require a dedicated central location for their management. They can be managed by different virtual organizations (VOs) through an appropriate mechanisms or protocols. Grid should be capable of integrating and coordinating resources that live within different control domains. And addressing the issues of security, policy, payment, membership, etc. Otherwise, we are dealing with a local management system.

- **Use standard, open, general purpose protocols and interfaces that address such fundamental issues** as authentication, authorization, resource discovery and resource access. It is important that these protocols and interfaces be standard and open. Otherwise, we are dealing with an application specific system

- **Deliver various nontrivial qualities of service which is able to meet complex user demands:** A Grid allows its constituent resources to be used in a coordinated fashion to deliver various qualities of service, relating for example to response time, throughput, availability, and security, and/or co-allocation of

multiple resource types to meet complex user demands, so that the utility of the combined system is significantly greater than that of the sum of its parts.

A widely acceptable definition was given by Raj Kumar Buyya in 2000, he stated: "Grid is a type of parallel and distributed system that enables the sharing, selection, and aggregation of geographically distributed autonomous resources dynamically at runtime depending on their availability, capability, performance, cost, and users' quality-of-service requirements"

1.2 Grid Security Issues

Grid computing offers unique security challenges. Close understanding to grid environments is required to address these issues. Grid resources are heterogeneous, decentralized and distributed under multiple administrative domains where resource sharing and coordination represent a real challenge. Security mechanisms that allow seamless access to Grid resources for users of different organizations is required.

Anirban Chakrabarti [1] has written a chapter entitled "*Taxonomy of Grid Security Issues*" in his book **Grid Computing Security**. On the top of his work, we briefly discuss various aspects in grid security emphasizing more on Grid Authorization. This section would provide a briefed overall landscape on Grid Security.

Figure 1.1 shows the categorization of different grid security issues presented earlier in [1]. They are categorized into three main categories: *architecture related issues*, *infrastructure related issues*, and *management related issues*.

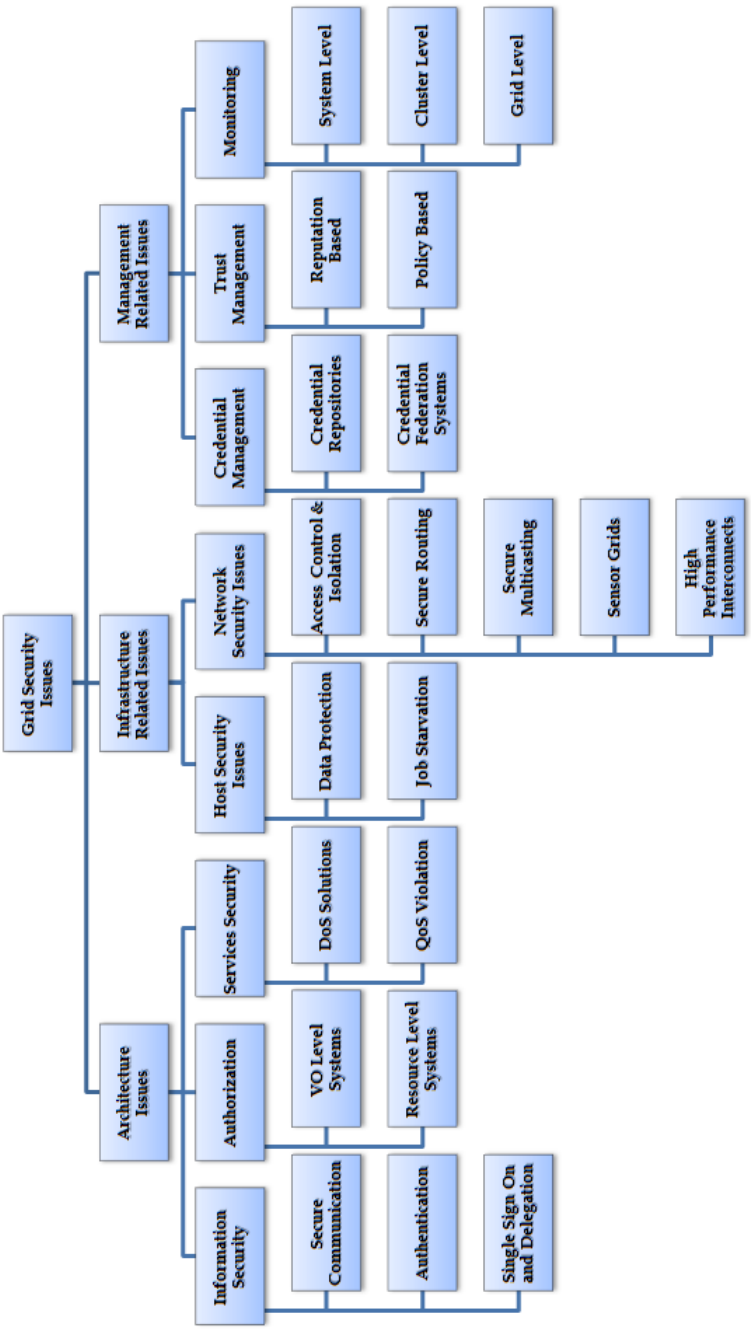

Figure 1.1: Taxonomy of Grid Security Issues

- **Architecture Related Issues:**

  These issues addresses the concern of the grid system as a whole. They generally destabilize the whole system and hence an architecture level solution is needed. Users of the grid are concerned about the data processed by the grid and hence there is a requirement to protect the data confidentiality and integrity, as well as user authentication. We categorize these requirements under *Information Security*. Similarly, *Authorization* is a critical requirement for grid systems. Finally, there are issues where users of the grid system may be denied the service of the grid *(DoS)* or the Quality-of-Service *(QoS) is violated*.

- **Infrastructure Related Issues:**

  These issues are relate to the network and host components which constitute the grid infrastructure. Host level security issues are those issues that make a host apprehensive about affiliating itself to the grid system. The main sub issues here are: *data protection*, *job starvation*, and *host availability*.

  A grid involves running alien code in the host system. Therefore, the host can be apprehensive about the part of the system which contains important data. Similarly, a host can also be concerned about the jobs that is running locally. The external jobs should not reduce the priority of the local jobs, and hence lead to job starvation. Similarly, if the host is a server, it can be concerned about its own availability. There should be mechanisms to prevent the system from going down resulting in denial-of-service to the clients attached to the host.

- **Management Related Issues:**

  The third set of issues pertains to the management of the grid. Managing trust is critical in any distributed system. Managing credentials is absolutely important in grid systems because of the heterogeneous nature of the grid. Grid systems also require some amount of resource monitoring for auditing purposes.

In the *Architecture Related Issues*, Authorization was a weak component. In our research we have enhanced the authorization techniques used in Grid to satisfy its requirements by introducing the Hierarchical Clustering Mechanism and the Grid Authorization Graph. Figure 1.2 shows the taxonomy of Grid security updated by our contributions (contributions appear in Red Color)

1.3 Motivation

The heterogeneity, massiveness and dynamism of the grid offers several challenges to authorization systems. A Grid system is composed of several autonomous domains, and it has huge number of resources and users which dynamically join and leave the grid. Authorization in such a system needs to be flexible and scalable to fulfill the requirement of the environment.

Authorization has been studied thoroughly all over the thesis. Generally, every resource in a grid has its own security policy, which may be identical or quite similar to other security policies of some other resources. This leads to huge repetitions in checking security rules while trying to find the user's authorized resource group.

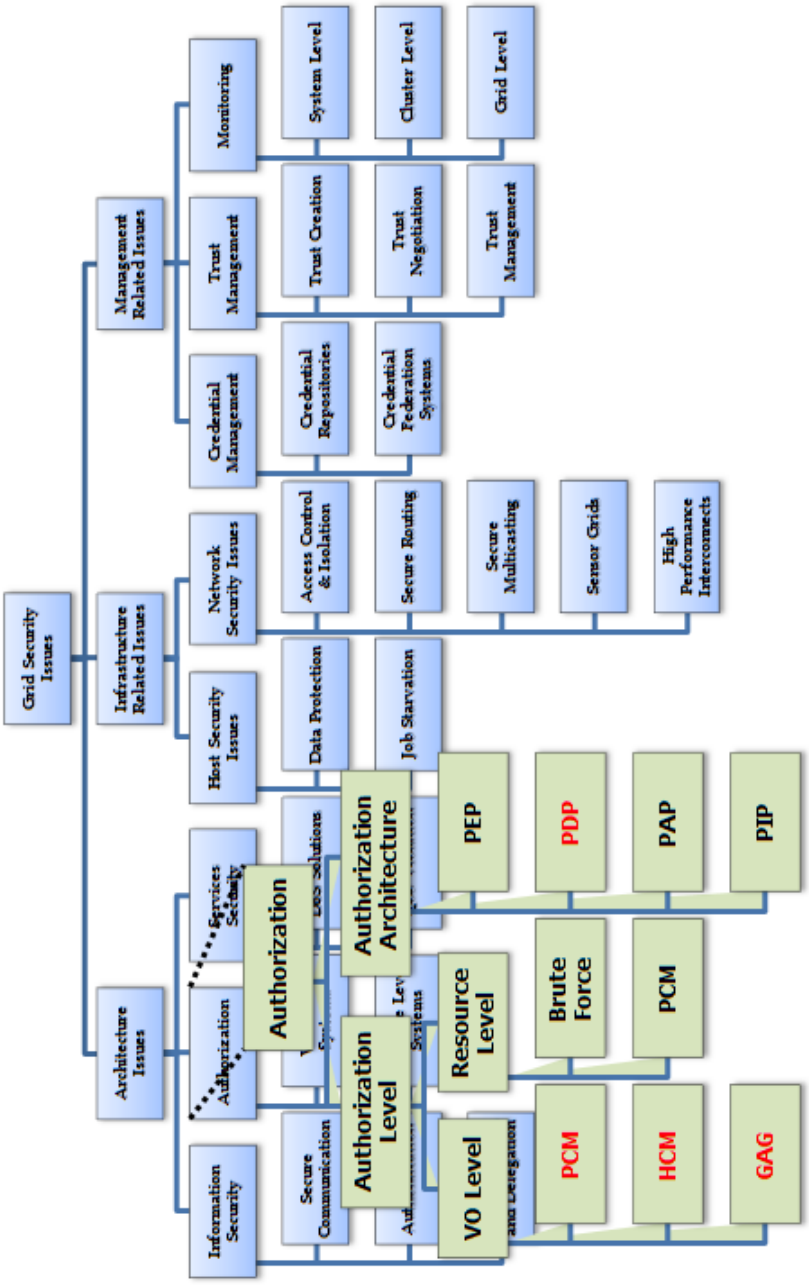

Figure 1.2: Extended Taxonomy of Grid Security Issues

This fact motivates the idea of the ability to *cluster the resources which have similar security policies in a hierarchical manner based on their shared security rules. So the authorization system can use a hierarchical decision tree to find the user's authorized resource group more quickly.* We have introduced this idea in the Hierarchical Clustering Mechanism (HCM)[8]. Proposed HCM increases the scalability of the authorization system a lot by reducing the redundancy in checking security rules compared to other mechanisms.

Current grid authorization systems seldom look at the way in which they store and organize the resources' security policies so to work more effectively. There is no well-defined data structure to store and manage the security policies intending to provide a quick response to the user.

One of the existing clustering mechanisms suggests dividing the security policies into general policies (GPs), which have to be checked on VO level, and local policies (LPs), which need to be checked on Resource level as appear in[9]. However, the proposed HCM can go more fine-grained clustering than that of only two levels.

One of the most accepted authorization system (Akenti) suggested clustering, when it allows the use-condition[10] to determine the rights, a user must have to access a **SET** of resources. Akenti uses a primitive kind of clustering (PCM) while the proposed HCM outperforms PCM in terms of redundancy in checking security rules.

In tune with the dynamic nature of grid environments, several tools need to be embedded with HCM to increase it scalability as illustrated in Chapter 4. However, even though HCM reduces the redundancy in checking security rules, it does not eliminate it entirely. This motivates us to introduce the Grid Authorization Graph (GAG) as our proposed solution to the Grid Authorization Problem.

1.4 Problem Definition

Design a structured, fine-grain, distributed and cross-domain access control mechanism which allows seamless access to grid resources with the least redundancy in checking security rules. This mechanism should fit easily into the present authorization systems with minimum overhead.

1.5 Major Contributions

The contributions of the thesis is shown in Figure 1.3 (appear with red color) and briefly explained in the following paragraphs:

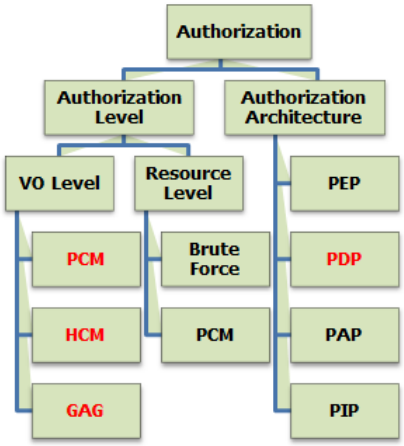

Figure 1.3: Authorization Taxonomy of Grid Security Issues

Grid authorization enhancement has been achieved by introducing the Hierarchical Clustering Mechanism (HCM). HCM is a kind of RBAC [11] specifically designed to suit in legacy systems. It efficiently reduces the redundancy in checking security rules compared to the existing mechanisms such as the Brute Force Approach and the Primitive Clustering Mechanism (PCM).

HCM is slightly expensive processes in terms of memory consumption and computations required to build the decision tree. That makes it unsuitable for small systems where it may cause an extra overhead which can lead to an authorization bottleneck. For that the Rough Set based PCM (RSPCM) is proposed. RSPCM is an alternative to HCM for those environments that are less dynamic and have less resources, and it is a superior to normal PCM in terms of redundancy.

Moreover, Grid is a heterogeneous environment. Thus the access control should work across domains. For that we have proposed the Distributed HCM (DHCM) with three different models each one suits in a particular scenario.

As grid has large number of users and they join and leave the grid dynamically, user scalability issues have to be addressed. Some caching mechanisms like the Temporal Caching Mechanism (TCM) and the Hamming Distance Caching Mechanism (HDCM) have been introduced to speed up the authorization process so that the system can scale up for more number of users. The concurrent HCM has been introduced to parallelize the authorization process so it can serve more than one user at a time.

All the above mentioned limitations and solutions make it convincing to consider HCM as an efficient security rules storage mechanisms to be embedded in the present popular grid authorizing systems like VOMS, Akenti, PERMIS, etc. However, HCM is not totally free of redundancy; moreover it is not easy to describe the OR-based security rules in HCM. For that the Grid Authorization Graph (GAG) has been introduced as our proposed solution to the grid authorization problem.

These contributions lead to the following papers:

1. ***"Enhancing the Hierarchical Clustering Mechanism of Storing Resources' Security Policies in a Grid Authorization System"*** in the proceeding of the 6th International Conference on Distributed Computing and Internet Technology (ICDCIT), LNCS 5966, pp(s):134-139.

2. ***"New Efficient Tree-Building Algorithms for Creating HCM Decision Tree in a Grid Authorization System"*** in the proceeding of the 2nd international conference on Network Applications Protocols and Services (NETAPPS), pp. 1-6, Malaysia, 22-23 Sept. 2010.

3. ***"A Rough Set based PCM for authorizing grid resources"*** in the proceeding of the IEEE 10th International Conference on Intelligent Systems Design and Applications (ISDA), Cairo, 29th Nov 2010, pp(s):391 - 396.

4. ***"Aspects in Grid Authorization"*** in the Proceeding of the International Conference on Nanoscience, Engineering & Advanced Computing (ICNEAC-2011).

5. ***"Concurrent HCM for Authorizing Grid Resources"*** in the proceeding of the 8th International Conference on Distributed Computing and Internet Technology (ICDCIT2012), LNCS, Springer.

6. A journal paper entitled ***"Grid Authorization Graph"*** is communicated to FGCS.

7. A chapter entitled ***"Aspects in Grid Authorization"*** is invited by iConcept Press as a part of the book entitled ***"Emerging Trends in Cloud Computing and Distributed Systems"***.

1.6 Organization of the Thesis

The thesis is organized into seven chapters as follows:

Chapter 1. gives an introduction to the grid authorization problem, our motivation to take up this research problem, briefly introduces the concept of Grid Computing. It also provides a briefed overall landscape on Grid Security Issues.

Chapter 2. provide an overview of authorization in general and grid authorization in particular. After that it introduces a brief literature survey about the most popular grid authorization systems. Then it surveys different access control models like the Discretionary Access Control (DAC), the Mandatory Access Control (MAC) and the Role-Based Access Control. It also explores the benefits and limitations of each access control model.

Chapter 3. introduces the mathematical foundation of the existing mechanisms used in current authorization systems. We proposes our HCM approach. Different HCM Tree-Building Algorithms are also introduced, a comparative analysis is made in a simulated environment.

Chapter 4. introduces various aspects of HCM scalability issues like performance of HCM in small or large grid, HCM stability against dynamic changes in the grid. The Distributed HCM has also been introduced as a solution to the Cross-Domain Authorization problem exists in Grid. User scalability issues have been addressed and caching mechanisms like the Temporal Caching Mechanism and Hamming Distance Caching Mechanism are proposed to enhance the user scalability. The concurrent model of HCM has been introduced to enable HCM to scale up to multiple number of authorization requests taking place at a time. It also shows that a single authorization process can be parallelized to decrease the response time.

Chapter 5. a Grid Authorization Graph is proposed in this chapter with several tools embedded to overcome all the unresolved limitations of HCM. As HCM reduces the redundancy in checking security rules compared to the Brute Force Approach as well as the Primitive Clustering Mechanism (PCM), it is not totally free of redundancy. Moreover, HCM cannot easily describe the OR-based security policies. GAG is introduced to over come HCM limitations and entirely eliminate the redundancy in checking security rule. Comparative studies are made in a simulated environment.

Chapter 6. shows how GAG can be embedded in GT4 authorization framework illustrating various components to be added or modified. It also introduces our Grid

Authorization Simulator (GAS) which is a C# based application used to simulate the authorization process of current used mechanisms like the Brute Force Approach and the PCM along with our proposed HCM as well as GAG. A quick user manual of GAS is given along with the technical specification which describes how the functional requirements of GAG will be translated into application components.

Chapter 7. concludes the thesis by summing up the contribution made and discusses the limitation and future scope of research.

Chapter 2

Literature Survey On Grid Authorization

This chapter gives an overview of authorization in general and grid authorization in particular. Essentials terms and definitions are provided in Section 2.1. The Authorization Sequences are provided in Section 2.2. Grid Authorization System Characteristics are given in Section 2.3. In Section 2.4, Authorization Architecture is discussed. A literature survey about the popular grid authorization systems are presented in Section 2.5. Access Control Mechanisms like the Discretionary Access Control (DAC), the Mandatory Access Control (MAC) and the Role-Based are explored in Section 2.6 along with the benefits and limitations of each model.

For the purpose of continuity of the content and consistency of the keywords used in Grid Authorization research, the content of these sections are adopted from the book titled *Grid Computing Security*[1], the GGF memo titled *Conceptual Grid Authorization Framework and Classification*[12] and the paper titled *Methods for Access Control: Advances and Limitations*.

2.1 Terms & Definition

- **Authorization:**

  Authorization is the function of specifying access rights to resources. It may be represented by:

  - A set of attributes (Roles) that describes a user characteristics or privileges.
  - A set of policies as the basis for a decision to grant access to a resource (access control).
  - A digitally signed documents that assert access rights to a resource.

- **Authorization Decision:**

  An authorization decision can be made at a number of places:

  - At the entrance of a service point (authorization may mean access control in this case).
  - At a (central) point outside the service point.

  Authorization decisions are made based on authorization information provided by authorities. These authorities must have a direct or a delegated relationship with the authorization subject (user), and the resource.

- **Authorization Server:**

  The term Authorization Server is however considered a rather vague term. Typically it may do:

  - An Authorization Decision (Sometimes the term Authorization Decision Server is used in this case). Authorization Decisions are typically the outcome of an evaluation of a policy.
  - An Authorization Lookup. A lookup of some entitys rights that are represented in some form. These rights may form the basis of a new Authorization Decision to be taken.
  - The delegation or proxy of an authorization decision to another Authorization Server.

- **Authorization Steps:**

  Authorization is frequently split into three distinct processes:

  - Defining an authorization policy at a high-level by a person or organization.
  - Implementing the high level policy into a digital representation that can be interpreted by computers.
  - Evaluating the digital representation of the policy by a process which subsequently decides to issue a specific authorization to a subject or take a specific action.

2.2 Authorization Sequences

2.2.1 Authorization Push Sequence

In these types of authorization systems, the authorization *credentials* are "pushed" to the *access controller*. Figure 2.1 & Figure 2.2 shows examples of push based authorization. In such a mechanism, there exists a *certificate generator* which checks the users credentials and generates a certificate so that the user can access the resource. The access controller allows access to the resource based on the certificate validity.

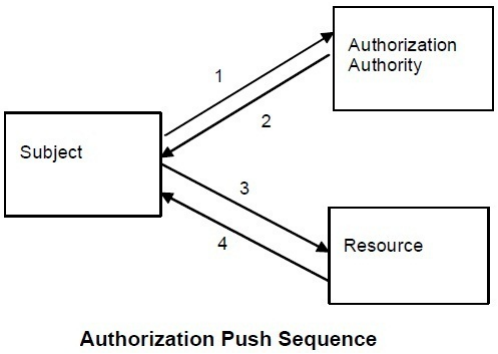

Figure 2.1: Authorization Push Sequence

- The Subject first requests an authorization from an Authority.
- The Authority may or may not honor the Subjects request. It then may issue and return secured message (token or certificate) that acts as a proof of right (***Authorization Assertion***).
- The assertion may subsequently be used by the Subject to request a specific service by contacting the Resource.
- The Resource will accept or reject the authorization assertion and will report this back to the requesting Subject.

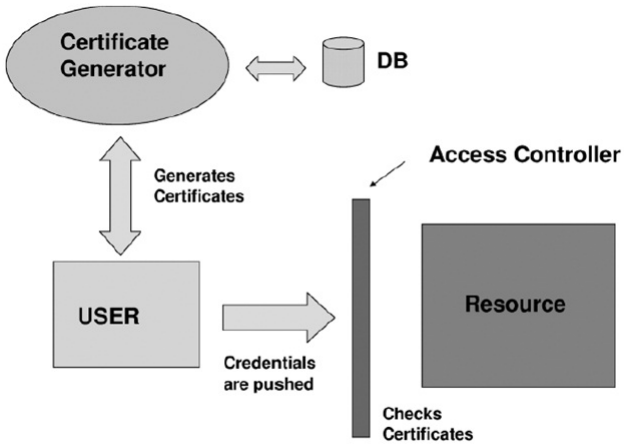

Figure 2.2: Authorization Push Sequence

The Push model is scalable but not as user friendly as the Pull model. It can be under DoS attack. Revocation of granted authorization is not easy. Examples of such sequences are found in many ticketing systems such as Kerberos[13] or Keynote[14]. In Grid, VOMS[15, 16] and CAS[17, 18] uses this model of authorization sequence.

2.2.2 Authorization Pull Sequence

In the pull model, the users provide the minimum credentials to the access controller and it is the responsibility of the controller to check the validity of the user based on the policies of the system.

The file systems in different operating systems employ pull mechanisms, where the user provides their minimum credentials, and the access policies corresponding to the user are "pulled" by the operating system. Based on the policies, the operating system either allows or disallows the user to access the resources.

Figure 2.3 & Figure 2.4 illustrates the pull model, where the user supplies the minimum credentials like the username, password, and the controller makes the access decision based on the user policies pulled from the database.

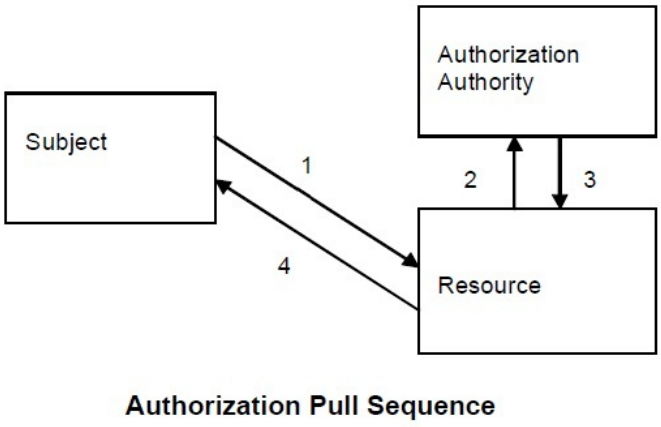

Figure 2.3: Authorization Pull Sequence

- The Subject contacts the Resource directly with a request.
- In order to admit or deny the service request, the Resource must contact its Authorization Authority.
- The Authorization Authority will perform an authorization decision and return a message containing the result of an authorization.
- The Resource will subsequently grant or deny the service to the Subject by returning a result message.

The Pull model is more user friendly. It is not under DoS threat. Revocation of granted authorization is easier. It is not as scalable as the Push model. Examples of such systems are found in the network world with systems using the RSVP [19] or RADIUS [20] protocol where requests typically are carried in-band.

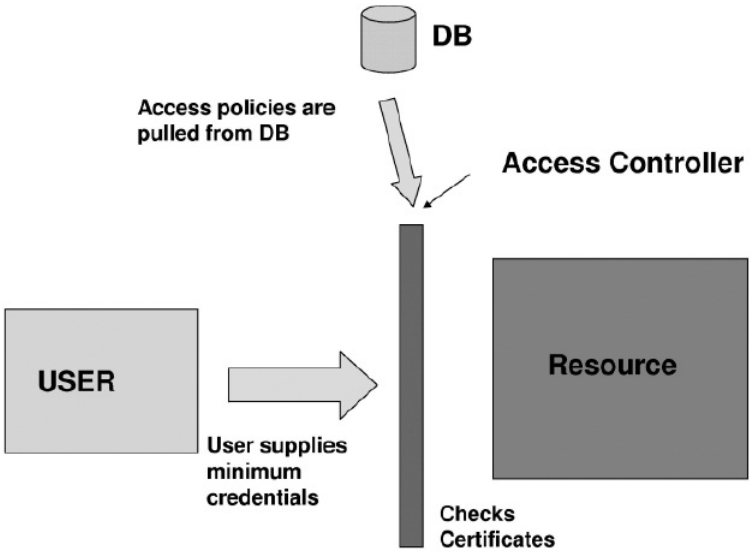

Figure 2.4: Authorization Pull Sequence

In the Grid environment this sequence is implemented in the PERMIS[21] and Akenti [2, 3] authorization systems. VOMS also can support this type of authorization sequence.

2.2.3 Authorization Agent Sequence

Figure 2.5 shows an example of the authorization pull sequence.

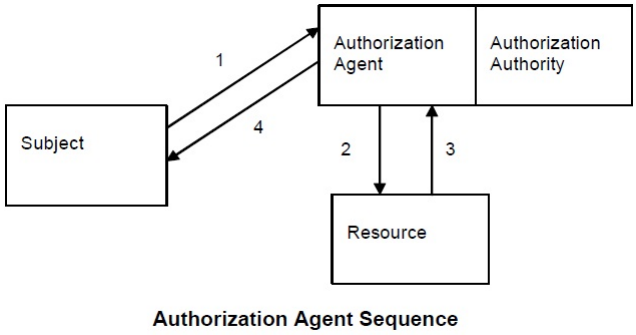

Figure 2.5: Authorization Agent Sequence

- The Subject contacts the agent with a request to obtain a service authorization.
- This agent makes an authorization decision based on the rules established by the authorization authority.
- If successful, it will contact the Resource to provide a certain state as to enable the service.
- After receiving successful feedback from the service, the agent will report success to the requesting Subject.
- The Subject then interacts directly with the Resource to access the service.

This model is relevant to Grid users when requesting a certain QoS from the Grid system (e.g., resource reservation through a scheduler).

2.3 Characteristics of Grid Authorization Systems

This section discusses the authorization requirements of a grid computing system.

- **Scalability Issues**

  Scalability is one of the most important characteristics of a grid authorization system. The grid authorization system should perform well when the number of users increases. In a grid system, users may join or leave the grid system quite frequently. Furthermore, resources may be added to or removed from the grid. These dynamic events have a significant effect on the design of the grid authorization system. There are two different types of scalability:

- **Performance Scalability:**

  Number of users is the primary measure for this type of scalability. The authorization system should not be a bottleneck in the grid infrastructure and should scale even if the number of users increase significantly. This aspect has a profound influence on choosing either the push based or pull based model.

- **Administrative Scalability:**

  This means that the system must be easily administrated even if it is highly dynamic. The centralized authorization mechanisms are generally used to tackle administrative scalability issues.

- **Security Issues**

  There are two types of compromises possible in a grid authorization system: user level and system level.

  - **User Level:**

    Here, an adversary poses as a user to the grid system. Then he can generate denial-of-service attacks. Gaining access into the system can be by tampering into the user credentials. So the authorization systems should provide *credentials* to the user through secure mechanisms like the **SSL** [22]. But still the *replication attack* (where the adversary sends an old credential) is possible here. To prevent it, a *time stamp* should be added to the communications.

- **System Level:**

  In case of centralized authorization systems, it is sometimes easy to compromise the authorization system rather than individual users. The adversary can employ two types of techniques to compromise an authorization server. Firstly, he can try to gain access to the server faking an administrative account. After getting access, the adversary can do all the things with the authorization system that an administrator is authorized to do. Secondly, the adversary can employ Denial-of-Service (DoS) attacks on the authorization system. So if authorization is mandatory to access the grid system, a DoS attack on a centralized grid authorization system will lead to a DoS attack on the grid system as a whole.

- **Revocation Issues**

Another important issue that needs to be considered before designing a grid authorization system is the issue of revocation of authorization. Consider the following scenario: A user logs into the grid system and is authorized to access the resources of the system. After some time, it is learned that the user has been compromised. In this case, the user should be denied access to the resources. Two mechanisms are generally designed to handle this problem.

- **Active Mechanism:**

  Here there is a communication between the user and the receiver access control mechanism based on which the user is denied further access to the resource. This type of mechanism can operate very quickly and the

revocation can happen as soon as the compromise is detected. Generally this is done through the use of *Certificate Revocation Lists* (CRL) issued by the authority, and the access controller needs to check whether a CRL exists for the credentials send by the user or not. There are two types of overheads associated with such systems. However, there is an overhead in using this mechanism because each time the *access controller* needs to see whether there is a CRL associated with each user credential. This may lead to a loss of scalability, especially if there are a huge number of users in the grid system.

– **Passive Mechanism:**

It is done through an expiration times provided in most certificates. During the generation of certificates, an expiration time is provided after which the certificate is considered to be invalid. In terms of scalability, these types of passive revocation mechanisms are better than the active ones, however, it comes at scalability cost. Assume a certificate is generated at time T and the expiration time is (T+t). Now the user is compromised just after time T. Then for a period of t, the adversary is capable of compromising the system. If the time t is small, then the system is more secure. However, smaller t also indicates that there are more number of authorizations required, and this means reducing the scalability of the system. Therefore, there is a trade-off between the scalability and security, which is tuned by the choice of the time t. Figure 2.6 illustrate the issue.

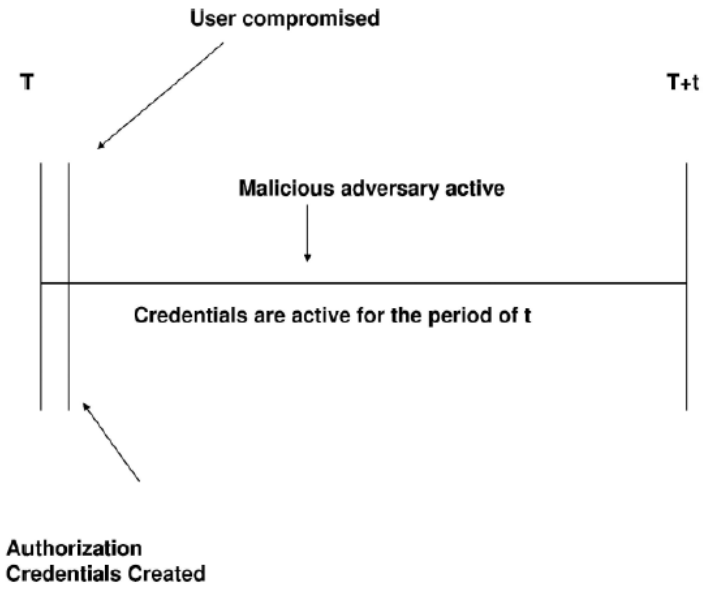

Figure 2.6: Expiration Time Problem

- **Inter-operability Issues**

  A grid system is characterized by heterogeneity not only at the resource level but also at the policy level. Therefore, the authorization systems should be inter-operable across multiple systems having heterogeneous policies. There may be issues of multiple authorization systems, as well as multiple communications protocols, and algorithms. One of the ways that most systems try to handle the inter-operability issues is through standardization.

- **Domain Considerations**

  - An administrative domain is a definition of the scope of authority.
  - In many distributed authorization scenarios there are at least two administrative domains: that of the Subject and that of the Resource.

- In Grid environments we frequently see scenarios where there are separate domains for identity, subject attributes, resource policy, and community policy authorities.
- In a simple Grid use case the Subject is in one administrative domain, its home domain, and the Resource is in another (the home domain of the resource).
- In more advanced scenarios a community or virtual organization (VO) domain is present.
- A VO domain can provide Authorities that perform privilege management for all the members of a VO.
- A typical Grid scenario is one where a Subject needs to use services from several domains. Sometimes this is accomplished by a Resource in one domain using a Resource in another domain on behalf of the Subject.
- Grid Service Providers may provide resources to users in multiple VO or home domains.

Figure 2.7 shows an example of the coordinated use of resources across different domains.

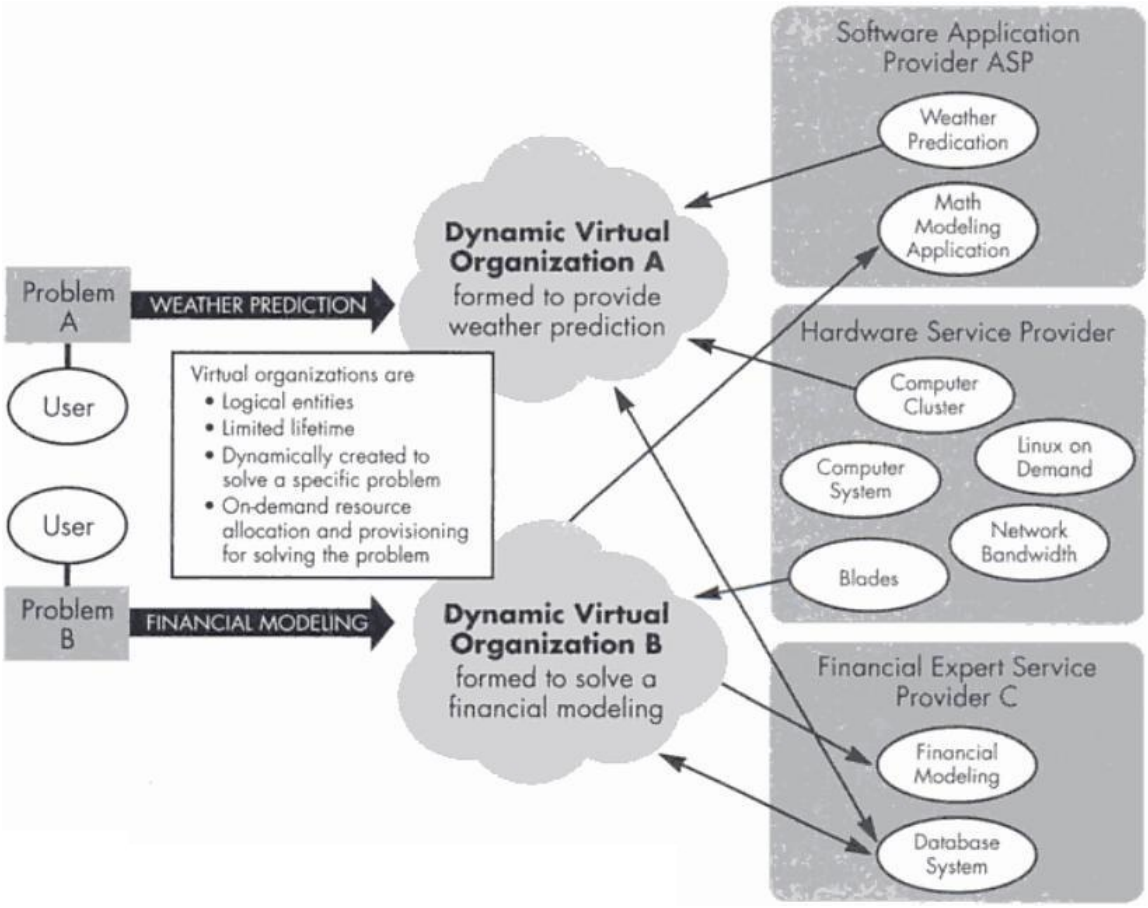

Figure 2.7: Coordinated use of resources across different domains

2.4 Authorization Architecture

The authorization architecture consists of a set of entities and functional components that allow authorization decisions to be made and enforced based on attributes, parameters and policies that define authorization conditions (Figure 2.8).

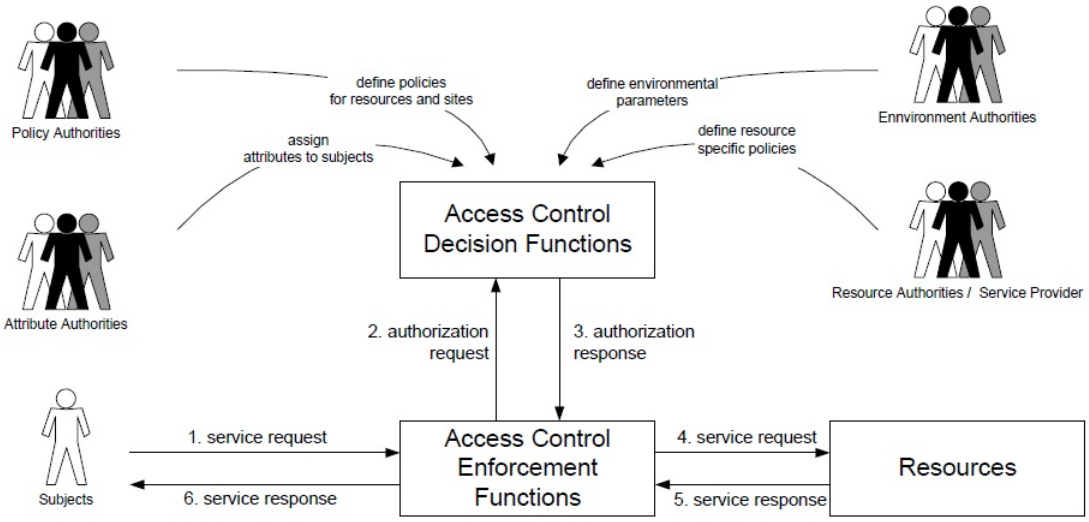

Figure 2.8: Authorization Architecture

2.4.1 Authorization Functions

There are two access control functions defined by ISO-101813:

- **Access Control Decision Function (ADF)**

  Makes authorization decisions about a subjects access to a service. It is equivalent to the Policy Decision Point (PDP) defined in RFC2904[23]. It is normally part of an Authorization Server and is independent of the Resource. Proposed methods in the thesis falls into this area.

- **Access Control Enforcement Function (AEF)**:

  Mediates access to a resource or service. It is equivalent to the Policy Enforcement Point (PEP) defined in RFC2904[23]. It is located in front of the Resource it protects.

The ADF, AEF, Subject and Resources may be embedded inside one or more administrative domains.

2.5 Grid Authorization Systems

As discussed earlier, grid authorization systems can be mainly divided into two categories: VO level systems and Resource level systems as shown in Figure 2.9. In this section, we will discuss popular grid authorization systems of both types.

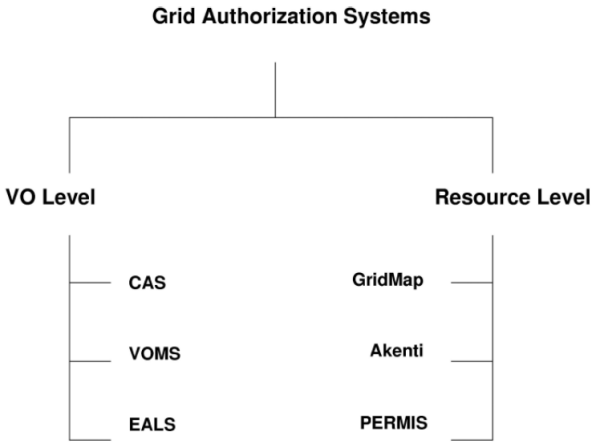

Figure 2.9: Grid Authorization Systems

2.5.1 VO Level Authorization Systems

VO level grid authorization systems are centralized. Whenever a user wants to access certain resources owned by a resource provider RP, he/she has to obtain a credential from the authorization system which allows certain rights to him/her. Then the user has to present the credentials to the RP to gain access to the resource. In this type of system, the resources has the final right to allow or deny the access to the user.

2.5.1.1 Community Authorization Service (CAS)

CAS [17, 18] has been developed as a part of the Globus toolkit. It works in a distributed virtual communities which consists of many communities, each participating as a resource provider and/or a resource consumer. The Authorization is handled using a trusted third party called the Community Authorization Service (CAS) server which is responsible for managing the policies and controlling access to the resources.

CAS can be viewed as a service which has been given the authority by the community to authorize users on its behalf. The CAS server provides capabilities to users based on the policies maintained at the CAS policy database. The users use the capabilities provided to them by CAS and show them to the resource server to grant them access to the resources based on their capabilities. It is important to mention that the resources may have their own local access control policy. So, assuming that the authority granted by the resource provider to the community is C, the capability provided to the user by the CAS server is U, and the resource allows R restrictions. Then the effective capability of the user on the resource is $C \bigcap U \bigcap R$. Figure 2.10 illustrates the process.

Thus, there are two steps to authorize a user in CAS:

1. **(Resource - Community) Interaction:**

   Here the resource provider provides certain capabilities to the community. For example, the resource provider allows "read" actions to 30% of its resources to the community. These policies are then put into the CAS policy database.

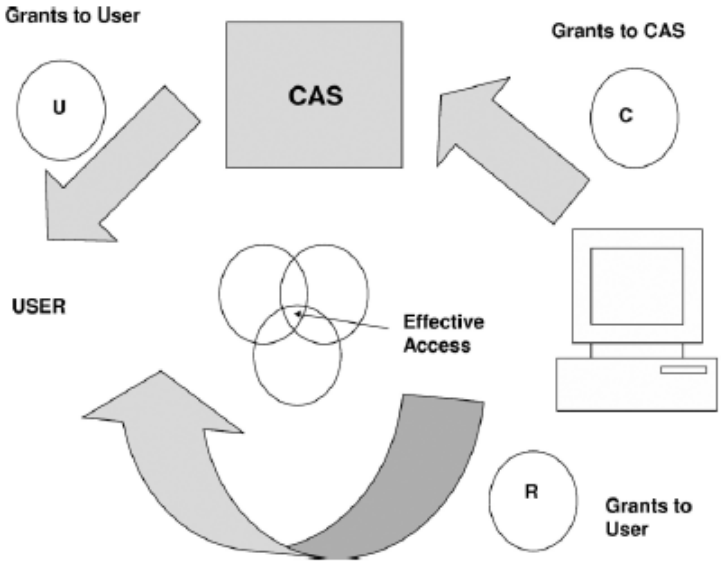

Figure 2.10: The Effective Capability

2. **(Community - User) Interaction:**

   Here, the user is granted certain capabilities by the community. Of course, the community cannot grant more capabilities to the user than what is provided to it by the resource providers.

Figure 2.11 gives an overview of a typical authorization of CAS. The user sends a request to the CAS server for a capability to access the resource, and presents a set of credentials to the CAS server to identify himself/herself. The CAS server delegates an appropriate capability to the user based on the policies stored in the policy database. The user then requests the resource with the capabilities given to it by the CAS. The resource server then allows or disallows the user to access the resource based on the local policies of the resource for the user. Therefore, in a CAS authorization system, the resource is the ultimate authority to decide which user can access its resources, and CAS is a facilitator in the process.

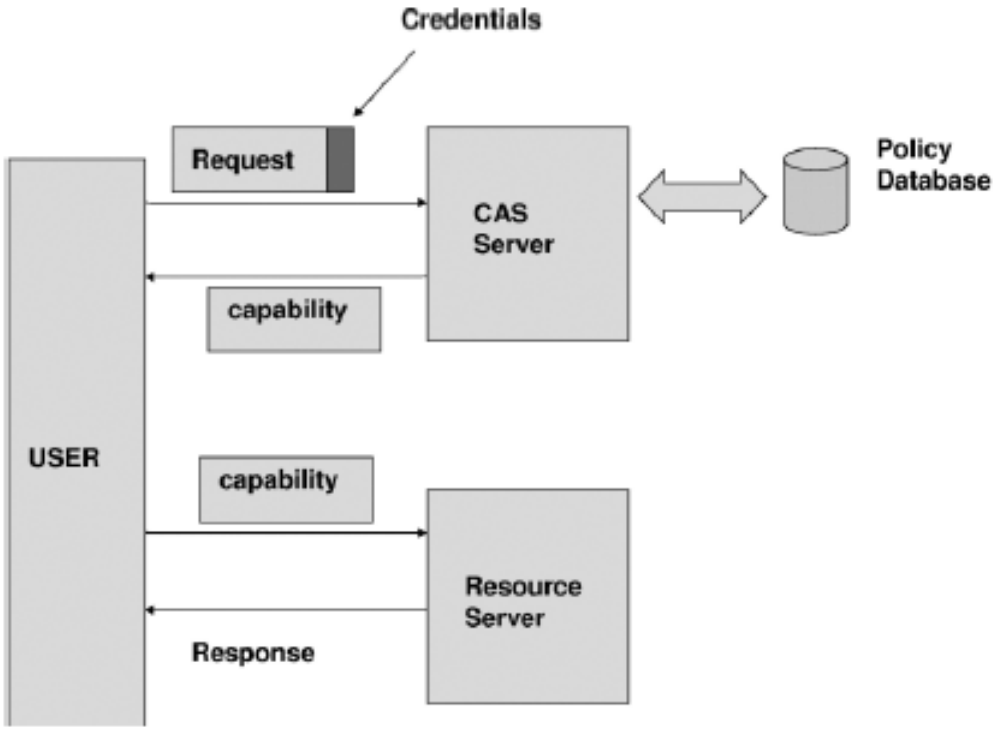

Figure 2.11: CAS Overview

CAS Drawbacks

CAS is a *push based* authorization system. Therefore, it has all the drawbacks of the push based authorization systems. The other drawbacks of CAS is lack of standardization. Current version of CAS does not support the standards like SAML and XACML which is supported by some other authorization systems like VOMS. This makes CAS less acceptable across enterprises, and raises an Inter-operability sort of problems.

2.5.1.2 Virtual Organization Membership Service (VOMS)

VOMS [16] is similar in many respects. VOMS has a policy database where the authorization policies are stored like CAS. However, in VOMS system, a user may be a member of as many VOs as required, and each VO can be a complex structure with groups and subgroups in order to clearly divide its users according to their tasks. A user, both at VO and group level, may be characterized by any number of roles and capabilities. He/She gets his certificate from the Certificate Authority, which is submitted to the VOMS systems. A user may get credentials from multiple VOMS systems, and present those credentials to the resources to gain access to the resources. The process is illustrated in figure 2.12.

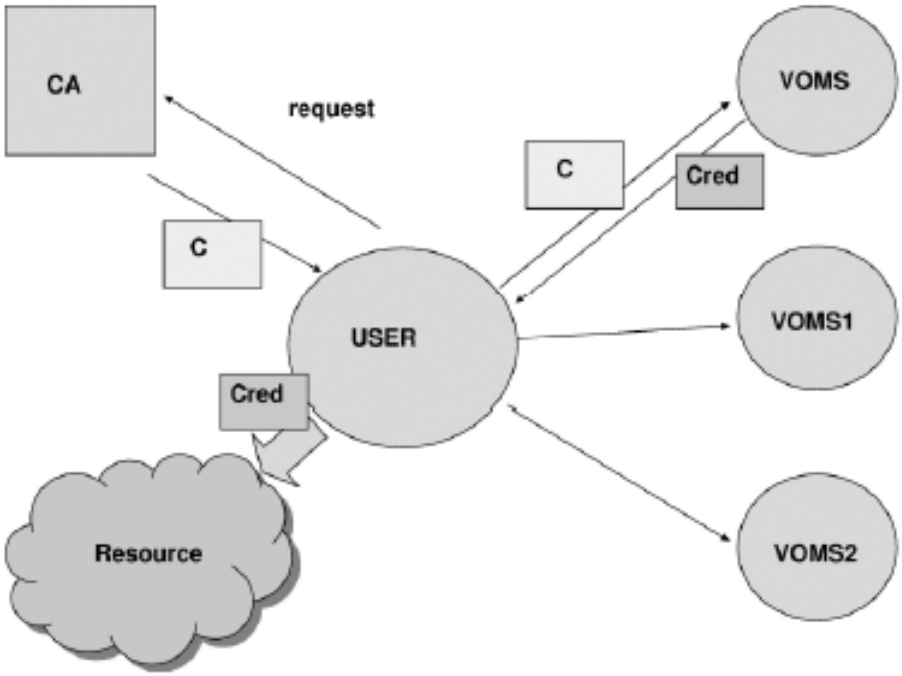

Figure 2.12: VOMS Overview

Even that VOMS is very similar to CAS, it has some unique features which make it different from CAS like:

- **Attribute Certificates:**

  VOMS uses a short lived credential which includes user information, server information, and the validity period. VOMS include the Attribute Certificate (AC), it puts the AC in the noncritical extension of the certificate that the user generates to make it compatible with "non-VOMS" system. The use of ACs makes VOMS quite compatible with other authorization systems like Akenti.

- **Multiple Roles:**

  VOMS allows a role based mechanism, where a user can take up multiple roles in multiple VOs.

2.5.2 Resource Level Authorization Systems

The resource level authorization systems implement the decision to authorize the access to a set of resource at the resource providers itself rather than a centralized authorization server. In this section we discuss popular resource level authorization systems like Akenti, PERMIS, and the GridMap.

2.5.2.1 Akenti

Akenti [2, 3] was initially designed to control Web resources, where there are multiple stakeholders for the resources and there are lots of users trying to access the resources. With the advent of grid and collaborative computing, Akenti has been seen as an important alternative to different centralized VO based systems like CAS,

VOMS, etc. Akenti model consists of resources that are being accessed via a resource gateway by a set of users who are the part of the virtual organization. It also assumes that each resource may have multiple stakeholders having a set of access constraints on the set of resources. The Akenti system allows the users to access the resource(s) based on the identity of the users and the access policy set on the resources by the resource stakeholders. The stakeholders express their access constraints through a set of self-signed certificates which are known to be stored in a secure remote server. The certificates express the attributes a user must have to access the resources. At the time of resource access, the resource gatekeeper asks the Akenti server what access the user has to the resource. The Akenti server finds all the relevant certificates, verifies the certificates, and the returns the decision to grant the access to the user. Figure 2.13 illustrates the above process.

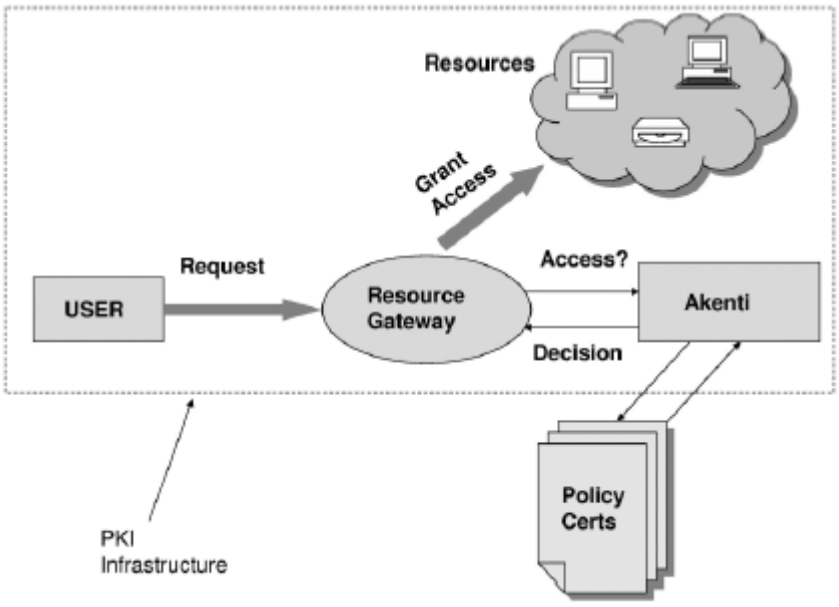

Figure 2.13: Akenti Overview

All certificates must follow the X.509 specification, and SSL/TLS channels are used to authenticate a user who wants to access the resource. The access to a certain resource is determined by the combined policy on the resource by the different stakeholders. Akenti policies are expressed in XML and stored in three types of signed certificates: policy certificates, use condition certificates, and attribute certificates.

- **Policy Certificates:**

  They are located with the resources, and contain information regarding the stakeholders and the location of the use-condition certificates of the stakeholders for the resource. They may also contain a list of URLs as a locations to search for the attribute certificates.

- **Use Condition Certificates:**

  They consist of a set of constraints that determine the rights a user must have to access a set of resources. By introducing this type of certificate, Akenti uses kind of Primitive Clustering Mechanism (PCM) which is discussed in the next chapter.

- **Attribute Certificates:**

  They contain an attribute-value pair and the principle to which the attribute applies. They generally apply to a single resource, or a group of resources.

It is important to observe that while the Akenti policy engine verifies the different types of certificates, it caches all the certificates so that the search latency is reduced for the next access request process. The lifetime of the cached certificates is set in the policy certificate for the resource.

Akenti's Advantages and Drawbacks:

Akenti does provide flexibility as the use condition certificates can be changed over time. Let us look at the different characteristics and applicability of the Akenti system.

- **Applicability:**

  Akenti can be applied in cases where there is an enterprise level grid having multiple enterprises or stakeholders managing the grid. However, in its current form, Akenti cannot be directly used in an enterprise scenario because of lack of support for standards like SAML and XACML.

- **Scalability:**

  Since Akenti uses a pull model, there is a loss of scalability when the number of users is large. However, the use of caching mechanisms can help to increase the scalability. Further, from the administrators' point of view, Akenti is more scalable and easy to be administered compared to the other systems by using the different types of certificate

- **Security and Revocation:**

  Akenti does not introduce any extra security vulnerability compared to the other systems proposed in the literature. Since Akenti applies a pull based mechanism, revocation is faster compared to CAS and VOMS as changes made to the use-conditions are effected immediately.

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

Similar to Akenti, PERMIS [21] is by default a resource level authorization system, but it is more suitable for enterprises as it supports standards. It has a Privilege Management Infrastructure (PMI) shown in Figure 2.14. A user asks for a certain application or resource and makes a request to the resource gateway or the application gateway. PERMIS uses a Role Based Access Control (RBAC) model, where each user is mapped to a role then policies are assigned to the roles.

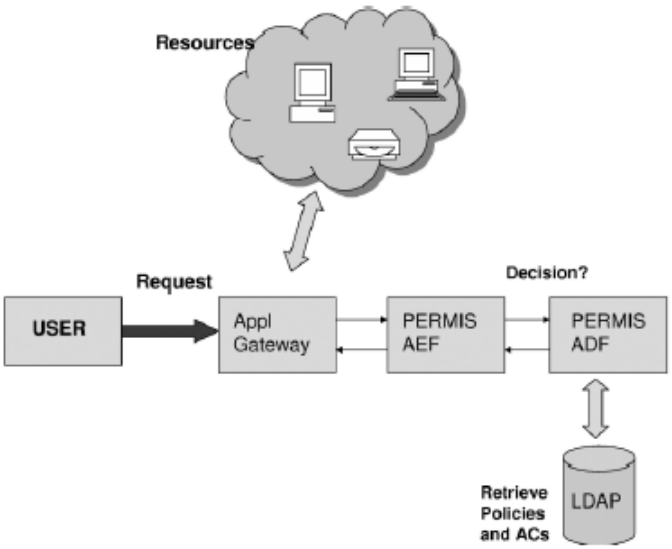

Figure 2.14: PERMIS Overview

Here, we briefly discuss the different components of the PERMIS PMI system:

- **Authorization Policy:**

  The authorization policy specifies *who* has access to *which* resources under *what* conditions. PERMIS uses XML for policy specification (SAML).

- **Privilege Allocator (PA):**

  It is used to allocate privileges to users. Since PERMIS uses RBAC [11], PA allocates roles to users in the form of Attribute Certificates (AC). The role assignment AC is signed by the Attribute Authority (AA). The ACs are then stored in an LDAP server and can be accessed publicly.

- **PERMIS AEF and ADF:**

  AEF authenticates a user and requests the ADF to make an authorization decision. The ADF accesses the LDAP servers, gets the policies and the attribute certificates, and then makes an authorization decisions based on the obtained information.

It is important to mention that PERMIS can be configured to use either a pull or a push model.

2.5.2.3 GridMap

GridMap [24] is the most widely used authorization system (mainly in Globus) due to its simplicity. In a GridMap system, the static policies of which user can access the resource and how it is accessed, is placed in each local resource. The decision to grant access to a resource is based on the information present in the GridMap file.

GridMap's Advantages ans Drawbacks:

- **Scalability:**

  The main reason for the development of CAS system is the lack of scalability of the GripMap authorization model. It scales very poorly specially in terms of administrative scalability. In a dynamic grid environment, where Globus is mostly deployed, Grid-Map file needs to be changed continuously, which affects the performance of the system tremendously.

- **Security:**

  Unlike CAS and VOMS systems, the denial-of-service is completely limited in GridMap. Revocation also can be done fast, as only one file needs to be changed to effect the changes. However, the GripMap file is stored without encryption in the host system.

2.5.3 Comparing the Different Authorization Systems

In this subsection, a comparison table between the different types of authorization systems taking care of many aspects is given. However it is important to notice that the two types of authorization systems (Resource level, and VO level) complement each other, and can be implemented together to provide a comprehensive authorization solution. table is shown in Figure 2.15.

Params	VO Based			Resource Based		
	CAS	VOMS	EALS	Akenti	PERMIS	Grid-Map
Push/ Pull	Push	Push	Pull	Pull	Push or Pull	Pull
Users Supported	High	High	Medium	Medium	High	Medium
Administrative Overhead	Low	Low	Low	Low	Low	High
Authentication	Using GSI	Using GSI	Passwords/Certificates	Certificates	Certificates	Using GSI
Revocation	No	No	Fast	Fast	Can be fast	Have to be updated
Interoperability	Uses SAML	Can use SAML	Through SAML, XACML	May be complex in some cases	Through SAML	Minimal
Decision Making	Requires separately	Requires separately	Integrated in Scheduler and License manager	Single step; through Capability certs	Two steps; "yes/no answer"	Based on policies
Multiple stakeholders	No	Yes	Yes	Yes	No	No

Figure 2.15: Comparisons between different authorization systems [1]

2.6 Access Controls

The fundamental goal of any access control mechanism is to provide a verifiable system for guaranteeing the protection of information from unauthorized and inappropriate access as outlined in one or more security policies. In general, this translation from security policy to access control implementation depends not only on the nature of the policy but involves the inclusion of at least one of the following controls:

- Confidentiality - Control disclosure of information
- Integrity - Control modification of information

Bishop[25] identifies military security policies and commercial security policies as two distinct types of policies that underline the difficulty in developing an all-purpose security model. Military security policies are defined as primarily concerned with preserving information confidentiality while commercial security policies primarily focus on guaranteeing information integrity.

The difference between governmental and commercial needs led to the development of two distinct access control mechanisms, Mandatory Access Control (MAC) and Discretionary Access Control (DAC).

MAC focuses on controlling disclosure of information by assigning security levels to objects and subjects, limiting access across security levels, and the consolidation of all classification and access controls into the system. Conversely DAC focuses on fine-grained access control of objects through Access Control Matrices and object level permission modes. Limitations in each model can be summarized as failings of one or more of three characteristics of an ideal security model:

- *Inescapable* inability to break security policies by frauding access controls of the model
- *Invisible* easy user and administrative interaction with model
- *Feasible* cost-effective and practical to implement model

These flaws in MAC and DAC led to research in new ways of modeling and implementing access control. Among the recent developments, Role Based Access Control (RBAC) incorporates features from both MAC and DAC. Our proposed HCM is considered as a type of RBAC (enhancement of RBAC).

Access control models are generally concerned with whether subjects, any entity that can manipulate information (i.e. user, user process, system process), can access objects, entities through which information flows through the actions of a subject (i.e. directory, file, screen, keyboard, memory, storage, printer), and how this access can occur.

Access control models are usually seen as frameworks for implementing and ensuring the integrity of security policies that mandate how information can be accessed and shared on a system.

The most common, oldest, and most well-known access control models are Mandatory Access Control and Discretionary Access Control. However, limitations inherent to each has stimulated further research into alternatives including Role Based Access Control, Dynamic Typed Access Control.

2.6.1 Mandatory Access Control (MAC)

In Mandatory Access Control (MAC) models, the administrator manages access controls. The administrator defines a policy, which users cannot modify. This policy indicates which subject has access to which object. This access control model can increase the level of security, because it is based on a policy that does not allow

any operation not explicitly authorized by an administrator. The MAC model is developed for and implemented in systems in which confidentiality has the highest priority, such as in the military. Subjects receive a clearance label and objects receive a classification label, also referred to as security levels. Mandatory Access Control is usually associated with the 1976 Bell-LaPadula Model[26] of multi-level security.

- **BellLaPadula Confidentiality Model**

  Bell-LaPadula assigns security labels to subjects and objects and uses two security properties, *simple security* property and **-property* to ensure military classification policies that restrict information flow from more secure classification levels to less secure levels.

  - The *simple security property* states that no process may access information labeled with a higher classification.
  - The **-property* prevents processes from writing to a lower classification.

  These two properties are supplemented by the *tranquility property*, which is stated in two forms: strong and weak. Under the strong tranquility property, security labels can never change during system operation. Under the weak tranquility property, however, labels can change during operation but never in a way that violates defined security policies.

  The benefit of the weak tranquility property is that it allows least privilege by starting a user session in the lowest security session, regardless of the users

clearance level, and only upgrades the session when objects requiring higher clearance levels are accessed. Once upgraded, the session can never have its classification level reduced and all objects created or modified will have the clearance level held by the session when the object is created or modified, regardless of the objects initial level. This is known as the *high water mark* principle. For example, an administrator has access level 65535, Alice level 100, and Guest level 1. There are two files, file1.doc has a level of 2, file2.doc a level of 200. Alice can access only file1, Guests can access neither file1 nor file2, and the administrator can access both files.

- **Biba Integrity Model**

  While Bell-LaPadulas model describes methods for assuring confidentiality of information flows, Biba[27] developed a similar method aimed at information integrity. Integrity is maintained through adherence to reading writing principles that can be thought of as a reverse of the Bell-LaPadula principles. In the Biba model, integrity levels are low to high with objects labeled high having high integrity. A subject can read objects at a higher level but can only write to objects of lower levels. This is known as the *low water mark* principle and assigns created objects the lowest integrity level that contributed to the creation of the object. Because the MAC method is primarily developed for purposes where confidentiality is far more important than integrity, Bibas influence was minor on further development of MAC models.

- **Advantages of MAC**

  Through its implementation of Bell-LaPadula in Multi-Layer Secure (MLS) systems, MAC is the main access control model used by the military and intelligence agencies to maintain classification policy access restrictions. Additionally, MAC is relatively straightforward and is considered a good model for commercial systems that operate in hostile environments (web servers and financial institutions) where the risk of attack is very high, confidentiality is a primary access control concern, or the objects being protected are valuable.

- **Disadvantages of MAC**

  MAC also suffers serious limitations. The assignment and enforcement of security levels by the system under the MAC model places restrictions on user actions that prevents dynamic alteration of the underlying policies, and requires large parts of the operating system and associated utilities to be trusted and placed outside of the access control framework.

Trusted components are processes, such as declassifying cryptographic processes, that need to violate MAC principles and thus must sit outside of the MAC model. In order to maintain the security policies and prevent unauthorized or inappropriate access, the code behind these components is assumed (hopefully with verification) to be correct and conforming to the underlying security policies of the system. Additional access control methods must be used to restrict access to these trusted components. Unfortunately, in practice it has been shown that it is virtually impossible to implement MLS using MAC

without moving essentially the entire operating system and many associated utilities outside the MAC model and into the realm of trusted components.

Additionally, MAC can unnecessarily over-classify data through the *high-water mark* principle and hurt productivity by limiting the ability to transfer labeled information between systems and restricting user control over data. MAC also does not address fine-grained least privilege, dynamic separation of duty or security or validation of trusted components.

2.6.2 Discretionary Access Control (MAC)

MAC is not the most widely used method of access control. The widely used type of access control model is Discretionary Access Control (DAC) integrated into UNIX, FreeBSD, and Windows 2000. DAC was developed to implement Access Control Matrices defined by Lampson in his paper on system protection [28].

Access Control Matrices are usually represented as three dimensional matrices where rows are subjects, columns are objects and the mapping of subject and object pairs results in the set of rights the subject has over the object.

Unlike the MAC framework where decisions to allow or deny access are made by the system according to pre-determined policies, DAC allows subjects the discretion to decide access rights on objects they own. Because Access Control Matrices have one row for every subject and one column for every object, the number of entries is intuitively the number of subjects times the number of objects.

This means that O(n) growth in subjects and objects results in O(n^2) growth in the size of the matrix. The size of the access control matrix would not be a concern if the matrix was dense, however, most subjects have no access rights on most objects so, in practice, the matrix is very sparse. If access control information was maintained in this matrix form, large quantities of space would be wasted and lookups would be very expensive.

Thus, DAC access settings are typically stored as either per-object file permission modes (default on UNIX) or as lists.

- **Lists**

  Access lists can be stored in a number of configurations with each configuration offering benefits and drawbacks under varying circumstances.

  - **Access Control Lists(ACLs):**

    Access control lists are the representation of object rights as a table of subjects mapped to their individual rights over the object. ACLs are the default representation of access rights on UNIX systems and essentially correspond to individual columns in the system Access Control Matrix. ACLs are effective but not time-efficient with a low number of subjects.

    Typically, the operating system knows who the user of a process is but does not know what rights the user has over objects on the system. ACLs require the operating system to either perform a rights lookup on each object access or somehow maintain the subjects active access rights.

Because of this rights management issue, and the difficulty in performing multi-object rights modifications for individual users, ACLs do not scale well on systems with large numbers of subjects or objects.

- **Capabilities Lists:** Capabilities Lists are similar to ACLs but instead of tables of subjects and rights, capability lists represent subject rights as mappings of objects to rights.

- **Advantages of DAC**

  A primary benefit associated with the use of DAC is enabling fine-grained control over system objects. Through the use of fine-grained controls, DAC can easily be used to implement least-privilege access. Individual objects can have access control restrictions to limit individual subject access to the minimum rights needed. DAC is also intuitive in implementation and is mostly invisible to users so it is regarded as the most cost-effective for home and small-business users.

- **Disadvantages of DAC**

  DAC, however, is not without issues. Allowing users to control object access permissions has a side-effect. Additionally, maintenance of the system and verification of security principles is extremely difficult for DAC systems because users control access rights to owned objects. The so-called Safety Problem, the lack of constraints on copy privileges, is another issue of DAC. The lack of constraints on copying info from one file to another makes it difficult to maintain safety policies.

2.6.3 Role Based Access Control(RBAC)

RBAC [29] is considered a much more generalized model than either MAC or DAC, encompassing both models as special cases while providing a policy neutral framework that allows RBAC to be customized on a per-application basis.

In RBAC, permissions are associated with roles as shown in Figure 2.16, and users are made members of the appropriate roles thereby acquiring the roles permissions. Roles can be created for the various job functions in an organization and users are then assigned roles based on their responsibilities and qualifications. Users can be easily reassigned from one role to another. Permissions can be revoked from roles when needed.

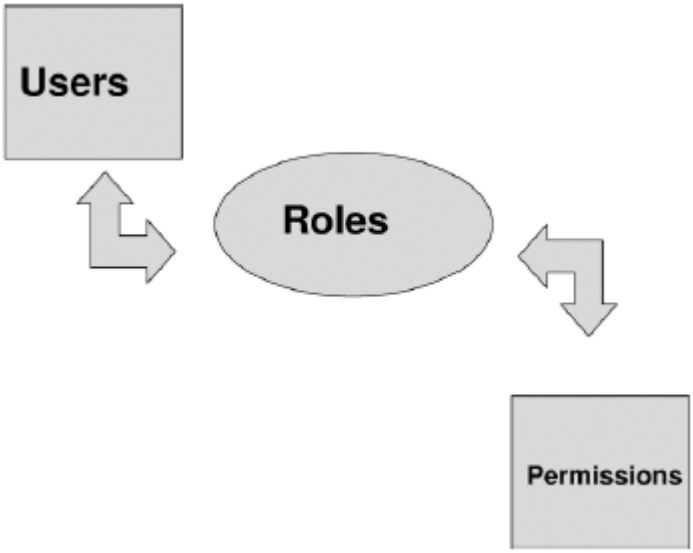

Figure 2.16: Role Based Access Control (RBAC)

Rule-Based Access Control model, which is sometimes referred to as Rule-Based Role-Based Access Control (RB-RBAC). It includes mechanisms to dynamically assign roles to subjects based on their attributes and a set of rules defined by a security

policy. For example, you are a subject on one network and you want access to objects in another network. The other network is on the other side of a router configured with access lists. The router can assign a certain role to you, based on your network address or protocol, which will determine whether you will be granted access or not.

- **Differences from MAC/DAC**

  First proposed by Ferraiolo and Kuhn in 1992 [11], RBAC addresses most of the failings of DAC while maintaining DACs focus on non-military systems. RBAC approaches commercial and civilian governmental security needs from an integrity first, confidentiality second position based on Clark and Wilsons research into commercial security policies.

  Security policies are maintained in RBAC through the granting of rights to roles rather than individuals. Right granting and policy enforcement is consolidated in the hands of a security administrator and users are prevented from transferring permissions assigned to a role they are allowed to perform to other users. This rule removes the ownership rights granted in DAC and thus behaves like a finer-grained version of the MAC model.

  RBAC also stands apart from the more traditional MAC and DAC by granted rights on transactions, not on underlying subjects. These rights are granted to roles, which at first glance appear to be a synonym for DAC groups. The difference lies in that groups consist of a collection of users while roles are a bridge between a collection of users and a collection of rights.

- **Advantages and Disadvantages of RBAC with Motivation to HCM**

  - **Advantages of RBAC**

    An important characteristic of RBAC is that by itself it is policy neutral. RBAC is a scalable and flexible mechanism for articulating access control policies. It is scalable, as the number of associations compared to a typical user to permissions mapping is less, and it is flexible because by tuning the different components of RBAC, it can be converted to different forms of DAC(Discretionary Access Control) and MAC(Mandatory Access Control) access mechanisms[30].

    Transaction based rights help ensure system integrity and availability by explicitly controlling not only which resources can be accessed but also how access can occur. In large organizations, the consolidation of access control for many users into a single role entry allows for much easier management of the overall system and much more effective verification of security policies. RBAC also supports principle of least-privilege, separation of duties, and central administration of role memberships and access controls.

    Separation of duties and least-privilege are not a part of MAC while central administration is loosely supported in MAC with trusted components and impossible in DAC due to the violation of the safety principle.

– **Disadvantages of RBAC**

While RBAC marks a great advance in access control, the administrative issues of large systems still exist. In large systems, memberships, role inheritance, and the need for finer-grained customized privileges make administration potentially impractical.

In the proposed research work, the Hierarchical Clustering Mechanism (HCM) is introduced to solve the problem of integrating RBAC in large systems as well as in multi-domain systems. HCM inherits the advantages of RBAC and solves its limitations in large systems. It reduces the authorization overhead to a great extend and serves huge number of user requests in a more effective way compared to RBAC. Simply, HCM is a new generation of RBAC which best suits in large and heterogeneous systems such as Grid.

Chapter 3

The Hierarchical Clustering Mechanism (HCM)

3.1 Introduction

Grid computing is concerned with shared and coordinated use of heterogeneous resources, belongs to distributed virtual organizations to deliver nontrivial quality of services [7]. The heterogeneity, massiveness and dynamism of grid environments complicate and delay the authorization process. This brings out the need for a fast and scalable fine-grained access control (FGAC) mechanism.

Every resource in a grid has its own security policy, which may be quite similar to other security policies of some other resources. This fact motivates the idea of the ability to cluster the resources which have similar security policies in a hierarchical manner based on their shared security rules. So the authorization system can use a hierarchical decision tree to find the user's authorized resource group more quickly. We have introduced this idea in the Hierarchical Clustering Mechanism (HCM)[8].

Current grid authorization systems seldom look at the way in which they store and organize the resources' security policies so to work more effectively. There is no well-defined data structure to store and manage the security policies intending to provide a quick response to the user. One of the existing clustering mechanisms suggests dividing the security policies into general policies (GPs), which have to be checked on VO level, and local policies (LPs), which need to be checked on Resource level[9], whereas it is shown that the proposed HCM can go more fine-grained clustering than two levels.

Currently, the main concentration in the literature is on the language used to write the resource's security policy, either it should be a standard specification language like SAML/XACML used in VOMS[16] to provide the interoperability property[31], or it is specific to a particular authorization system (as in Akenti [10]). Furthermore, they have discussed the authorization process, either to be centralized (push model[1] as VOMS and CAS[17, 18]), or decentralized (pull model[1] as PERMIS[21] and Akenti[2, 3]). Some systems adopt transport level security rather than message level security as the later involves slow XML manipulations, which make adding security to grid services a big performance penalty[32].

Akenti authorization system has first suggested clustering when it allows the use-condition[10] to determine the rights a user must have to access a SET of resources. However, that was a primitive kind of clustering (PCM). This chapter demonstrates how HCM outperforms PCM.

In this chapter, we first introduce the mathematical foundation of the existing mechanisms used in current authorization systems. Then we introduce the proposed HCM approach. Different HCM Tree-Building Algorithms are also introduced.

3.2 Existing Mechanisms

In this section, a detailed overview of the existing mechanism is given along with mathematical definitions for each mechanism, supported by a working example.

3.2.1 The Brute Force Approach

The classical example of this approach is authorization using Grid-Map file[24]. In this approach, the resource's Security Policy (SP) is represented as a set of security rules. Each of these security rules has to be checked before one can say that the security policy is satisfied. Considering Definition 1,

Definition 1. *Resources & Security Rules Groups*

- *Let* $R = \{r_j \mid j = 1, \ldots, k\}$ *be the set of resources.*
- *Let* $SR = \{sr_j \mid j = 1, \ldots, l\}$ *be the set of all security rules.*

For each resource $r_j \in R$ there will be a corresponding security policy $SP_j \in SR$. All (resource to security policy) assignments can be represented by the Rule Mapping group (RM) as shown in Definition 2:

Definition 2. *The Rule Mapping group (RM)*

- $RM = \{(\mathrm{r_j},\ \mathrm{SP_j}) |\ \mathrm{r_j} \in\ \ \mathrm{R},\ \mathrm{SP_j} \subseteq\ \ \mathrm{SR},\ \mathrm{j} = 1,\ ...,\ \mathrm{k}\}$*is the set of resource to security policy assignments.*

3.2.2 Working Example Illustration

Consider a grid environment with 12 resources $\mathrm{R} = r_1, r_2, ..., r_{12}$ and four security rules $\mathrm{SR} = sr_1, sr_2, sr_3, sr_4$ where:

- sr_1 requires the user to be from XYZ University.
- sr_2 requires the user to have a teacher role.
- sr_3 requires the user to have a student role.
- sr_4 requires the user to be in second year.

All the 12 resources are deployed with the following security policies:

- r_1, r_2 require the user to be from XYZ University to be able to access them. So $SP_1 = SP_2 = \{sr_1\}$.
- r_3, r_4 require the user to be from XYZ and to have a teacher role in order to access them. So $SP_3 = SP_4 = \{sr_1, sr_2\}$.
- r_5, r_6, r_7, r_8, r_9 require the user to be a student in XYZ. So $SP_5 = SP_6 = SP_7 = SP_8 = SP_9 = \{sr_1, sr_3\}$.
- r_{10}, r_{11}, r_{12} require the user to be a second year student in XYZ. So $SP_{10} = SP_{11} = SP_{12} = \{sr_1, sr_3, sr_4\}$.

Then for this grid environment, the Rule Mapping group RM will be as follows:

$$\text{RM} = \{(r_1, \{sr_1\}), (r_2, \{sr_1\}), (r_3, \{sr_1, sr_2\}), (r_4, \{sr_1, sr_2\}), (r_3, \{sr_1, sr_2\}),$$
$$(r_5, \{sr_1, sr_3\}), (r_6, \{sr_1, sr_3\}), (r_7, \{sr_1, sr_3\}), (r_8, \{sr_1, sr_3\}), (r_9, \{sr_1, sr_3\}),$$
$$(r_{10}, \{sr_1, sr_3, sr_4\}), (r_{11}, \{sr_1, sr_3, sr_4\}), (r_{12}, \{sr_1, sr_3, sr_4\})\}.$$

The Brute Force Approach for the proposed example stores the security policies as shown in Figure 3.1. So, the authorization system will have 12 security policies, each of them consists of a set of security rules, all need to be checked for every user request. This procedure has a sequential **search complexity** of $O(k \times l)$; where k is the number of resources in the grid environment ($k = |R|$), l is the maximum number of security rules in each security policy ($l = |SR|$). The immediate improvement is possible by clustering the resources which have identical security policies as discussed in the following subsection.

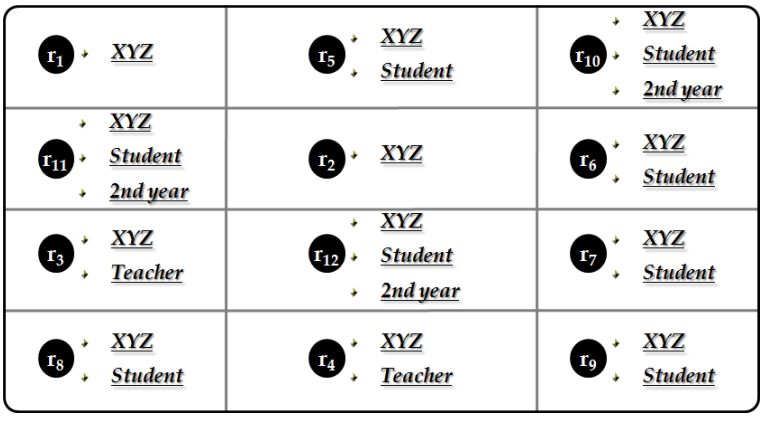

Figure 3.1: Example of Brute Force Security Policy Storing Mechanism

3.2.3 The Primitive Clustering Mechanism

It is clear in the previous example that several resources share the same security policies; for example, r_1 and r_2 have identical security policies SP_1 and SP_2. In general, the Brute Force Approach requires a number of security policies checking processes equal to the number of resources which exist in the grid, irrespective of whether these security policies are identical or similar. Instead, if one looks at the dual system of the Rule Mapping group, it is possible to find a method which reduces the number of security policies to be checked by introducing the Reverse Rule Mapping group (RRM) as shown in Definition 3:

Definition 3. *The Reverse Rule Mapping group (RRM)*

- *Let* $R_j \subseteq \ \mathrm{R} = \{\mathrm{r_i} \mid \mathrm{r_i} \in \ \mathrm{R},\ (\mathrm{r_i},\ \mathrm{SP_j}) \ \in \mathrm{RM}\}$ *be the set of all resources that share the same security policy*SP_j.

- *Then* $RRM = \{(\mathrm{SP_j}, \mathrm{R_j}) \mid \mathrm{SP_j} \subseteq \ \mathrm{SR}\}$ *is the set of security policy to a group of resources assignments.*

With this definition, the authorization system stores the resources' security policies in a clustering manner. This reduces the number of security policies to be checked from $k = |R|$ (the number of resources in the grid) to $n = |RRM|$ (the number of distinct security policies used in the system). Usually k is very large compared to n, so with the introduction of RRM, the **search complexity** is reduced from $O(k \times l)$ to $O(n \times l)$.

It is interesting to observe that after applying the Primitive Clustering Mechanism to the previous example, the RRM group will have only four elements:

$\text{RRM} = \{(\{sr_1\}, \{r_1, r_2\}), (\{sr_1, sr_2\}, \{r_3, r_4\}), (\{sr_1, sr_3\}, \{r_5, r_6, r_7, r_8, r_9\}),$

$(\{sr_1, sr_3, sr_4\}, \{r_{10}, r_{11}, r_{12}\})\}$

Figure 3.2 shows how the Primitive Clustering Mechanism stores the security policies of the proposed example. It is obvious that the number of security policies to be checked is reduced from 12 to 4.

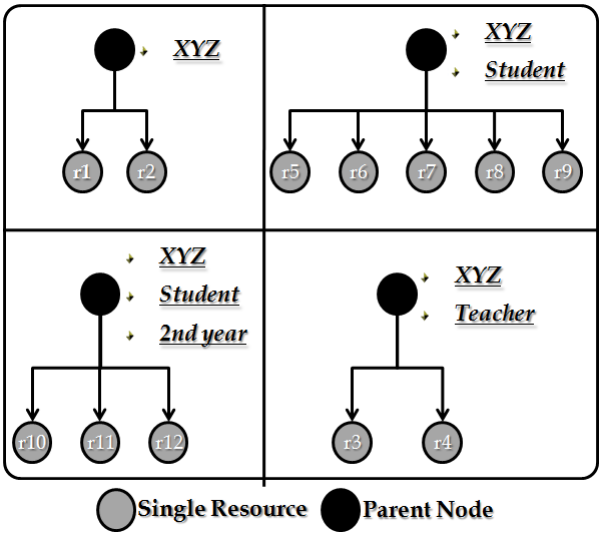

Figure 3.2: Example of the Primitive Clustering Mechanism

3.3 The Hierarchical Clustering Mechanism (HCM)

PCM removes the redundancy of checking identical security policies, but it cannot remove the redundancy of checking identical security rules. In other words, it avoids checking identical security policies SPs more than once; since each security policy SP

is a set of security rules, the security rule (sr) level of redundancy is still prevailing in PCM. As an example, the redundancy of checking identical security policies has been removed by clustering r_5, r_6, r_7, r_8, r_9 in one parent node (as depicted in Figure 3.2). However, it is still required to check the XYZ security rule four times. In general, redundancy in PCM is depicted mathematically as shown in Definition 4:

Definition 4. *Redundancy in PCM*

- *Let* $SP_j = \{sr_i \mid sr_i \in SR\}$ *be the set of rules that represent a single security policy.*
- *PCM is redundant if* $SP_j \bigcap \mathrm{SP_i} \neq \Phi$ *for any* $i \neq j$ *where* $\boldsymbol{SP}_j$ *and* $\boldsymbol{SP}_i$ *are two different security policies.*

In a real grid environment, one can observe that this intersection, seen in Definition 4, is mostly not equal to ϕ. (it will be equal to ϕ only if all the security policies are entirely different), thus PCM still has redundancy.

Like PCM, HCM clusters the resources in parent nodes based on their shared security policies. However, it also achieves a hierarchical clustering of the parent nodes themselves based on their shared security rules. Figure 3.3 shows how HCM organize the security policies of the same proposed example. The search complexity of HCM (number of security rules to be checked to find the user's authorized resource group) is less than that of PCM, as HCM does the same primitive clustering like PCM; moreover, it avoids checking the same security rule shared by different security policies.

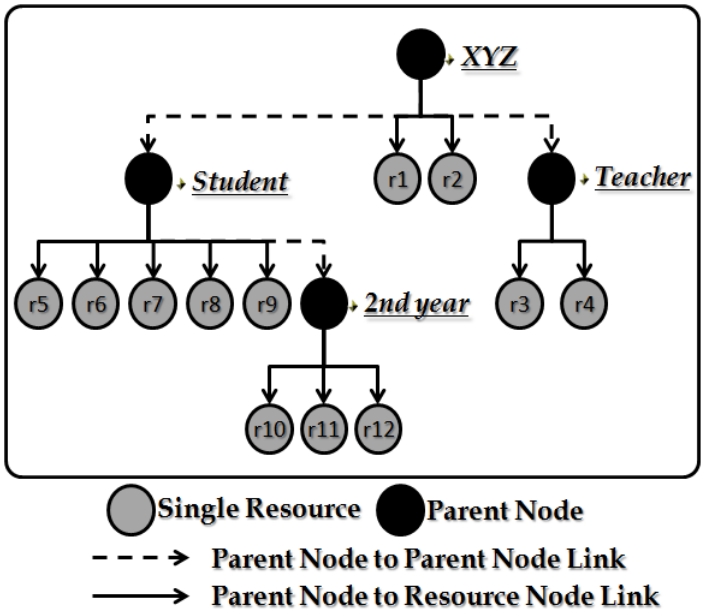

Figure 3.3: Example of Hierarchical Clustering Mechanism

3.3.1 Comparisons

Table 3.1 presents a comparison between the three mechanisms on the same example discussed earlier. The following parameters are used in the comparison:

- The number of security policies to be checked, say = x.
- The number of security rules to be checked, say = y.
- The average number of security rules to be checked per single resource = (y/total number of resources), say = z.

Table 3.1: Comparisons of the three mechanisms

	x	y	z
Brute Force	12	25	2.08
Primitive Clustering	4	8	0.67
Hierarchical Clustering	-	4	0.33

3.4 HCM Tree-Building Algorithms

Building HCM decision tree is not a trivial process. An algorithm to properly choose the root security rule of the tree and its sub-trees is required. In this section, we discuss different algorithms which can be used to build HCM decision tree.

3.4.1 The Counting Algorithm

In this section, a modified version of the Counting Algorithm (first appear in [8]) is proposed (Algorithm 1). It is a single-pass, depth-first algorithm developed to build the *hierarchical decision tree* based on the data of the Security Table (ST). The Security Table (ST) is a table representation of the Resource Mapping group (RM); where security rules are considered as attributes, and resources as objects, with table entries of (i, j)th cell as 1 if the jth security rule is an element of the security policy of the ith object (ith resource). Table 3.2 is the corresponding Security Table for the example considered in Section 3.2.2. The Counting algorithm constructs a tree. If the root node of the tree is NULL then HCM is a forest.

Table 3.2: Security Table Example (Resources Vs Security Rules)

R_{id}	XYZ	Teacher	Student	2^{nd} Year
r_1	1	0	0	0
r_2	1	0	0	0
r_3	1	1	0	0
r_4	1	1	0	0
r_5	1	0	1	0
r_6	1	0	1	0
r_7	1	0	1	0
r_8	1	0	1	0
r_9	1	0	1	0
r_{10}	1	0	1	1
r_{11}	1	0	1	1
r_{12}	1	0	1	1

Table 3.3: Count Table

	XYZ	Teacher	Student	2^{nd} Year
Count	12	2	8	3

Algorithm 1 The Counting Algorithm

Input:

Resources' Security Table

Output:

HCM Decision Tree

Variable:

Each node in the tree is a structure having two fields:

- the *security rule* **sr**,
- an interim *security table* **ST**

Begin

Step 1: (Initialization)

- Initialize the *decision tree* by a root node with NULL *security rule* (**sr**).
- Build the security table which represents the entire security policies of the system (Table 3.2). Assign it to the **ST** property of the root node.
- Execute Step 2 for the root node.

Step 2: (Processing of a node N)

Step 2.1: (Adding N's Resources)

- Add each resource, whose corresponding row in N's **ST** is having zero cells only, as a child resource to N then delete its corresponding row from ST.

***Step 2.2:* (Processing of N's security table (ST))**

- Sum the cells of each column of N's **ST** and refer it as **Count** (Table 3.3).
- Choose the security rule sr_j with the highest **Count**.
- Divide **ST** into two tables excluding the j^{th} column as the following:
  - The first table (T_1) contains the rows of the resources which demand sr_j (each row whose j^{th} cell = 1).
  - The second table (T_2) contains the rows of the resources which do not demand sr_j (each row whose j^{th} cell = 0).

***Step 2.3:* (N Bifurcation)**

- Add a left child node to N with sr_j as the *security rule* (**sr**) and T_1 as the *security table* (**ST**).
- Add a right child node to N with NULL as the *security rule* (**sr**) and T_2 as the *security table* (**ST**).

Step 3: (Recurring)

- Repeat step 2 for each child node until a node with **empty** *security table* is reached.

Step 4: (Pruning)

- Prune the derived tree at nodes labeled NULL.
- Delete all the interim *security tables* (**ST**s) to free space.

End

Computational Complexity of the Counting Algorithm	
Step 1	O($M \times N$): Where M is the number of resources and N is the number of security rules.
Repeated	1 time
Step 2	O($\frac{M}{2^i} \times (N - i)$): As an average complexity, where i is the node's level
Repeated	In average, we have 2^i nodes in each level and (0 to N) levels
Step 3	O(1): Simple condition check.
Repeated	(N + 1) times
Step 4	O(N): Maximum number of NULL nodes in the tree is N
Repeated	1 time
Algorithm Complexity: O($M \times N^2$). (usually N << M)	

3.4.2 Functional Dependency Based Algorithm

Security rules are often related to each other, so it would be more efficient to check a particular security rule before another. This idea by default is considered in the HCM decision tree structure. As an example, when the decision tree shown in Figure 3.3 is parsed for a particular user, it always checks *student* security rule before checking 2^{nd} *Year*. If *student* is not satisfied, then 2^{nd} *Year* won't be checked.

Although the decision tree embraces security rules dependencies, the Counting Algorithm is not completely based on it. When a security rule is selected to be the root node of the decision tree (as XYZ) or of a sub-tree (as *student*), it is chosen based on the maximum number of 1's exists in its column in the security table.

A functional dependency based Algorithm (Algorithm 2) is introduced to choose the root security rule purely based on functional dependencies. But in many real grid environments, it is not possible to find a valid functional dependency applicable for ALL the tuples of the security table. For that, one can think of using an association rule that approximates the functional dependency as shown in the next section.

Algorithm 2 The Functional Dependency Algorithm

Input:

Resources' Security Table

Output:

Root Security Rule of a HCM Decision Tree

Begin

- Let $SR = \{sr_j \mid j = 1 \text{ to } m\}$ be the set of all security rules, where **m** is the total number of security rules exist in the system.
- The functional dependency $sr_i \rightarrow sr_j$ holds on schema ST (*Security Table*), if for all pairs of tuples t_1 and t_2 in ST such that $t_1[sr_i] = t_2[sr_i]$, it also the case that $t_1[sr_j] = t_2[sr_j]$.
- Let n_i be the number of the *functional dependencies* hold on ST for sr_i security rule; it means how many valid $sr_i \rightarrow sr_j$ relations exist, where j=1 to m excluding i.
- Choose the security rule with the highest n_i to be the root node of the *decision tree* or of the *sub-tree*.

End

3.4.3 Association Rules Based Algorithm

Finding valid functional dependencies applicable to ALL tuples of the security table is not possible in many real grid environments. In this section, a mechanism that uses association rules is applied to approximate the functional dependencies and build the HCM decision tree (Algorithm 3).

Algorithm 3 The Association Rules Based Algorithm

Input:

Resources' Security Table

Output:

Root Security Rule of a HCM Decision Tree

Begin

1. Let k = 1 to N, // where N is the number of security rules. //
2. Let j = 1 to N.
3. Compute O2O, O2Z, Z2O, and Z2Z values out of the *security table* as the following (depicted in Table 3.4):
   - Let O2O be the number of 1s in the $\mathbf{k}^{th}$ column which correspond to 1s in the $\mathbf{j}^{th}$ column.
   - Let O2Z be the number of 1s in the $\mathbf{k}^{th}$ column which correspond to 0s in the $\mathbf{j}^{th}$ column.
   - Let Z2O be the number of 0s in the $\mathbf{k}^{th}$ column which correspond to 1s in the $\mathbf{j}^{th}$ column.
   - Let Z2Z be the number of 0s in the $\mathbf{k}^{th}$ column which correspond to 0s in the $\mathbf{j}^{th}$ column.
4. Define fdp_j by the following formula: fdp_j = Max(O2O, O2Z) / (O2O + O2Z) + Max(Z2O, Z2Z) / (Z2O + Z2Z).
5. Repeat steps 3, and 4 for each j.
6. Define fdp_k = sum (fdp_j)/N.
7. Repeat steps 2, 3, 4, 5 and 6 for each k.
8. Choose the $\mathbf{k}^{th}$ security rule with the highest fdp_k to be the root of the *decision tree* or of the *sub-tree*.

End

Table 3.4: (kth Column Vs jth Column)

	0	**1**
0	Z2Z	Z2O
1	O2Z	O2O

Computational Complexity of the Association Rules Based Algorithm	
Step 1, 2	are 2 nested loops of total O(N^2) complexity
Step 3, 4	O(M): Where M is the number of resources
Repeated	N^2 times
Step 5	O(1): End of the inner loop body
Repeated	N^2 times
Step 6	O(N)
Repeated	N times
Step 7	O(1): End of the outer loop body
Repeated	N times
Step 8	O(N)
Repeated	1 time
Algorithm Complexity: O($M \times N^2$)	

3.4.4 Experiments and Results

A grid environment of 120 resources and 15 security rules has been considered and 100 instances of security policies are simulated (100 security tables are generated) to study the effectiveness of HCM.

For each security table, the posterior analysis of the Brute Force Approach (BFA) and HCM has been done and depicted in Figure 3.4. The average number of the security rules to be checked in the BFA was 902 with a standard deviation 20.6184; while it was 402 in HCM with a standard deviation 9.8489. The complexity range ([min, max] number of security rules to be checked over all experiments) of the BFA was [853, 952], while it was [377, 427] for HCM.

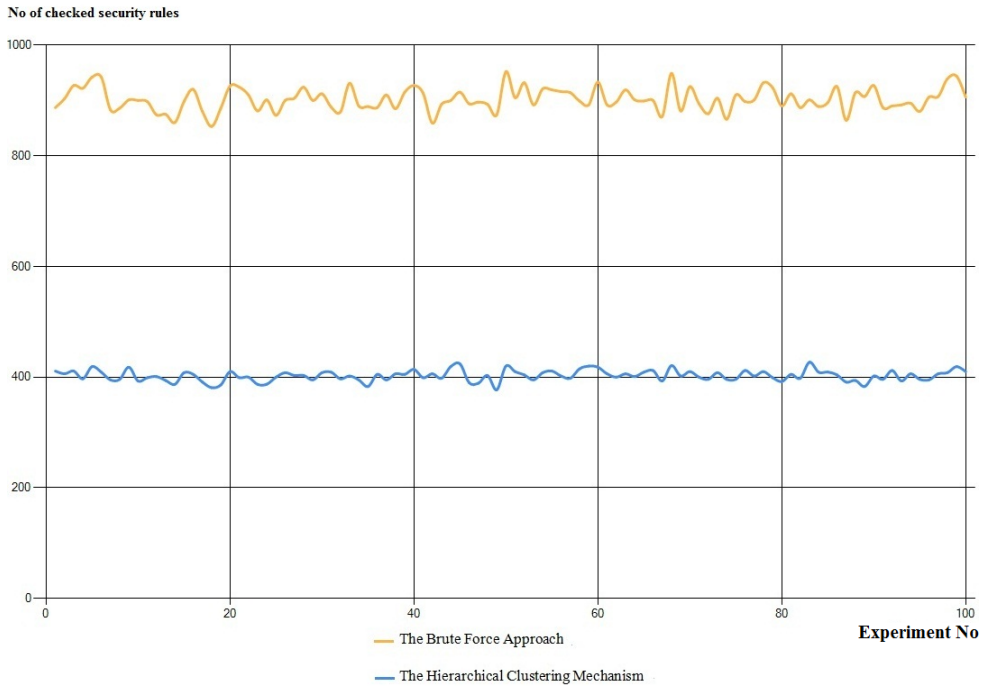

Figure 3.4: Experiments and Results

PCM did not show any considerable improvements over the BFA so it is not included in the experiments. The possibility of having two identical rows in a security table with 120 rows and 15 columns is negligible (since the number of distinct rows with 15 columns is $2^{15} = 32768$, which is quite bigger than 120). Thus, in this simulated environment, PCM shows almost the same performance as of the BFA. The efficiency of PCM depends on two issues:

- The nature of a real time grid environment is intended to have more identical security policies compared to a synthetic one.
- If $k >> 2^l$, where k is the number of resources in the grid and l is the number of security rules, then PCM will perform better even in a synthetic environment.

Experiments have not shown any considerable improvement of the other criteria over the Counting criterion. However, the other criteria may outperform the Counting criterion in real time environments where the effect of functional dependencies is more manifest compared to the randomized environment used in our experiments.

3.5 Summary and Limitations

This chapter introduces the Hierarchical Clustering Mechanism as a fine-grained access control mechanism which best suits in heterogeneous and legacy systems such as Grid. It reduces the authorization overhead to a great extend and serves huge number of user requests in a more effective way compared to the Brute Force Approach as well as the Primitive Clustering Mechanisms. Experiments are conducted in a simulated environment.

However, HCM is considered to be an expensive process in terms of memory consumption and computation required to build the decision tree. In a dynamic environment as the Grid, regenerating the decision tree for every dynamic change occurs will cause a computational overhead. Further, Grid is a multi-domain system, while HCM requires a centralized policy repository to build its decision tree.

For the above limitations, next chapter introduces various tools, which can be embedded in HCM to increase its scalability and make it more amenable to the dynamic changes in the Grid.

Chapter 4

HCM Scalability Issues

In a heterogeneous, dynamic and cross-domain environment such as Grid, access control scalability issues are the most important issues to be analyzed. In this chapter, we discuss all of them. They are divided into the architectural related scalability issues and User Related Scalability Issues.

In the architectural related scalability issues, we have analyzed the aspects related to Grid size and cross-domain authorization. As HCM is slightly an expensive process in terms of the computations required to build the decision tree and memory consumption, so embedding HCM in a small grid may add complexity to the system. A new Rough Set based PCM (RSPCM) is proposed as an alternative to HCM for those environments which are less dynamic and have less or medium number of resources, and is superior to normal PCM in terms of redundancy. A paper has been published based on RSPCM [33].

HCM has more benefits in dynamic and huge grid environments. However, in such environments, deletion, updation, and addition of resources take place quite often. If

HCM is used in its present form, these processes require rebuilding the *decision tree* every time. This chapter shows how we can avoid this by adding simple constructs to it that makes it more amenable for implementation.

Moreover, HCM demands a centralized authorization database where all the resources' security policies have been kept in order to build the decision tree. This is difficult to achieve in a heterogeneous environment like Grid where the security policies are distributed across different domains. This chapter addresses this issue and presents the Distributed HCM as an efficient solution.

User related scalability issues are also important in grid, grid users are many and they dynamically join and leave it. This chapter introduces several tools which can be embedded in HCM for more efficient authorization. One of these tools is the Temporal Caching Mechanism (TCM) which helps to avoid reparsing the decision tree for every user request. In some grid environments, few roles of the user can be changed frequently, thus it is important for the caching mechanism to handle this case. Another caching mechanism which uses the Hamming Distance (HDCM) is also proposed. A paper based on this work has been published[34].

Apart from caching, the capability of plain HCM, as a centralized agent, in scaling up to huge number of users' authorization requests also need to be analyzed. This chapter addresses this issue and introduces the concurrent model of HCM, where it can process multiple authorization requests simultaneously. Moreover, this model can also parallelize a single authorization request in order to reduce the response time. A paper based on that has been published in ICDCIT 2012[35].

HCM was mainly designed to find the **SET** of resources which the user is authorized to access with the least redundancy in checking security rules. However, when a user intends to use a particular resource, HCM spends more time in parsing the whole decision tree, when checking a **SINGLE** resource security policy itself is sufficient. This chapter addresses this issue and presents efficient solutions for it.

4.1 Architectural Related Scalability Issues

Architectural related scalability issues are about grid size and cross-domain authorization. This section shows how HCM can scale for size and heterogeneity.

4.1.1 Issues Related to Grid Size

This section discuss the grid size related scalability issues. HCM is slightly an expensive process in terms of computations required to build the decision tree and memory consumption. So embedding HCM in a small grid may add complexity to the system. A new Rough Set based PCM (RSPCM) is proposed as an alternative to HCM for those environments that are less dynamic and have less number of resources.

4.1.1.1 The Rough Set based PCM

Normal PCM involves clustering the resources which have similar security policies. Then for authorizing a user, all of these clustered security policies have to be checked. In this section a Rough Set based algorithm is proposed which helps to selectively check a specific clustered security policies rather than checking all of them.

The algorithm is divided into two stages. In ***Stage 1*** (Algorithm 4), it prepares the Hashed Clusters Table which is going to be used in ***Stage 2*** of authorization.

Algorithm 4 The Rough Set based PCM - Stage 1

Input:

Resources' Security Table

Output:

The Hashed Clusters Table

Begin

1. Let R be the set of ***reducts*** of the *security table*.
2. Let CR be the ***reduct*** which has the least number of attributes (security rules).
3. Cluster the *security table* into C clusters based on CR attributes to get the *Resources Clusters Table*.
4. Order and hash (enumerate) the clusters as per the values of CR attributes to get the *Hashed Clusters Table*.

End

Computational Complexity of the Rough Set based PCM - Stage 1	
Step 1	O($M \times lg(M) \times N^2$): Where M is the number of resources and N is the number of security rules.
Step 2	O($\|R\|$)
Step 3, 4	O($M \times N$)
Algorithm Complexity: O($M \times lg(M) \times N^2$)	

Consider the following security table (ST) shown in Table 4.1. Apply ***Stage 1*** on **ST** to build the *Hashed Clusters Table*, we get the following results:

Table 4.1: Security Table (ST)

$\mathbf{R}_{id}$	$\mathbf{sr}_1$	$\mathbf{sr}_2$	$\mathbf{sr}_3$	$\mathbf{sr}_4$
$\mathbf{R}_1$	0	0	0	1
$\mathbf{R}_2$	1	0	1	0
$\mathbf{R}_3$	0	1	0	1
$\mathbf{R}_4$	0	0	0	1
$\mathbf{R}_5$	1	1	0	0
$\mathbf{R}_6$	0	1	0	1
$\mathbf{R}_7$	0	0	0	1
$\mathbf{R}_8$	1	0	1	0
$\mathbf{R}_9$	1	1	0	0
$\mathbf{R}_{10}$	1	0	1	0

1. Find the set of ***reducts*** R= {R_1, R_2} where R_1= {sr_1, sr_2} and R_2= {sr_2, sr_4}.

2. Let CR = R_1.

3. Cluster the resources based on CR attributes as shown in Table 4.2.

Table 4.2: Resources Clusters

$\mathbf{R}_{group}$	$\mathbf{sr}_1$	$\mathbf{sr}_2$	$\mathbf{sr}_3$	$\mathbf{sr}_4$
$\mathbf{R}_1$, $\mathbf{R}_4$, $\mathbf{R}_7$	0	0	0	1
$\mathbf{R}_2$, $\mathbf{R}_8$, $\mathbf{R}_{10}$	1	0	1	0
$\mathbf{R}_5$, $\mathbf{R}_9$	1	1	0	0
$\mathbf{R}_3$, $\mathbf{R}_6$	0	1	0	1

4. Order and hash the clusters as per the values of CR attributes (Table 4.3).

Table 4.3: HashedClustersTable

$\mathbf{R}_{group}$	$\mathbf{sr}_1$	$\mathbf{sr}_2$	$\mathbf{sr}_3$	$\mathbf{sr}_4$	**Cluster**
$\mathbf{R}_1$, $\mathbf{R}_4$, $\mathbf{R}_7$	0	0	0	1	0
$\mathbf{R}_3$, $\mathbf{R}_6$	0	1	0	1	1
$\mathbf{R}_2$, $\mathbf{R}_8$, $\mathbf{R}_{10}$	1	0	1	0	2
$\mathbf{R}_5$, $\mathbf{R}_9$	1	1	0	0	3

Now, for authorizing a user, we need to follow the steps of ***Stage 2*** (Algorithm 5). In this stage, each user has a *Privileges Vector* and a *flag vector* of $|C|$ bits initialized with zeros. The *domination* relationship between 2 vectors of binary attributes is defined in the following statement: $S1\ dom\ S2 \Leftrightarrow S1\ \Lambda\ S2 = S2$

Algorithm 5 The Rough Set based PCM - Stage 2

Input:

The Hashed Clusters Table & The User's Privileges Vector

Output:

User's Authorized Resource Group (UARG)

Begin

1. Find the set of *clusters* (SC) whose *reduct* attributes are *dominated* by the *reduct* attributes of the user's *Privileges Vector.*
2. For every *cluster* in SC whose *non-reduct* attributes are *dominated* by the *non-reduct* attributes of the user's *Privileges Vector*, set the corresponding bit of the *flag vector* to 1.
3. Add the resource group (R_{group}) of every *cluster*, whose corresponding bit in the *flag vector* is set to 1, to the UARG.

End

Computational Complexity of the Rough Set based PCM - Stage 2	
Step 1	O($\|C\| \times \|CR\|$): Where C is the set of clusters and CR is the reduct attributes.
Step 2	O($\|SC\|$)
Step 3, 4	O($\|C\|$)
Algorithm Complexity: O($\|C\| \times \|CR\|$)	

- **Example 1**

  Assume a user of *Privileges Vector* shown in Table 4.4. The initial *flag vector* of this user is shown in Table 4.5. For finding the ARG of this user, let us apply the steps of ***Stage 2***.

Table 4.4: User Privileges Vector

$\mathbf{sr}_1$	$\mathbf{sr}_2$	$\mathbf{sr}_3$	$\mathbf{sr}_4$
1	0	1	1

Table 4.5: Flag Vector

$\mathbf{C}_0$	$\mathbf{C}_1$	$\mathbf{C}_2$	$\mathbf{C}_3$
0	0	0	0

1. SC for this user = $\{C_0, C_2\}$.
2. As the *non-reduct* attributes of the user's *Privileges Vector* (which are sr_3 and sr_4 in this example) *dominate* the *non-reduct* attributes of C_0 and C_2, set the corresponding bits of the *flag vector* to 1 as shown in Table 4.6.

Table 4.6: Flag Vector

$\mathbf{C}_0$	$\mathbf{C}_1$	$\mathbf{C}_2$	$\mathbf{C}_3$
1	0	1	0

3. So the UARG = $\{\mathbf{R}_1, \mathbf{R}_4, \mathbf{R}_7, \mathbf{R}_2, \mathbf{R}_8, \mathbf{R}_{10}\}$.

- **Example 2**

  Assume a user of *Privileges Vector* shown in Table 4.7. The initial *flag vector* of the user will be the same as shown in Table 4.5.

Table 4.7: User Privileges Vector

$\mathbf{sr}_1$	$\mathbf{sr}_2$	$\mathbf{sr}_3$	$\mathbf{sr}_4$
1	0	0	1

1. SC for this user = {C_0, C_2}.

2. As the *non-reduct* attributes of the user's *Privileges Vector dominate* the *non-reduct* attributes of C_0 the corresponding bits of the *flag vector* is set to 1 as shown in Table 4.8. As the *non-reduct* attributes of the user's *Privileges Vector* **do not** *dominate* the *non-reduct* attributes of C_2 then leave the *flag vector* unchanged.

Table 4.8: Flag Vector

$\mathbf{C}_0$	$\mathbf{C}_1$	$\mathbf{C}_2$	$\mathbf{C}_3$
1	0	0	0

3. UARG = **{$\mathbf{R}_1$, $\mathbf{R}_4$, $\mathbf{R}_7$}.**

- **Example 3**

  Assume a user of *Privileges Vector* shown in Table :4.9. The initial *flag vector* of the user will be the same as shown in Table 4.5.

Table 4.9: User Privileges Vector

$\mathbf{sr}_1$	$\mathbf{sr}_2$	$\mathbf{sr}_3$	$\mathbf{sr}_4$
1	0	0	0

1. SC for this user = {C_0, C_2}.

2. As the *non-reduct* attributes of the user's *Privileges Vector* **do not** *dominate* the *non-reduct* attributes of C_0 and C_2 then leave the *dynamic flag vector* as it is.

3. UARG = ϕ

Experiments are done using the *Rough Set Exploration System ver. 2.2*[36].

4.1.1.2 Analyzing the Advantages of the Rough Set based PCM

In this section, the advantages of the Rough Set based PCM are studied over the normal PCM. For simplifying the discussion, refer to the binary value of the *reduct* attributes of the user's *Privileges Vector* as X. The following points are the advantages of the Rough Set based PCM:

1. Finding the set of *clusters* (SC) whose *reduct* attributes are *dominated* by X does not require checking the *reduct* attributes of all the rows in the *Hashed Clusters Table*. It is enough to check those rows whose *reduct* attributes values are less than X. As the row whose reduct attributes value is equal to X is of course *dominated* by X and the rows whose reduct attributes values are greater than X cannot be *dominated* by X. As an example, consider the user whose *Privileges Vector* is shown in Table 4.10. The *reduct* attributes value of this user is X = $(01)_b$ = $(1)_d$. So, for this user, it is enough to check the *domination* relationship with the *reduct* attributes value of only row 0 of the *Hashed Clusters Table*. As row 1 will satisfy the *domination* relationship by equality and rows {2, 3} can never satisfy it.

Table 4.10: User Privileges Vector

$\mathbf{sr}_1$	$\mathbf{sr}_2$	$\mathbf{sr}_3$	$\mathbf{sr}_4$
0	1	0	0

2. As it is obvious in ***Stage 2***, once a row is chosen to be checked whether it is *dominated* by X or not, it is not always required to check the whole columns (*security rules*) of it. Only if the *domination* relationship is hold for the *reduct* attributes of this row, then we need to further investigate the *domination* relationship between the remaining *non-reduct* attributes (remaining security rules). As an example, consider the user whose *Privileges Vector* is shown in Table 6. The rows which are going to be checked for the *domination* relationship between the *reduct* attributes are {0, 1}. As the *reduct* attributes {sr_1, sr_2} of row 0 is *dominated* by X, then we need to further investigate the *non-reduct* attributes {sr_3, sr_4} of row 0. However, the *reduct* attributes of row 1 is not *dominated* by X, then we need not to further investigate the *non-reduct* attributes of row 1.

3. Normal PCM does not define the algorithm by which it clusters the resources based on their similar security policies. It can be any primitive clustering algorithm. In the Rough Set based PCM, it clearly specifies the use of *reduct* to cluster the resources. This is a well known clustering mechanism in the literature[37, 38, 39].

4. The user's *flag vector* can be used as a caching mechanism through which the user need not go into the authorization stage more than once unless a change

is occurred either in his *Privileges Vector* or in the *Security Table*. Consider **Example 1**, if the same user is willing to access the grid in the future, then the only thing which is required to find his UARG is checking his previous *flag vector* shown in Table 4.6.

For the above mentioned points, the Rough Set based PCM is considered to be more efficient compared to the normal PCM as it avoids checking of many unnecessary security rules. It smartly chooses the appropriate security rules which need to be checked for a particular user. In normal PCM, it blindly checks all the security rules of all the clusters which increases redundancy and decreases efficiency.

It is worth mentioning that the proposed examples are made simple to clarify the algorithm, but in real time scenario, the *security table* will be more complicated and the advantages of RSPCM will become more evident.

4.1.1.3 HCM Stability Against Dynamic Changes

In huge grid environments deletion, updation, and addition of resources take place quite often. If HCM is used in its present form, these processes require rebuilding the *decision tree* every time. We can avoid this by adding simple constructs to HCM that makes it more amenable for implementation.

- ***Resource Deleting:*** This is the simplest manipulation process. It can be done by just removing the resource from the *decision tree*.
- ***Resource Adding:*** Any new resource (r_k) can be added to the *decision tree* without the need of rebuilding the tree entirely by applying the following steps:

1. Parse the *decision tree* according to the security policy of r_k until a parent node with an identical security policy is found. Add r_k as a child resource to it and exit.

2. If this parent node is not found. Choose the parent node (CP) with the closest security policy to r_k. Make a new parent node (NP) having the same security policy as r_k. Add r_k as a child resource to NP. Add NP as a child node to CP.

3. After **X** number of additions, the *decision tree* will not remain as efficient as it was earlier. So at this stage, we need to rebuild the *decision tree* from the original Security Table (ST).

- ***Resource Updating:*** For any resource updating process, **t**he *decision tree* can be synchronized by combining the previous (deleting/adding) algorithms:

  1. Delete the old version of the resource from the tree.

  2. Add the modified version of the resource as a new resource to the tree.

By following the previous algorithms, one can observe that the *decision tree* remains usable for many numbers of modifications before the system is advised to rebuild it again. Thus they can be called as *incremental algorithms*. To study the efficiency of these algorithms, a Java program is written to simulate 100 instances of HCM *decision tree* with 700 resources and 30 different security rules. For each instance, the time required to add 50 resources is calculated and compared with the time required to rebuild the decision tree entirely. Results are depicted in Figure 4.1. The average time required for *the rebuilding approach* was 79.5ms with a standard

deviation as 27.79, while it was only 17.7ms for *the proposed algorithm* with a standard deviation as 5.70.

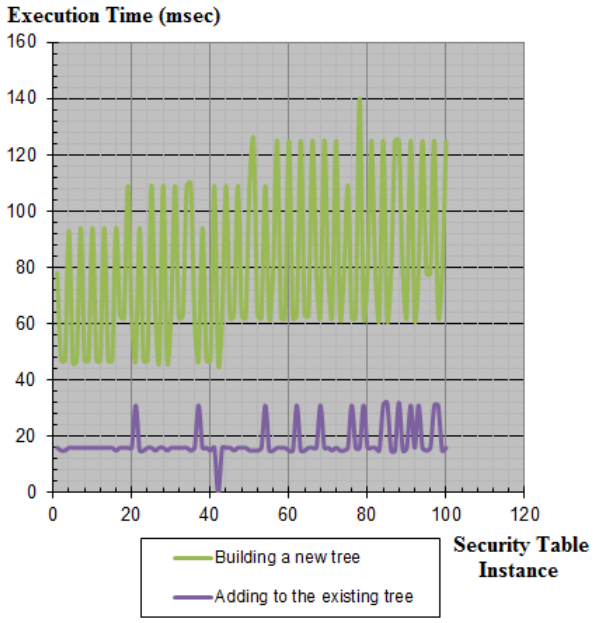

Figure 4.1: Incremental Algorithm Experiments and Results

Determining the value of **X** (number of additions before the system is advised to rebuild the *decision tree* entirely) is also an important parameter to study the efficiency of HCM in a dynamic environment. Many experiments have been done to estimate **X's** value before the system is advised (with a possibility of 50%) to rebuild the *decision tree* entirely. Results are depicted in Table 4.11. It is seen from Table 4.11 that the percentage of **X** to the total number of resources is reasonable enough to consider HCM as an efficient mechanism in a dynamic environment.

Table 4.11: Experiment to determine the value of X

Resource Number	Security Rules Number	**X**	Modification Percentage
550	7	474	46%
572	7	1476	72%
1210	8	838	41%
1300	8	2796	68%
456	9	568	55%
1310	9	738	36%
6000	10	2192	27%
6912	10	9472	58%

4.1.2 HCM Cross-Domain Authorization

Grid is an infrastructure that enables the integrated collaborative use of high-end computers, networks, databases and scientific instruments owned and managed by multiple organizations. Each virtual organization (VO) has its own administrative domain. An administrative domain is a scope of authority. In a simple grid use case, the Subject can be in one administrative domain and the Resource in another. Where, in a typical grid scenario, the Subject can use services from several domains (VOs). Grid Service Providers may provide resources to users from multiple VOs.

In order to build the HCM *decision tree*, an access to the security policies of whole grid resources is a MUST. However, grid systems never maintain such a database as the resources' security policies are distributed across different domains as per their resources. This paper addresses this issue and introduces the Distributed HCM (DHCM) as a solution to it with three different models.

4.1.2.1 Light Central HCM and Heavy Agents

In this model, each domain has a HCM Agent. This Agent runs the Counting Algorithm on the domain's resources security policies, this results in a local HCM *decision tree*. Like that we will have multiple local HCM *decision trees* distributed in different administrative domains. Figure 4.2 shows an example of this model.

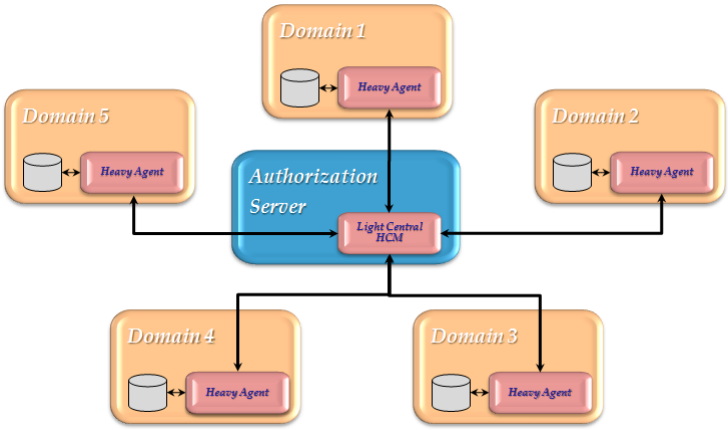

Figure 4.2: Example of the DHCM in Heavy Agent Model

Once an authorization request is fired, the Central HCM has to propagate the request to its registered agents across the grid domains. Local HCM Agents are responsible for parsing their local HCM *decision tree* to get the set of authorized local resources, then they have to reply with this set to the Central HCM. The Central HCM concatenates the different sets to get the whole set of resources which the user is authorized to access.

This model has the advantage of fast parsing of the *decision tree* as it is distributed across several domains (type of parallel processing). Whereas, it has the disadvantage of network latency due to communications required between the Central HCM and its Agents for each authorization request.

4.1.2.2 Heavy Central HCM and Light Agents

In this model, the HCM Agent in each domain is responsible for propagating its resources' security policies information to the Central HCM via a secured channel. The Central HCM is responsible for running the Counting Algorithm on the collected information, this results in a global HCM *decision tree* (Figure 4.3).

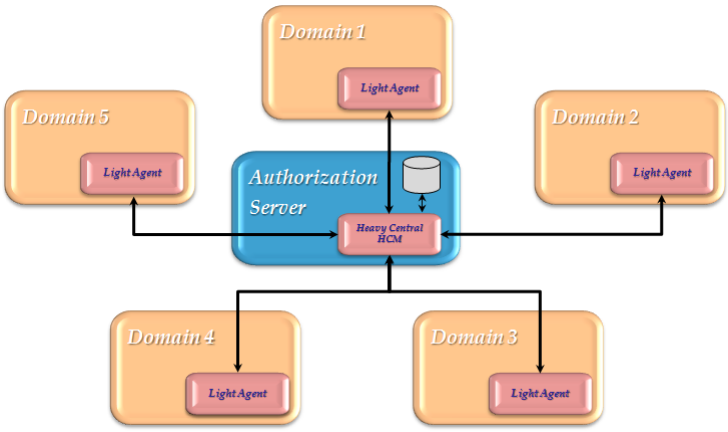

Figure 4.3: Example of the DHCM in Light Agent Model

This model serves the incoming authorization request faster as it does not require any network communication. However, it may cause an overhead to the authorization server if the global HCM *decision tree* was big enough. Moreover, it requires network communication to propagate any updates which may occur to local resources.

4.1.2.3 Hybrid Model

If the domain's resources are few, then running the Heavy HCM Agent in such domains unnecessarily complicates the process. On the other hand, if the domain's resources are numerous, propagating and maintaining updates of these resources to the Heavy Central HCM is an expensive process. So, one can think of a hybrid

model where the Central HCM has to deal with two types of agents. It has to synchronize resources information with the Light Agents for those domains who have few resources, and it has to propagate authorization requests to the Heavy Agents in those domains with numerous resources and wait for their replies. Figure 4.4 shows an example of this model.

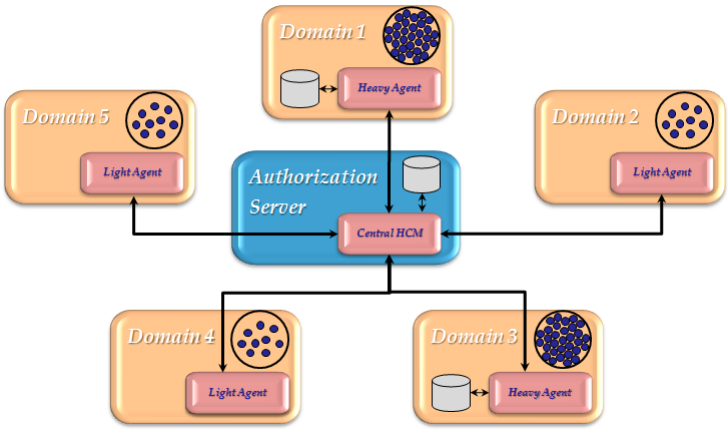

Figure 4.4: Example of the DHCM in Hybrid Model

4.2 User Related Scalability Issues

User scalability issues are very important in grid, grid users are many and they dynamically join and leave the grid. This section introduces several tools which can be embedded in HCM for more efficient authorization. One of these proposed tools is the Temporal Caching Mechanism (TCM) which helps to avoid reparsing the decision tree for every user request. In some grid environments, few roles of the user can be changed frequently, thus it is important for the caching mechanism to handle this case. Another caching mechanism which uses the Hamming Distance (HDCM) is also proposed in this section.

4.2.1 Temporal Caching Mechanism (TCM)

Each parent node in the decision tree can be represented as shown in Figure 4.5; where N_id is a unique number for each parent node, and the timestamp refers to the time in which the Security Rules List of that particular parent node (or its ancestors) has most recently been modified. The ChildNodes List is a list of all internal child nodes except the resources, and Resource's List is a list of all child resources.

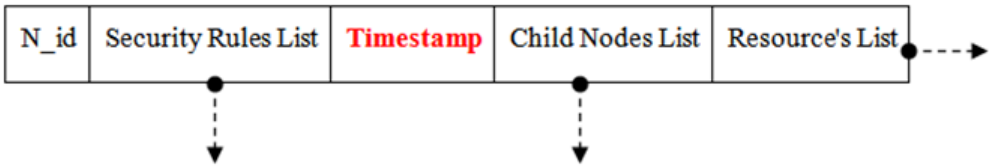

Figure 4.5: Parent Node Structure

While each user has a special record represented as shown in Figure 4.6; where the timestamp refers to the time in which the authorization system has parsed the decision tree most recently for that particular user. The Parent Nodes List field is a linked list of the allocated parent nodes for the user.

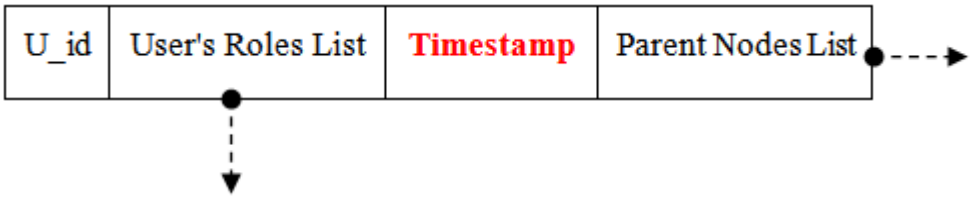

Figure 4.6: User's Record Structure

Now when a user requests for a grid service, the authorization system (embedded with the TCM) will provide him his UARG by following Algorithm 6:

Algorithm 6 The Temporal Caching Mechanism

Input:

HCM Decision Tree & User Record

Output:

UARG

Begin

1. If the *timestamp* field of the user's record is NULL (See Figure 4.7), then according to the associated user's roles, parse the *decision tree* starting from the root node (Figure 4.8 & Figure 4.9)

   (a) For each parent node which the user satisfies its security policy; and does not satisfy the security policy of any of its child nodes; add its unique id to the *Parent Nodes List,* along with a *timestamp* as shown in Figure 4.10.

   (b) Continue parsing the tree until there are no more authorized parent nodes.

   (c) Exit.

2. Directly point to the first parent node of the *Parent Nodes List* of the user's record (Figure 4.12).

   (a) Check if the *timestamp* field of the user's record is greater than the *timestamp* field of the corresponding parent node. If no, update the user's *timestamp* to NULL and go to step 1.

   (b) Move up starting from that parent node to the root of the *decision tree*; add all the single resources to the UARG (Figure 4.13).

3. Repeat step 2 for all other parent nodes in the *Parent Nodes List* field.

4. Exit.

End

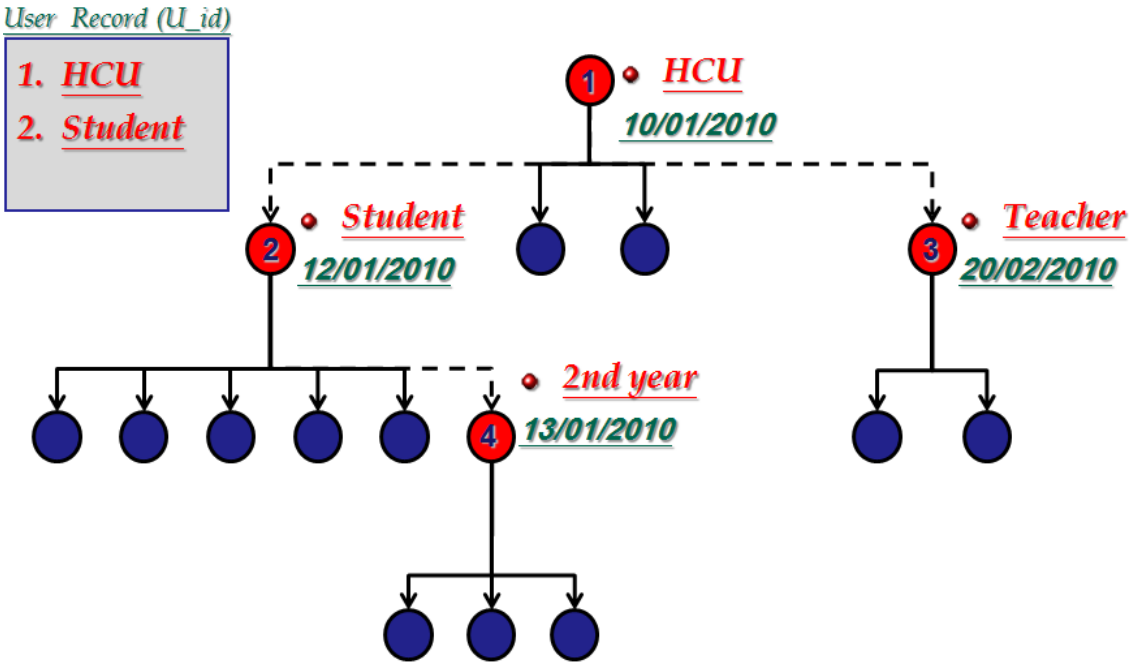

Figure 4.7: Example on Temporal Caching (Initial State)

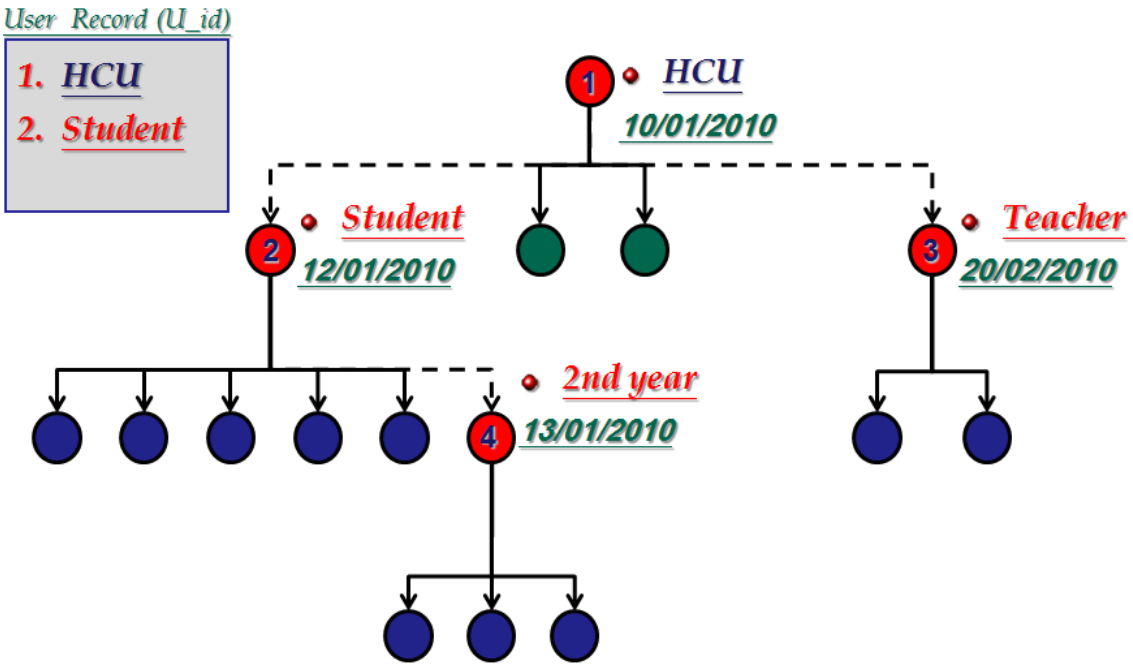

Figure 4.8: Example on Temporal Caching (Parsing the Tree)

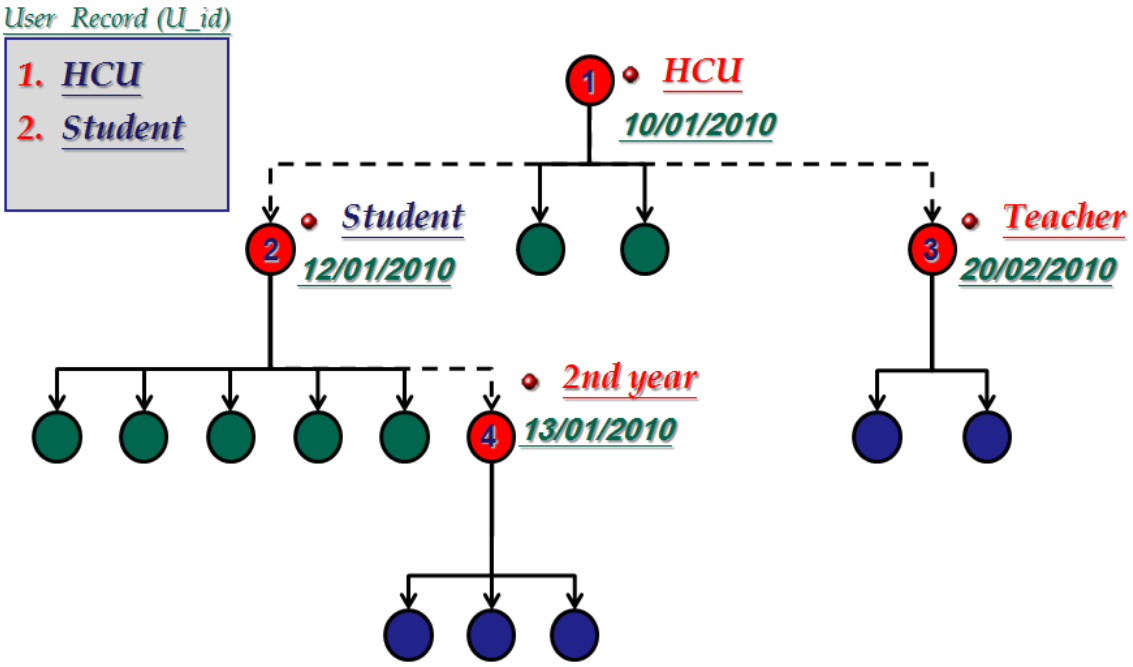

Figure 4.9: Example on Temporal Caching (Parsing the Tree)

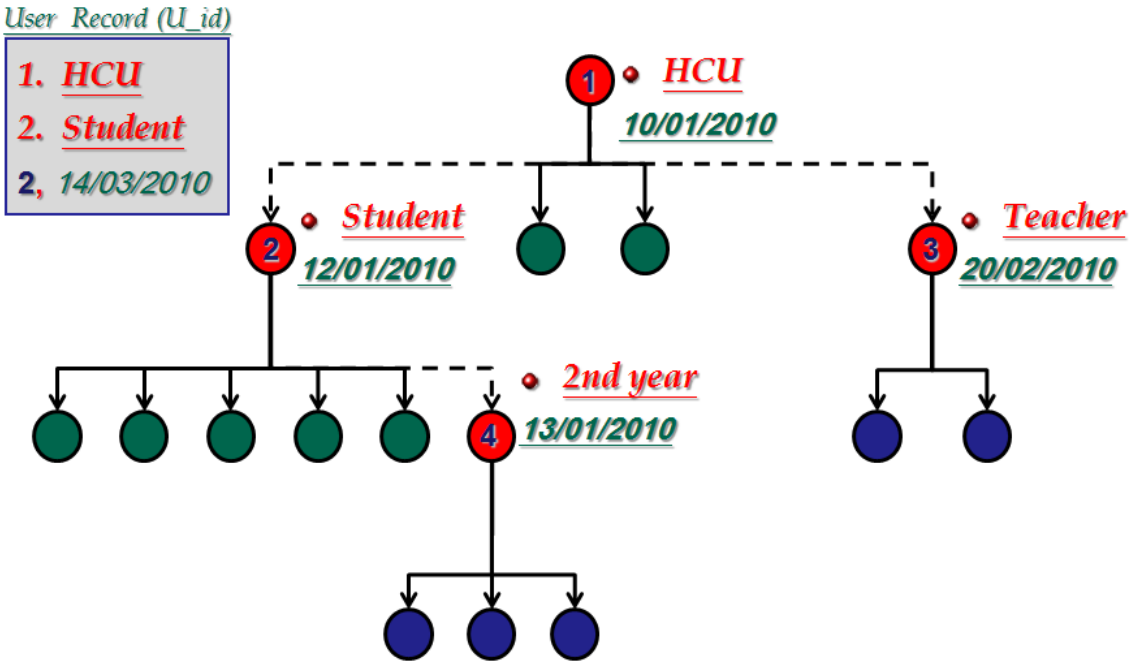

Figure 4.10: Example on Temporal Caching (Caching)

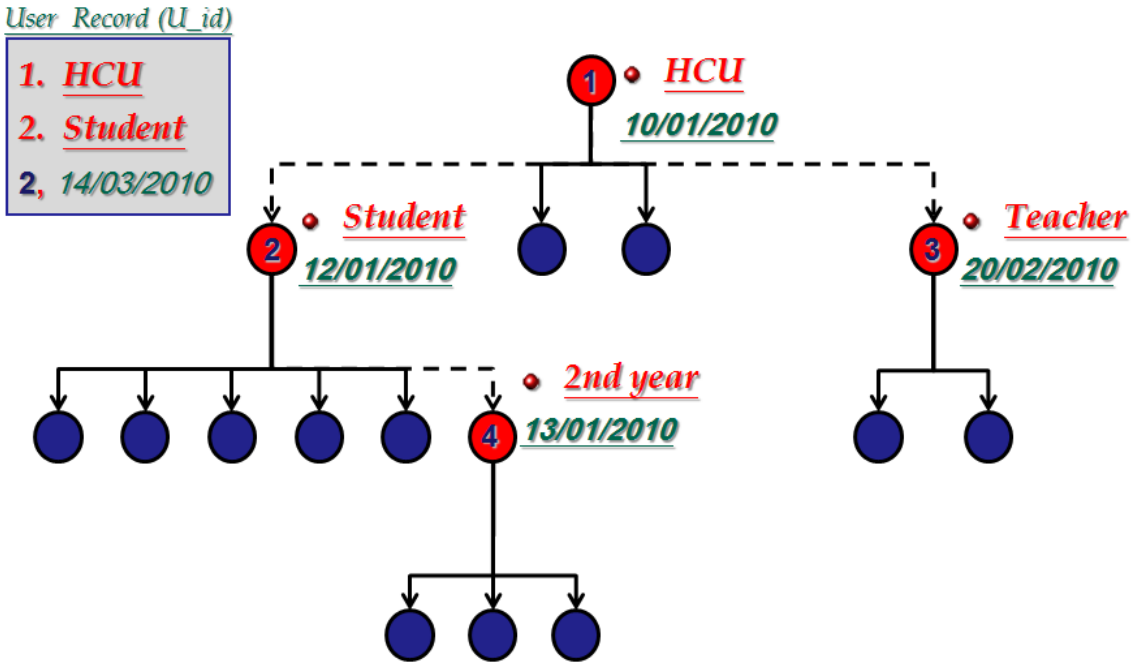

Figure 4.11: Example on Temporal Caching (Pre to the 2nd Request)

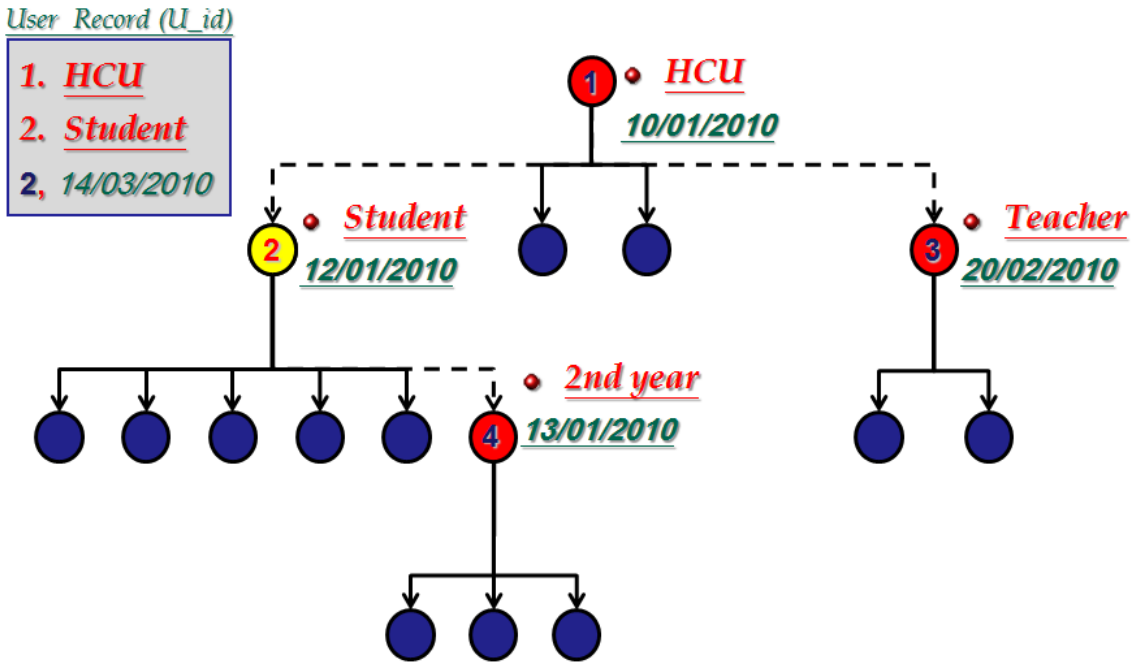

Figure 4.12: Example on Temporal Caching (Pointing to the cached parent node)

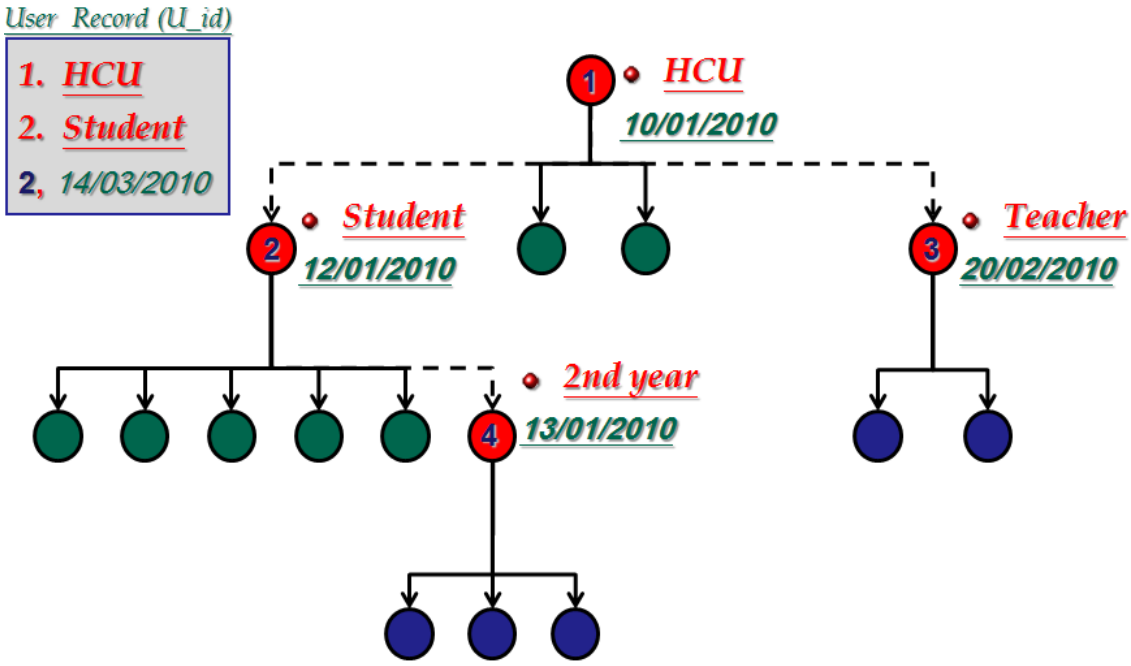

Figure 4.13: Example on Temporal Caching (Up Pass)

Computational Complexity of TCM
Any authorization process can not take more than $O(2^N)$ complexity for the first pass and O(1) complexity for the next pass as far as there is no change.

It is important to mention that the efficiency of this caching mechanism depends on how frequently the user's roles or the resources security policies change. Because, whenever the user's roles list is changed, it releases the Parent Nodes List field (which is allocated for the previous roles list) and reparse the decision tree again for the incoming requests. In the same way if the security policy of one parent node in the Parent Nodes List changes, then the timestamp will not match the user's record timestamp, and it requires to reparse the tree again. In the next section, we address the problem of user's frequently changeable roles.

4.2.1.1 User's Frequently Changeable Roles

In some grid environments, user's roles are frequently changed. The temporal caching mechanism is efficient only if it distinguishes between the user's frequently changeable roles and the user's static roles. As an example, being a teacher in an XYZ university can be considered as a static role, but being an administrative in charge is a changeable role. For these two types of roles, the system has to parse the decision tree in two steps:

- First, it parses the tree for the static roles producing the pair (Timestamp1, Parent Node List1).
- Then, it continues parsing the tree for the remaining changeable roles producing the pair (Timestamp2, Parent Node List2).

Thus the user's record has to save 2 Parent Nodes Lists along with 2 timestamps as shown in Figure 4.14.

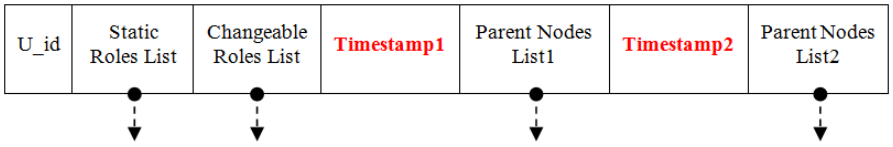

Figure 4.14: User's Record Structure

If the timestamp of the whole user's roles does not match with the timestamp of the corresponding parent nodes, it directly points back to the parent nodes that corresponds to the user's static roles and checks the timestamp again. If it matches, the system continues parsing the tree from that parent node only. Figures 4.15, 4.16, 4.17, 4.18 and 4.19 shows an example of this process (The blue color roles are the user's frequently changeable roles).

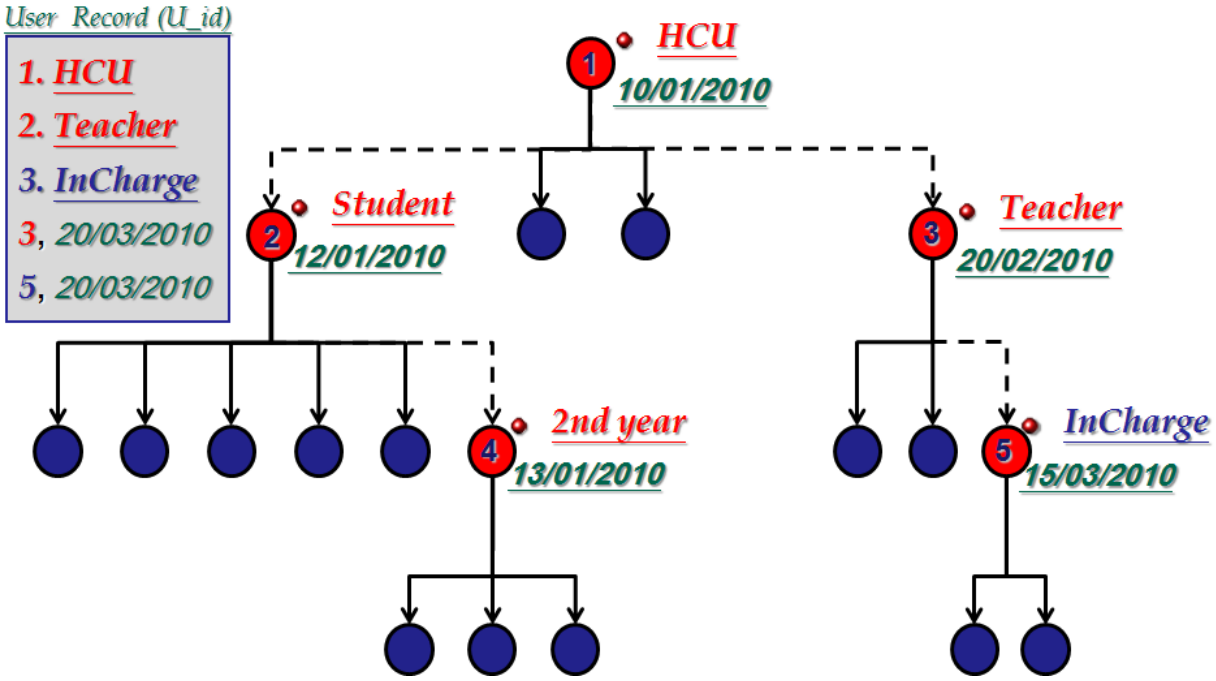

Figure 4.15: Example on User's frequently changeable roles (Initial State)

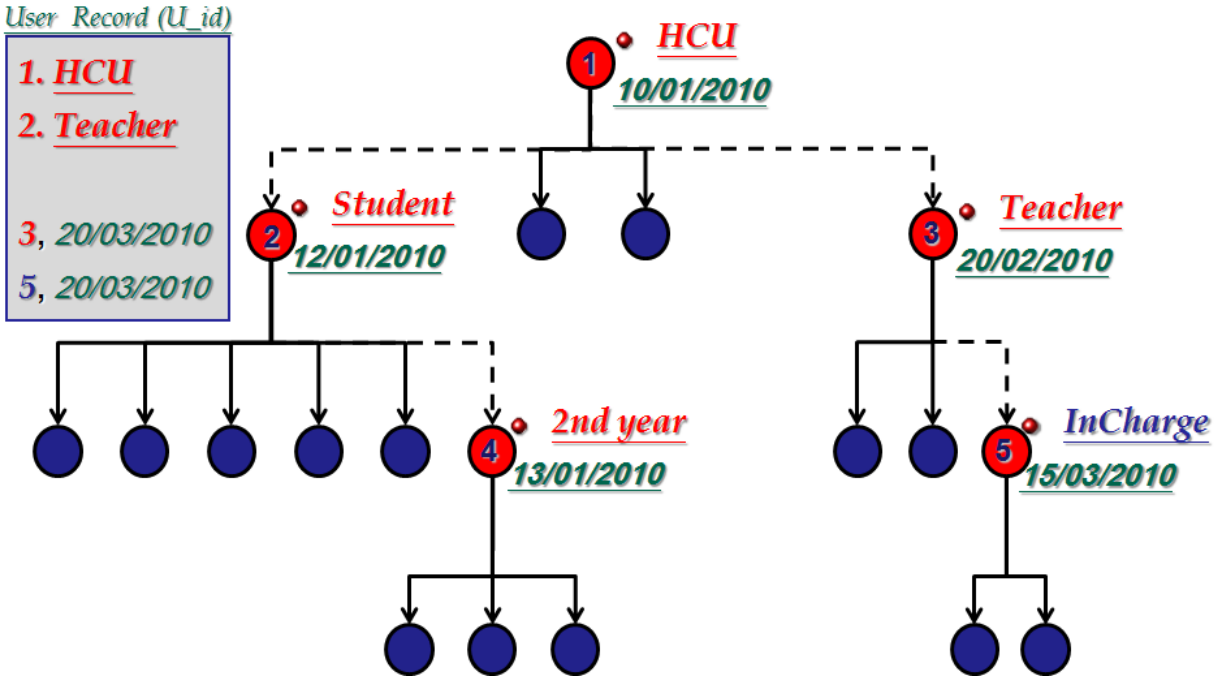

Figure 4.16: Example on User's frequently changeable roles (The changeable role is revoked)

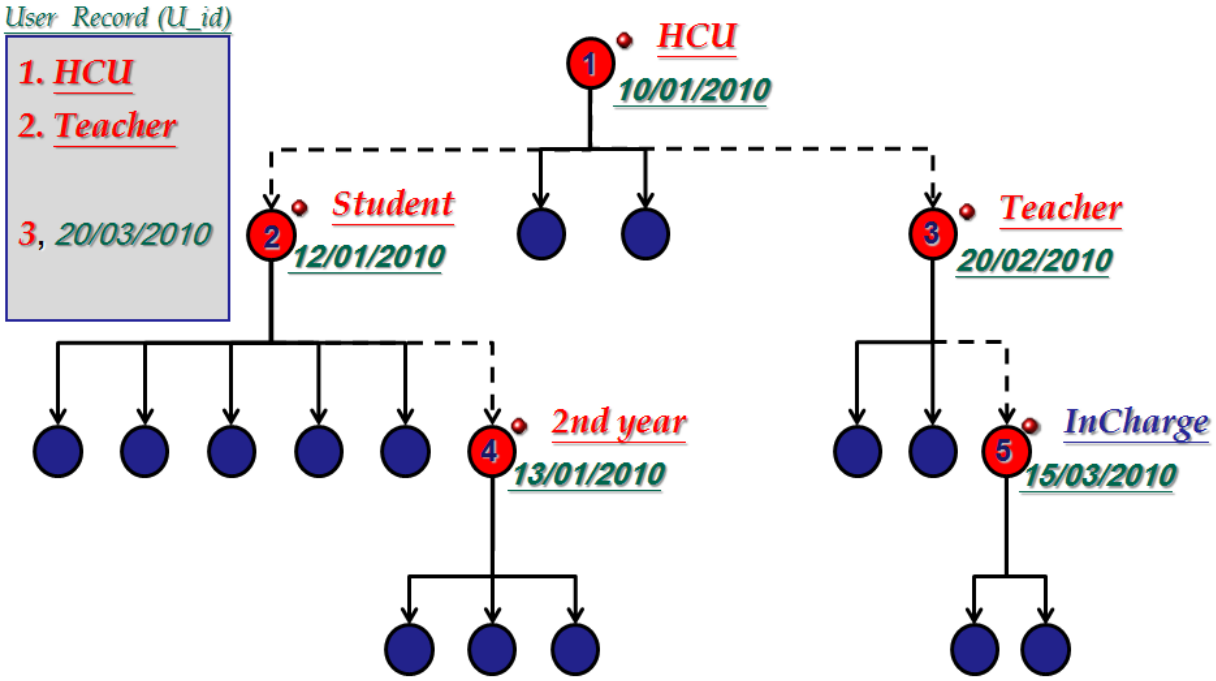

Figure 4.17: Example on User's frequently changeable roles (The corresponding Parent Node List is deleted)

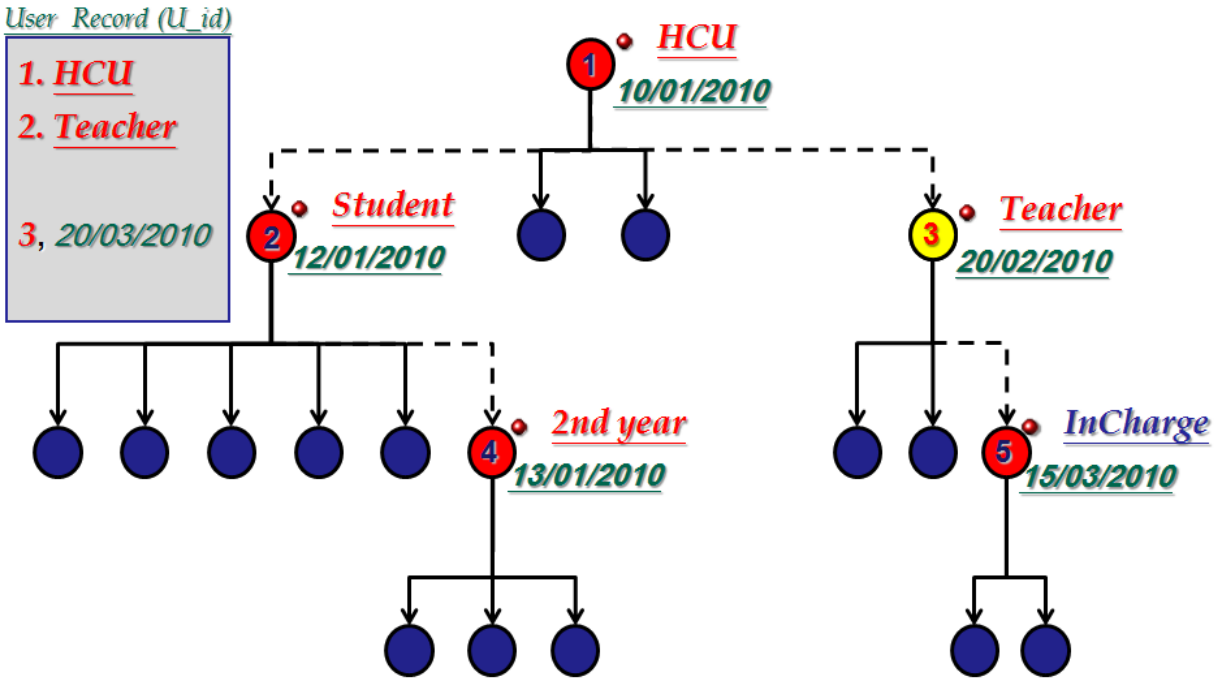

Figure 4.18: Example on User's frequently changeable roles (Using the static roles' Parent Node List)

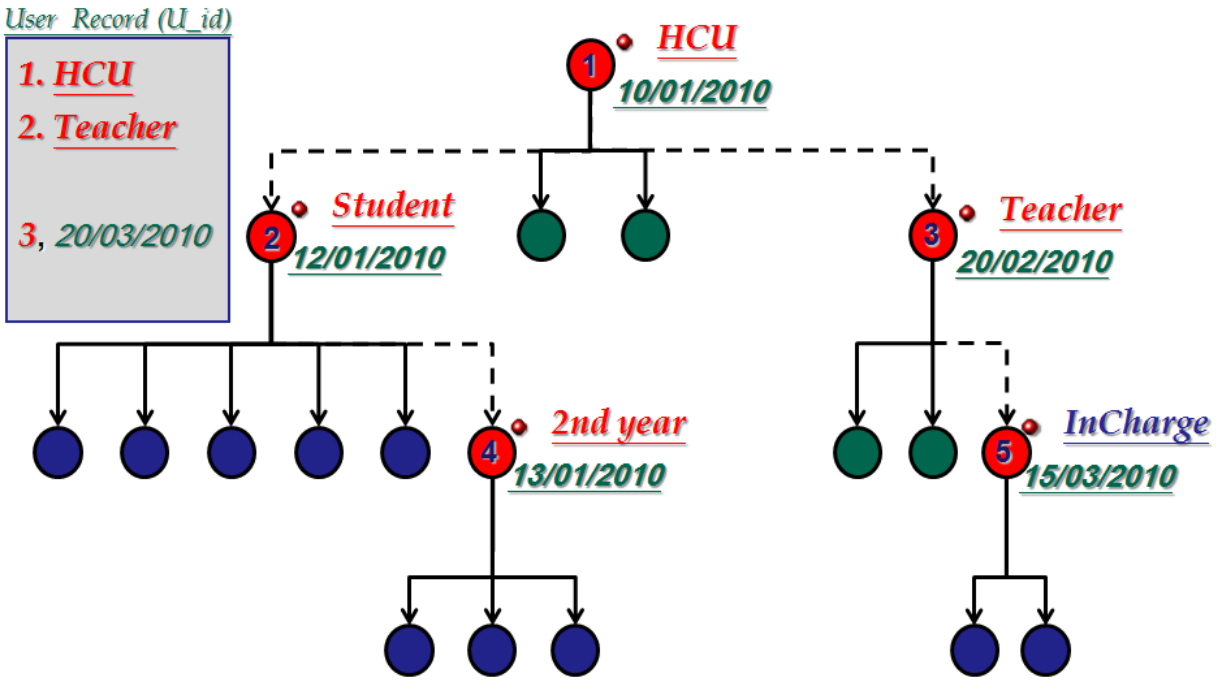

Figure 4.19: Example on User's frequently changeable roles (Allotting the resources to the user without reparsing the tree)

4.2.2 Hamming Distance Caching Mechanism (HDCM)

In the real scenario, it is observed that the user's roles are not completely changed. The change is only for adding or deleting few roles, so the new allocated parent nodes are very close in the tree to the old one (either it will be one more higher level, if an old role has been revoked from the user, or one more lower level if a new role is granted to the user). Considering this fact, the current authorized parent nodes group can be derived from the previous one as shown in Algorithm 7:

Table 4.12: One parent node Security Policy Vector

N_id	Role 1	Role 2	. . .	Role n
150	1	0	. . .	1

Algorithm 7 The Hamming Distance Caching Mechanism (HDCM)

Input:

HCM Decision Tree & User Record

Output:

UARG

Predefined Functions:

The domination relationship ***dom*** between two vectors is defined as follows: V1 *dom* V2 $\Leftrightarrow$ V1 $\wedge$ V2 = V2

Begin

- Each parent node's security policy is represented as a vector SPV of zeros and ones (Table 4.12). URV(t) is the user's roles vector at t point of time.
- For the first authorization, the decision tree will be parsed normally. Then the authorized parent nodes will be saved in the Parent Node List of the user record (Figures 4.20, 4.21, 4.22 and 4.23).
- For the second authorization, the system computes the hamming distances between the user's roles vector URV at that time and the SPV of each parent node in the Parent Nodes List.
- If the hamming distances are all zeros, this indicates that there was no change. Therefore the new Parent Nodes List is the same as the old one (Figures 4.24 and 4.25). Otherwise, we have two cases:
  - **Case 1**: If (SPV *dom* URV for any parent node in the Parent Nodes List) then move one level up in the decision tree, starting from that particular parent node, until you find the parent node for which (URV *dom* SPV). Then this node replaces the old parent node in the Parent Nodes List (Figures 4.26, 4.27, 4.28 and 4.29).
  - **Case 2**: If (URV *dom* SPV) then do the same process in reverse direction (move one level down).

End

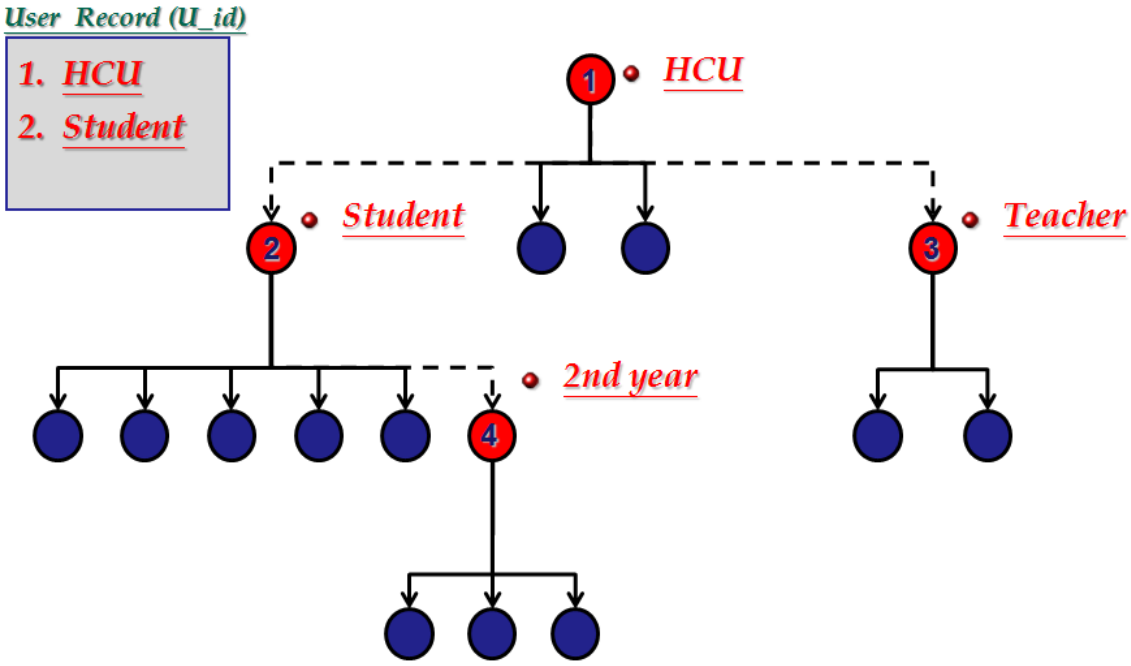

Figure 4.20: Example on the Hamming Distance Caching Mechanism (Initial State)

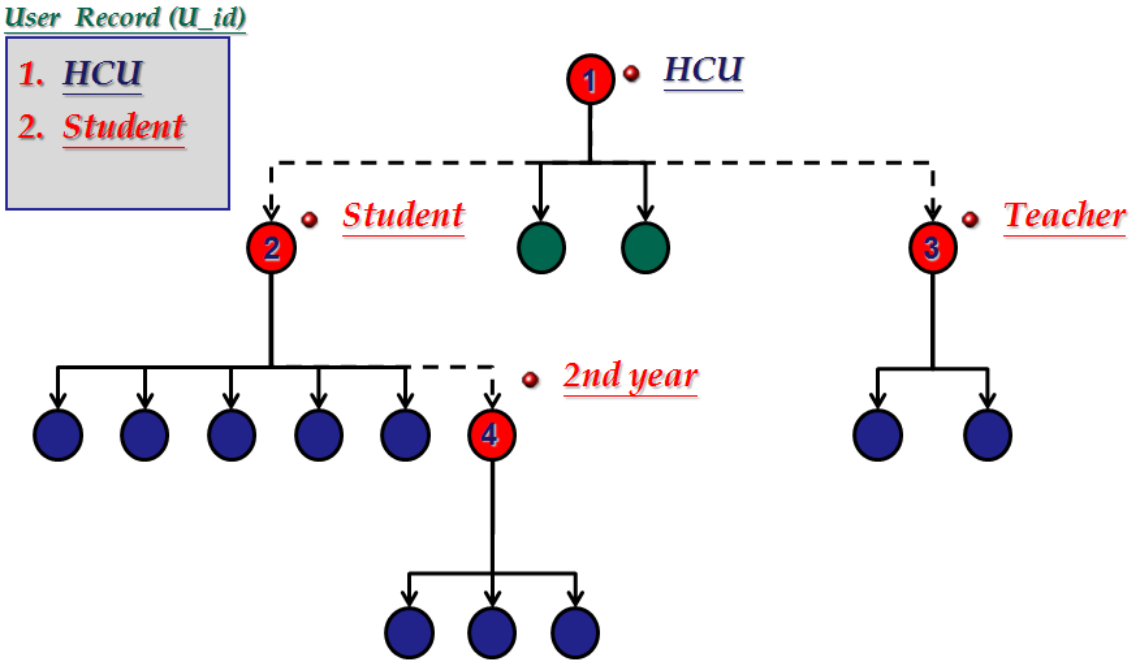

Figure 4.21: Example on the Hamming Distance Caching Mechanism (Parsing the Tree)

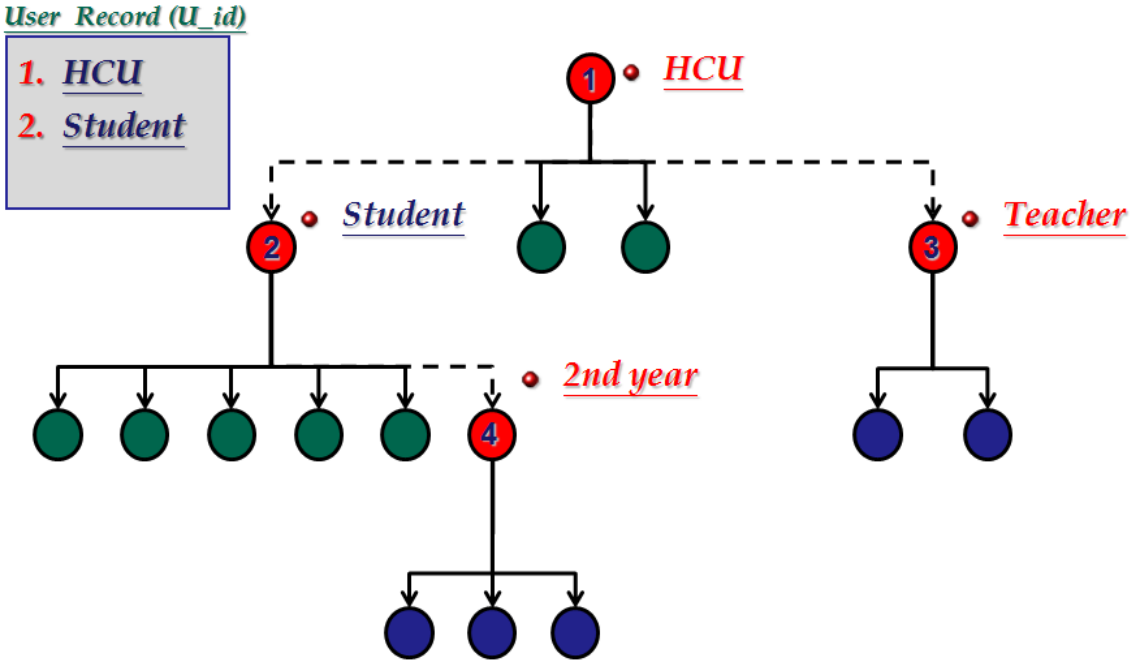

Figure 4.22: Example on the Hamming Distance Caching Mechanism (Parsing the Tree)

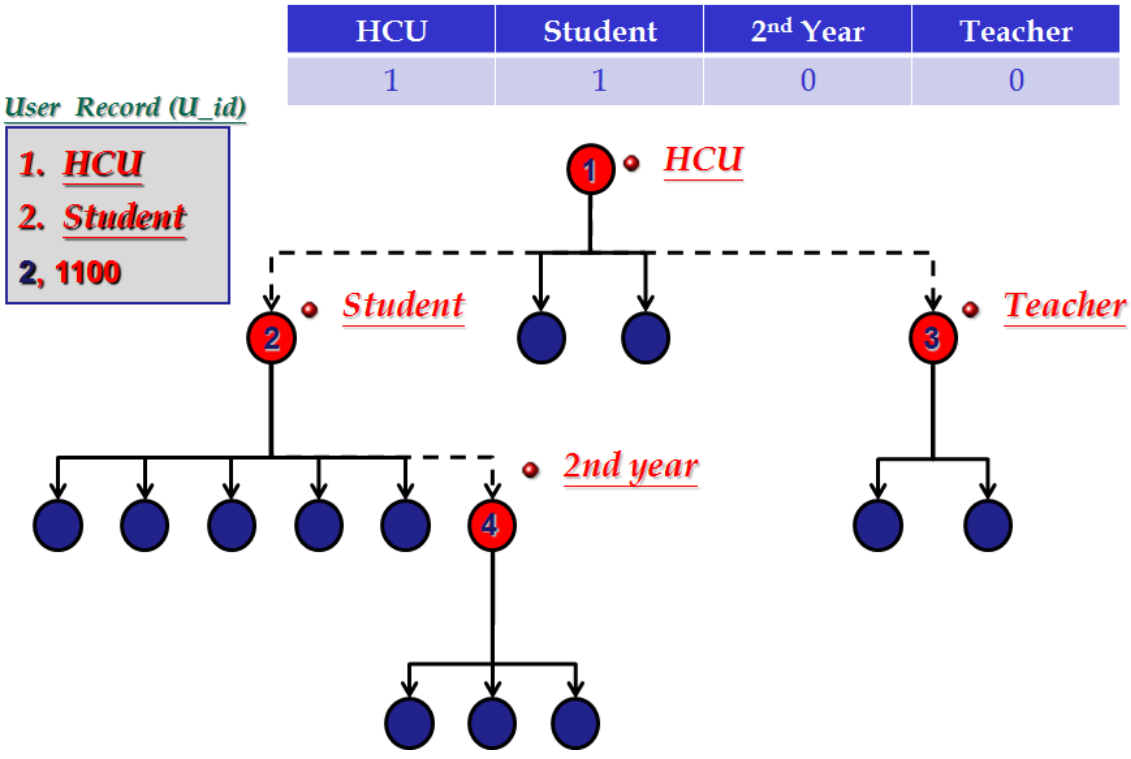

Figure 4.23: Example on the Hamming Distance Caching Mechanism (Caching)

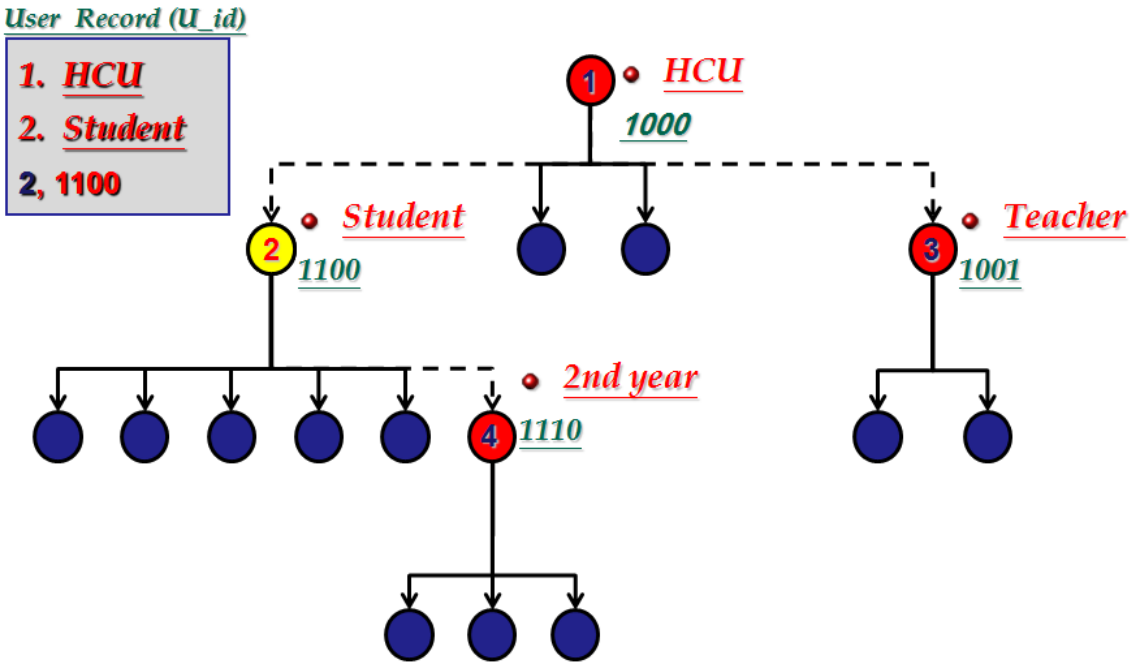

Figure 4.24: Example on the Hamming Distance Caching Mechanism (Comparing URV and SPV)

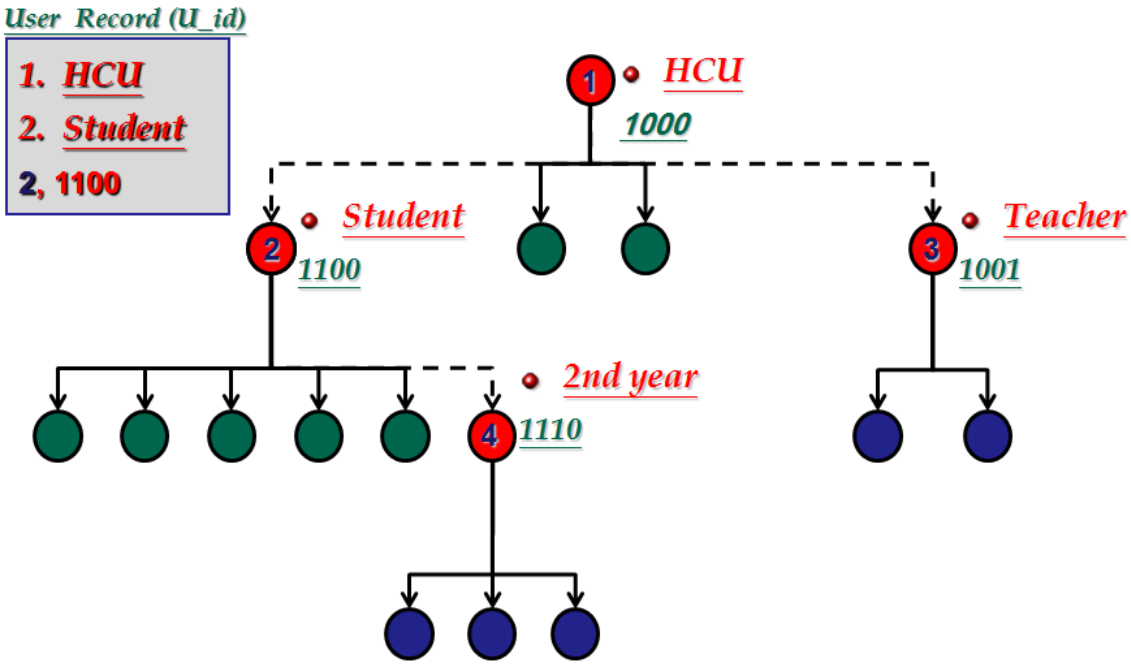

Figure 4.25: Example on the Hamming Distance Caching Mechanism (Getting the UARG)

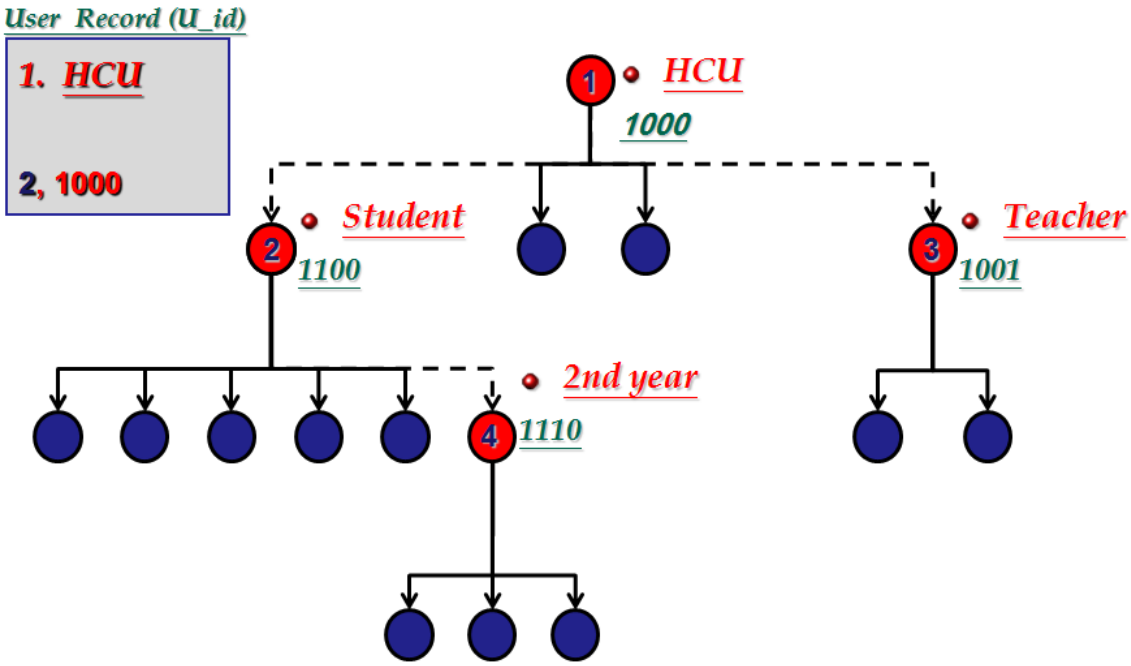

Figure 4.26: Example on the Hamming Distance Caching Mechanism (One Role has been revoked)

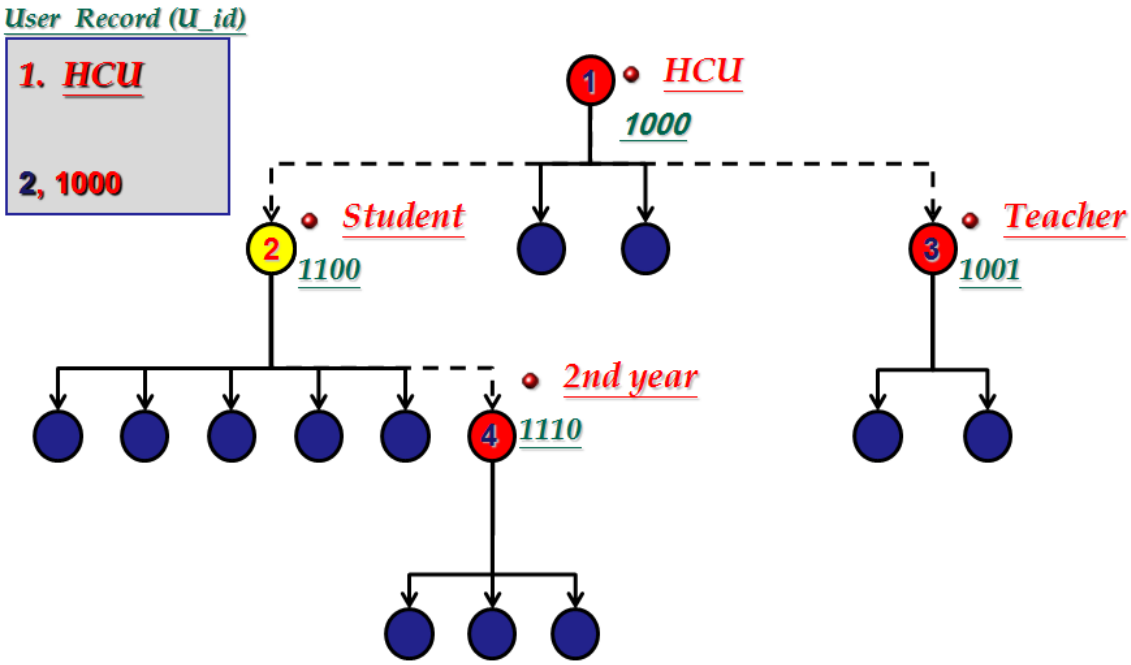

Figure 4.27: Example on the Hamming Distance Caching Mechanism (Comparing URV and SPV, SPV *dom* URV)

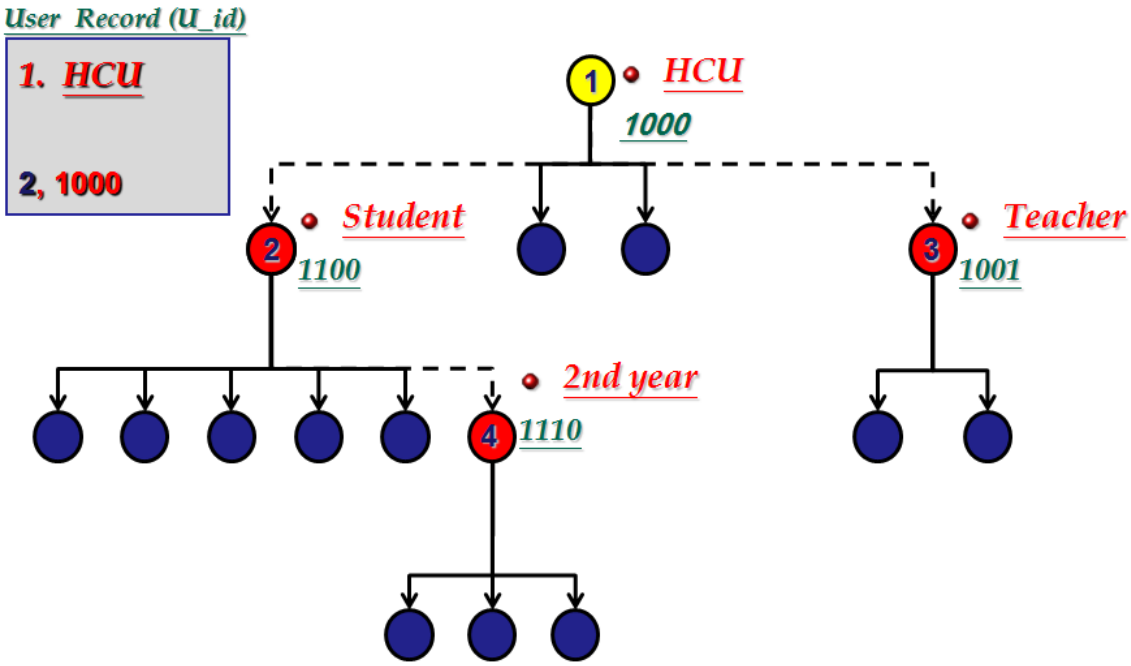

Figure 4.28: Example on the Hamming Distance Caching Mechanism (Go one level up till the hamming distance is ZERO)

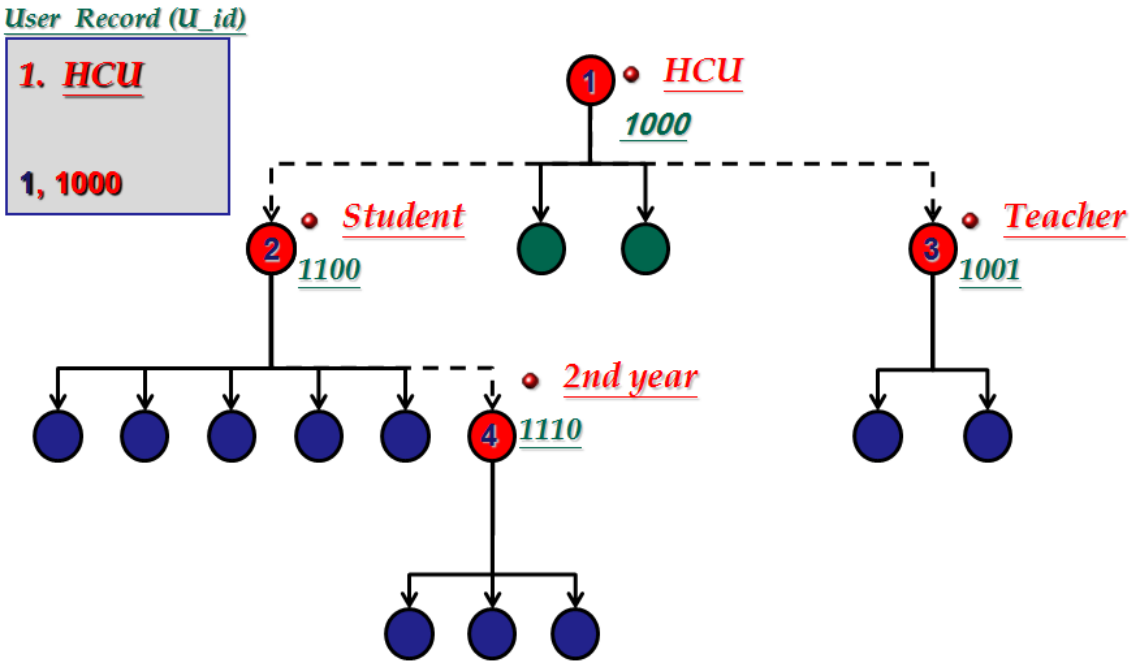

Figure 4.29: Example on the Hamming Distance Caching Mechanism (Derive the new UARG)

Computational Complexity of HDCM
Any authorization process can not take more than $O(2^N)$ complexity for the first pass and expected O(1) complexity for the next pass even if there is a change.

This caching mechanism (HDCM) can use the same data structures of the parent node, and the user's record shown in Figure 4.5, and Figure 4.6 respectively without the timestamp field.

4.2.3 The Concurrent Model of HCM

By default, HCM fits in an environment which has two properties:

1. The ability to access a centralized authorization database, where all security policies have been saved, through which HCM can build its decision tree.

2. Using a push model sequence of authorization where the subject (user) has to first request an authorization from the authority before he/she can communicate with the target resource.

Considering the huge number of users and resources which may exist in the grid, using HCM may cause a bottleneck to the authorization system as a centralized agent targeted by all authorization requests. The immediate and direct solution to this problem is by replicating the *decision tree* into several authorization servers to reduce the load on the centralized agent. However, one can think of enhancing the HCM mechanism itself to process multiple authorization requests simultaneously . If this enhancement is not sufficient to avoid the bottleneck in a very huge grid, then one can think of replicating HCM.

4.2.3.1 Processing Multiple Authorization Requests Simultaneously

This section explains how the HCM decision tree can be processed in parallel to serve multiple authorization requests at a time.

The authorization server boots with a head process named "**Welcoming Process**" (See Figure 4.30). This process is responsible for capturing the authorization requests and forking a new authorization process for every request to be served concurrently. Code 1 represents an outline C pseudo code of typical authorization server that uses concurrent HCM. When a new request comes, "**accept**" method returns. Then the server calls "**fork**" to create a child process named "**Authorization Process**". The child process (the Authorization Process) serves the incoming request while the parent process (the Welcoming Process) waits for another request to come.

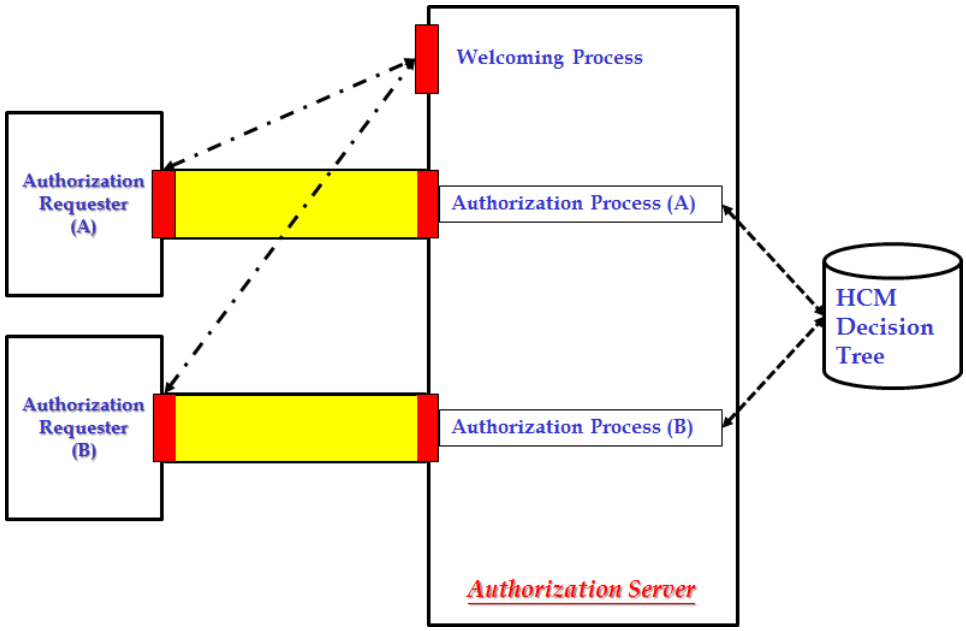

Figure 4.30: Processing Multiple Authorization Requests Simultaneously

Code 1 Outline C pseudo code for typical HCM authorization server.

```
pid_t pid;
int   welProfd, authProfd;

/* Fill with the Welcoming
welProfd = socket( ... );
bind(welProfd, ... );
listen(welProfd, 1);

while(true) {
  authProfd = accept (welProfd, ... );
  if( (pid = fork()) == 0) {

/*  The child process closes the welcoming process' socket */
    close(welProfd);

/*  Process the request & return the user's authorized resource group */
    UARG rg = auhtorize(authProfd);

    close(authProfd); /* Done with this authorization request */
    exit(0);          /* Child terminates */
  }
}
```

Creating a new process for every authorization request might be expensive in terms of memory consumption as every process has its own separated segments in memory. Thus the use of "**clone**" rather than "**fork**" is strongly recommended as the first one creates a "**light weight**" process which shares parts of its execution context with the parent process, such as the memory space.

4.2.3.2 Parallelizing a Single Authorization Request

This section shows how the "**Authorization Process**" itself can be parallelized for maximum utilization of the authorization server' CPUs which leads to speed up the response time.

The main thread of the authorization process starts at the root node of the decision tree. Then for each branch of the root node, a new thread is created to parse that particular branch. When a thread encounters an unfulfilled security rule, it stops and terminated. Code 2 shows an outline C pseudo code for a typical authorization process on the HCM decision tree.

It might not be practical to create a new thread for each branch of the HCM decision tree if it has a lot of branches. That may cause too much *context-switching* overhead and too much overhead in the scheduler. Thus one can use the "MAXN-THREADS" parameter to control the maximum number of threads created by a single authorization process. Like that, if the HCM decision tree has a lot of branches, then only the high level nodes of the decision tree will be associated with new threads. After that, no more threads are created, and each of those branches will be handled by one thread in a serialized manner.

One more advantage in parallelizing the authorization process is that it allows what we can call as "**Authorization on the fly**" where a thread can make its associated resources directly available for the scheduler once the user is authorized without waiting to complete parsing the whole tree.

Code 2 Outline C pseudo code for typical HCM authorization process.

```
/* Obtain the credentials of the user who has issued the
authorization request. */
Credentials getUserCredentials(authProfd);

/* Verify whether a user satisfies the security rule of a TreeNode.
It is assumed that ‘‘getUserCredentials’’ and ‘‘verify’’ are
predefined functions in one library. */
bool verify(Credentials userCred, TreeNode tn);

/* Expected maximum number of threads */
#define MAXNTHREADS 100

/* The User’s Authorized Resource Group (UARG) variable is shared
among all threads. As all of them has to alter this variable in order
to add resources from different branches of the decision tree. Thus a
mutex variable is required. */
struct {
  pthread_mutex_t mutex;
  UARG rg = NULL;
  int nthreads = 0;
} mrg = {PTHREAD_MUTEX_INITIALIZER};

/* Thread start routine’ parameters. */
typedef struct {
  TreeNode * tn;
  Credentials userCred;
} thread_arg_t;

/* Prototyping the thread’s start routine */
void *check(void * arg);
```

```
/* This is the main body of the authorization process */
UARG authorize(int authProfd) {

  Credentials uc = getUserCredentials(authProfd);

  thread_arg_t * arg = malloc(sizeof(thread_arg_t));
  arg->tn = root;
  arg->userCred = uc;
  pthread_t tid[MAXNTHREADS];

  pthread_create(&tid[mrg.nthreads], 0, check, arg);

  for (int i=0; i< mrg.nthreads; i++) {
    pthread_join(tid[i], NULL);
  }

  exit(0);
}

/* Assume that childResourceNode and childSecurityRuleNode are
predefined data type in one library like TreeNode and UARG data
types. */
void * check(void * arg) {
  thread_arg_t * tArg = arg;
  if(verify(tArg->userCred, tArg->tn)) {

/*  Add all child resources to UARG safely (thread-safe) */
    pthread_mutex_lock(&mrg.mutex);
    foreach(childResourceNode r of tArg->tn)
      mrg.rg.insert(r);
    pthread_mutex_unlock(&mrg.mutex);
```

```
/*  Create new threads to process every branch of the current node */
    foreach(childSecurityRuleNode sr of tArg->tn) {
      pthread_mutex_lock(&mrg.mutex);
      mrg.nthreads ++;
      pthread_mutex_unlock(&mrg.mutex);
      thread_arg_t * nArg = malloc(sizeof(thread_arg_t));
      nArg->tn = sr;
      nArg->userCred = tArg->userCred;
      pthread_create(&tid[mrg.nthreads], 0, check, nArg);
    }
  }
}
```

4.2.4 Single Resource Authorization Problem

HCM was mainly designed to find the **SET** of resources which the user is authorized to access with the least redundancy in checking security rules. However, the user may intend to use a particular resource that means HCM spends more time and complicates the process of authorization in parsing the whole decision tree, when checking a **SINGLE** resource security policy itself is sufficient. This section addresses this issue and provides efficient solutions for it.

The problem of **SINGLE** resource authorization can be solved by mainly two methods:

4.2.4.1 Implementing Hybrid Authorization Mechanism

The immediate improvement in the solution of the problem can be achieved by allowing the authorization system to use a hybrid authorization mechanism where

it can choose between HCM and a single resource authorization mechanism based on the properties of the incoming request. Figure 4.31 illustrates the hybrid authorization, if the incoming request is looking for a single resource then the Request Dispatcher directs it to the typical Grid-Map file way of authorization. Otherwise, it uses HCM.

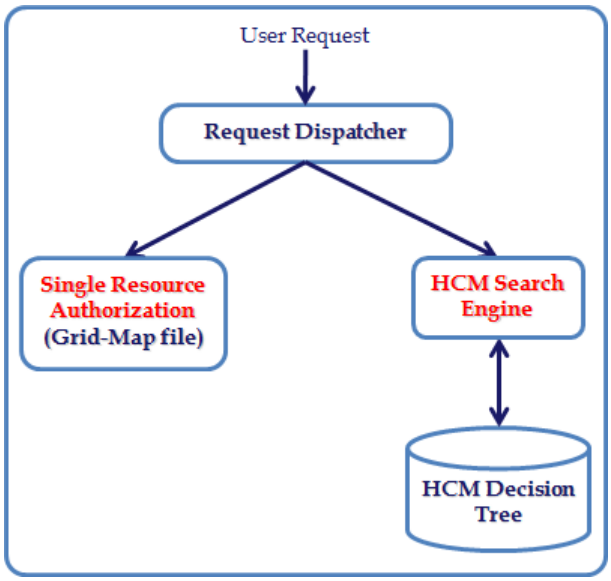

Figure 4.31: Hybrid Authorization

This solves the problem and reduces the overhead caused by HCM when it handles a single resource authorization request. However, the next section shows that HCM can be tuned to handle single resource authorization requests without involving any extra complexity. Moreover it can prepare a cached data to help in future authorization processes.

4.2.4.2 HCM Single Resource Authorization Mode

HCM enables a special authorization mode for authorizing a single resource in such a way that won't parse the entire *decision tree* unnecessarily and will also prepare a cached data that helps to speed up further authorization in the future.

Figure 4.32 shows an example HCM in Single Resource Authorization Mode. Each path from the root node to a resource node in the *decision tree* represents the security policy of that resource. As an example, the security policy of resource R_4 is represented by the following formula: $sr_1 \wedge sr_2 \wedge sr_3$. So, if the user satisfies this formula then he/she is authorized to access R_4.

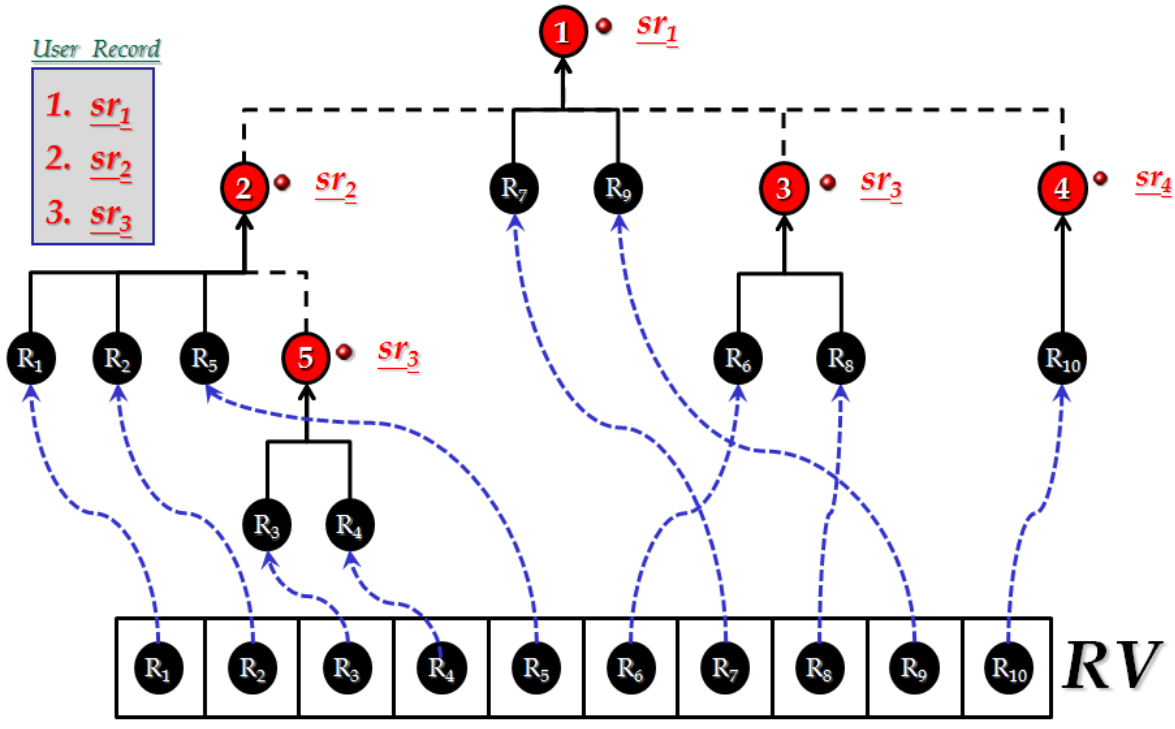

Figure 4.32: Example of HCM Single Resource Authorization Mode

A Resource Vector (RV) is used to map each resource to its correspondent node in the *decision tree*. Once an authorization request for a particular resource is fired, the *decision tree* is parsed bottom-top way starting from the resource node in RV up to the root node of the *decision tree*. Like this, only the security rules which exist in the resource's security policy are going to be checked. Moreover, every resource which exists in the same level of the required resource or in the same level of a satisfied security rule exists in the path to the root node will be marked with the same authorization decision as of the required resource. This helps to speed up further authorization requests issued by the same user targeting any of these resources.

Figures 4.33, 4.34, 4.35, 4.36, 4.37, 4.38, 4.39, 4.40 and 4.41 show an example *decision tree* parsed in a single resource authorization mode for user X who is requesting to access R_4 only, and whose credentials satisfy the security rules sr_1, sr_2, and sr_3. As user X is authorized to access R_4, all the resources {R_1, R_2, R_3, R_5, R_7 and R_9} which exist in the same level of R_4 or in the same level of any satisfied security rule placed on the path from R_4 to the root node will also be marked as authorized.

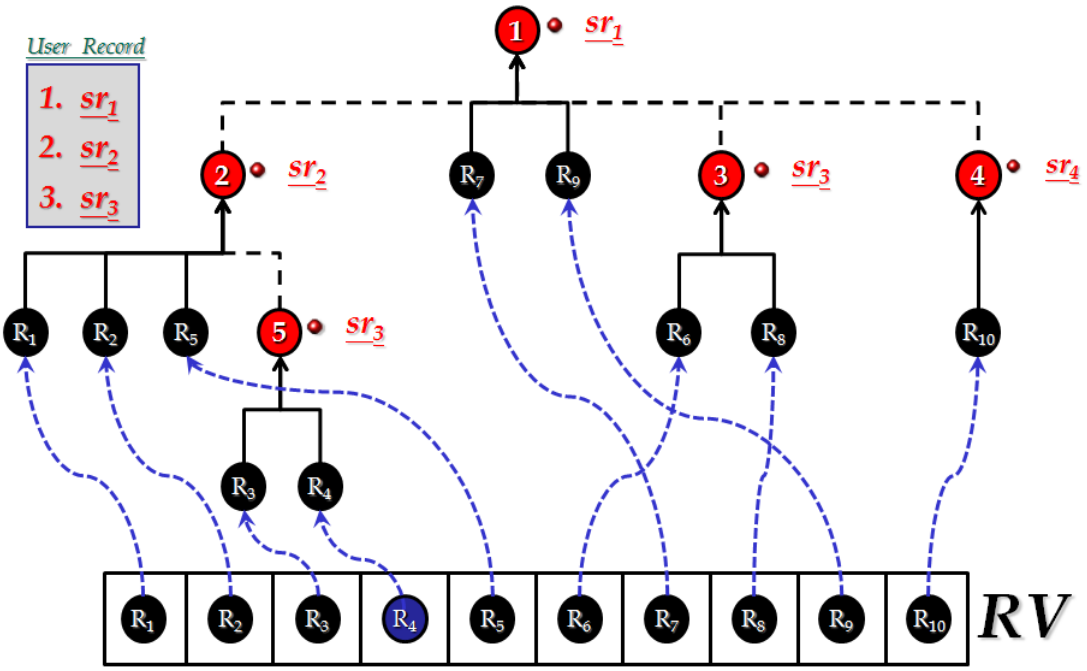

Figure 4.33: Parsing HCM in Single Resource Authorization Mode (User requested R_4)

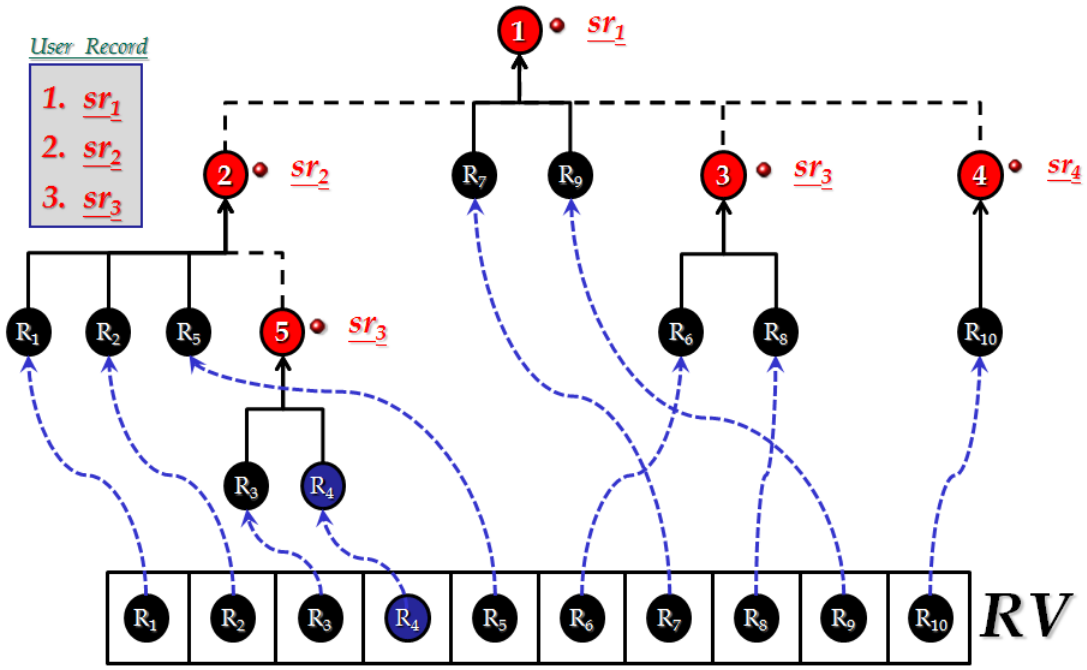

Figure 4.34: Parsing HCM in Single Resource Authorization Mode (Point to the corresponding R_4 in the decision tree)

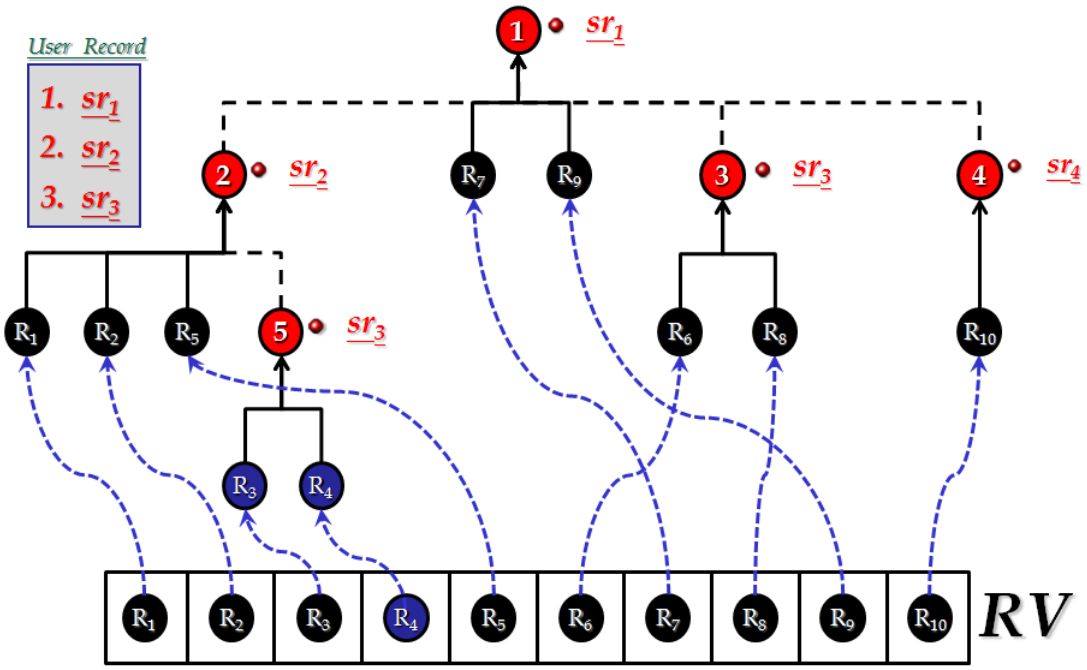

Figure 4.35: Parsing HCM in Single Resource Authorization Mode (Mark R_3 to have the same authorization decision as R_4)

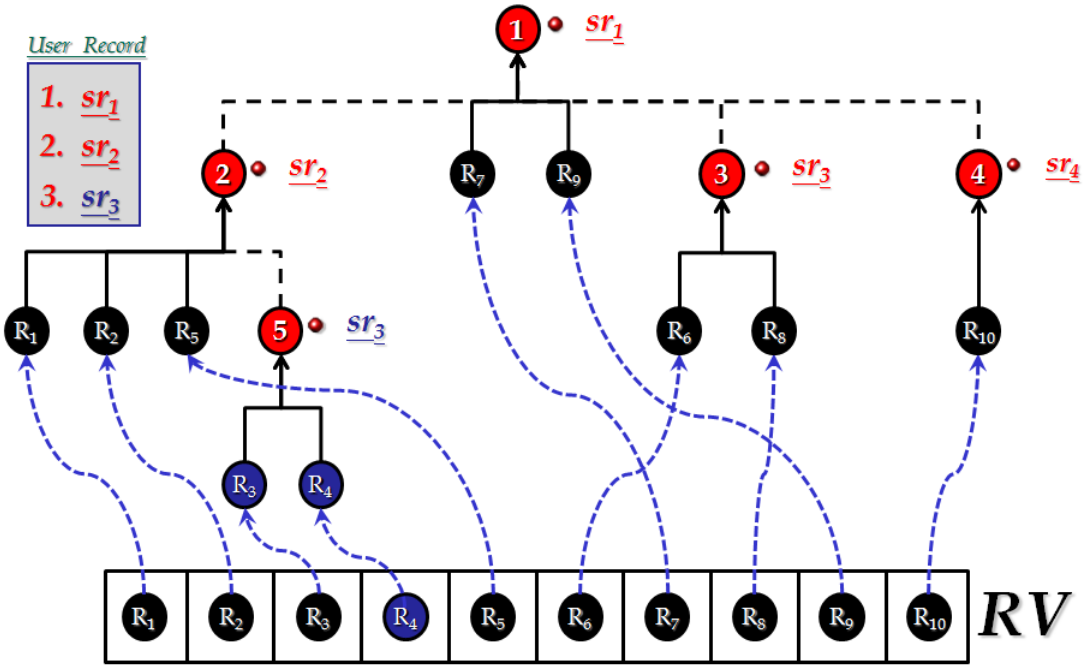

Figure 4.36: Parsing HCM in Single Resource Authorization Mode (Go up and check sr_3)

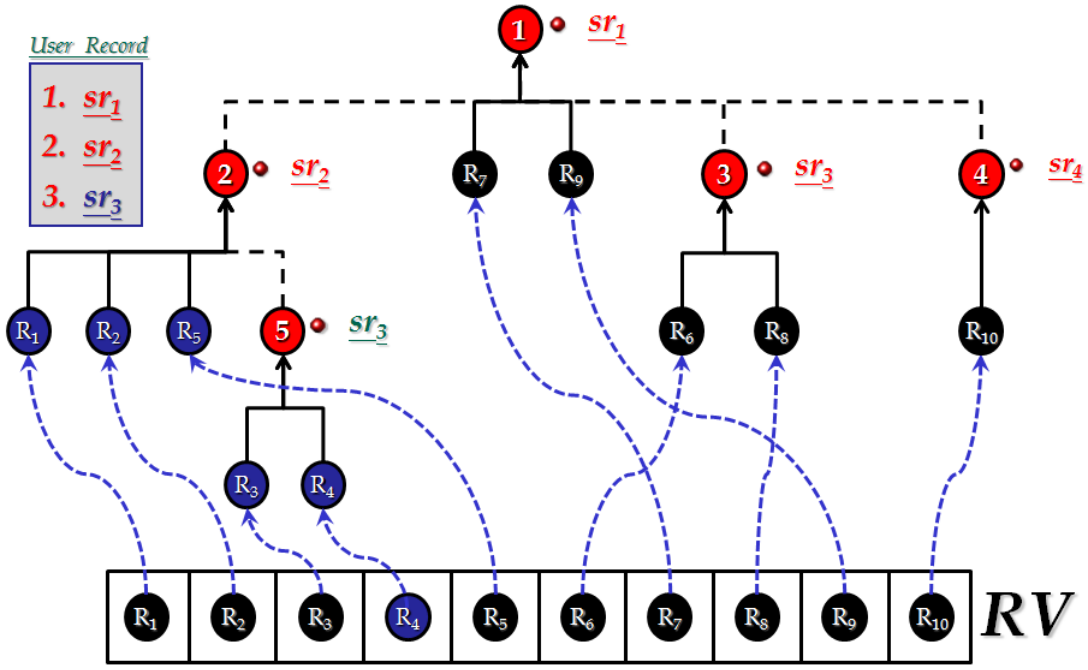

Figure 4.37: Parsing HCM in Single Resource Authorization Mode (As sr_3 is satisfied, Mark R_1, R_2 and R_5 to have the same authorization decision)

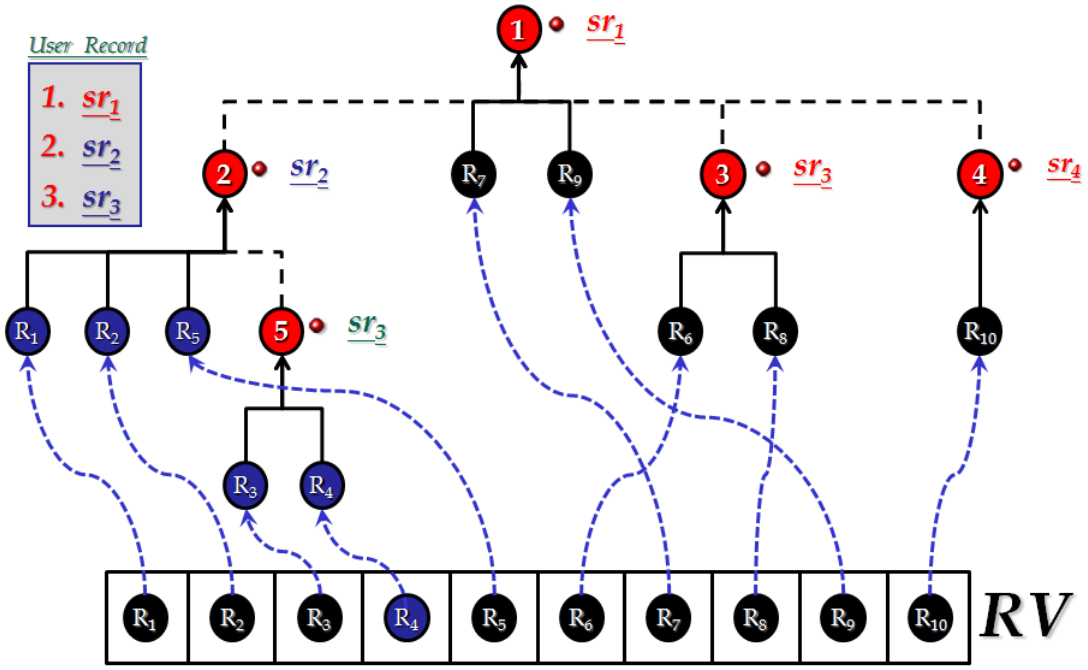

Figure 4.38: Parsing HCM in Single Resource Authorization Mode (Go up and check sr_2)

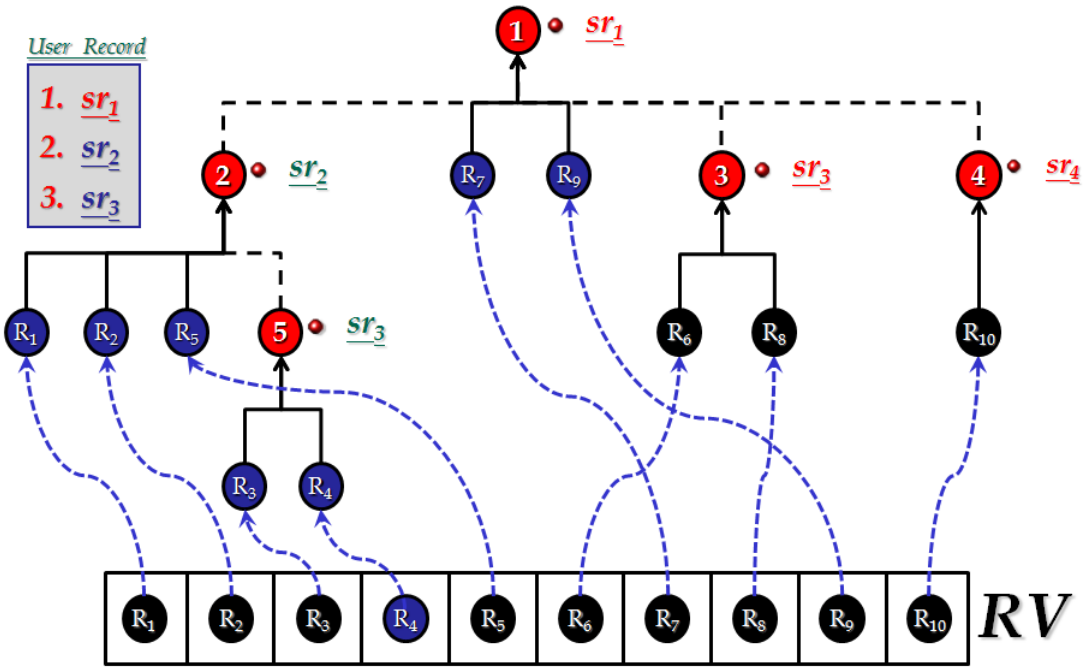

Figure 4.39: Parsing HCM in Single Resource Authorization Mode (As sr_2 is satisfied, Mark R_7 and R_9 to have the same authorization decision)

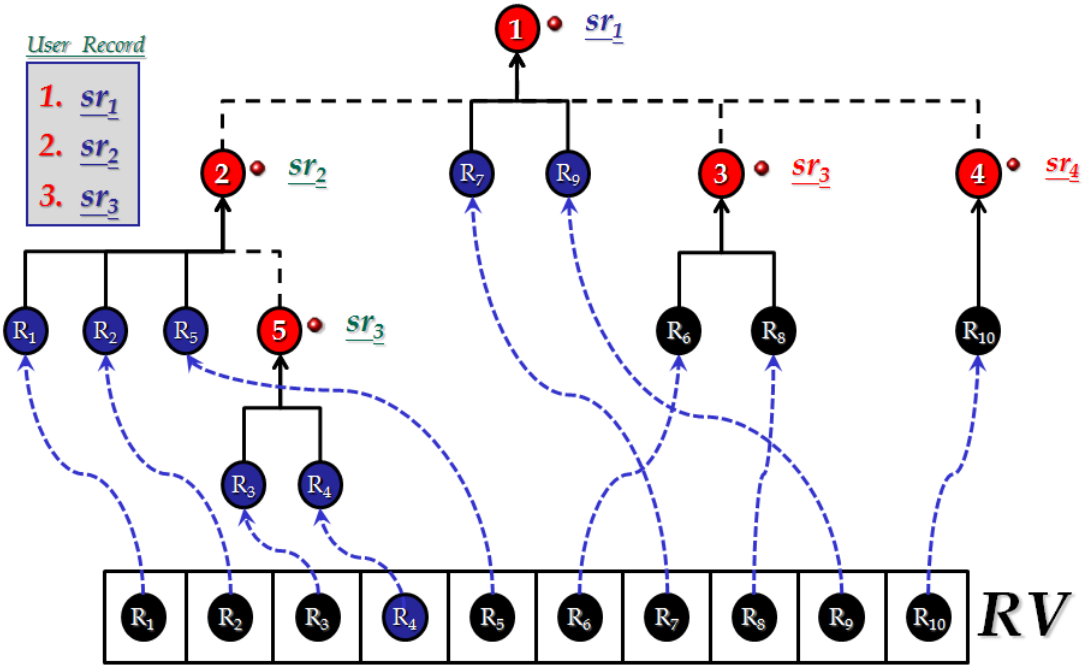

Figure 4.40: Parsing HCM in Single Resource Authorization Mode (Go up and check sr_1)

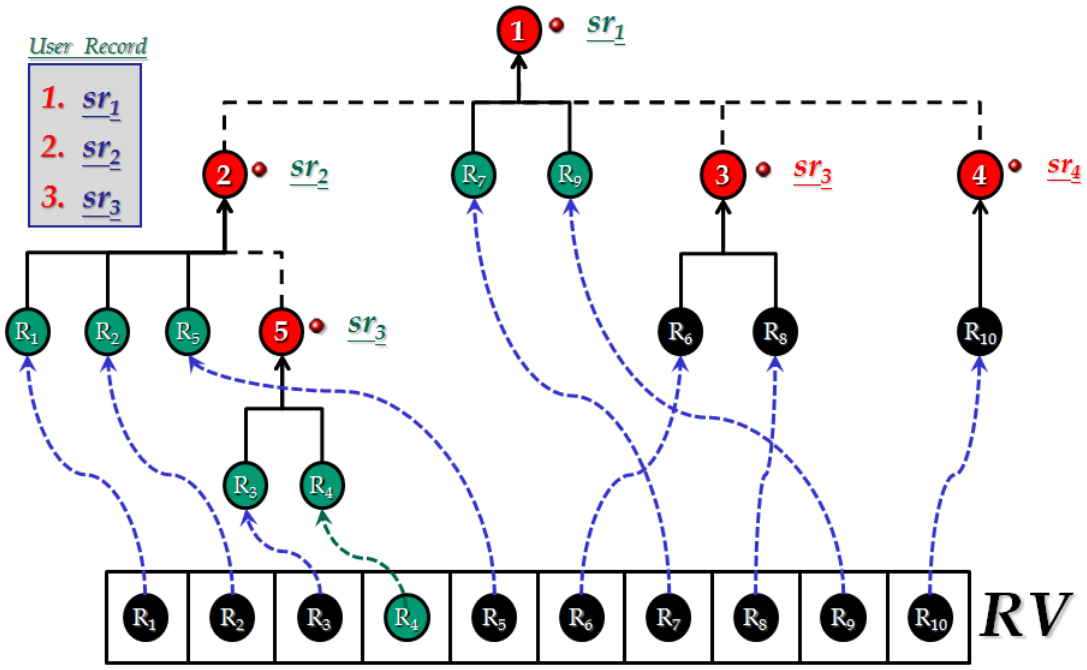

Figure 4.41: Parsing HCM in Single Resource Authorization Mode (As sr_1 is satisfied and it is the root node, then set the authorization decision to `GRANTED`)

Like this, the system is able to prepare a set of resources which the user is authorized to access while it is parsing only a single resource security policy rather than parsing the whole *decision tree*. This set of resources, named UARS, does not represent the whole set of the UARG but a subset of it (ex: user X is also authorized to access resources R_6 and R_8, but they do not belong to UARS).

In the future, if user X asks to access resource R_2 as an example, the set UARS is first checked. As R_2 belongs to it, then no further authorization test is required. However, if the user asks for a resource which does not belong to UARS, then a new single resource authorization process has to be triggered for that particular resource.

4.3 Summary

In this chapter, various aspects of HCM scalability issues have been introduced. The Rough Set based PCM is proposed as an alternative to HCM which suits better for less dynamic, small or medium size grid environments. Normal PCM tests all the rows of the *Resources Clusters Table*. After applying the Rough Set theory, PCM tests only those rows whose *reduct* attributes is *dominated* by the *reduct* attributes of the user's *Privileges Vector*. If N is the number of resources, M is the number of security rules and C is the number of *clusters*, then the Rough Set *reduct* attributes are of O $(\log_2(C))$ complexity. When M is greater than $\log_2(C)$, the Rough Set based PCM gives a better throughput. Moreover, PCM here clusters the resources based on the ***reduct*** attributes. This is a fast clustering mechanism which is not yet adopted in normal PCM. A new caching mechanism based on the *flag vector* can be used in the Rough Set based PCM.

This chapter also addressed the stability of HCM against dynamic changes in the Grid. The proposed simulation studies show how the decision tree of the enhanced HCM is stable and usable for a reasonable rate of dynamic modifications.

The Distributed HCM has been introduced as a solution to the Cross-Domain Authorization problem exists in Grid. Three different models of DHCM are proposed.

Novel basic caching algorithms, Temporal Caching Mechanism and Hamming Distance Caching Mechanism are proposed to enhance HCM user scalability.

The concurrent model of HCM has been introduced to enable HCM to scale up to huge number of authorization requests taking place at a time. It also shows that a single authorization process can be parallelized to decrease the response time.

Finally, this chapter discusses the problem of single resource authorization in HCM. It clearly shows that HCM can still be used even in single resource authorization scenarios without adding extra complexity to the authorization process. Moreover, HCM can help to prepare cached data in order to speed up further authorization in the future.

Thus HCM now, is shown an efficient access control mechanism which cauld be integrated in the present popular Grid authorizing systems like VOMS, Akenti, PERMIS, etc.

Chapter 5

Grid Authorization Graph

HCM efficiently reduces the redundancy in checking security rules compared to the Brute Force Approach as well as the Primitive Clustering Mechanism (PCM), still HCM is not totally free of redundancy. Moreover, it cannot easily describe the OR-based security policies. In order to overcome these limitations of HCM, an extension to it "The Grid Authorization Graph" is proposed in this chapter. Comparative studies are made in a simulated environment.

5.1 Unresolved Issues in HCM

This section discusses two issues which HCM does not incorporate:

5.1.1 Describing OR-based Security Policy in HCM

Generally, a grid resource can have multiple ways to access it. For example, consider a grid environment with six resources and four security rules represented by a security table ST shown in Table 5.1. Resource (r_4) has two different ways to access it, that is why it has two rows in the security table. A user can access the

resource (r_4) if he/she is a ***student*** in ***XYZ University*** **OR** a ***programmer*** in ***XYZ Software Company***.

Table 5.1: Security Table Example (Resources Vs Security Rules)

R_{id}	XYZ Univ.	Student	XYZ Soft. Co.	Programmer
r_1	1	0	0	0
r_2	1	0	0	0
r_3	1	1	0	0
r_4	1	1	0	0
r_4	0	0	1	1
r_5	0	0	1	0
r_6	0	0	1	1

One can observe in the decision tree representation (Figure 5.1) of the security table (Table 5.1) that r_4 resource node has to be duplicated to represent its security policy. In general, if a resource has x ways to access it, then the HCM *decision tree* has to repeat its resource node x times, this is why HCM cannot easily describe the OR-based security policies.

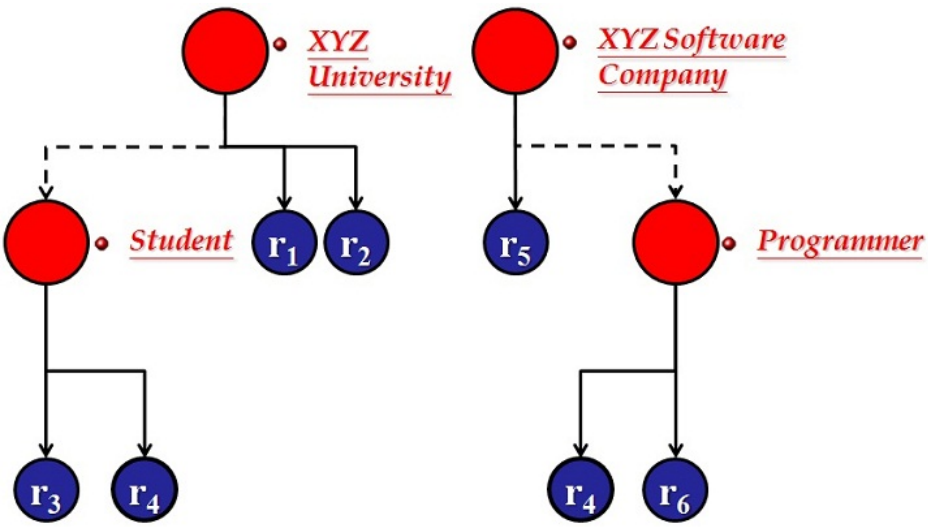

Figure 5.1: Describing OR-based Security Policy in HCM

5.1.2 Redundancy in HCM

Even though HCM reduces the redundancy compared to the *Brute Force Approach* as well as the *Primitive Clustering Mechanism*, it does not entirely eliminate the redundancy. As an example, consider a grid environment with 20 resources and five security rules. Table 5.2 represents the resources' security policies.

Table 5.2: Security Table Example (Resources Vs Security Rules)

R_{id}	sr_1	sr_2	sr_3	sr_4	sr_5
r_1	1	0	0	0	0
r_2	1	0	0	0	0
r_3	1	1	0	0	0
r_4	1	1	0	0	0
r_5	1	1	0	0	0
r_6	1	1	0	0	0
r_7	1	1	0	1	0
r_8	1	1	0	1	0
r_9	1	1	0	0	1
r_{10}	1	1	0	0	1
r_{11}	1	0	1	0	0
r_{12}	1	0	1	0	0
r_{13}	1	0	1	0	0
r_{14}	1	0	1	1	0
r_{15}	1	0	1	1	1
r_{16}	1	0	1	1	1
r_{17}	1	0	0	0	1
r_{18}	1	0	0	0	1
r_{19}	1	0	0	1	1
r_{20}	1	0	0	1	0

Figure 5.2 shows the output *decision tree* when the *Counting Algorithm* (Algorithm 1) runs on Table 5.2. It can be observed that the security rule $\mathbf{sr}_4$ has to be checked four times and $\mathbf{sr}_5$ has to be checked three times. This shows that even though HCM reduces the redundancy compared to the Brute Force Approach as well as the PCM, it does not completely eliminate the redundancy. This motivated us to propose the Grid Authorization Graph, which is discussed in the next section.

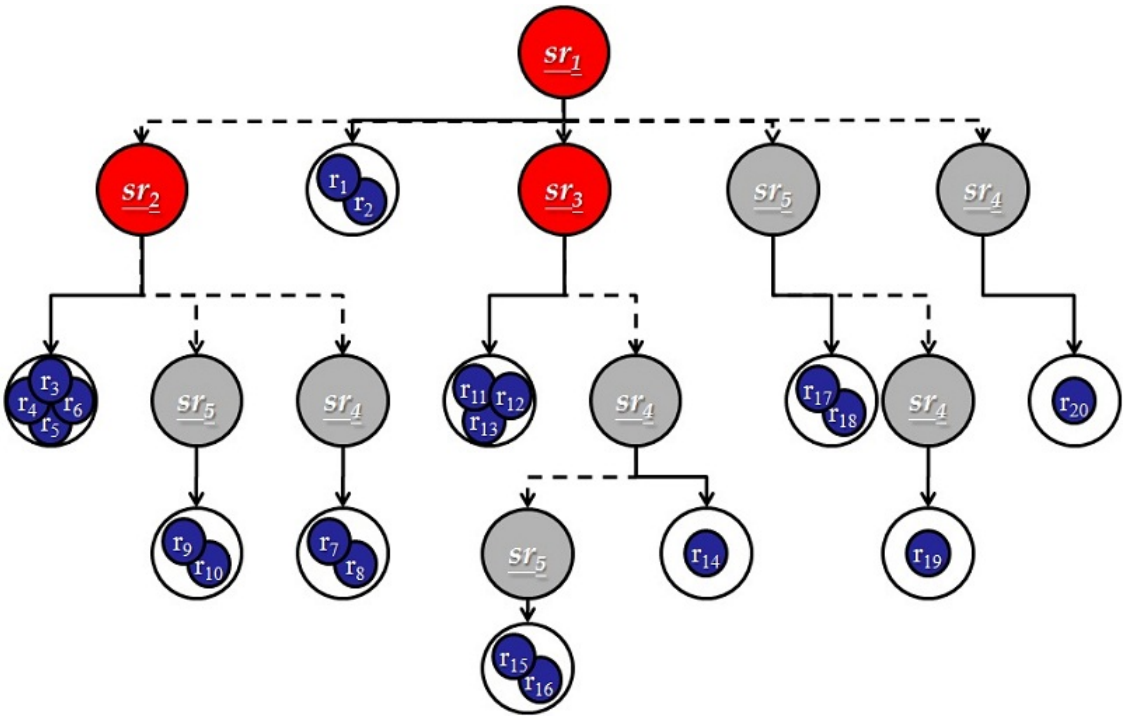

Figure 5.2: Redundancy in HCM (Redundant rules are in gray color nodes)

5.2 Grid Authorization Graph (GAG)

In this section, the Grid Authorization Graph, a *decision graph* derived from the HCM *decision tree* by embedding various edges and tools, is introduced.

5.2.1 Describing OR-based Security Policies using GAG

A graph data structure allows a node to be a child for more than one parent node. Thus it can easily describe the OR-based security policy and avoid the appearance of a resource node for multiple times. Figure 5.3 shows how GAG represents the security policies of Table 5.1 without duplicating the resource's node.

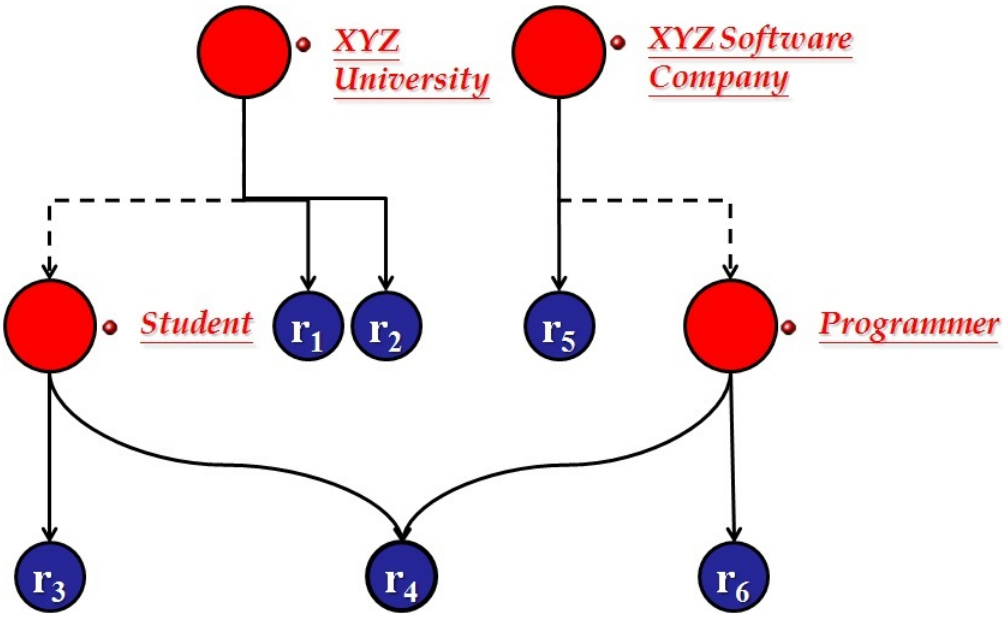

Figure 5.3: Describing OR-based Security Policy in GAG

5.2.2 Eliminating Redundancy of HCM

Figure 5.2 shows an example of redundancy in the HCM *decision tree*. To encounter this issue, a special type of edges named "***Correspondence Edge***" has been introduced in GAG which can be used to entirely eliminate the redundancy. Figures 5.4 and 5.5 represents the *Correspondence Edge* with a red dotted line. It can be used as the following:

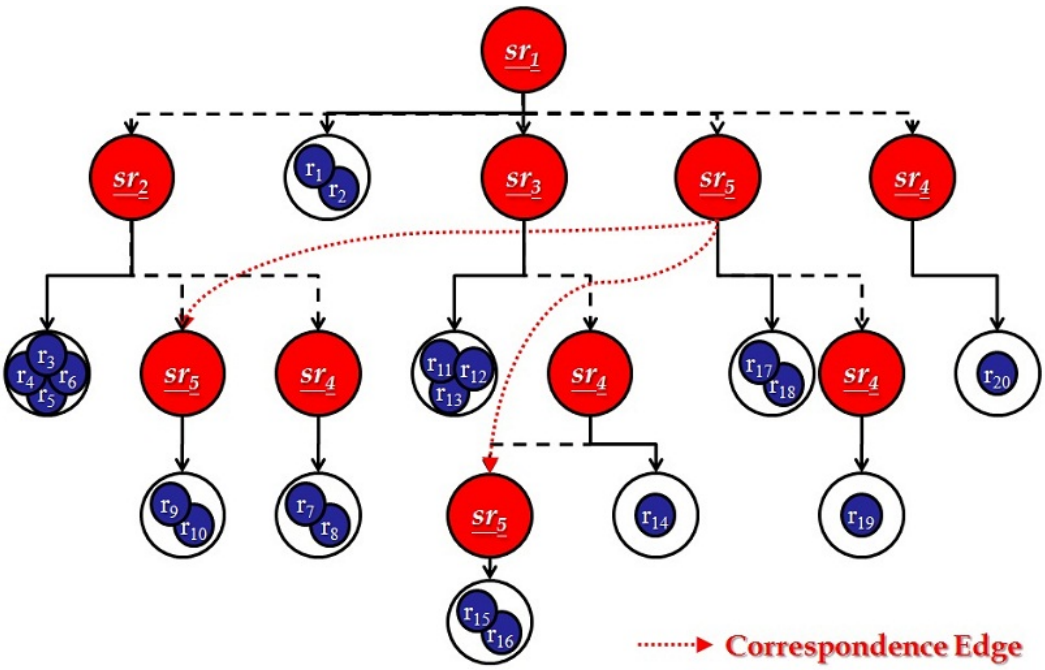

Figure 5.4: Eliminating Redundancy in GAG (Adding sr_5 Correspondence Edges)

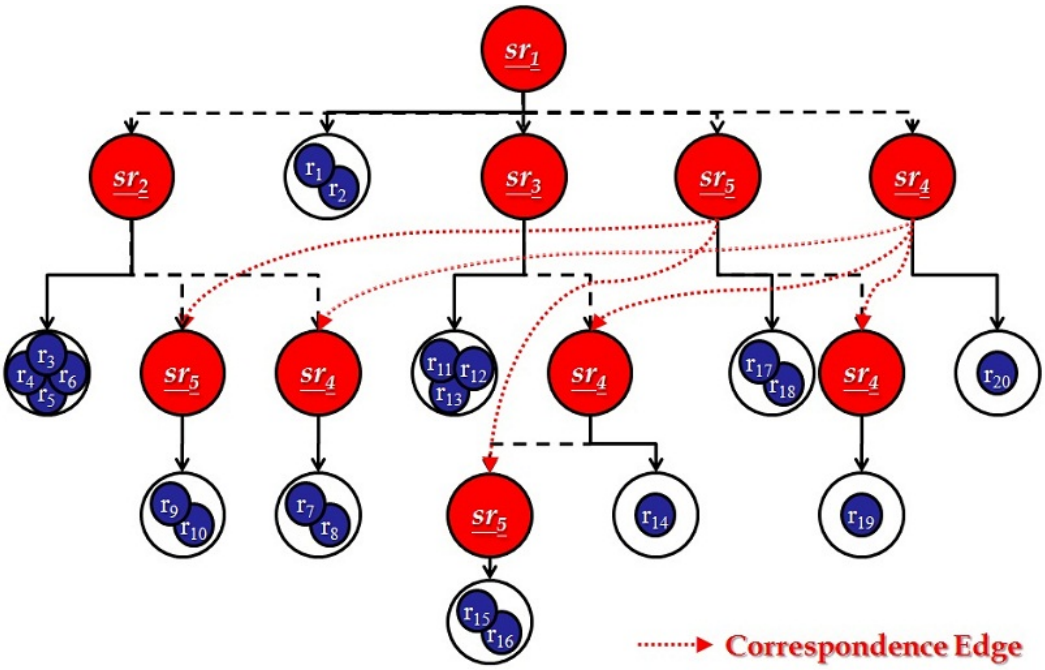

Figure 5.5: Eliminating Redundancy in GAG (Adding sr_4 Correspondence Edges)

*Once a security rule, as an example sr_5, is checked for the first time, a one level **BFS** (Breadth-First Search) [40] on its **Correspondence Edges** is enough to mark all the redundant security rules with the result of the first check. Thus, when the authorization process reaches a redundant node it will find it already marked with the result of the first check and there is no need to do a further checking.*

Working Example

Consider a user whose credentials satisfy sr_1, sr_2 and sr_5 security rules. Figure 5.6 shows the *decision graph* parsed for this user. First, sr_1 is checked (Figure 5.7). As the user satisfies sr_1, the system adds resources r_1 and r_2 to the UARG (Figure 5.8). Then it proceeds to check sr_2 (Figure 5.9). As the user satisfies sr_2, the system adds resources r_3, r_4, r_5, and r_6 to UARG (Figure 5.10). It proceeds to check sr_3 (Figure 5.11). As the user does not satisfy sr_3, then the whole sr_3' sub-tree is marked as *unauthorized* (Figure 5.12). Then it proceeds to sr_5 (Figure 5.13). As the user satisfies sr_5, the system adds resources r_{17} and r_{18} to UARG (Figure 5.14), then it makes a BFS on the *Correspondence Edges* of sr_5 node to mark all redundant sr_5 nodes as *authorized* so all child resources of these redundant nodes, like r_9 and r_{10}, are added to UARG (Figure 5.15). It is important to mention that the *unauthorized* marking **dominates** the *authorized* marking, so a node can not be marked as *authorized* when it has been previously marked as *unauthorized* (ex: sr_5 node in the sub-tree of sr_3 node). The system proceeds to sr_4 node (Figure 5.16). As the user satisfies sr_4, the resource r_{20} is added to UARG (Figure 5.17) then it makes a BFS on the *Correspondence Edges* to mark all the redundant sr_4 nodes as *authorized* so all child resources of these redundant nodes r_7, r_8 and r_{19} are added to UARG (Figure 5.18).

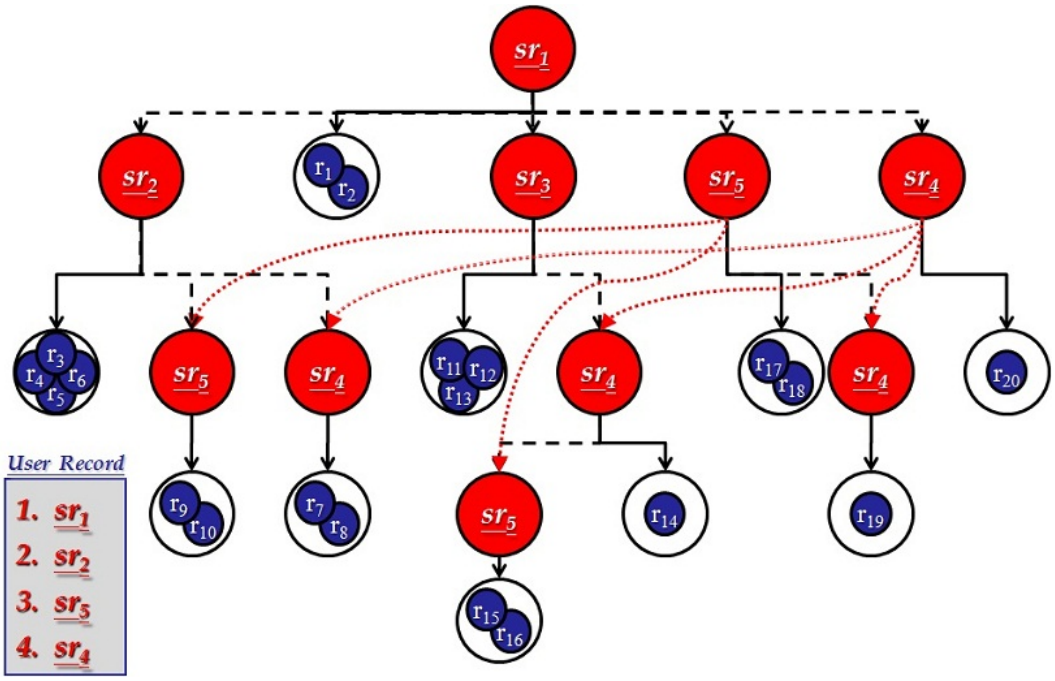

Figure 5.6: Eliminating Redundancy in GAG (Parsing the decision graph for a particular user)

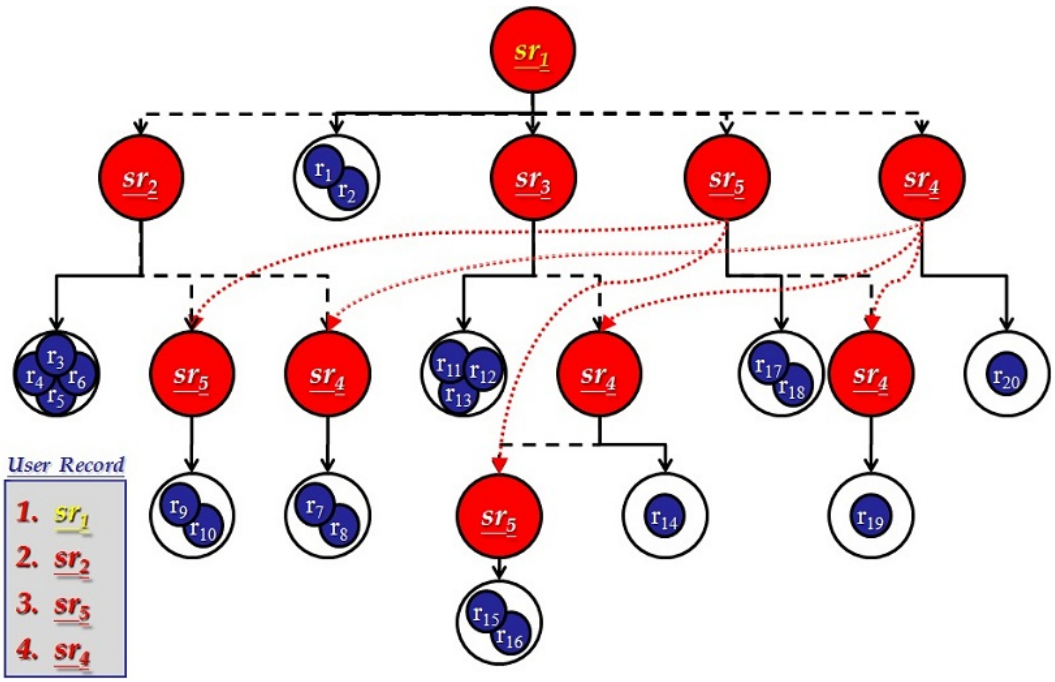

Figure 5.7: Eliminating Redundancy in GAG (checking sr_1 security rule)

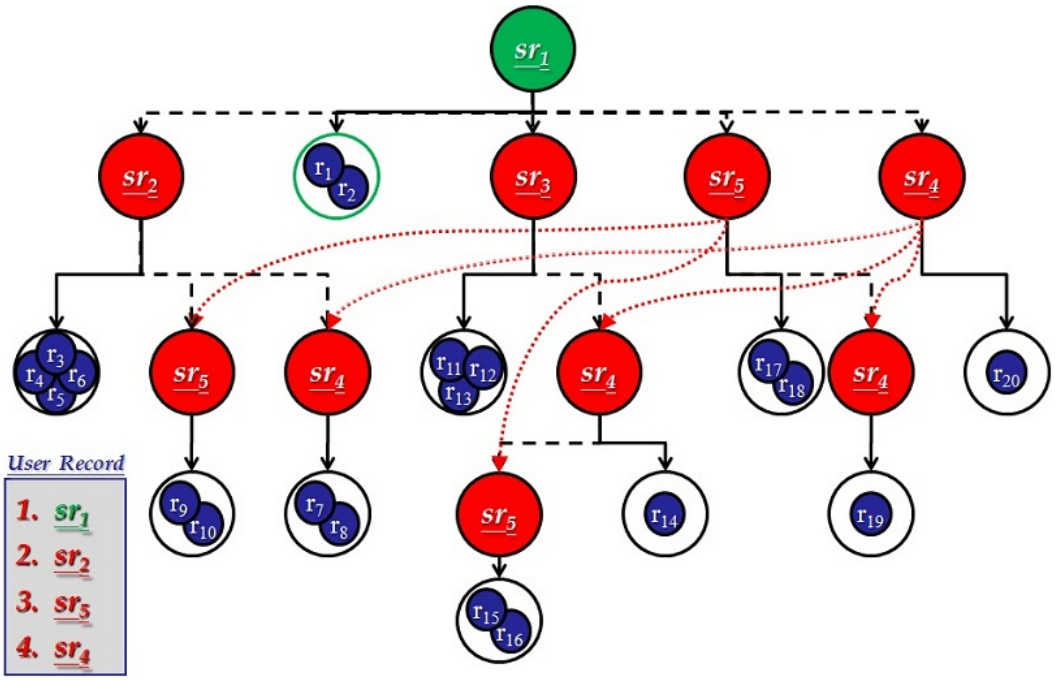

Figure 5.8: Eliminating Redundancy in GAG (As sr_1 is satisfied, add r_1 and r_2 to the UARG)

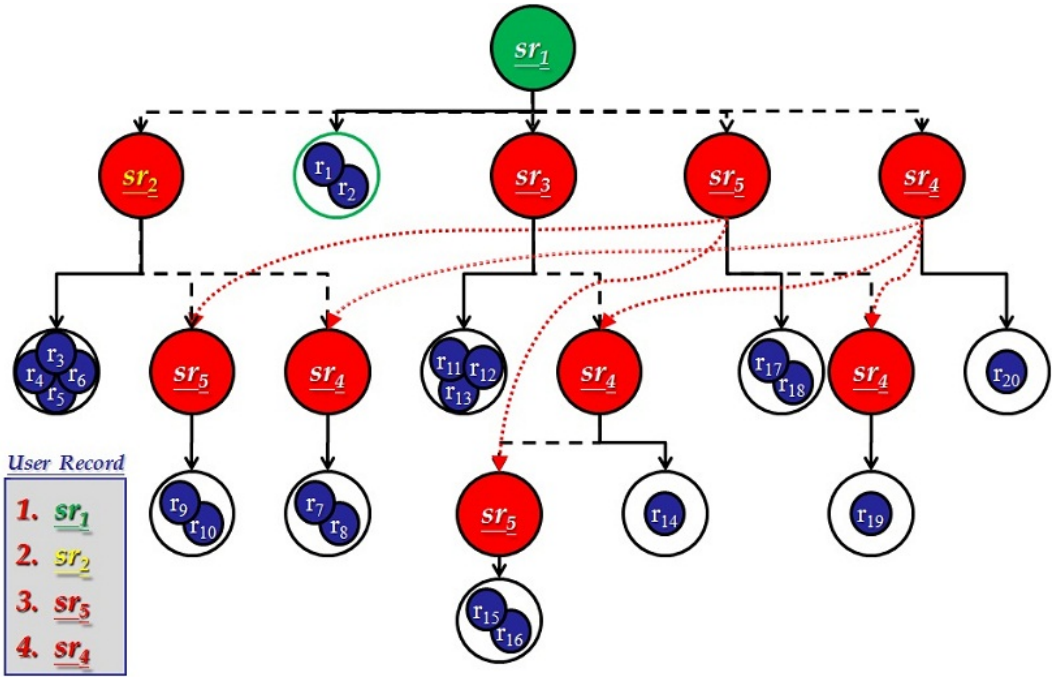

Figure 5.9: Eliminating Redundancy in GAG (checking sr_2 security rule)

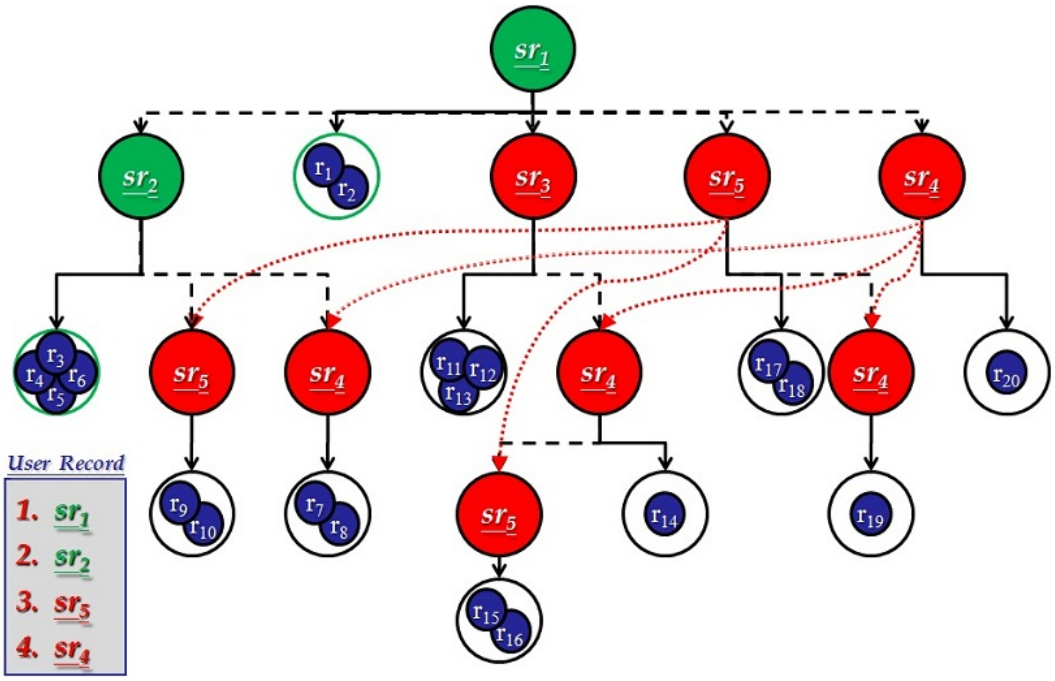

Figure 5.10: Eliminating Redundancy in GAG (As sr_2 is satisfied, add r_3, r_4, r_5 and r_6 to the UARG)

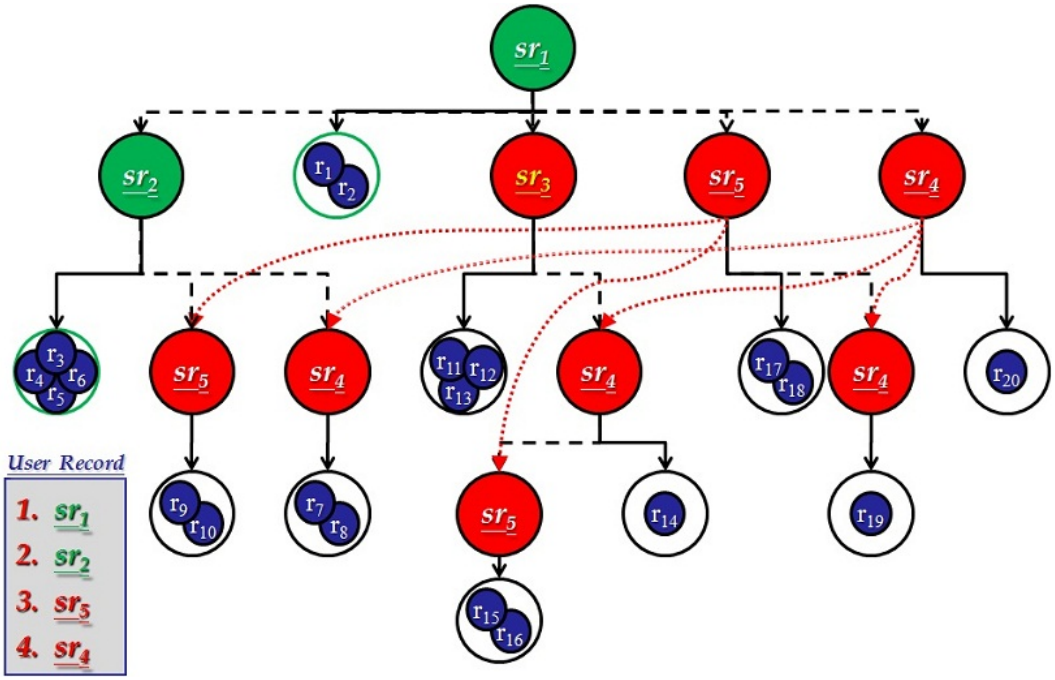

Figure 5.11: Eliminating Redundancy in GAG (checking sr_3 security rule)

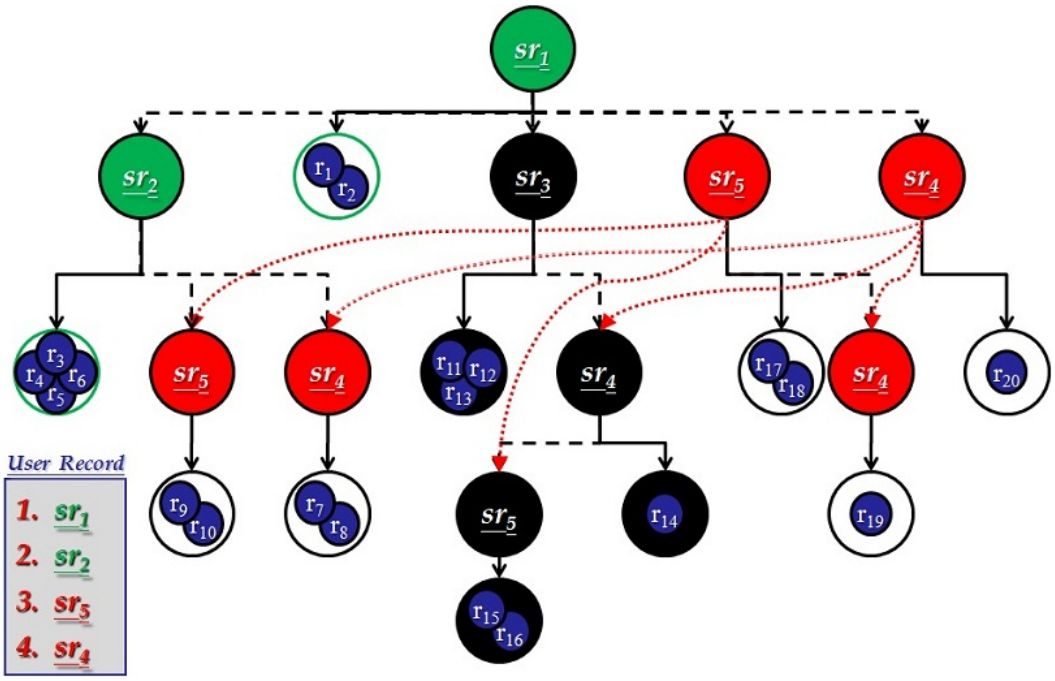

Figure 5.12: Eliminating Redundancy in GAG (As sr_3 is not satisfied, mark the whole sr_3 sub-tree as unauthorized branch)

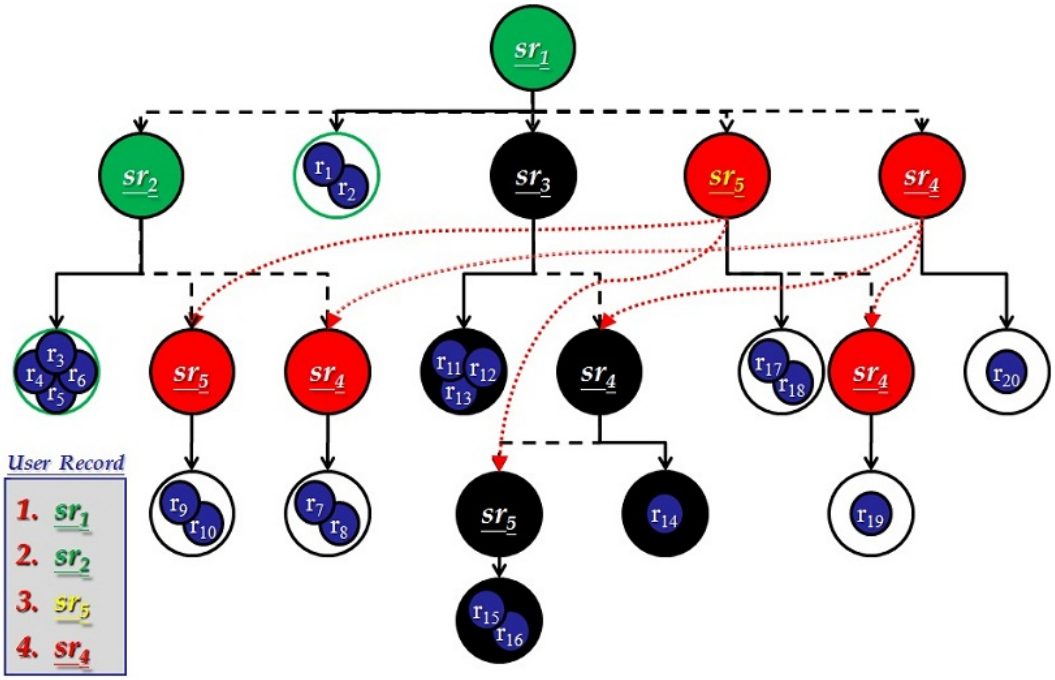

Figure 5.13: Eliminating Redundancy in GAG (checking sr_5 security rule)

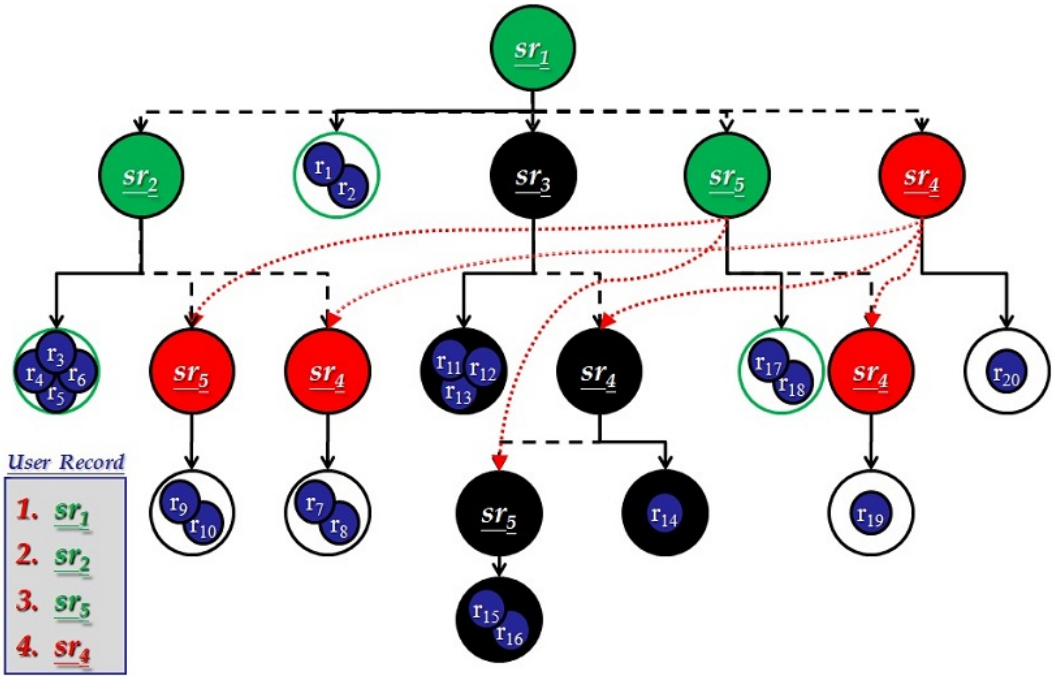

Figure 5.14: Eliminating Redundancy in GAG (As sr_5 is satisfied, add r_{17} and r_{18} to the UARG)

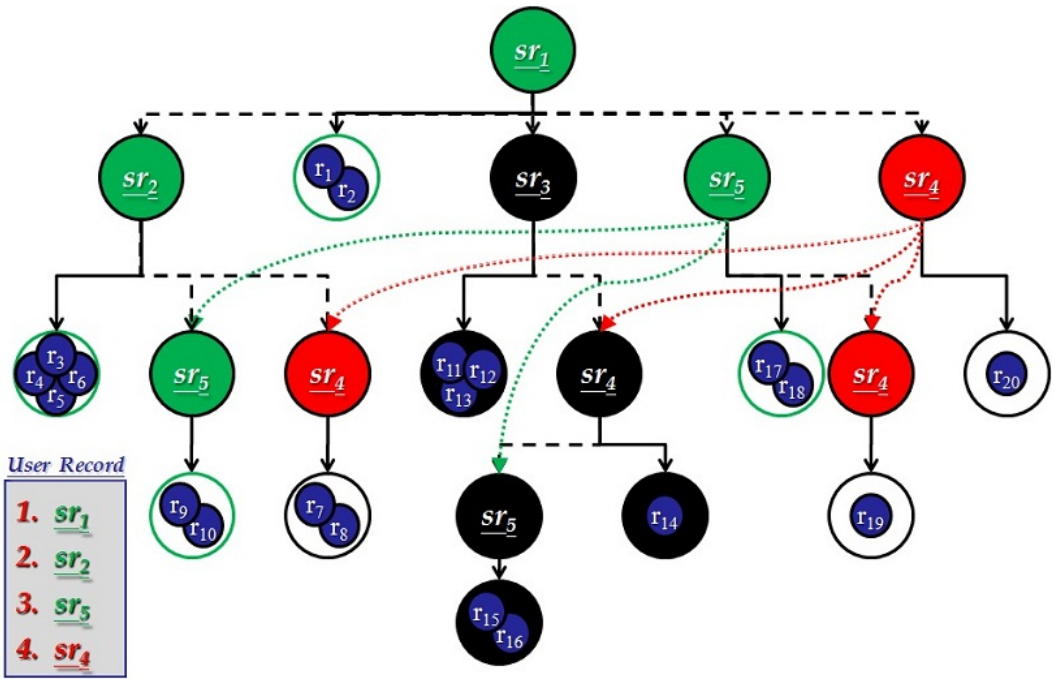

Figure 5.15: Eliminating Redundancy in GAG (Perform BFS on the Correspondence Edges of sr_5, add r_9 and r_{10} to the UARG)

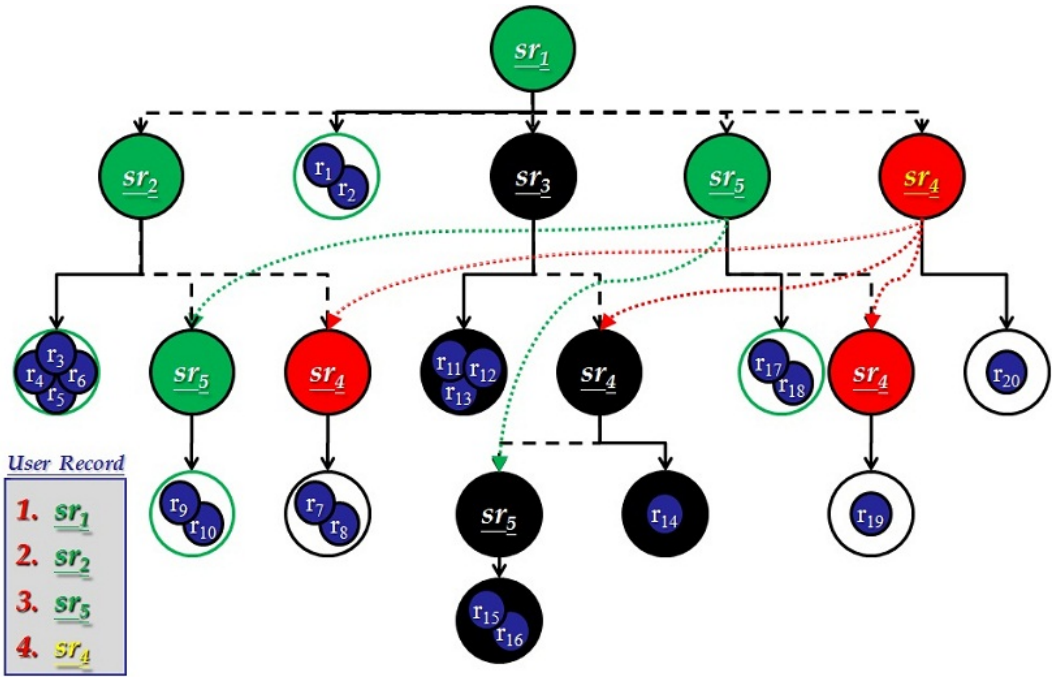

Figure 5.16: Eliminating Redundancy in GAG (checking sr_4 security rule)

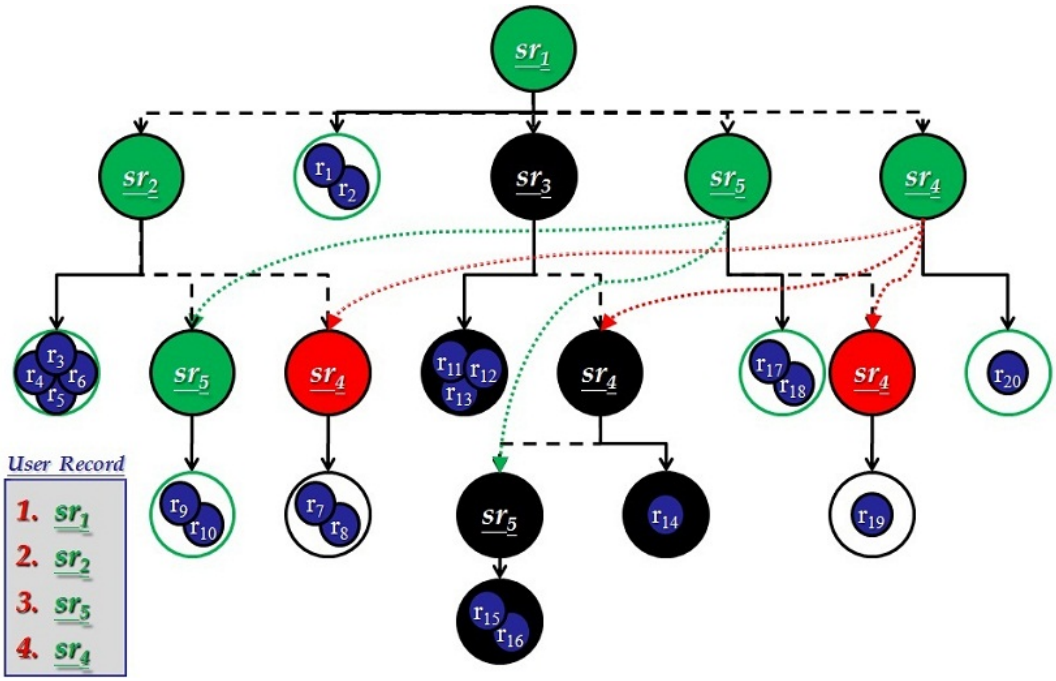

Figure 5.17: Eliminating Redundancy in GAG (As sr_4 is satisfied, add r_{20} to the UARG)

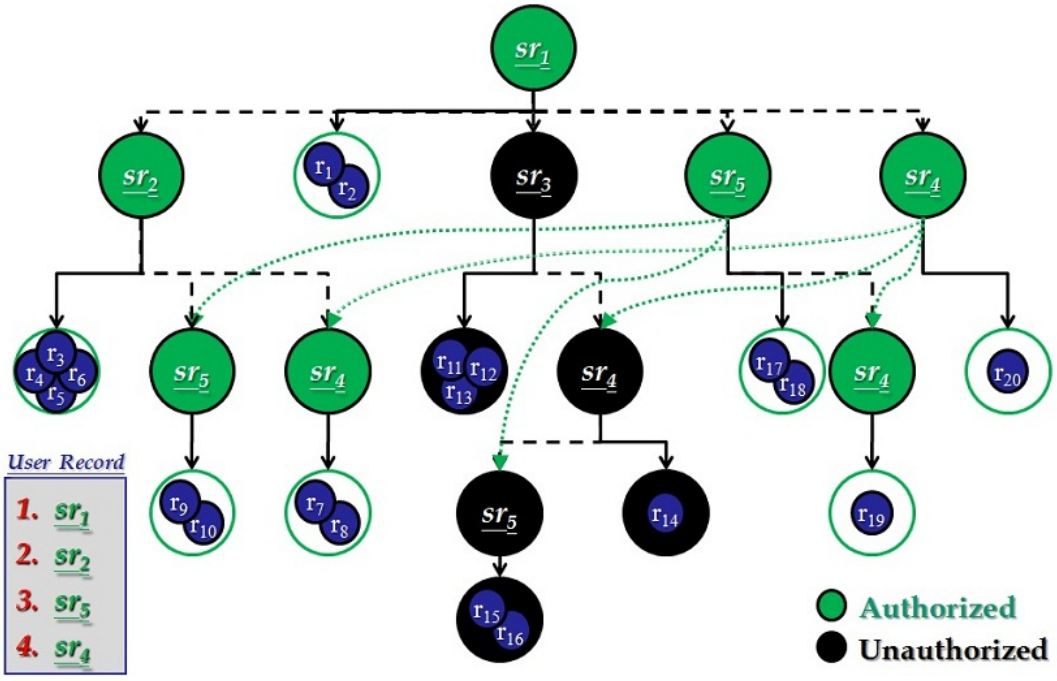

Figure 5.18: Eliminating Redundancy in GAG (Perform BFS on the Correspondence Edges of sr_4, add r_7, r_8 and r_{19} to the UARG)

It is useful to mention that, apart from eliminating redundancy, "Correspondence Edges" provide more functionality listed below:

- Consistency in checking security rules, and
- Can also be used for compatible security rules; that is when different security rules share the same status. Compatible security rules have to be defined by the administrator and cannot be discovered automatically.

5.2.3 Handling Mutually Exclusive Security Rules

Another type of edges which reflects different meaning of *dependency* can be introduced to handle the case where a set of security rules are mutually exclusive. This type of edge is named as "***Discrepancy Edge***". Figure 5.19 represents the *Discrepancy Edge* with a black dotted line. It can be read as the following:

"If sr_3 is satisfied then sr_4 and sr_5 cannot be satisfied".

So, it is evident not to check sr_4 and sr_5 if sr_3 is satisfied as it is already known to the system that sr_4 and sr_5 are mutually exclusive with sr_3 and they cannot be satisfied all together. Therefore, with the help of the *Discrepancy Edges*, once sr_3 is checked and satisfied, a one level BFS on the *Discrepancy Edges* is enough to mark all the set of mutually exclusive security rules, sr_4 and sr_5 as *unauthorized*.

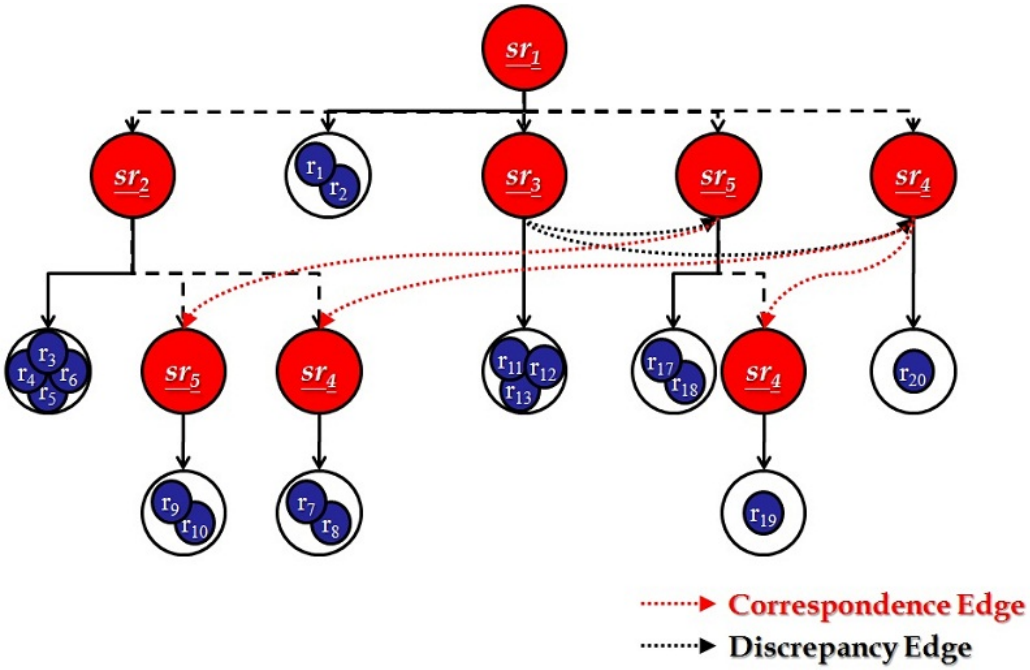

Figure 5.19: Handling Mutually Exclusive Rules (Adding sr_3 Discrepancy Edges)

Consider a user whose credentials satisfy security rules {sr_1, sr_2, sr_3}. Figure 5.20 shows the *decision graph* parsed for this particular user. First, sr_1 is checked (Figure 5.21). As the user satisfies sr_1, the system adds resources r_1 and r_2 to the UARG (Figure 5.22) then it proceeds to check sr_2 (Figure 5.23). As the user satisfies sr_2, the system adds resources r_3, r_4, r_5, and r_6 to UARG (Figure 5.24) then it proceeds to check sr_3 (Figure 5.25). As the user satisfies sr_3, the system adds resources r_{11}, r_{12} and r_{13} to UARG (Figure 5.26) then it makes a BFS on the *Discrepancy Edges* of sr_3 node (Figure 5.27) to mark all the set of mutually exclusive security rules sr_4 and sr_5 as *unauthorized* (Figure 5.28). Following it *Correspondence Edges* are used in its turn to mark all the redundant sr_4 and sr_5 security rules as *unauthorized* also (Figure 5.29).

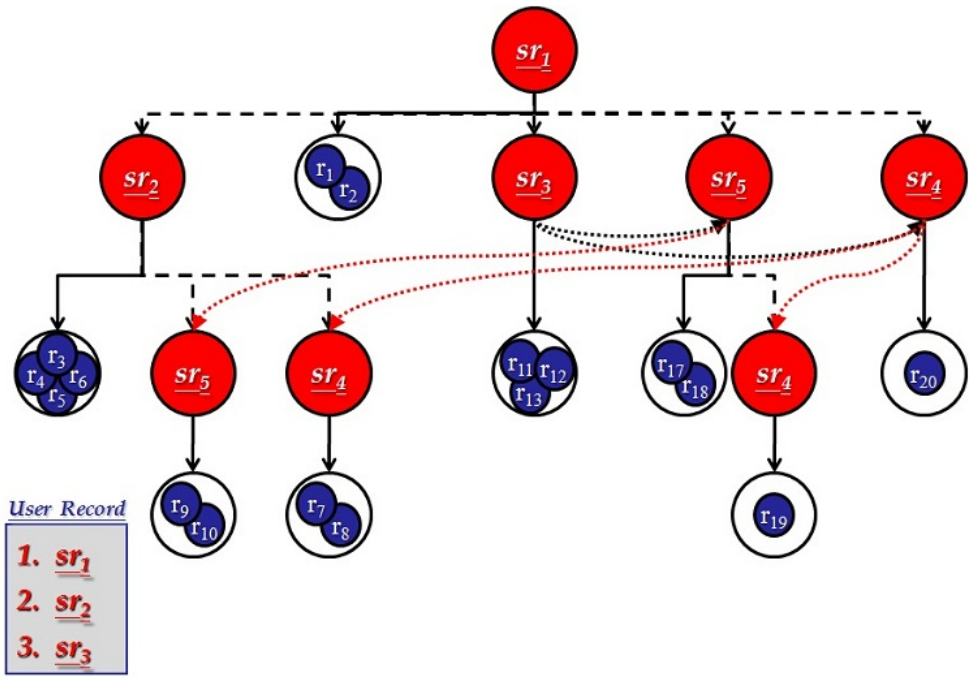

Figure 5.20: Handling Mutually Exclusive Rules (Parsing the decision graph for a particular user)

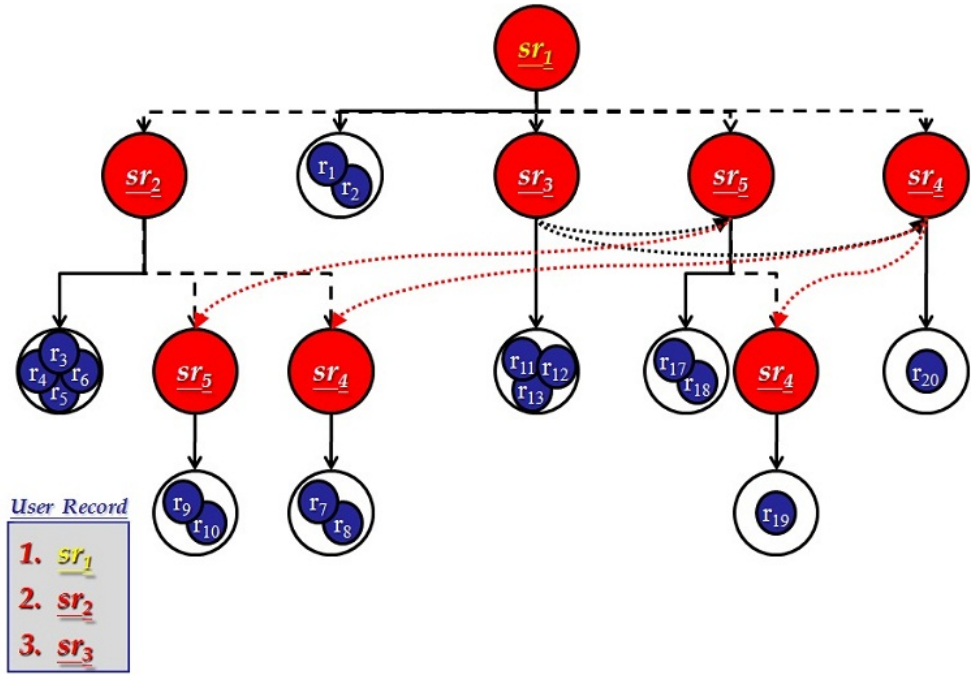

Figure 5.21: Handling Mutually Exclusive Rules (checking sr_1 security rule)

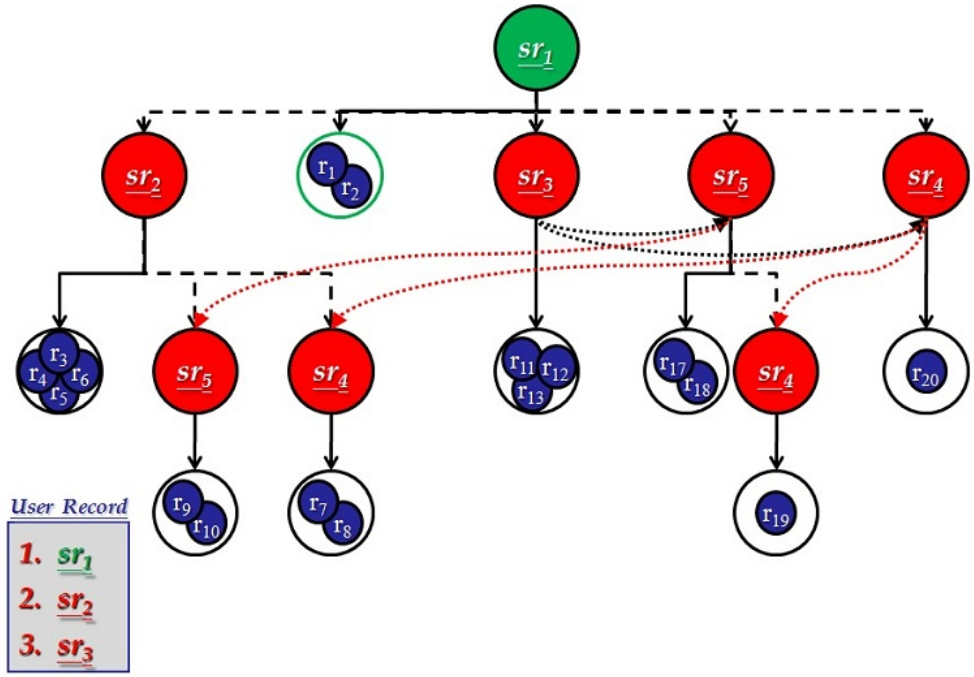

Figure 5.22: Handling Mutually Exclusive Rules (As sr_1 is satisfied, add r_1 and r_2 to the UARG)

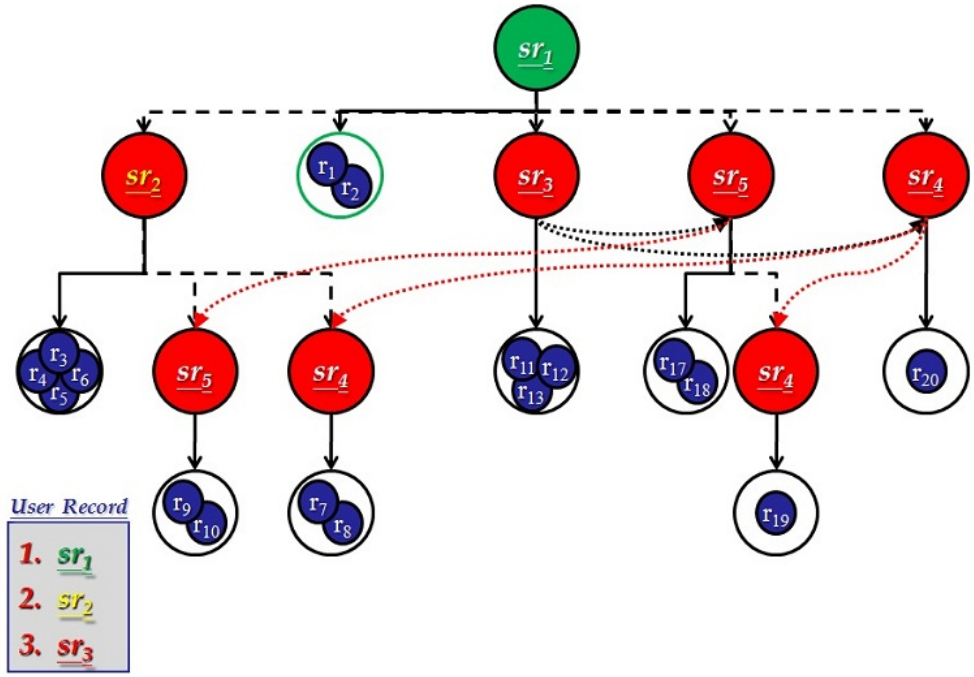

Figure 5.23: Handling Mutually Exclusive Rules (checking sr_2 security rule)

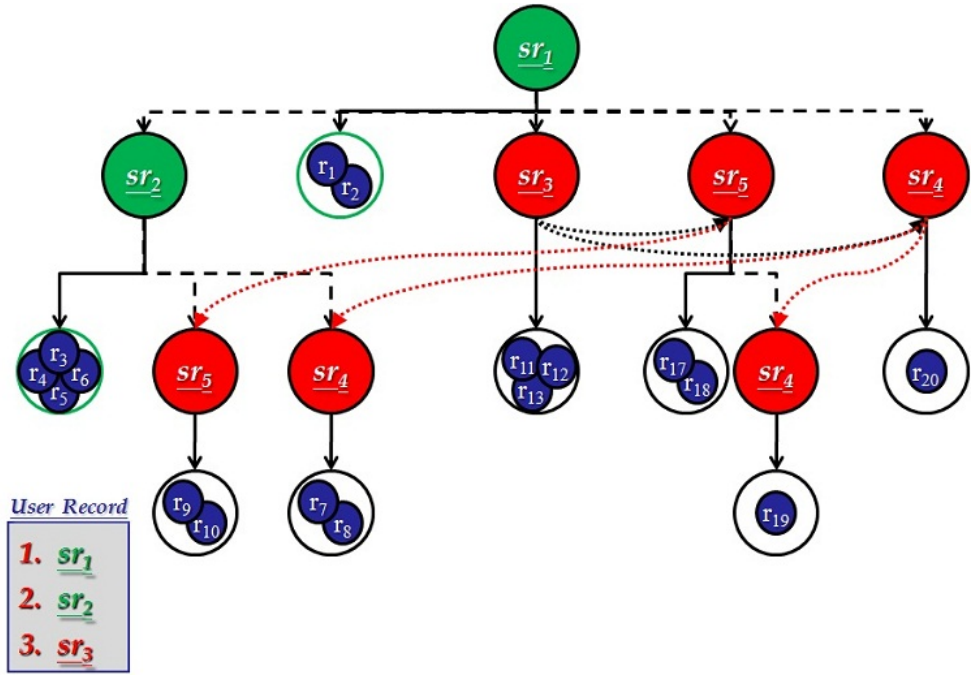

Figure 5.24: Handling Mutually Exclusive Rules (As sr_2 is satisfied, add r_3, r_4, r_5 and r_6 to the UARG)

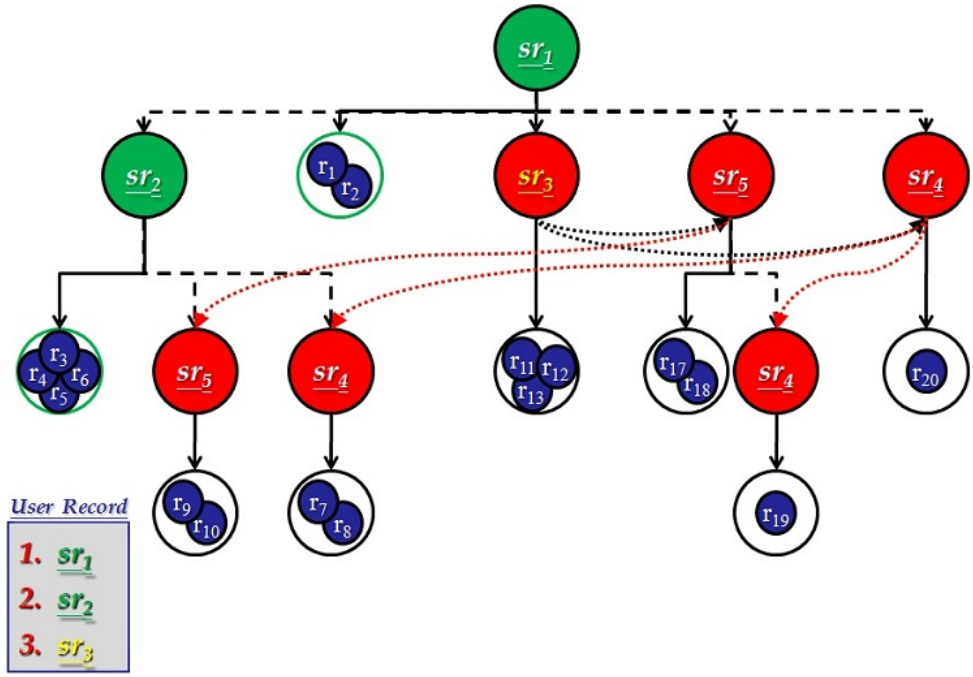

Figure 5.25: Handling Mutually Exclusive Rules (checking sr_3 security rule)

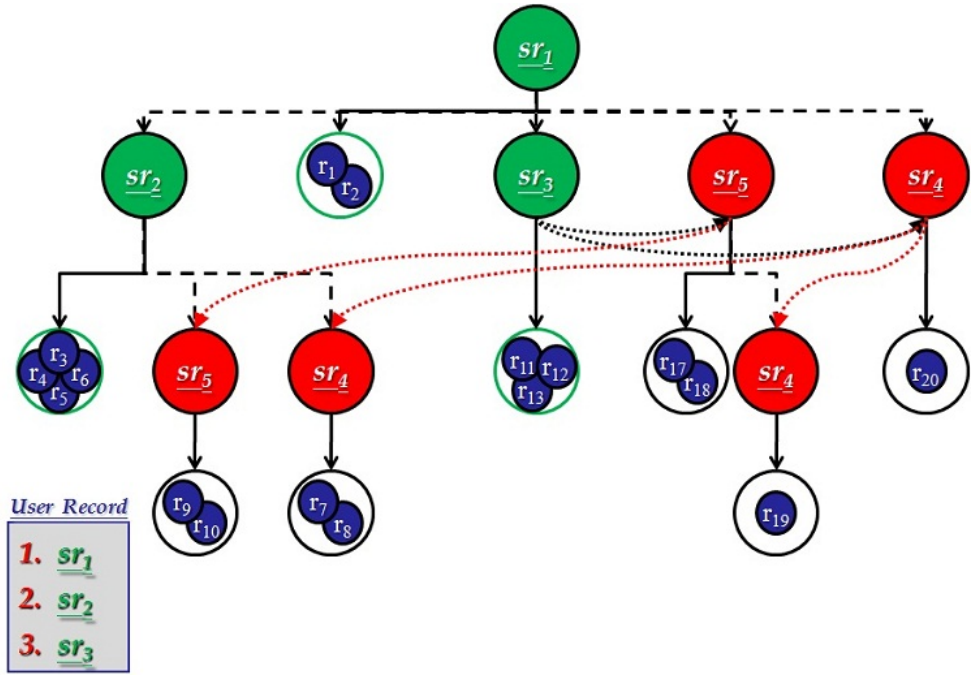

Figure 5.26: Handling Mutually Exclusive Rules (As sr_3 is satisfied, add r_{11}, r_{12}, and r_{13} to the UARG)

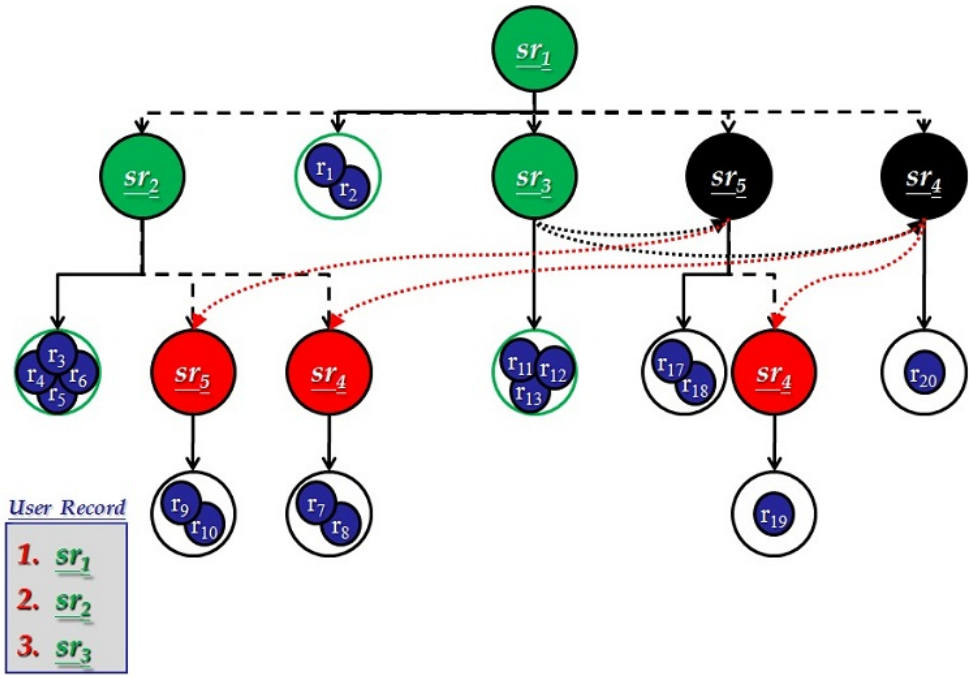

Figure 5.27: Handling Mutually Exclusive Rules (Perform BFS on the Discrepancy Edges of sr_3)

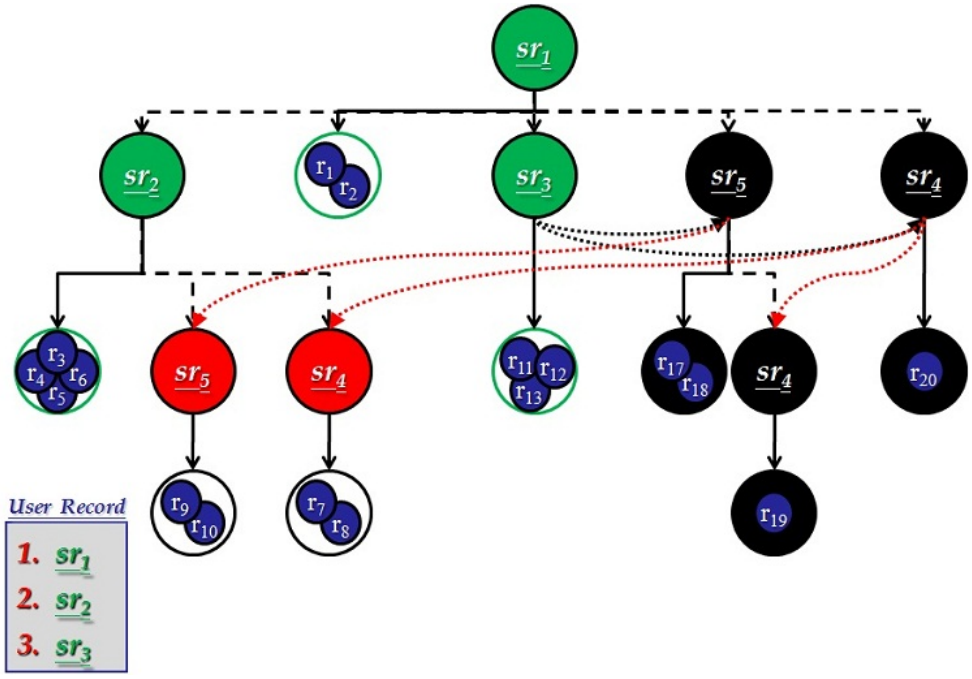

Figure 5.28: Handling Mutually Exclusive Rules (Mark sr_4 and sr_5 sub-trees as unauthorized)

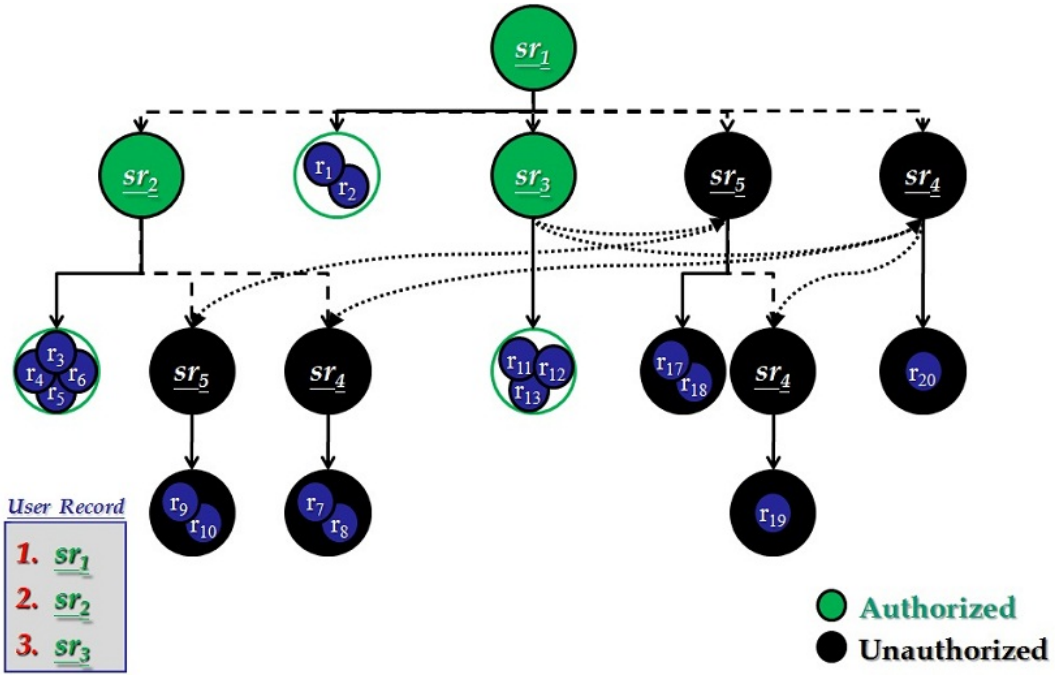

Figure 5.29: Handling Mutually Exclusive Rules (Perform BFS on the Correspondence Edges of sr_4 and sr_5 to mark the redundant nodes as unauthorized)

Dependency Edges (Correspondence/Discrepancy) have seven forms (Figure 5.30). The "One Way" form propagates the result in one direction only, while the "Two Ways" form propagates in two directions. The "Positive" form propagates the *authorized* result only while the "Negative" form propagates the *unauthorized* result.

5.3 The GAG Generator Algorithm

Dependency Edges (Correspondence / Discrepancy) can be added manually by the administrator based on system requirements. However, it is unfeasible to add all the *Correspondence Edges* between the redundant nodes manually. Thus, an algorithm which automatically tracks the redundant nodes and draws the *Correspondence Edges* between them is required.

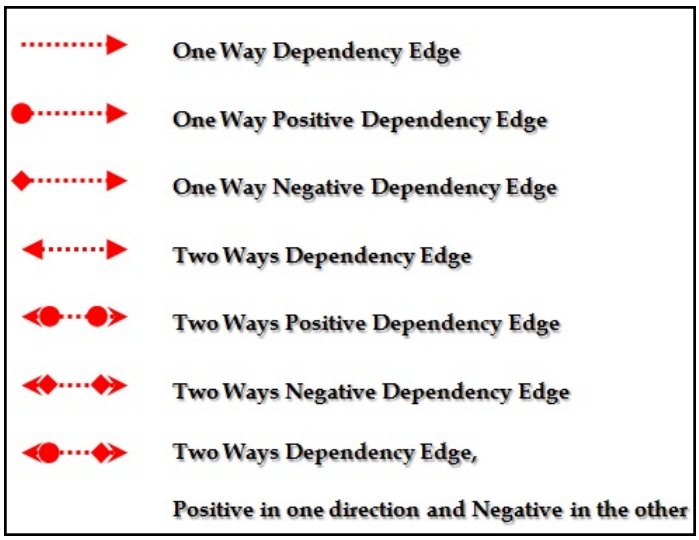

Figure 5.30: Different forms of Dependency Edges

The **Counting Algorithm** (Algorithm 1) is upgraded in this section to add the *Correspondence Edges* automatically between redundant nodes. It is named as: "The GAG Generator algorithm" (Depicted in Algorithm 8). This algorithm uses the Security Rules Vector (SRV) which helps easing up updating of security rules (update at one site). Without SRV, a clique between redundant nodes has to be maintained, as we do not know which redundant node is going to be checked first.

The output of the GAG Generator algorithm, when it runs on the security table (Table 5.2) is depicted in Figure 5.31. During the authorization process, when the security rule of a particular node is checked, the result is propagated through the undirected edge of that node to its correspondent cell in the SRV. Then the result is further propagated through the correspondent SRV cell to all redundant nodes via one level BFS.

Algorithm 8 The GAG Generator Algorithm

Input:

Resources' Security Table

Output:

Grid Authorization Graph (GAG)

Variable:

[1] A vector, named "SRV", of all security rules is maintained (Figure 5.31)

[2] Each node in the graph is a structure having three fields:

- the *security rule* **sr**,
- an interim *security table* **ST** and
- an *undirected edge* to the correspondent security rule in SRV.

Begin

Step 1: (Initialization)

- Initialize the *decision tree* by a root node with NULL *security rule* (**sr**).
- Build the security table which represents the entire security policies of the system. Assign it as the *security table* property (**ST**) of the root node.
- Assign NULL to the *correspondent edge* of the root node.
- Execute Step 2 for the root node.

Step 2: (Processing of one node N_i)

Step 2.1: (Adding N_i's Resources)

- Add each resource, whose representative row in N_i's **ST** is having zero cells only, as a child resource to N_i.

Step 2.2: (Processing of N_i's security table (ST))

- Sum the cells of each column of N_i's **ST** and refer it as **Count**.
- Choose the security rule sr_j with the highest **Count**.
- Divide **ST** into two tables excluding the j^{th} column as the following:

- The first table (T_1) contains the rows of the resources which demand sr_j (each row whose j^{th} cell's = 1).
- The second table (T_2) contains the rows of the resources which do not demand sr_j (each row whose j^{th} cell's = 0).

Step 2.3: *(N_i* ***Bifurcation)***

- Add a left child node, named "**LCN**", to N_i with sr_j as the *security rule* (**sr**) and T_1 as the *security table* (**ST**). Let the ***correspondent edge*** of **LCN** refer to sr_i's cell in **SRV**.
- Add a right child node, named "**RCN**", to N_i with NULL as the *security rule* (**sr**) and T_2 as the *security table* (**ST**). Let the ***correspondent edge*** of **RCN** refer to NULL.

Step 3: (Recurring)

- Repeat step 2 for each child node until a node with **empty** *security table* is reached.

Step 4: (Pruning)

- Prune the graph at nodes labeled NULL.
- Delete all the interim *security tables* (**ST**s) to free space.

End

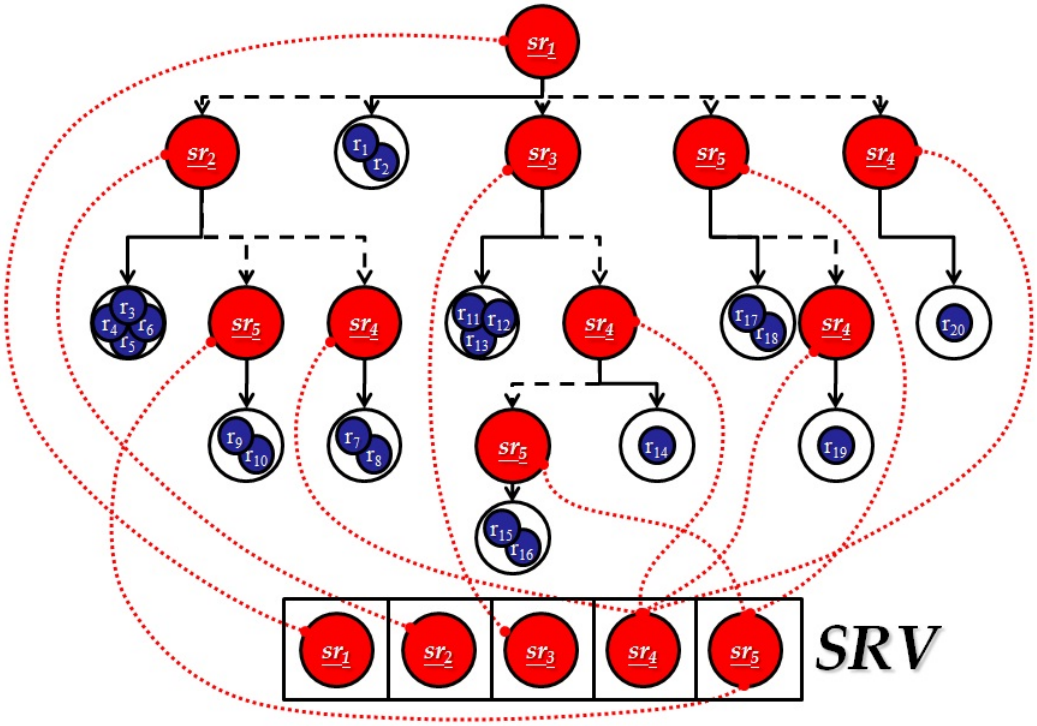

Figure 5.31: Example of the GAG Generator Algorithm

Computational Complexity of the GAG Generator Algorithm	
Step 1	O($M \times N$): Where M is the number of resources and N is the number of security rules.
Repeated	1 time
Step 2	O($\frac{M}{2^i} \times (N - i)$): As an average complexity, where i is the node's level
Repeated	In average, we have 2^i nodes in each level and (0 to N) levels
Step 3	O(1): Simple condition check.
Repeated	(N + 1) times
Step 4	O(N): Maximum number of NULL nodes in the tree is N
Repeated	1 time
Algorithm Complexity: O($M \times N^2$). (usually N << M)	

After introducing GAG with its powerful services, we can notice that "HCM *decision tree* is still at the core of GAG". Figure 5.2 and 5.5 show an example of it. This means all caching mechanisms, which were designed to work in HCM as TCM [34] and HDCM [34] are still valid to work in GAG. Thus TCM and HDCM modules of HCM can be embedded directly in GAG Search Engine described in Section 6.1. Moreover, all the analysis which has been done to prove the stability of HCM *decision tree* against the dynamic changes in the grid environment[34] are also valid for GAG.

5.4 Results and Experiments

For a grid environment of 30 resources and 10 security rules, 100 different authorization processes have been initiated. For each authorization process, the posterior analysis of the *Hierarchical Clustering Mechanism (HCM)* and the *Grid Authorization Graph (GAG)* has been done and depicted in Figure 5.32 and Table 5.3.

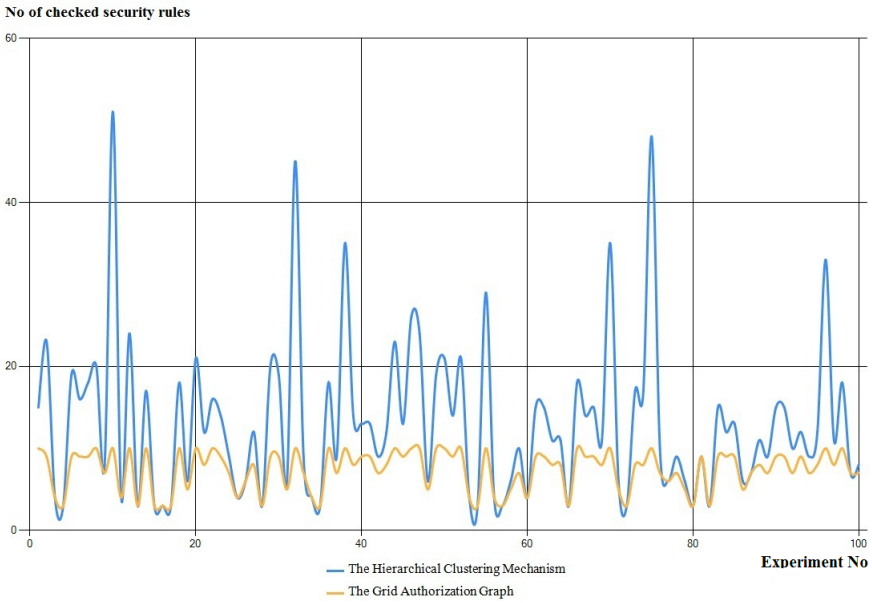

Figure 5.32: Experiments and Results

Table 5.3: Experiments and Results

	AVG	Standard Deviation	Checked security rules Range [min, max]
HCM	13	9.5708	[3, 51]
GAG	7	2.4658	[3, 10]

It is important to notice that while GAG eliminates the redundancy in checking security rules, it adds extra complexity when it does BFS on the redundant nodes. However, the cost of the BFSs operations compared to the cost of checking the redundant security rules is negligible. As checking a security rule requires checking of user credentials (*attributes assertions*[41] issued by the *Attributes Authorities*[42]), this further requires PKI[43] operations, which are well known as expensive processes[44].

5.5 Weighted GAG

As each edge in a graph can have a weight, one can think of assigning a weight (nonnegative integer) to every edge in the *decision graph* which reflects the *importance degree* of the security rule which the edge emerges from. Then an attribute named "***Classification Level***" can be assigned to every resource which is a numerical value that equals the weight of the shortest path from the root node to the resource's node (sum of the weight of the edges), as shown in Figure 5.33. Every user in the system has an attribute named "***Security Clearance***" derived out of user's set of roles in the system. One can avoid parsing the *decision graph* for authorizing a user u_i whose *security clearance* ($\mathrm{USC}(\mathrm{u}_i)$) is less than the *classification level* of the resource r_j ($\mathrm{RCL}(\mathrm{r}_j)$), that is $\mathrm{USC}(\mathrm{u}_i) < \mathrm{RCL}(\mathrm{r}_j)$. Thus the developed **weighted GAG** with the following details shown in Definition 5 helps to implement Bell-LaPadula (BLP) [45] model of enforcing access control.

Definition 5. *Weighted GAG Parameters*

- *Let SR*= $\{\mathrm{sr_j}|\mathrm{j} = 1,\ ..., l\}$ *be the set of all security rules.*
- *Let IDSR: SR* $\rightarrow$ *N: IDSR(sr_j) = the Importance Degree of security rule sr_j.*
- *Let G(V, E) be the Grid Authorization Graph (GAG), where V is the set of vertices (security rules) and E is the set of edges.*
- *Let W: E* $\rightarrow$ *N: W(e_{ij}) = IDSR(sr_i) be a function defines edges' weights in GAG. Where sr_i is the security rule which the edge e_{ij} emerges from.*
- *Let* $R = \{\mathrm{r_j}|\mathrm{j} = 1,\ ..., k\}$ *be the set of grid resources.*
- *Let SP: R* $\rightarrow$ *N: SP(r_j) = weight of the Shortest Path to resource r_j.*
- *Let RCL: R* $\rightarrow$ *N: RCL(r_j) = SP(r_j) be a function maps each resource to its Classification Level.*
- *Let Roles* $= \{\,\mathrm{role_j}\,|\mathrm{j} = 1,\ ..., l\}$ *be the set of all roles.*
- *Let IDR: Roles* $\rightarrow$ *N: IDR($role_j$) = the Importance Degree of $role_j$.*
- *Let U* $= \{\,\mathrm{u_i}\,|\mathrm{i} = 1,\ ...,\ \mathrm{m}\}$ *be the set of all users.*
- *Let UR* $\subseteq$ *U* $\times$ *Roles be the set of relations of user to role assignments.*
- *Let USR: U* $\rightarrow$ *Roles: USR(u_i) =* $\{role_j|(u_i, role_j) \in UR\}$ *be a function derived from UR mapping each user to a set of roles.*
- *Let USC: U* $\rightarrow$ *N: USC(u_i)=* $\sum_{role_j \in USR(u_i)}$ *IDR($role_j$) be a function maps each user to his security clearance.*

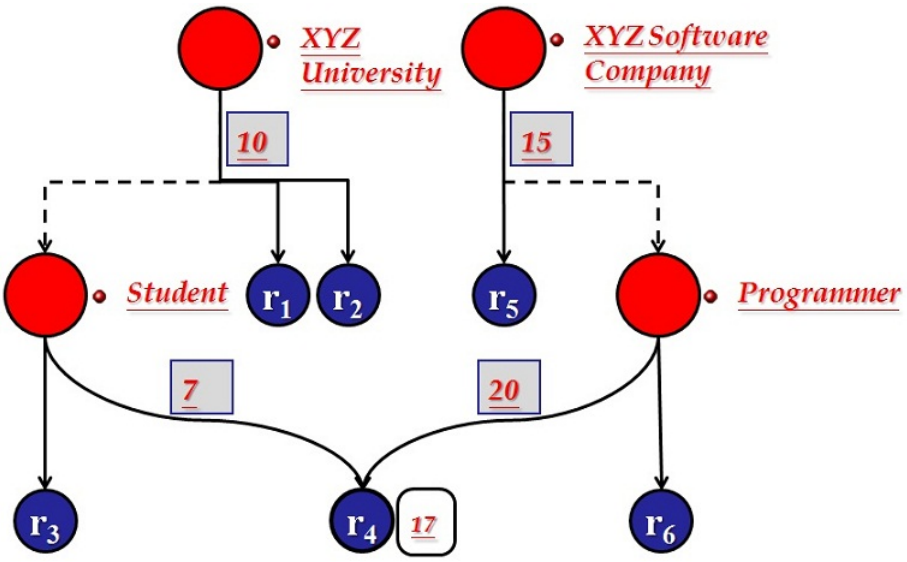

Figure 5.33: Classification Level example of r_4

5.6 Summary

In this chapter, a novel grid authorization enhancement is proposed by introducing the *Grid Authorization Graph* (GAG). While HCM reduces the redundancy in checking security rules compared to the *Brute Force Approach* and the *Primitive Clustering Mechanism*, GAG eliminates it completely. As HCM is still at the core of GAG, TCM and HDCM caching mechanisms are still valid to work in GAG. Moreover, GAG introduces several new tools to speed up the authorization process like the *Dependency Edges* and the use of *Classification Level* and *Security Clearance* in the weighted GAG. Thus GAG is shown as an efficient and superior access control mechanism which can be integrated easily in the present popular grid authorizing systems like VOMS, Akenti, PERMIS, etc.

Chapter 6

Design & Developement of Grid Authorization Simulator

This chapter shows how GAG can be embedded in GT4 authorization framework illustrating various components to be added or modified. Also it introduces the Grid Authorization Simulator (GAS), designed and implemented as a part of this thesis. GAS is a C# based application used to simulate the authorization process of currently used mechanisms like the Brute Force Approach and the PCM as well as our proposed HCM and GAG.

A quick user manual of GAS is prepared with the technical specifications which describes how the functional requirements of GAG are translated into application components. Its goal is to ensure a clear understanding of what the developers are supposed to build in satisfying overall functional requirements of GAG, and to ensure internal standards and best practices.

6.1 Embedding GAG in GT4 Authorization Framework

GT4[46] authorization framework[47] was constructed based on the OASIS XACML and SAML standards[48]. It contains the PEP (*Policy Enforcement Point*)[23], the PDP (*Policy Decision Point*)[23], the PIP (*Policy Information Point*)[23] and the PAP (*Policy Administration Point*)[23]. To make the framework compatible with GAG, five more components were added to the architecture as shown in Figure 6.1 (red color). These components are:

- RAP (*Request Analyzer & Processing*) and GAG Search Engine in the PDP.
- the XML Analyzer, the GAG Generator Engine and the GAG Database.

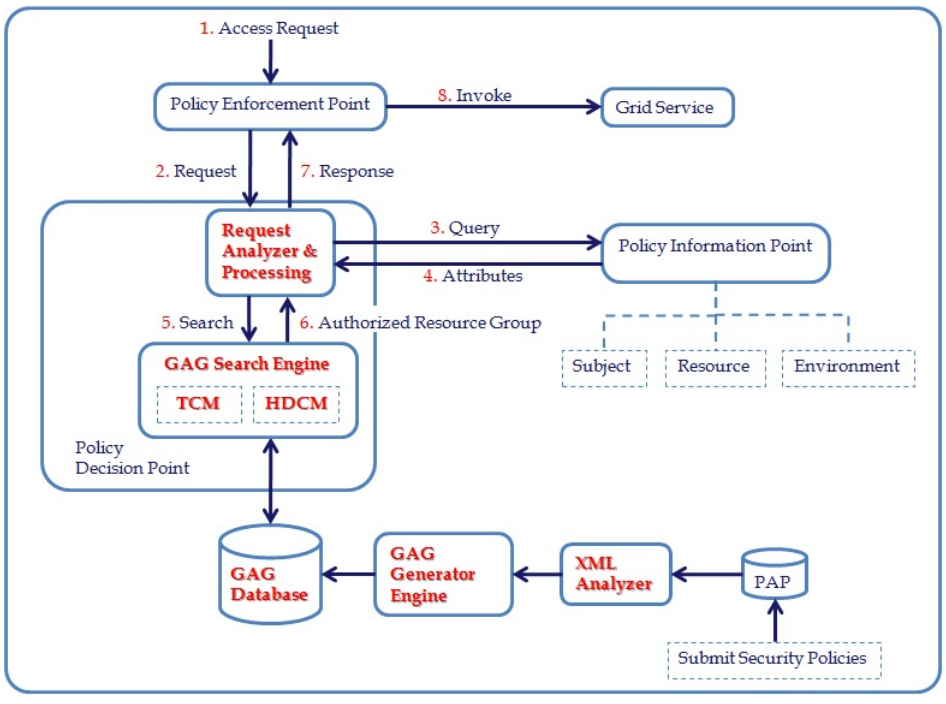

Figure 6.1: GAG Enabled Authorization Framework

The resource's security policy is submitted by the stakeholder to the PAP through SAML or XACML specification language. Thus an XML Analyzer is required to parse the security policies' files, pick up the security rules and provide a simplified input to the GAG Generator Engine. The GAG Generator Engine is responsible to build the Grid Authorization Graph (GAG) using Algorithm 8. Following this it maintains the output *decision graph* in the GAG Database to be used by the GAG Search Engine.

When a user raises an access request, the PEP intercepts the request and propagates it to the PDP. The RAP makes access decisions using the UARG obtained by querying the GAG Search Engine and the authorization attributes[49] of the subject, the resource and the environment obtained by querying the PIP. The access decision given by the PDP is sent to the PEP. The PEP fulfills the obligations and either permits or denies the access request according to the decision of PDP.

6.2 GAS Quick User Manual

6.2.1 File Menu

Figure 6.2 shows the main form of GAS. The **File** menu enables us to create the security table (It is a table whose each row of it represents a resource' security policy).

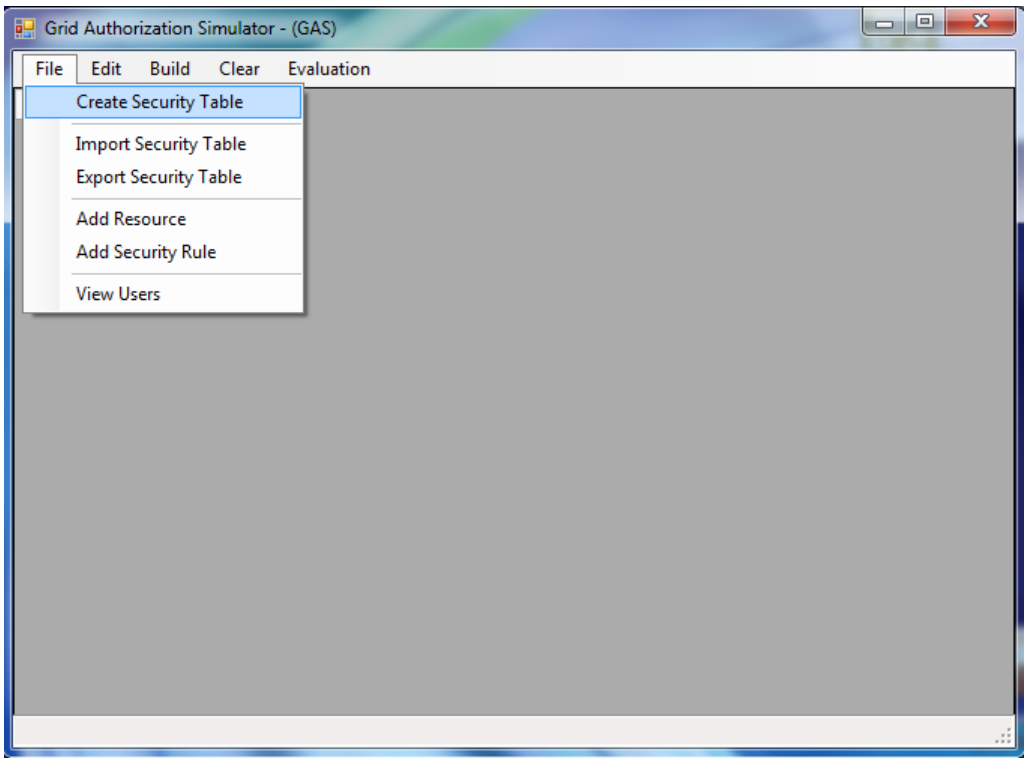

Figure 6.2: GAS - File Menu

If we select "create security table", we have to specify the number of resources and the number of security rules existing in the system as shown in Figures 6.3 and 6.4 respectively. Finally, we get a blank security table as shown in Figure 6.5.

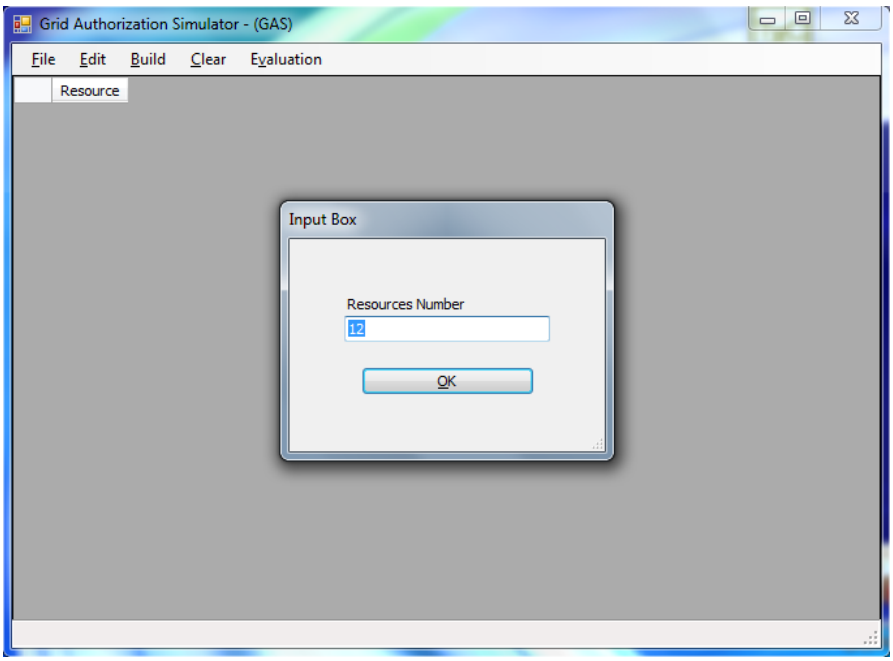

Figure 6.3: GAS - Specify number of resources

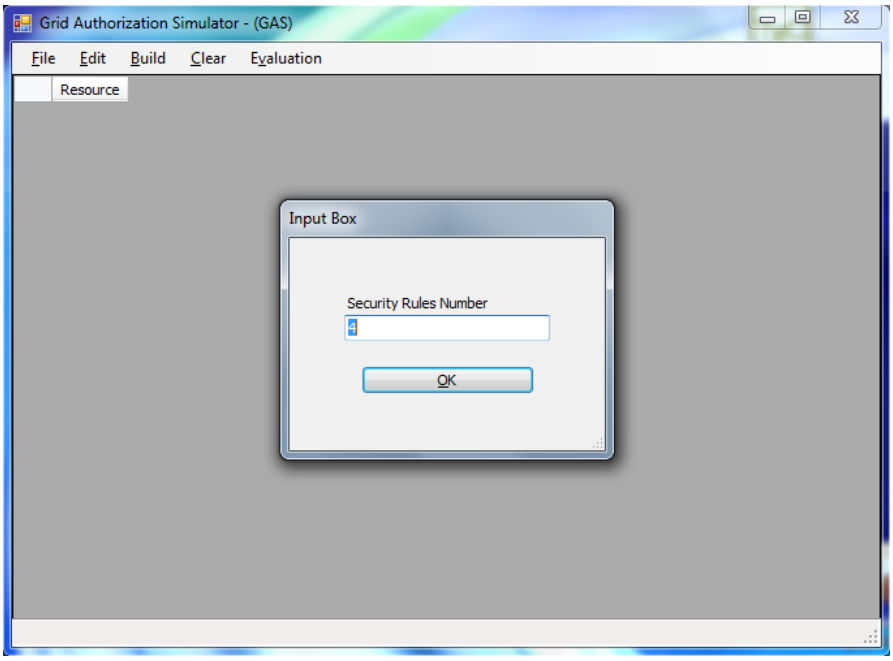

Figure 6.4: GAS - Specify number of security rules

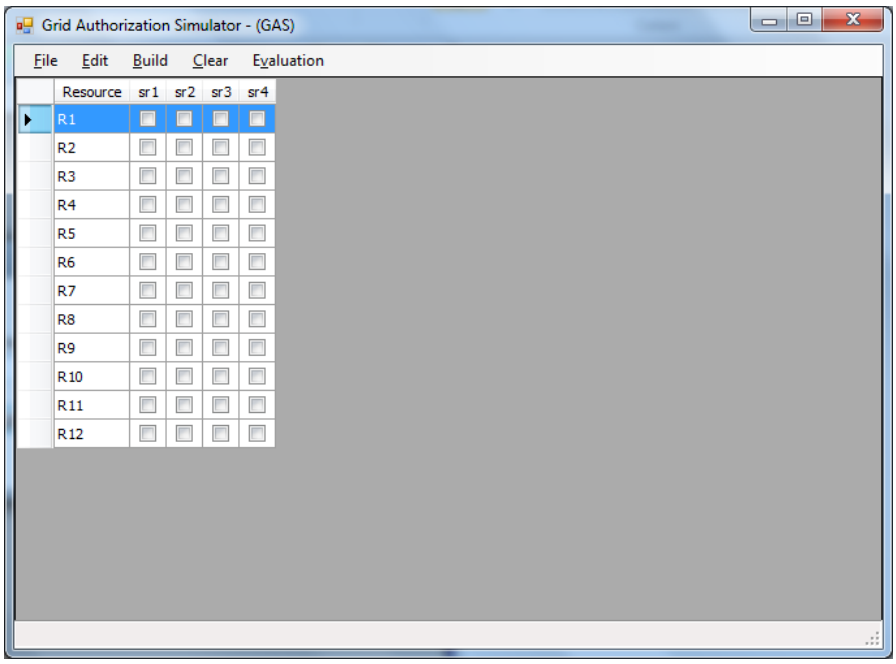

Figure 6.5: GAS - A blank security table

After that we need to setup our security policies or we may initialize the table randomly just for testing purpose from **Edit** menu as shown in Figure 6.6 to get the resulted security table shown in Figure 6.7.

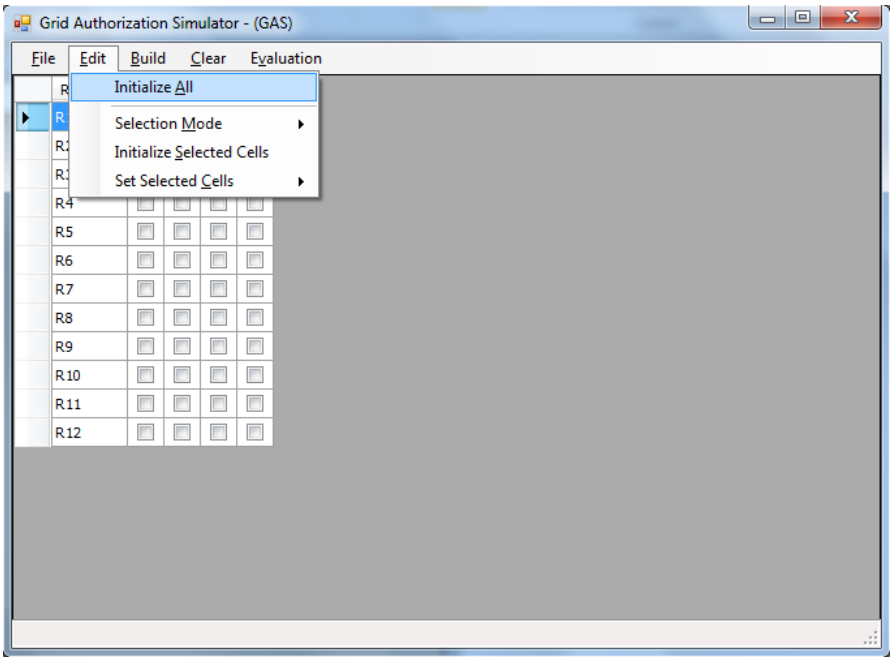

Figure 6.6: GAS - Initialize the security table

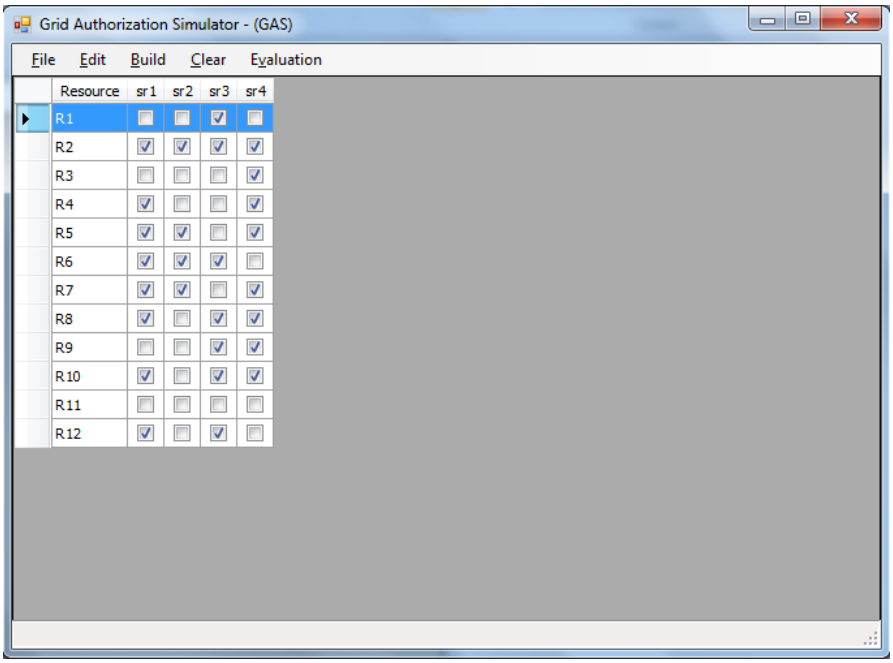

Figure 6.7: GAS - The initialized security table

The **File** menu also enables us to import the security table from a security policy file as shown in Figure 6.8. In this example, we are going to import the same security table of the grid environment discussed in Section 3.2.2 from a security policy file, output is shown in Figure 6.9.

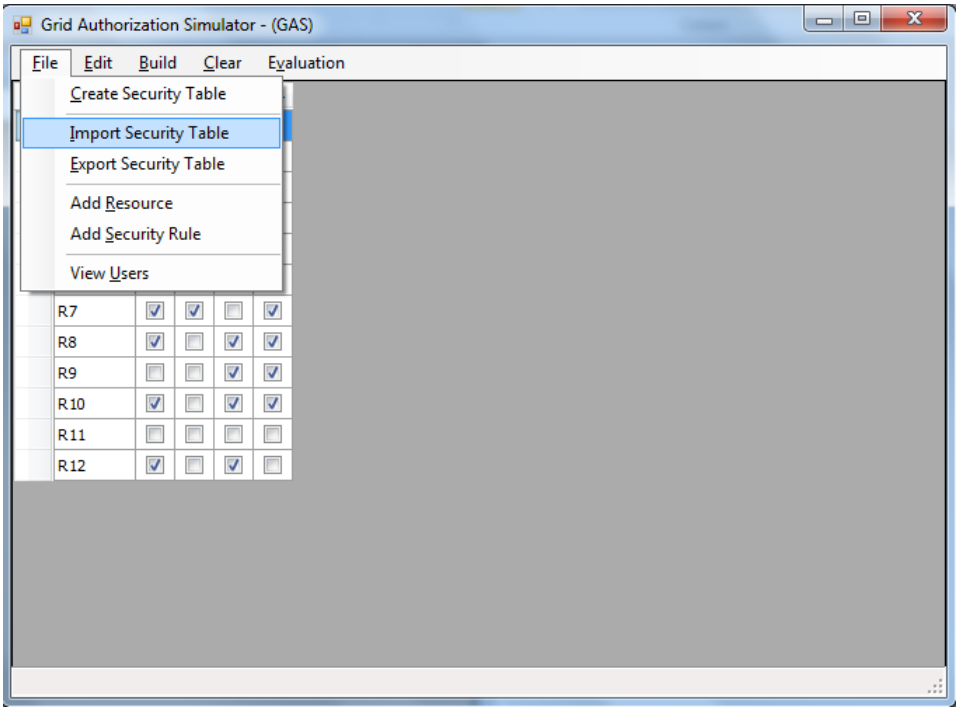

Figure 6.8: GAS - Import a security table

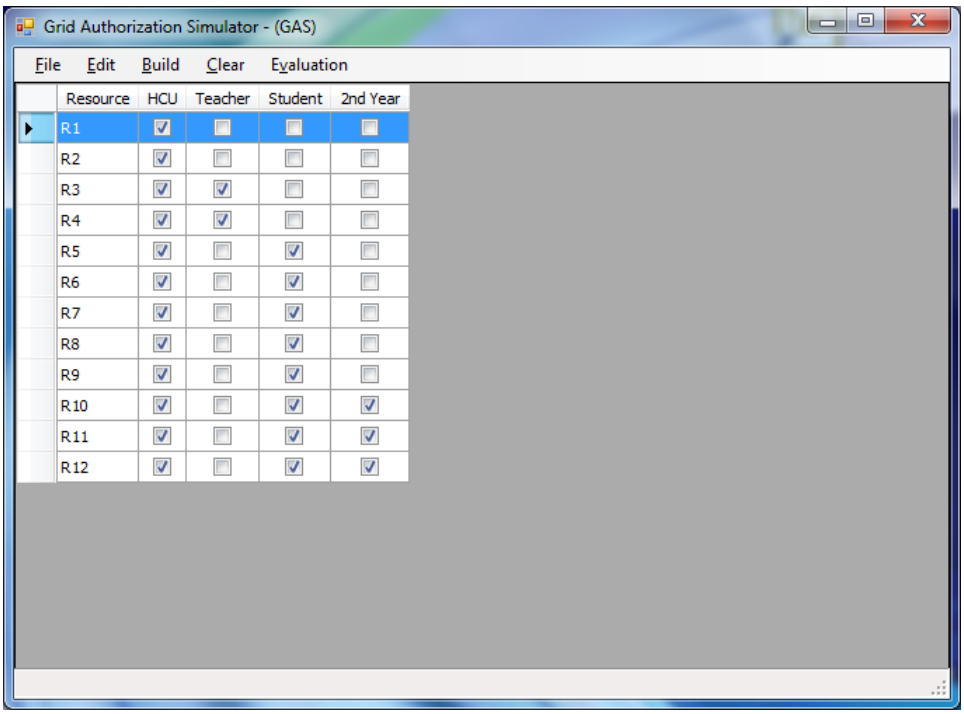

Figure 6.9: GAS - The imported security table

6.2.2 Build Menu

The **Build** menu enables us to navigate through different security policy storing mechanisms as shown in Figure 6.10.

6.2.2.1 BFA Submenu

From the **BFA** submenu, we can calculate the maximum complexity of the Brute Force Approach (BFA) as shown in Figure 6.11. We can also authorize a random user as shown in Figure 6.12. It shows what resources this user is authorized to access (Figure 6.13), and the complexity consumed in this authorization (Figure 6.14).

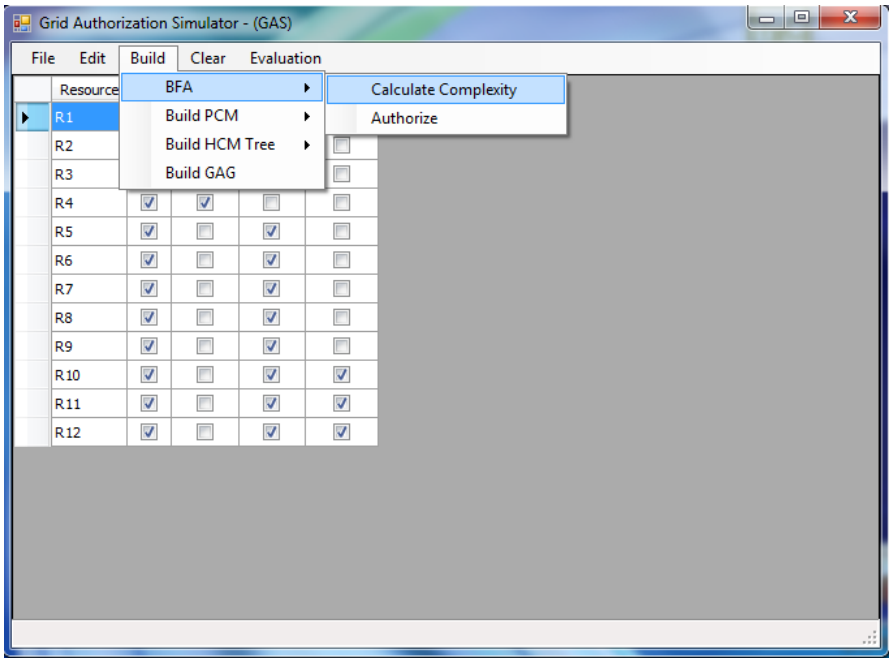

Figure 6.10: GAS - Build Menu

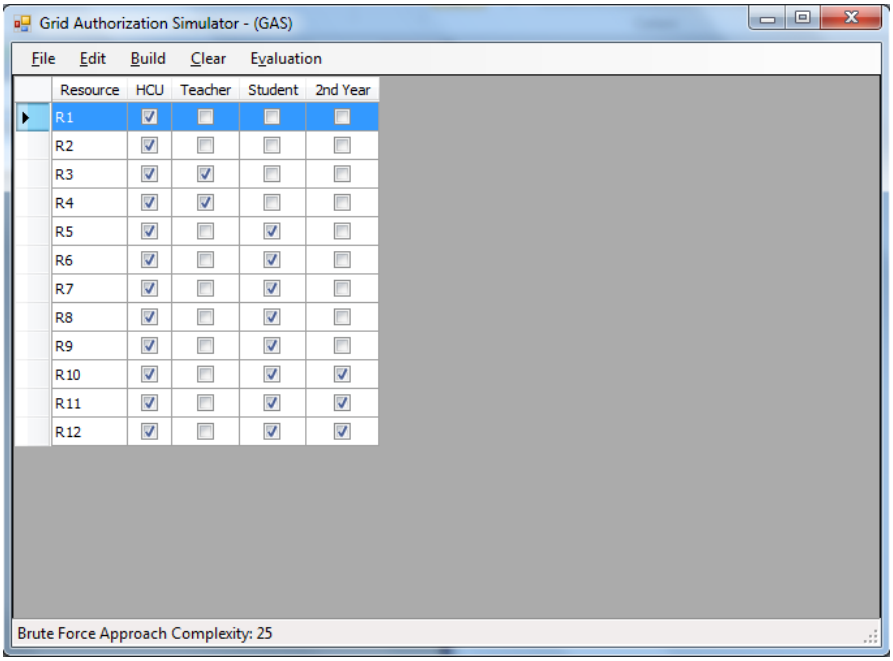

Figure 6.11: GAS - Calculate the Brute Force Approach Complexity

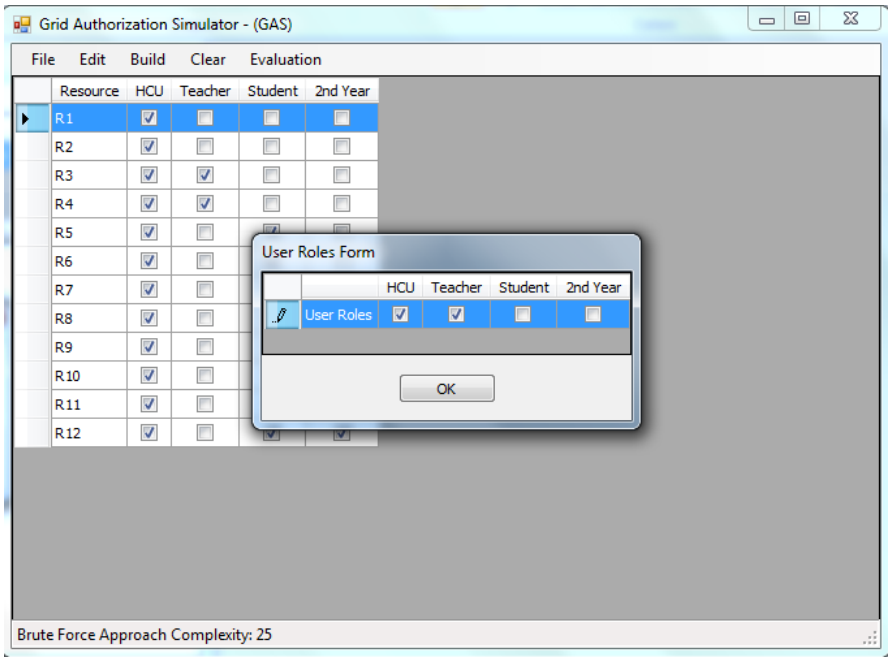

Figure 6.12: GAS - Authorizing a Random user using the Brute Force Approach

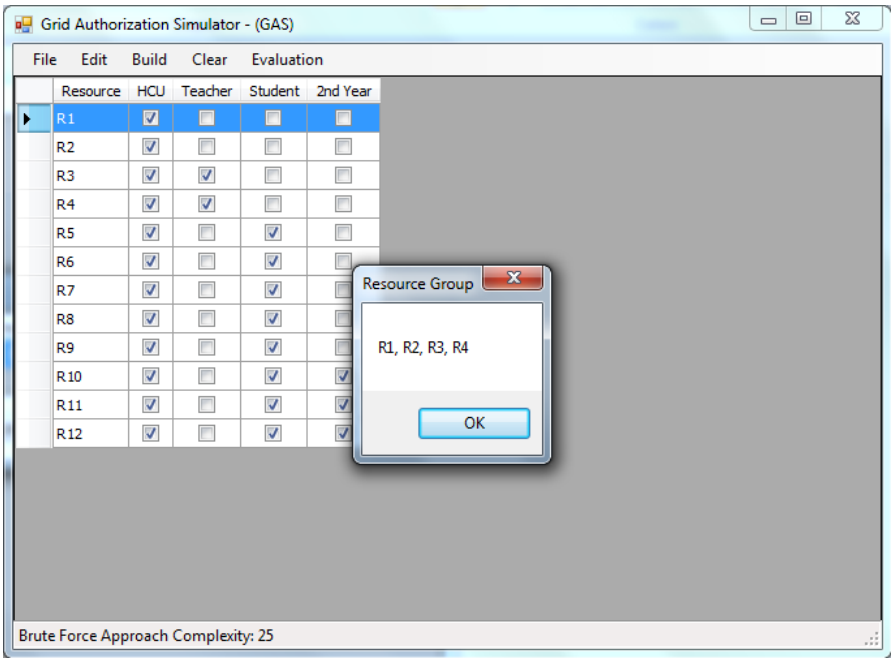

Figure 6.13: GAS - Showing the User's Authorized Resource Group (UARG)

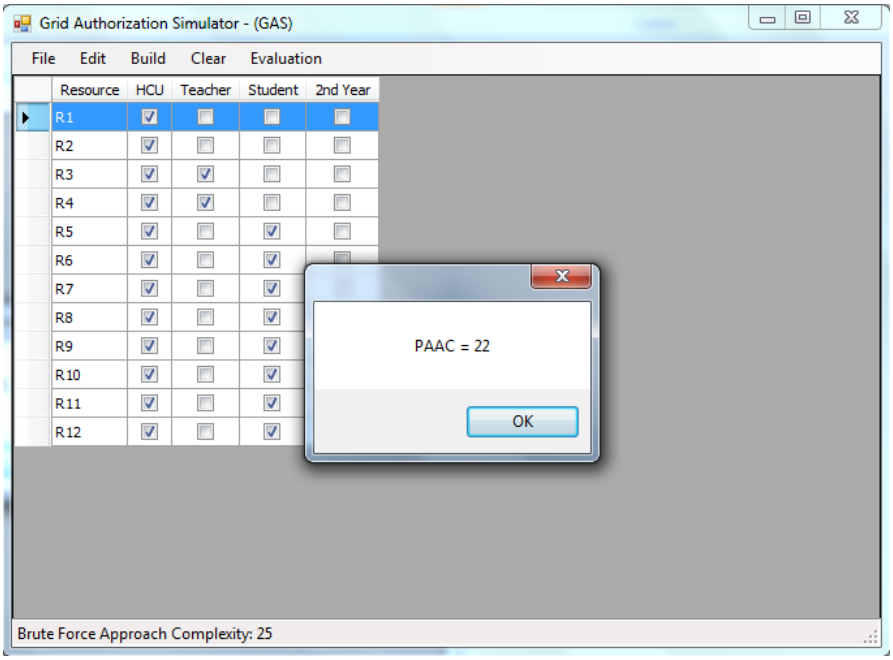

Figure 6.14: GAS - Showing PAC for the latest authorization process

6.2.2.2 Build PCM Submenu

From the **Build PCM** submenu (Figure 6.15), we can see how the Primitive Clustering Mechanism (PCM) stores the resources' security policies of the proposed grid environment. Figure 6.16 shows the PCM management window. The left TreeView component shows the resources' clusters. By navigating through different clusters, the GridView component, placed on the top right corner, shows the security policy of the selected cluster. The TreeView component, placed on the bottom right corner shows the resources belonging to the selected cluster.

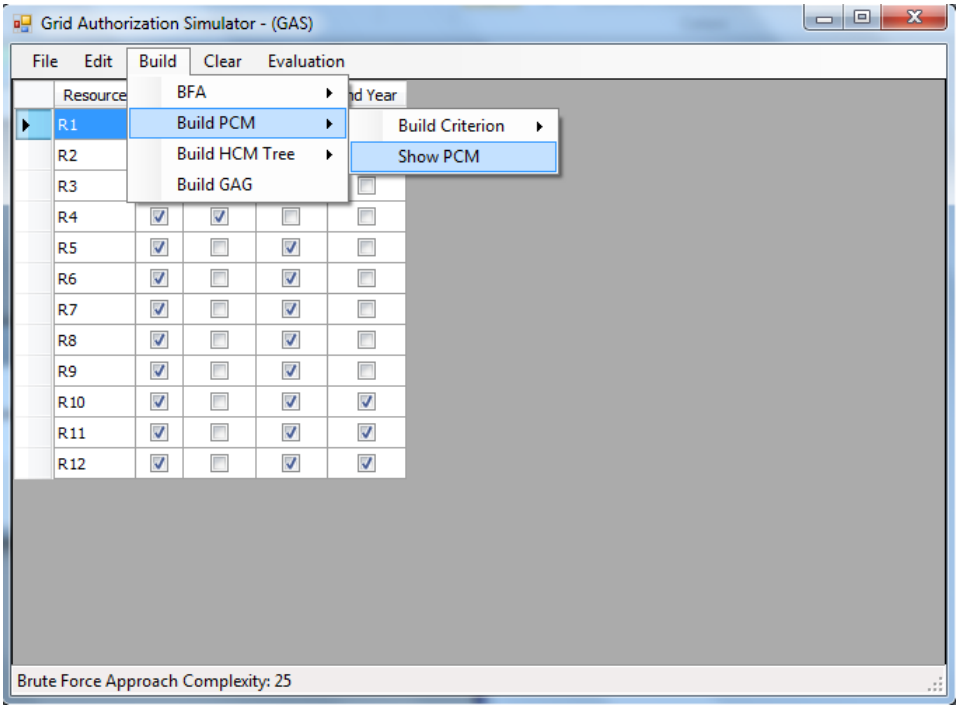

Figure 6.15: GAS - Show PCM Clusters

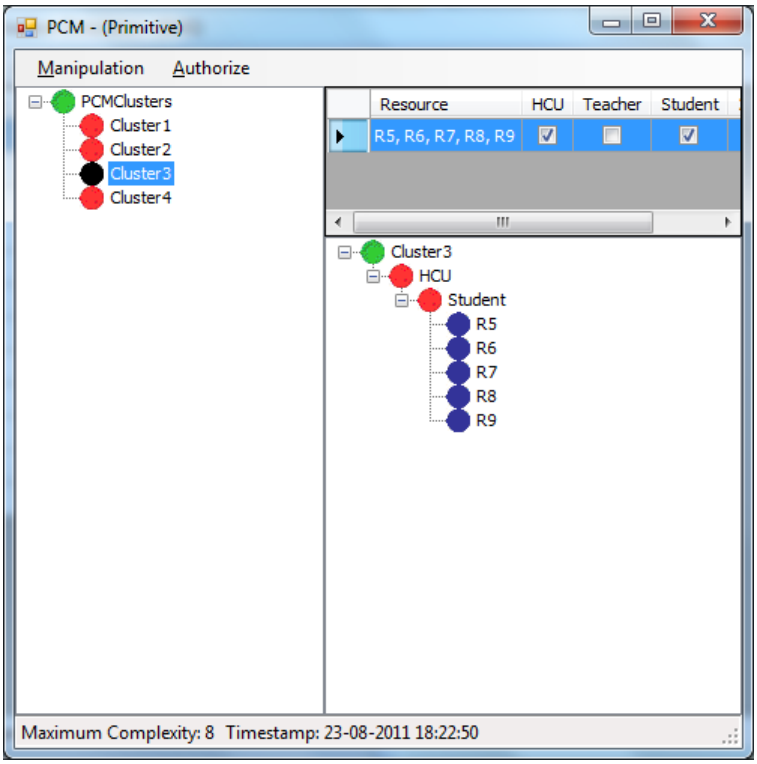

Figure 6.16: GAS - The Primitive Clustering Mechanism Management Window

Through the **Manipulate** submenu (Figure 6.17), we can Add a resource, Update the security policy of a particular resource or Delete a selected resource. Whatever the operation we select, the affected resource will be replaced into an appropriate cluster.

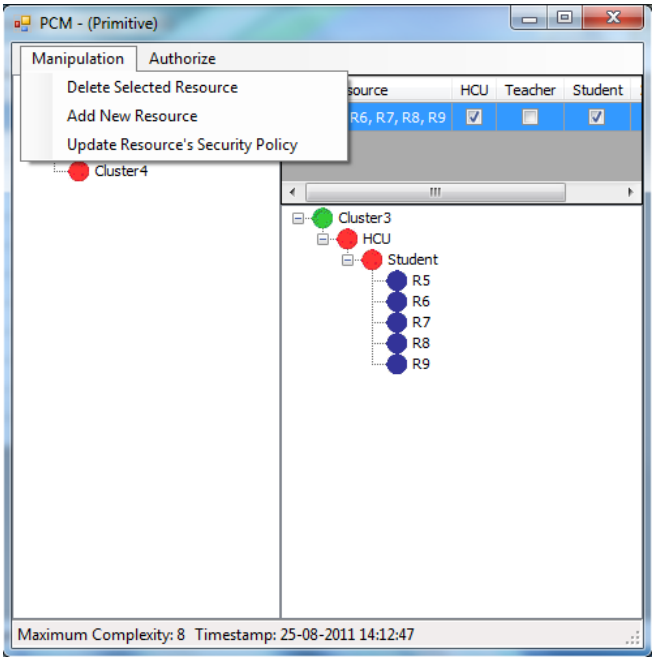

Figure 6.17: GAS - The PCM Manipulation Processes

Through the **Authorize** submenu (Figure 6.18), we can authorize a random user or an existing user in the database. If we choose to authorize a random user, then we have to specify his roles as shown in Figure 6.19. Then we will get the UARG as shown in Figure 6.20. Finally, the complexity of the authorization process will be displayed as shown in Figure 6.21.

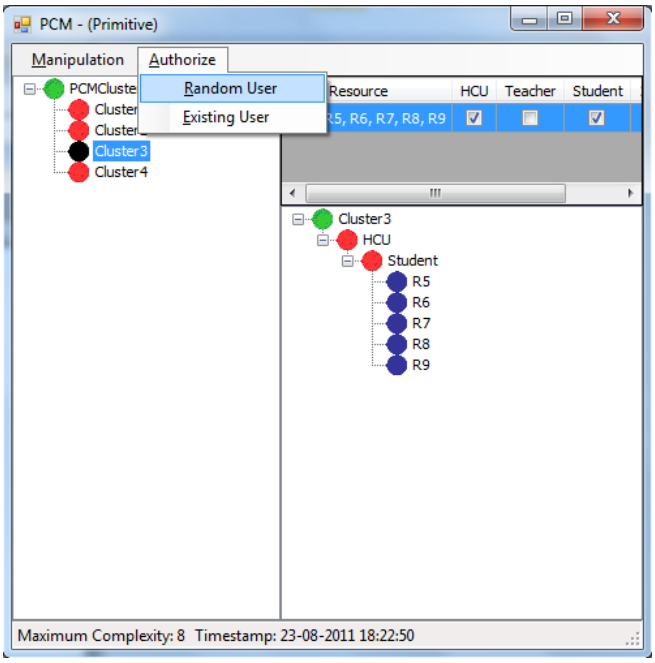

Figure 6.18: GAS - The PCM Authorize Submenu

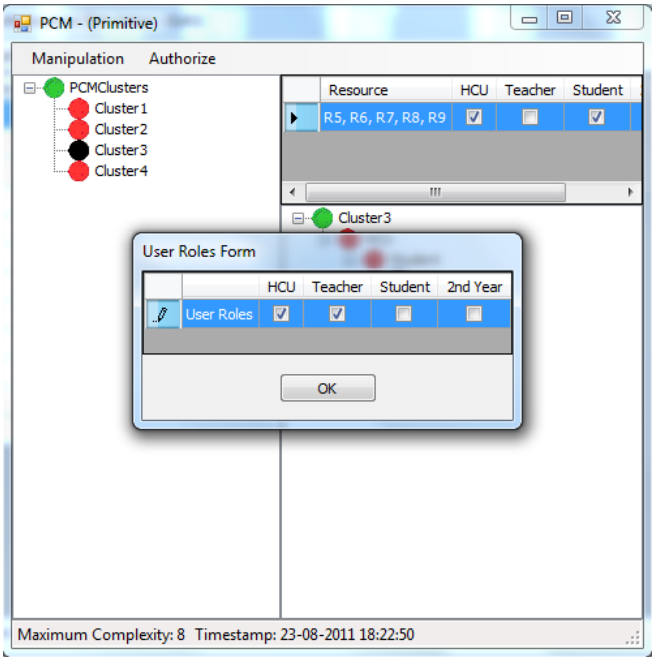

Figure 6.19: GAS - Specifying the roles of a random user

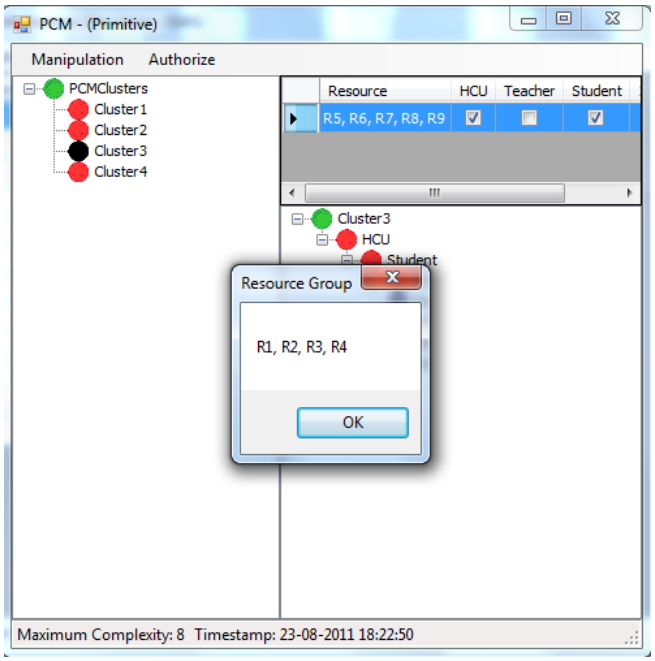

Figure 6.20: GAS - The User's Authorized Resource Group (UARG)

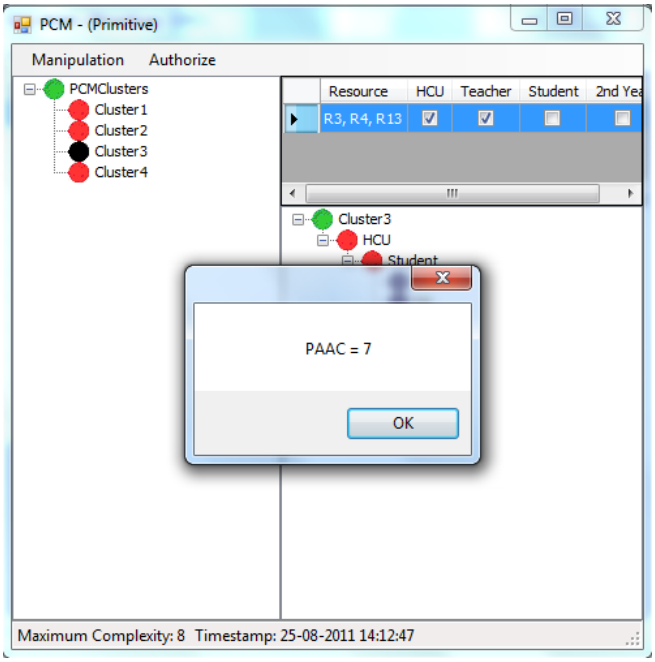

Figure 6.21: GAS - PCM PAAC for the latest authorization

Or we may authorize an existing user from the database. Click the Authorize button (Figure 6.22). A message box will display the PAAC (Figure 6.23). The UARG field will show the user's authorized resource group. The timestamp will be used for caching purpose, so that, in the future if we try to authorize the same user, the PAAC will be zero.

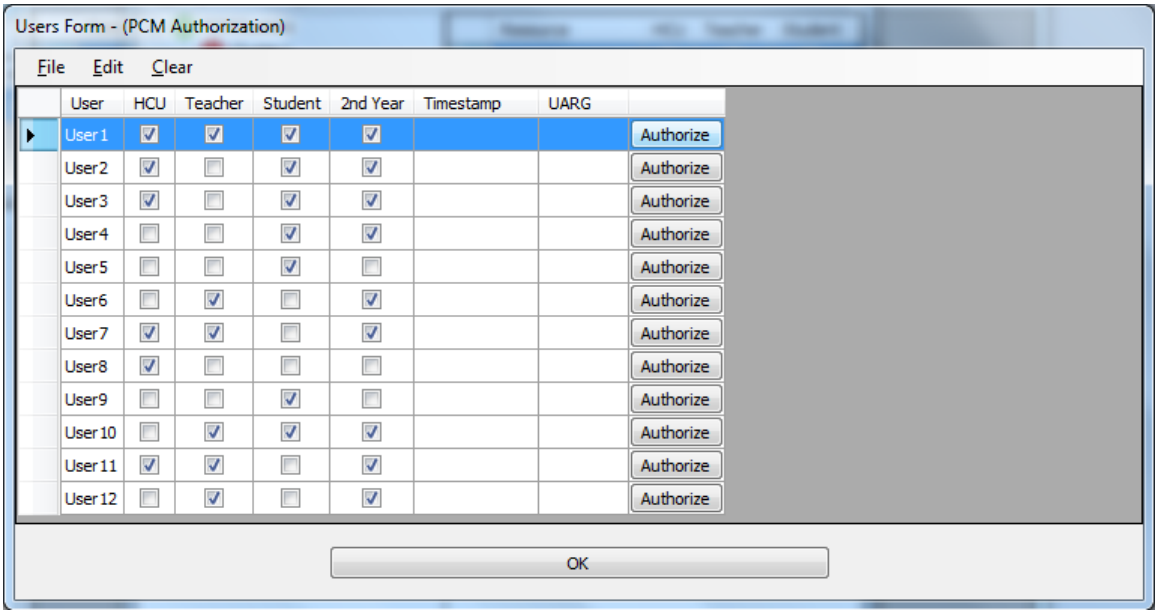

Figure 6.22: GAS - Authorizing an Existing user

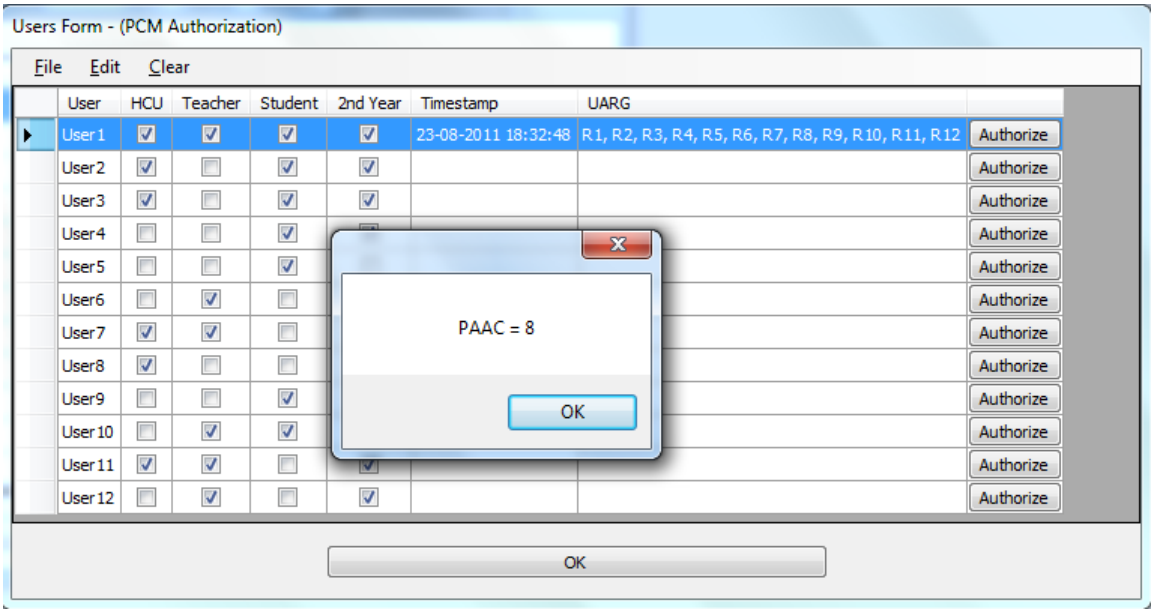

Figure 6.23: GAS - Showing the PAAC, the timestamp, and the cached UARG

6.2.2.3 Build HCM Submenu

Build HCM Tree submenu (Figure 6.24) allows us to build the HCM *decision tree* for the proposed grid environment. We can choose to prune the tree on NULL nodes or not. It also enables us to select the building criteria.

Figure 6.25 shows the HCM management window. The left TreeView component shows the HCM pruned *decision tree*. By navigating through different parent nodes (security rules), the GridView component, placed on the right side, shows the security table associated with the selected parent node.

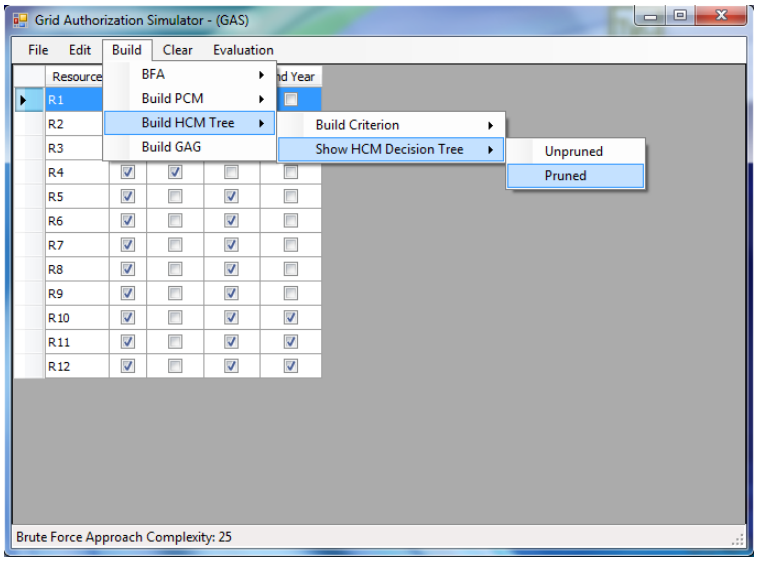

Figure 6.24: GAS - Build HCM Tree Submenu

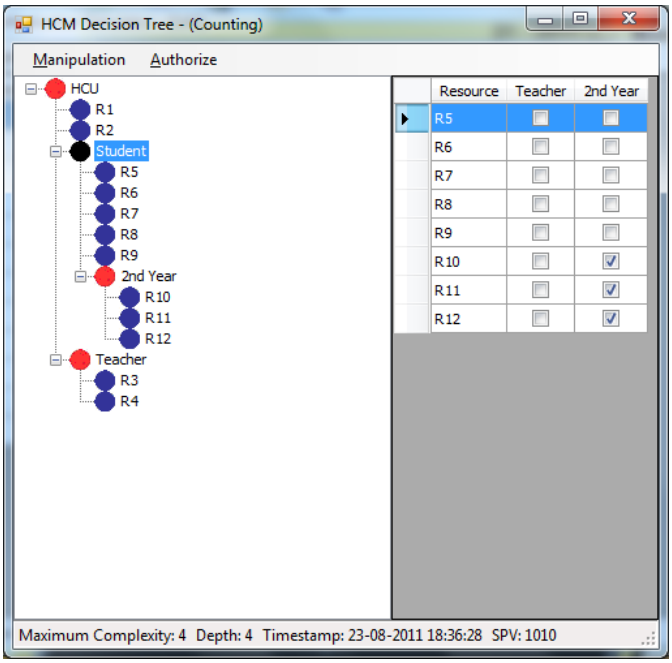

Figure 6.25: GAS - The HCM Management Window

In the HCM management window, we have the same functionalities as provided in PCM management window, that are Manipulation processes and Authorizing random or existing users. However, HCM uses TCM (Section 4.2.1) as well as HDCM (Section 4.2.2). This enables HCM to find the UARG more fast in case of any change. Figure 6.26 shows the value of PAAC = 4 for a user. After that, the user's roles have been altered by adding one extra role (2^{nd} *year*). Then, an authorization process is fired for the same user. Figure 6.27 shows how the new UARG is alloted to the user with the value of PAAC = 1.

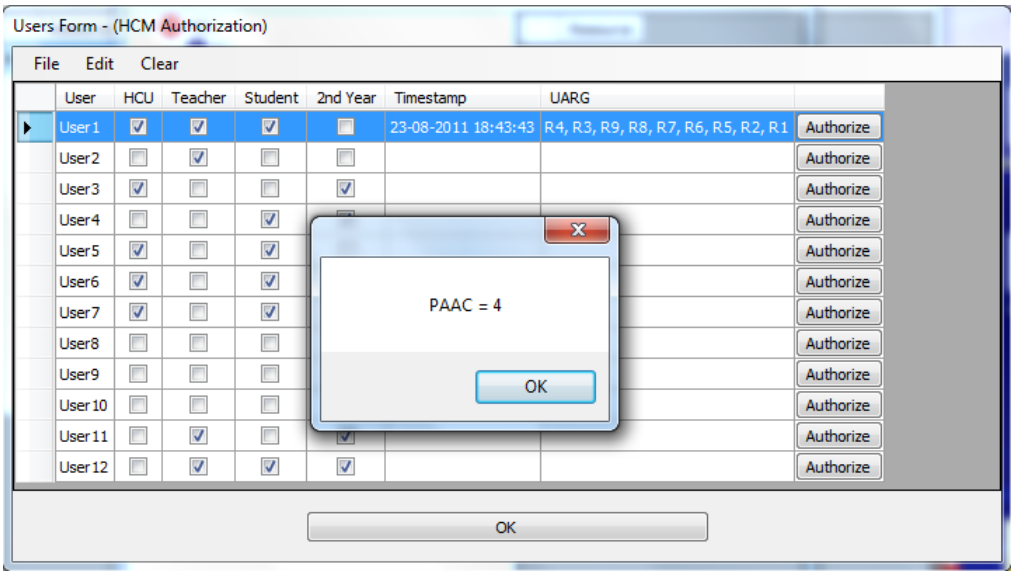

Figure 6.26: GAS - Authorizing an existing user

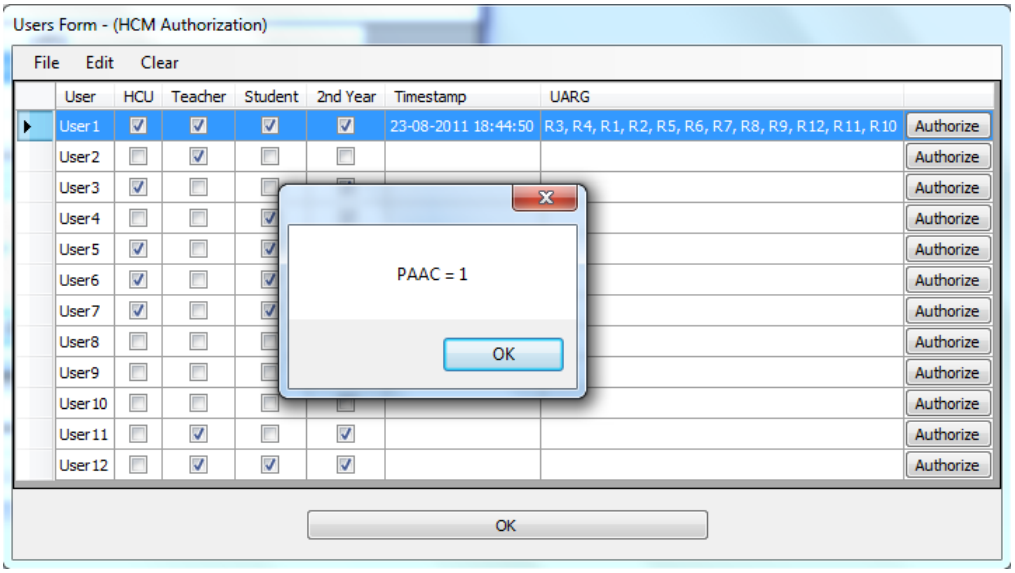

Figure 6.27: GAS - Reauthorizing the same user after altering his roles

6.2.2.4 Build GAG

The **Build** menu, enables us to build the Grid Authorization Graph (GAG). Figure 6.28 shows GAG for the proposed environment. The most left TreeView component shows the **SRV** vector. The middle TreeView component shows the core of GAG which is the HCM decision tree.

The Dependency Edges is linked in hidden mode as we can not show them on the TreeView component (A GraphView component is required which does not come in the IDE). So, we have used the node color as an indicator of the hidden Dependency Edges. Once we click on any security rule in the SRV, it will be highlighted with black color along with all correspondent security rules in the *decision tree* as an indicator of the Dependency Edges between them (Observe *Student* security rule in Figure 6.28). Figure 6.29 gives better view for another grid environment (Observe sr_4).

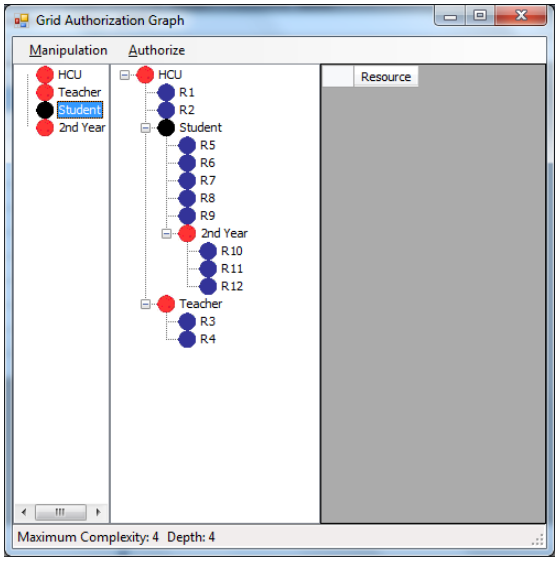

Figure 6.28: GAS - The Grid Authorization Graph

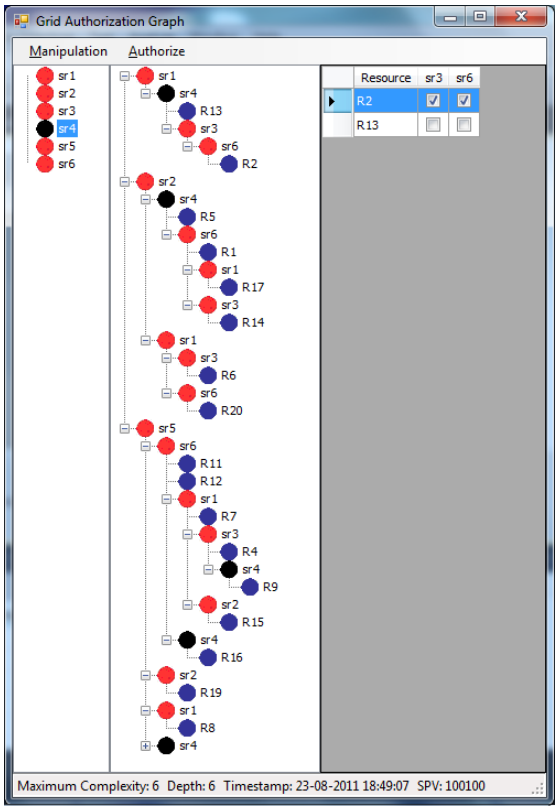

Figure 6.29: GAS - GAG for another grid environment

6.2.3 Evaluation Menu

The **Evaluation** menu (Figure 6.30) enables us to make a comparative study between different mechanisms.

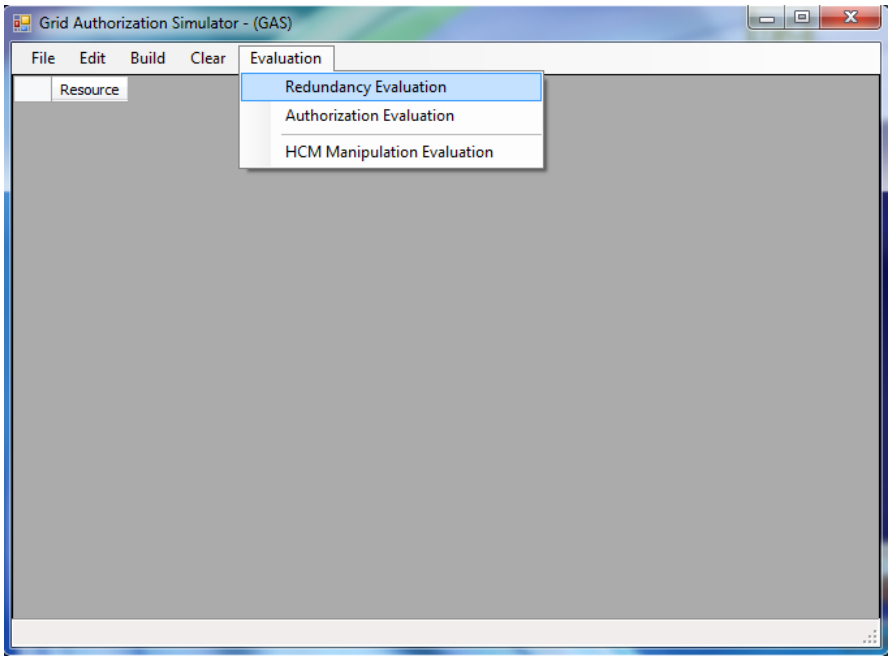

Figure 6.30: GAS - Evaluation & Comparative Studies

Figure 6.31 shows the available evaluation criteria. Once we select a set of criteria, we will be asked about the number of experiments we wish to conduct (Figure 6.32). Then we need to specify the number of resources exist in the simulated environment (Figure 6.33). Also we need to specify number of security rules (Figure 6.34). The simulator is going to process our request (Figure 6.35). Finally, we will get the evaluation results depicted on the chart (Figure 6.36). Section 7.2 provides a detailed result analysis of such experiments, here we are just showing how to use the simulator.

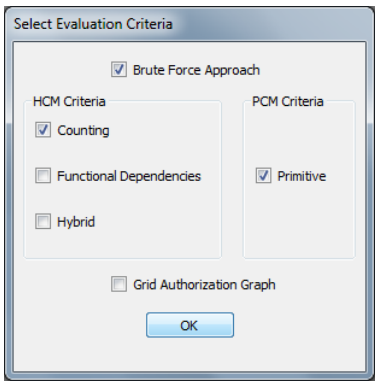

Figure 6.31: GAS - Select Evaluation Criteria

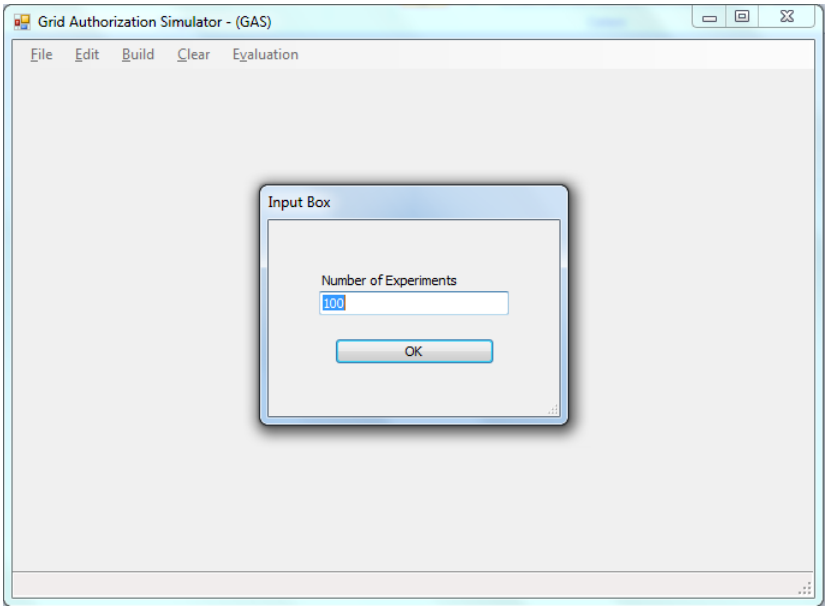

Figure 6.32: GAS - Enter number of experiments

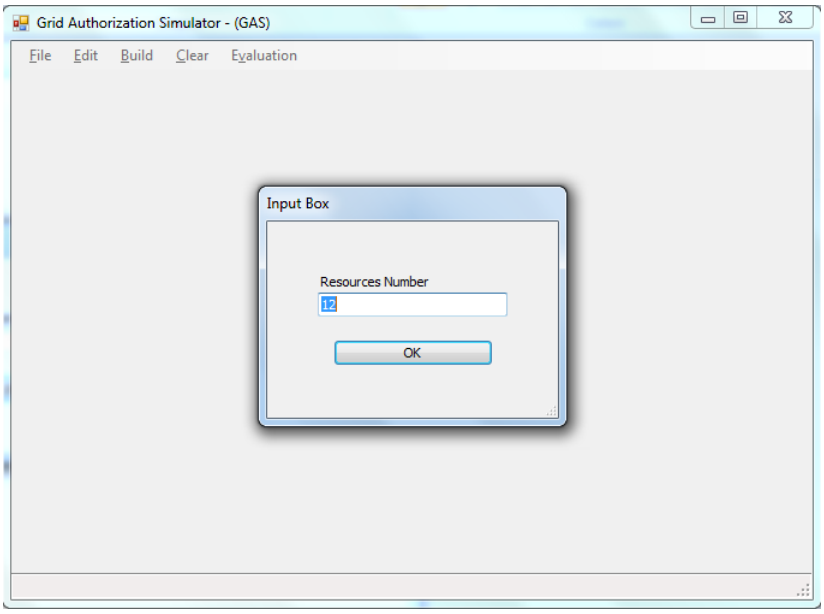

Figure 6.33: GAS - Enter number of resources

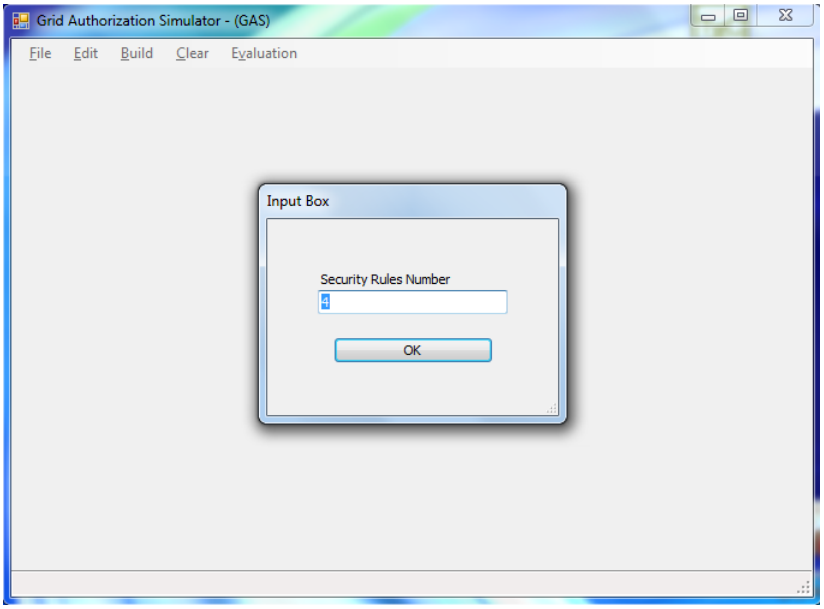

Figure 6.34: GAS - Enter number of security rules

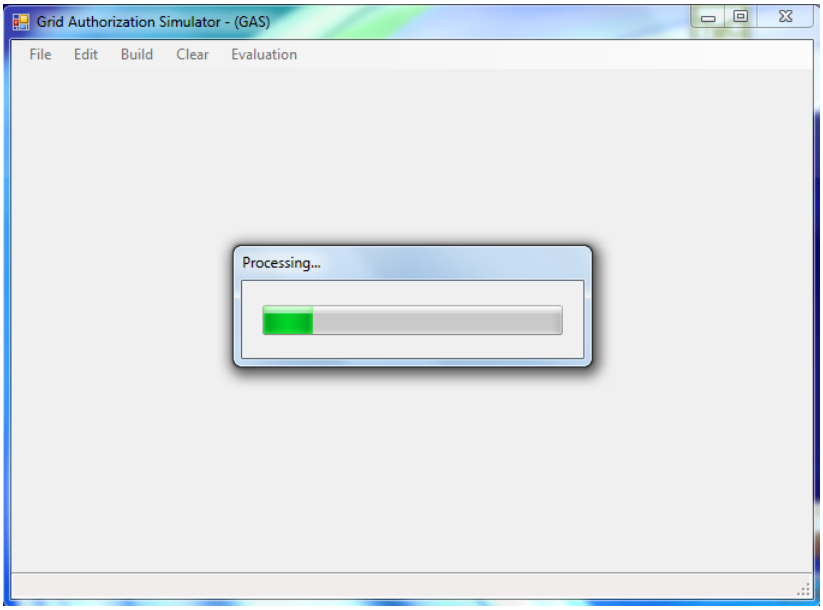

Figure 6.35: GAS - Processing your request

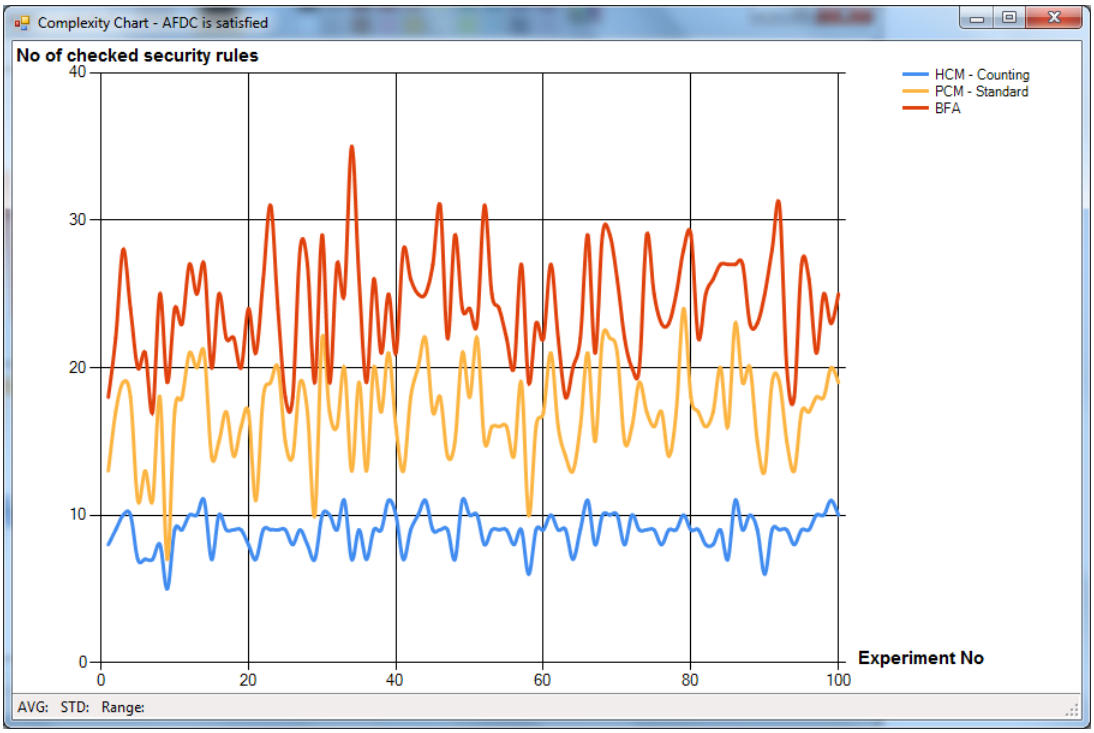

Figure 6.36: GAS - Evaluation Results

We can also compute the Average (AVG), the standard deviation (STD) and the Range of any of the chosen criteria by right click on the chart (Figure 6.37) and select the criterion (Figure 6.38).

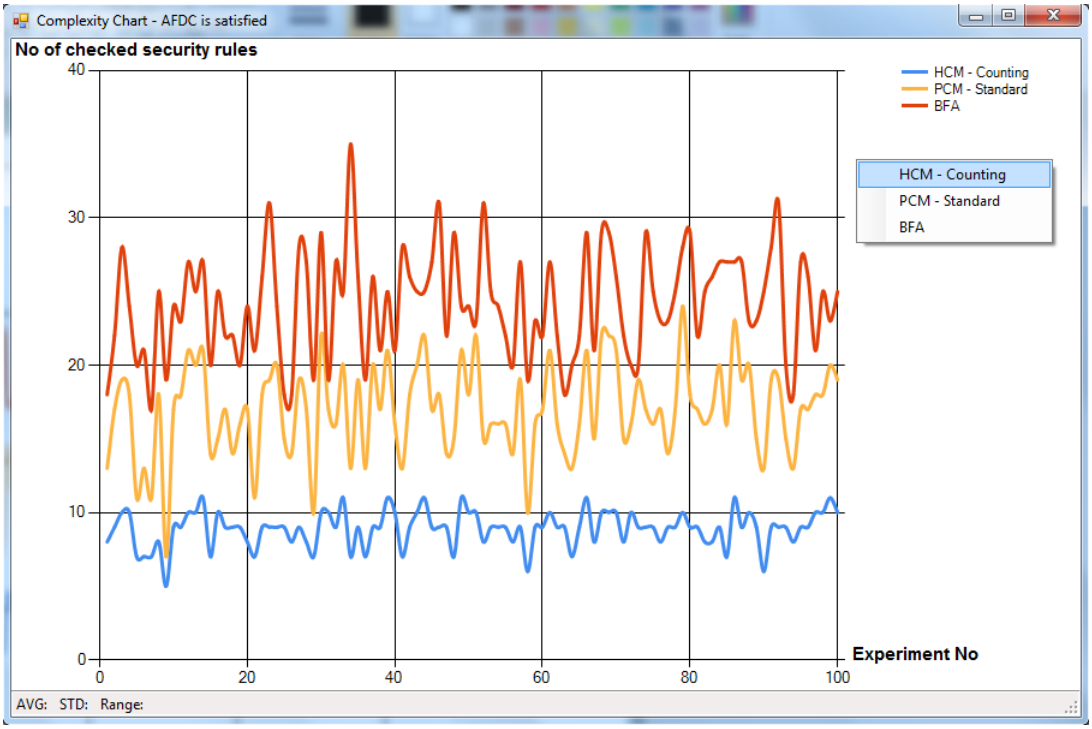

Figure 6.37: GAS - Select Criterion

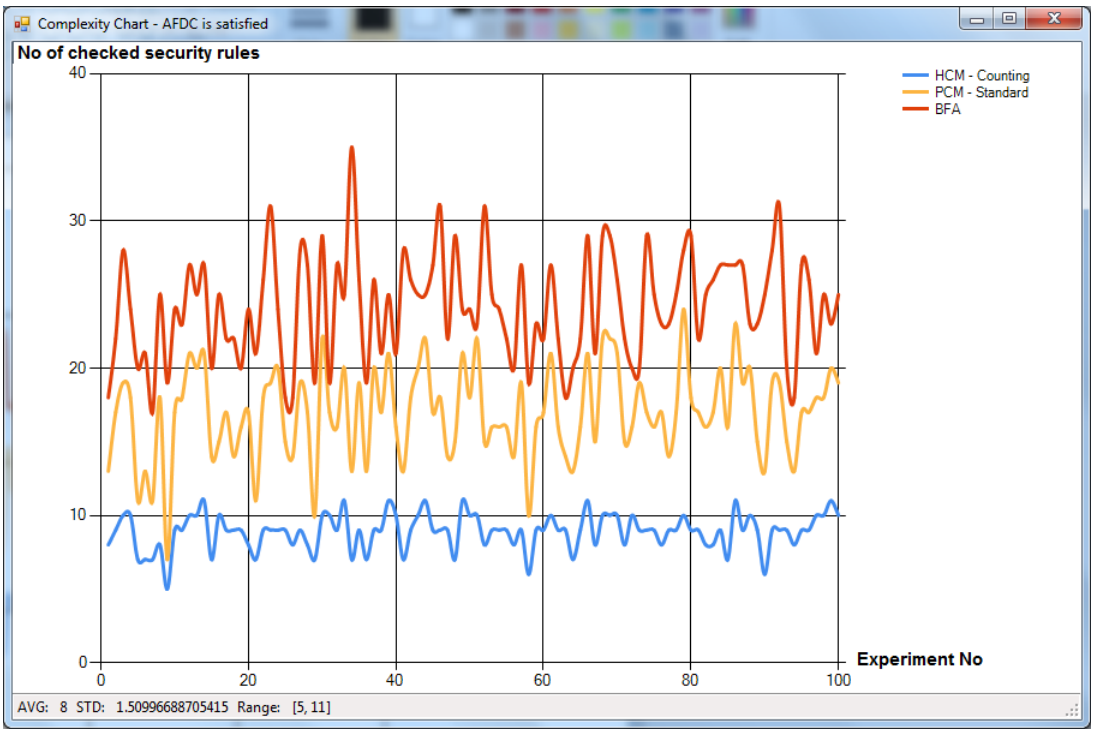

Figure 6.38: GAS - Compute the AVG, STD and Range of the selected criterion

Evaluation menu also allows us to study the efficiency of HCM *decision tree* against dynamic changes in the grid (Figure 6.39). Figure 6.40 compares the redundancy of the updated decision tree (Adding) with the redundancy of a new tree rebuilt for every change (Rebuilding). This study shows that they are almost similar, so HCM decision tree is still thought to be effective in dynamic environments.

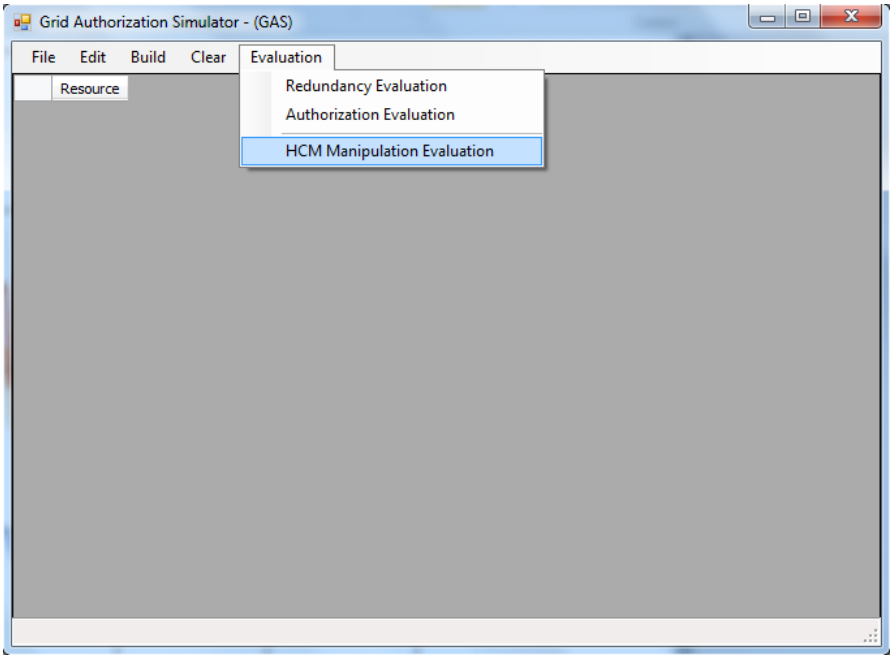

Figure 6.39: GAS - Select HCM Manipulation Evaluation

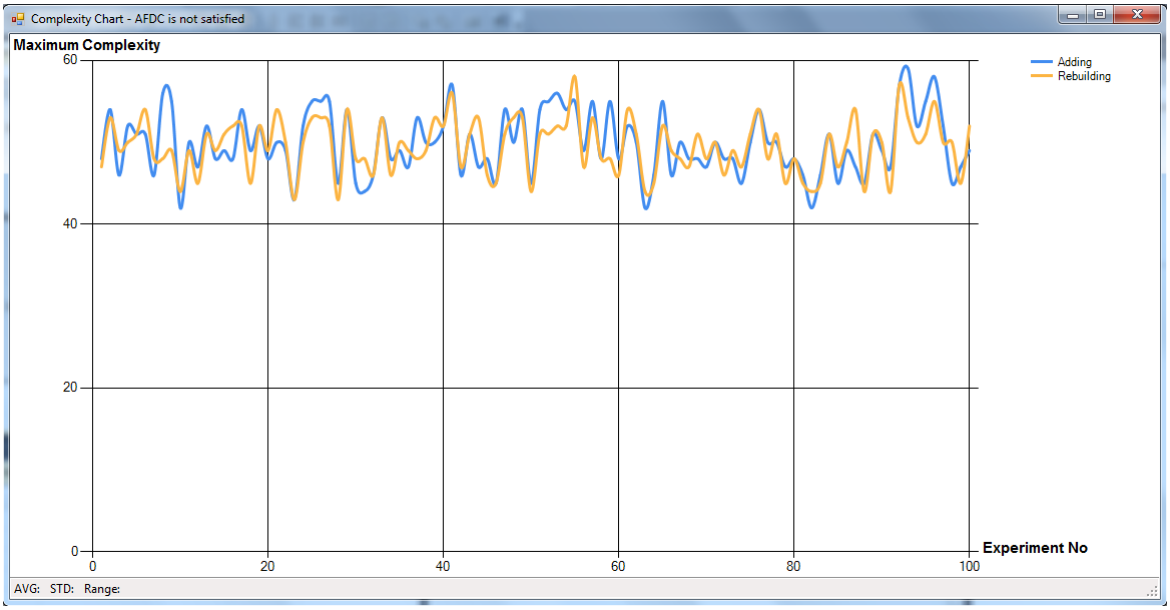

Figure 6.40: GAS - Chart Result

6.3 Technical Specification

Figure 6.1 shows an overall abstract view of HCM / GAG enabled authorization framework. In this subsection, we are going to provide more details about the technical specification of such framework. What data types are expected to be used, what different messages are going to be exchanged between the modules and the core implementation of HCM / GAG modules is also provided.

6.3.1 HCM Tree Building

In this subsection, we are going to show the implementation of the Counting Algorithm shown in Algorithm 1. It consists of different functions and procedures. The first procedure to be called is **BuildHCM** (See code 3) which is responsible for initiating the *decision tree* and call the **Manipulate** procedure. The **Manipulate** procedure is the core part, it is responsible for manipulating the tree nodes recursively (See code 4). Internally it calls **AddParentNode** (See code 5) and **AddResources** (See code 6) to build the tree structure. After that **BuildHCM** invoke the **Prune** procedure (See code 7) to prune the tree at NULL nodes.

Code 3 BuildHCM Procedure

```
public void BuildHCM(DataGridView dgv) {
    SecurityTable ST = new SecurityTable();
    ST.CopyFrom(dgv);

    DecisionTreeTV.Tag = CreateNodeInfo(ST, null);

    dgv.Tag = ST;

    DecisionTreeTV.Nodes.Clear();

    complexity = 0;

    int resNo = securityTable.GetResourcesCount();
    int srNo = securityTable.GetSecurityRulesCount();
    if ((resNo > 0) && (srNo > 0)) {
        TreeNode rootNode = DecisionTreeTV.Nodes.Add("NULL");
        rootNode.ImageIndex = 1;

        string firstRow = securityTable.Keys.First();
        SecurityTableRow SPV = new SecurityTableRow(securityTable[firstRow]);
        SPV.Zeroing();

        rootNode.Tag = CreateNodeInfo(securityTable, SPV);

        Manipulate(rootNode);

        if (pruned)
            Prune(rootNode);

        UpdateStatusStrip();
    }
}
```

Code 4 Manipulate Procedure

```
private void Manipulate(TreeNode TN) {
    SecurityTable ST = GetNodeSecurityTable(TN);
    AddResources(TN);

    if (ST.GetSecurityRulesCount() == 1) {
        string firstRow = ST.Keys.First();
        string sr = ST[firstRow].Keys.First();

        if (ST.Tag == 1)
            return;

        complexity++;

        SecurityTable CST = new SecurityTable();
        TreeNode child = AddNode(TN, sr, 0, CST);

        string[] rowsEn = ST.Keys.ToArray();
        foreach (string currentRow in rowsEn) {
            if (ST[currentRow][sr] != 0) {
                TreeNode res = AddNode(child, currentRow, 2, null);
                CST[currentRow] = new SecurityTableRow();
            }
        }
        return;
    }

    string chosenSecurityRule = ST.Evaluate(criterion);

    SecurityTable T1 = new SecurityTable(), T2 = new SecurityTable();
    ST.SplitSecurityTable(chosenSecurityRule, T1, T2);
    AddParentNode(TN, chosenSecurityRule, 0, T1);
    AddParentNode(TN, "NULL", 1, T2);
}
```

Code 5 AddParentNode Function

```
private TreeNode AddParentNode(TreeNode TN, string nodeName, int
imageIndex, SecurityTable ST) {
    int resNo = ST.GetResourcesCount();
    int srNo = ST.GetSecurityRulesCount();
    TreeNode child = null;
    if ((resNo > 0) && (srNo > 0)) {
        if (nodeName != "NULL")
            complexity++;
        child = AddNode(TN, nodeName, imageIndex, ST);
        Manipulate(child);
    }
    return child;
}
```

Code 6 AddResources Procedure

```
private void AddResources(TreeNode TN) {
    SecurityTable ST = GetNodeSecurityTable(TN);
    ST.Tag = 1;
    string[] rowsEn = ST.Keys.ToArray();
    foreach (string currentRow in rowsEn) {
        if (ST[currentRow].IsZero()) {
            TreeNode res = AddNode(TN, currentRow, 2, null);
            if (res.Level > maxDepth)
                maxDepth = res.Level;
        }
        else {
            ST.Tag = 0;
        }
    }
}
```

Code 7 Prune Procedure

```
void Prune(TreeNode TN) {
    TreeNode TNP = TN.Parent;
    if (TN.Text == "NULL") {
        // Recurse through the child nodes...
        for (int ndx = TN.Nodes.Count; ndx > 0; ndx--) {
            TreeNode node = TN.Nodes[ndx - 1];

            TN.Nodes.Remove(node);
            if (TNP != null)
                TNP.Nodes.Add(node);
            else
                DecisionTreeTV.Nodes.Add(node);

            Prune(node);
        }

        if (TNP != null)
            TNP.Nodes.Remove(TN);
        else DecisionTreeTV.Nodes.Remove(TN);
    }
    else {
        for (int ndx = TN.Nodes.Count; ndx > 0; ndx--) {
            TreeNode node = TN.Nodes[ndx - 1];
            Prune(node);
        }
    }
}
```

6.3.2 User Record

Once a user submits a request, his/her record is passed to the RAP. **UserRecord** data type is implemented as a structure of different fields listed as the following: **userRoles**, **Timestamp**, **PNL** and **UARG**.

The **userRoles** is of *SecurityTableRow* data type that is shown in code 8. It is a Dictionary $< string, int >$ based data type enhanced with various functionalities like indexing, CalculateComplexity, CopyFrom, ExtractRules, View on a DataGrid, IsEqual and Initialize.

The **Timestamp** field is a simple *DateTime* data type used to hold the timestamp when the *decision tree* was parsed for the user. It is used for implementing the TCM.

The **PNL** is a simple *ArrayList* data type used to hold the Parent Node List of the resources alloted to the user in the last authorization. It is also used for implementing the TCM as well as the HDCM.

UARG is the user's authorized resource group which will hold the result of authorization from the RAP.

Code 8 SecurityTableRow Datatype

```
using Enumerator = System.Collections.IEnumerator;
using SecurityTableRowType = Dictionary<string, int>;

public class SecurityTableRow {
    private SecurityTableRowType STR;
    public int Tag = 0;

    public SecurityTableRow() {
        this.STR = new SecurityTableRowType();
    }

    public SecurityTableRow(SecurityTableRow oldRow) {
        this.STR = new SecurityTableRowType(oldRow.STR);
    }

    public int this[string res] {
        set { STR[res] = value; }
        get { return STR[res]; }
    }

    public SecurityTableRowType.KeyCollection Keys {
        get { return STR.Keys; }
    }

    public SecurityTableRowType.ValueCollection Values {
        get { return STR.Values; }
    }

    public int Count {
        get { return STR.Count; }
    }
```

```
public void RemoveColumn(string name) {
    STR.Remove(name);
}

public bool ContainsKey(string sr) {
    return STR.ContainsKey(sr);
}

public void Initialize() {
    string[] securityRules = STR.Keys.ToArray();
    foreach (string sr in securityRules) {
        STR[sr] = (GAS.Randoms.GenerateBoolean()) ? 1 : 0;
    }
}

public void Zeroing() {
    string[] securityRules = STR.Keys.ToArray();
    foreach (string sr in securityRules) {
        STR[sr] = 0;
    }
}

public bool IsZero() {
    string[] securityRules = STR.Keys.ToArray();
    foreach (string sr in securityRules) {
        if (STR[sr] != 0)
            return false;
    }

    Tag = 1;

    return true;
}
```

```
public bool IsEqual(SecurityTableRow TR) {
    string[] securityRules = STR.Keys.ToArray();
    foreach (string sr in securityRules) {
        if (STR[sr] != TR[sr])
            return false;
    }

    return true;
}

public override string ToString() {
    string result = "";
    string[] securityRules = STR.Keys.ToArray();
    foreach (string sr in securityRules)
    {
        result += (STR[sr] == 1)? "1": "0";
    }

    return result;
}

public bool Dominates(SecurityTableRow SPV) {
    string sURV = this.ToString();
    string sSPV = SPV.ToString();

    for (int i = 0; i < sSPV.Length; i++)
        if (sSPV[i] == '1')
            if (sURV[i] != '1')
                return false;
    return true;
}
```

```
public int CalculateComplexity() {
    int complexity = 0;
    Enumerator columnsEn = STR.Keys.GetEnumerator();
    while (columnsEn.MoveNext()) {
        string currentColumn = (string)columnsEn.Current;
        if(STR[currentColumn] == 1)
            complexity++;
    }

    return complexity;
}

public void CopyFrom(DataGridViewRow dataGridViewRow) {
    DataGridView dataGridView = dataGridViewRow.DataGridView;
    int colNum = dataGridViewRow.Cells.Count;

    string res = dataGridViewRow.Cells[0].Value.ToString();

    for (int j = 1; j < colNum; j++) {
        if (dataGridView.Columns[j].CellType.Name ==
            "DataGridViewCheckBoxCell") {

            string sr = dataGridView.Columns[j].HeaderText;
            STR[sr] = Convert.ToInt32(
                      dataGridViewRow.Cells[j].Value);
        }
    }
}
```

```
    public void ExtractRules(DataGridView dataGridView) {
        DataGridViewOperations dataGridViewOperations =
                    new DataGridViewOperations(dataGridView);

        dataGridViewOperations.Clear();

        Enumerator columnsEn = STR.Keys.GetEnumerator();
        while (columnsEn.MoveNext()) {
            string currentColumn = (string)columnsEn.Current;
            dataGridViewOperations.AddColumn(currentColumn);
        }
    }

    public void View(DataGridView dataGridView, string rowName) {
        DataGridViewOperations dataGridViewOperations =
                    new DataGridViewOperations(dataGridView);

        DataGridViewRow dgvr = null;

        dgvr = dataGridViewOperations.AddRow(rowName);

        int j = 1;
        Enumerator columnsEn = STR.Keys.GetEnumerator();
        while (columnsEn.MoveNext()) {
            string currentColumn = (string)columnsEn.Current;
            // Look at it later...
            dgvr.Cells[j].Value = (STR[currentColumn] == 1) ?
                                  true : false;
            j++;
        }
    }
}
```

6.3.3 HCM Search Engine

Once the user's record is submitted to the RAP, it initiates an authorization process in the HCM Search Engine. The authorization code is shown in code 9. It checks if the user has recently passed through an authorization process, then it applies TCM & HDCM to get his/her resource group with simple up-pass to the decision tree (See code 10), otherwise it performs a new authorization. Once the process is done, the **PNL**, **Timestamp** along with the **UARG** is returned back to the RAP for making entries in the **userRecord**. It is the responsibility of the PEP to restrict user's access to the grid resources as per his/her record.

GAG technical specification is similar to HCM, as the HCM *decision tree* is still at the core of GAG. The only difference is that, GAG uses an SRV vector which is nothing but an **ArrayList** of redundant parent nodes. Also GAG has to perform BFS after authorizing each parent node as refered in Section 5.2.2.

Code 9 HCM Search Engine Authorization Process

```
public void Authorize(SecurityTableRow userRoles, string timestamp,
ref ArrayList PNL, ref string UARG, ref int UAC) {
        if (timestamp != "") {
                Object[] aux = PNL.ToArray();
                foreach (TreeNode TN in aux) {
                        TreeNode n = TN;
                        SecurityTableRow SPV = GetNodeSPV(n);
                        while (!userRoles.Dominates(SPV)) {
                                n = n.Parent;
                                if (n == null)
                                        break;
                                SPV = GetNodeSPV(n);
                        }

                        if (n != TN) {
                                PNL.Remove(TN);
                                bool add = true;
                                foreach (TreeNode T in PNL) {
                                        if(T.FullPath.
                                             StartsWith(n.FullPath))
                                                add = false;
                                }
                                if (add)
                                        PNL.Add(n);
                                else
                                        continue;
                        }
                        UpPass(n, userRoles, timestamp, ref PNL,
                               ref UARG, ref UAC);
                }
```

```
                if (UARG.Length > 0)
                        UARG = UARG.Substring(0, UARG.Length - 2);
                else
                        UARG = "This user is not authorized to access
                                any resource.";
        }
        else {
                PNL = new ArrayList();
                Authorize(userRoles, ref UARG, ref PNL, ref UAC);
        }
}
```

Code 10 Up Pass of the decision tree

```
private void UpPass(TreeNode TN, SecurityTableRow userRoles, string
timestamp, ref ArrayList PNL, ref string UARG, ref int UAC) {

    TreeNodeCollection TNC = (TN == null)?
                            DecisionTreeTV.Nodes:TN.Nodes;

    foreach (TreeNode n in TNC) {
        if (n.ImageIndex == 2) {
            if (!UARG.Contains(n.Text + ", "))
                UARG += n.Text + ", ";
            continue;
        }

        HCMNodeInfo ni = GetNodeInfo(n);
        if (ni.Timestamp.CompareTo(Convert.ToDateTime(timestamp))
            > 0) {
            if (NodeAuthorization(n, userRoles, ref UARG, ref PNL,
                                  ref UAC))
                if (PNL.Contains(TN))
                    PNL.Remove(TN);
            continue;
        }

        if (userRoles.Dominates(ni.SPV)) {
            bool existed = false;
            foreach (TreeNode T in PNL) {
                if (T.FullPath.StartsWith(n.FullPath)) {
                    existed = true;
                    break;
                }
            }
```

```
            if (!existed) {
                NodeAuthorization(n, userRoles, ref UARG, ref PNL,
                                  ref UAC);
                if (PNL.Contains(TN))
                    PNL.Remove(TN);
                continue;
            }
        }
    }

    if (TN != null)
        UpPass(TN.Parent, userRoles, timestamp, ref PNL, ref UARG,
               ref UAC);
}
```

6.3.4 Summary

This chapter shows how GAG components can be embedded in GT4 authorization framework. The user manual of easy to use Grid Authorization Simulator and the technical specification of HCM or GAG enabled authorization framework is also proposed. All the implementation is done in C#.

Chapter 7

Thesis Summary

7.1 Summary of Contributions

A novel grid authorization enhancement has been developed by introducing the Hierarchical Clustering Mechanism (HCM). HCM is a kind of RBAC specifically designed to suit in legacy systems. It efficiently reduces the redundancy in checking security rules compared to the existing mechanisms such as the Brute Force Approach and the Primitive Clustering Mechanism (PCM).

HCM is slightly expensive processes in terms of memory consumption and computations required to build the decision tree. Thus HCM is not economical for small systems where it may cause an extra overhead which can lead to an authorization bottleneck. For such systems, the Rough Set based PCM (RSPCM) is proposed. RSPCM is an alternative to HCM for those environments that are less dynamic and have less resources, and it is a superior to normal PCM in terms of redundancy.

Moreover, Grid is a heterogeneous environment, thus the access control should work across domains. For that, The Distributed HCM (DHCM) is proposed with 3 different models to suit different grid scenarios.

As grid has large number of users who join and leave the grid dynamically, user scalability issues have to be addressed. Some caching mechanisms have been introduced like the Temporal Caching Mechanism (TCM), and the Hamming Distance Caching Mechanism (HDCM) to speed up the authorization process so that the system can scale up for more number of users. The concurrent HCM has been introduced to parallelize the authorization process to serve more than one user at a time.

All the above mentioned limitations and solutions make it convincing to consider HCM as an efficient access control mechanism to be embedded in the present popular grid authorizing systems like VOMS, Akenti, PERMIS, etc. However, HCM still has some redundancy; moreover it is not easy to describe the OR-based security rules in HCM. The Grid Authorization Graph (GAG) has been developed to eliminate HCM redundancy and to accommodate the OR-based security rules.

Our schema is scalable since it is dependent on roles and not on the number of users. It has no computational overhead to find the user's authorized resource group since it uses effective caching mechanisms such as TCM and HDCM. It is developed by considering all dynamic aspects of the Grid. This makes our scheme better than others in terms of many performance parameters. A Grid Authorization Simulator has been developed to experiment various authorization mechanisms and attain a comparative study between the existing mechanisms and our proposed mechanisms.

7.2 Results Analysis

In this section, we are going to analyze the overall experience with the present Grid access control mechanisms with compare to our proposed mechanisms through a simulated experiments. A grid environment of 30 resources and 7 security rules, 100 different authorization processes have been initiated. For each authorization process, the posterior analysis of the *Brute Force Approach (BFA)*, the *Primitive Clustering Mechanism (PCM)*, the *Hierarchical Clustering Mechanism (HCM)* and the *Grid Authorization Graph (GAG)* has been done and depicted in Table 7.1 and Figure 7.1.

The average number of the checked security rules in BFA was 49 with a standard deviation 18.711; while it was 45 in PCM with a standard deviation 17.058. The complexity range ([min, max] number of checked security rules over all experiments) of BCA was [30, 109], while it was [28, 101] for PCM.

PCM did not show any considerable improvements over BFA as the possibility of having two identical rows in a security table with 30 rows and 7 columns is low (since the number of possible distinct rows with 7 columns is $2^7 = 128$). Thus, in this simulated environment, PCM shows similar performance as of BFA.

HCM and GAG performs much better with compare to the existing mechanisms BFA and PCM. The average number of the checked security rules in HCM was 8 with a standard deviation 6.229; while it was 5 in GAG with a standard deviation 1.741. The complexity range of HCM was [3, 32], while it was [3, 7] for GAG.

Table 7.1: Experiments and Results

	AVG	Standard Deviation	Checked security rules Range [min, max]
BFA	49	18.711	[30, 109]
PCM	45	17.058	[28, 101]
HCM	8	6.229	[3, 32]
GAG	5	1.741	[3, 7]

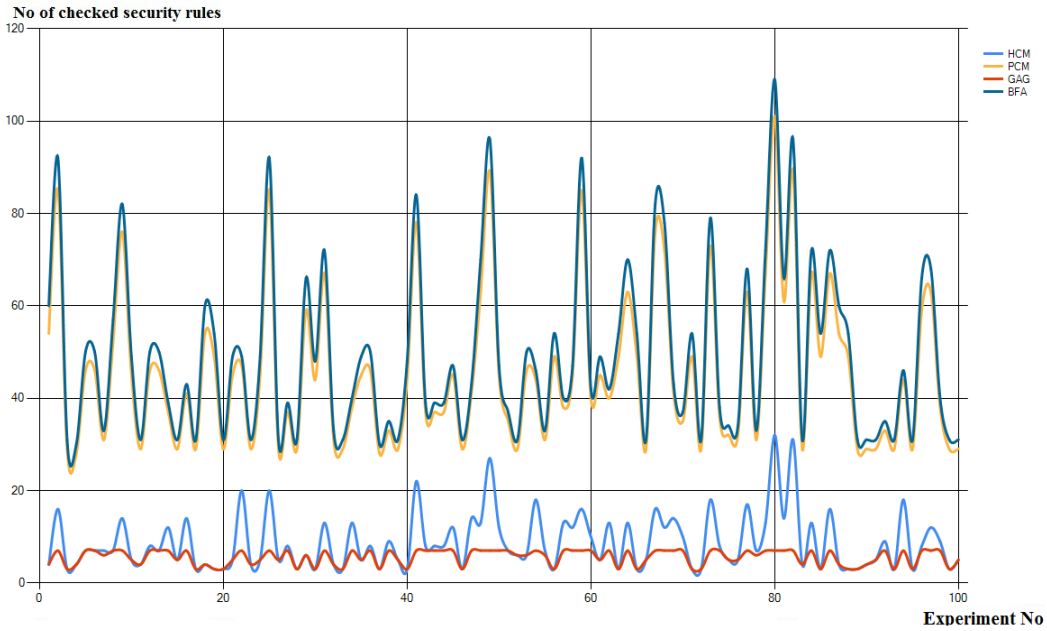

Figure 7.1: Experiments and Results

The above experiments and the mathematical analysis all over the thesis shows that the proposed Grid access control mechanisms HCM and GAG are superior to the existing mechanisms like BFA and PCM.

Section 6.1 also shows that HCM or GAG can be embedded in present Grid authorization framework. Thus they can be considered as an efficient mechanisms to be included in the present authorization systems such as Akenti, VOMS, etc.

The following table brings out snap-shots of all algorithms discussed in this thesis where M is the number of resources, N is the number of security rules and SPs is the number of distinct security policies. The entries of the security table are all boolean values.

Algorithm	Purpose	Context in which it is used	Authorization Time complexity	Uniqueness
BFA	Access Control Mechanism	Resource Level Authorization	O($M \times N$) in all cases	*
PCM	Access Control Mechanism	Resource Level Authorization	O($SPs \times N$) in all cases	*
HCM	Access Control Mechanism	VO Level Authorization	O(2^N) in the worst case	proposed
RSPCM	Access Control Mechanism	VO Level Authorization	O($SPs \times N$) in the worst case	proposed
TCM	Access Control Caching Mechanism	Along with HCM/GAG	O(1) #	proposed
HDCM	Access Control Caching Mechanism	Along with HCM/GAG	O(1) in the average case	proposed
GAG	Access Control Mechanism	VO Level Authorization	O(N) in the worst case	proposed
* Techniques exists in the literature but mathematical treatment/formulation is given for the first time for the best of our knowledge.				
# TCM is used only if there is no change in resources' security policies, thus alway runs in O(1) if used.				

7.3 Future Scope

Authorization provides the UARG on which a scheduling algorithm has to run to coordinate job execution among these resources. As HCM or GAG mechanisms runs across domains and cover the entire Grid resources, one can think of utilizing the authorization process to provide initial information about resources' availability and other important scheduling parameters to the scheduler which helps to speed up the scheduling process.

Grid is one of the most highly distributed systems, so managing trust in a grid is really a critical issue. Furthermore, the very dynamic nature of a grid system, where members frequently join and leave, makes managing and maintaining trust extremely a difficult job. A Trust Management System (**TMS**), especially in grids, needs to be scalable. Insuring scalability in such a global distributed system is

extremely important as well as a difficult job. Scalability of a TMS appears in 3 domains: the amount of exchanged messages over the network, the trust information stored at each node and the trust computations carried at each node. HCM or GAG decision trees provide a wide base over which trust parameters can be distributed across different grid node which ease Trust computations as an add-on functionality to their structure.

One of the things which leads us to develop our own simulator is that **GridSim** [50] does not have an authorization module where we can integrate our mechanisms and test them. So one of the future works which we like to accomplish is implementing an authorization module in **GridSim**. Moreover, a realtime implementation of HCM and GAG in a real grid systems such as **GridBus** and **Globus** can be the next step, especially after having them implemented in a simulated environment.

Finally, **Cloud** are so close to Grid, the very next step of research could be in rectifying our schema to fit well in Cloud environment.

Bibliography

[1] A. Chakrabarti, *Grid Authorization Systems*, pp. 66 – 104. Springer Berlin Heidelberg, 2007.

[2] M. Thompson, A. Essiari, and S. Mudumbai, "Certificate-based authorization policy in a pki environment," *ACM Trans. Inf. Syst. Secur.*, vol. 6, pp. 566 – 588, nov. 2003.

[3] M. R. Thompson, A. Essiari, K. Keahey, V. Welch, S. Lang, and B. Liu, "Fine-grained authorization for job and resource management using akenti and the globus toolkit," *CoRR*, vol. cs.DC/0306070, 2003.

[4] A. S. Tanenbaum and M. V. Steen, *Distributed Systems: Principles and Paradigms*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1st ed., 2001.

[5] D. Thain and M. Livny, "Building reliable clients and servers," in *The Grid: Blueprint for a New Computing Infrastructure* (I. Foster and C. Kesselman, eds.), Morgan Kaufmann, 2003.

[6] I. Foster, C. Kesselman, and S. Tuecke, "The anatomy of the grid - enabling scalable virtual organizations," *International Journal of Supercomputer Applications*, vol. 15, 2001.

[7] I. Foster, "What is the grid? a three point checklist," *GRIDtoday*, vol. 1, jul. 2002.

[8] M. Kaiiali, R. Wankar, C. Rao, and A. Agarwal, "Design of a structured fine-grained access control mechanism for authorizing grid resources," in *Computational Science and Engineering Workshops, 2008. CSEWORKSHOPS '08. 11th IEEE International Conference on*, pp. 399 – 404, jul. 2008.

[9] E. Bertino, P. Mazzoleni, B. Crispo, and S. Sivasubramanian, "Towards supporting fine-grained access control for grid resources," in *Distributed Computing Systems, 2004. FTDCS 2004. Proceedings. 10th IEEE International Workshop on Future Trends of*, pp. 59 – 65, may 2004.

[10] W. Johnston, S. Mudumbai, and M. Thompson, "Authorization and attribute certificates for widely distributed access control," in *Enabling Technologies: Infrastructure for Collaborative Enterprises, 1998. (WET ICE '98) Proceedings., Seventh IEEE International Workshops on*, pp. 340 – 345, jun. 1998.

[11] D. Ferraiolo and R. Kuhn, "Role-based access control," in *In 15th NIST-NCSC National Computer Security Conference*, pp. 554–563, 1992.

[12] R. B. L. G. P. M. A. M. L. R. K. S. D. S. M. R. T. Markus Lorch, Bob Cowles, "Conceptual grid authorization framework and classification," 2004.

[13] B. Neuman and T. Ts'o, "Kerberos: an authentication service for computer networks," *Communications Magazine, IEEE*, vol. 32, pp. 33 –38, sep. 1994.

[14] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. Keromytis, "The keynote trust-management system version 2," RFC 2704, Internet Engineering Task Force, sep. 1999.

[15] R. Alfieri, R. Cecchini, V. Ciaschini, L. dell'Agnello, k. Frohner, A. Gianoli, K. Lrentey, and F. Spataro, "Voms, an authorization system for virtual organizations.," in *European Across Grids Conference* (F. F. Rivera, M. Bubak, A. G. Tato, and R. Doallo, eds.), vol. 2970 of *Lecture Notes in Computer Science*, pp. 33–40, Springer, 2003.

[16] R. Alfieria, R. Cecchinib, V. Ciaschinic, L. dell'Agnellod, A. Frohnere, K. Lorenteyf, and F. Spatarog, "From gridmap-file to voms: managing authorization in a grid environment," *Future Generation Comp. Syst.*, vol. 21, no. 4, pp. 549 – 558, 2005.

[17] L. Pearlman, V. Welch, I. Foster, C. Kesselman, and S. Tuecke, "A community authorization service for group collaboration," in *Policies for Distributed Systems and Networks, 2002. Proceedings. Third International Workshop on*, pp. 50 – 59, 2002.

[18] L. Pearlman, C. Kesselman, V. Welch, I. Foster, and S. Tuecke, "The community authorization service: Status and future," in *In Proceedings of Computing in High Energy Physics 03 (CHEP '03*, 2003.

[19] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin, "Resource reservation protocol (rsvp) – version 1 functional specification," RFC 2205, Internet Engineering Task Force, sep. 1997.

[20] C. Rigney, S. Willens, A. Rubens, and W. Simpson, "Remote authentication dial in user service (radius)," RFC 2865, Internet Engineering Task Force, jun. 2000.

[21] D. W. Chadwick and O. Otenko, "The permis x.509 role based privilege management infrastructure," *Future Generation Comp. Syst.*, vol. 19, no. 2, pp. 277 – 289, 2003.

[22] T. Dierks and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2," RFC 5246, Internet Engineering Task Force, Aug. 2008.

[23] J. Vollbrecht, P. Calhoun, S. Farrell, L. Gommans, G. Gross, B. de Bruijn, C. de Laat, M. Holdrege, and D. Spence, "Aaa authorization framework," RFC 2904, Internet Engineering Task Force, aug. 2000.

[24] A. Hoheisel, S. Mueller, and B. Schnor, "Fine-grained security management in a service-oriented grid architecture," in *Proceedings of the Cracow Grid Workshop*, (Poland), pp. 433 – 440, 2006.

[25] M. Bishop, *Computer Security: Art and Science*. Addison-Wesley, Dec. 2002.

[26] D. E. Bell and L. J. Lapadula, "Secure Computer System: Unified Exposition and MULTICS Interpretation," Tech. Rep. ESD-TR-75-306, The MITRE Corporation, 1976.

[27] Biba, "Integrity Considerations for Secure Computer Systems," *MITRE Co., technical report ESD-TR 76-372*, 1977.

[28] B. W. Lampson, "Protection," *SIGOPS Oper. Syst. Rev.*, vol. 8, pp. 18–24, January 1974.

[29] R. Sandhu, E. Coyne, H. Feinstein, and C. Youman, "Role-based access control models," *Computer*, vol. 29, pp. 38 –47, feb 1996.

[30] S. Osborn, R. Sandhu, and Q. Munawer, "Configuring role-based access control to enforce mandatory and discretionary access control policies," *ACM Trans. Inf. Syst. Secur.*, vol. 3, pp. 85–106, May 2000.

[31] B. Lang, I. Foster, F. Siebenlist, R. Ananthakrishnan, and T. Freeman, "A multipolicy authorization framework for grid security," in *Network Computing and Applications, 2006. NCA 2006. Fifth IEEE International Symposium on*, pp. 269 – 272, jul. 2006.

[32] S. Shirasuna, A. Slominski, L. Fang, and D. Gannon, "Performance comparison of security mechanisms for grid services," in *Grid Computing, 2004. Proceedings. Fifth IEEE/ACM International Workshop on*, pp. 360 – 364, nov. 2004.

[33] M. Kaiiali, R. Wankar, C. Rao, and A. Agarwal, "A rough set based pcm for authorizing grid resources," in *Intelligent Systems Design and Applications (ISDA), 2010 10th International Conference on*, pp. 391 – 396, 29 2010-dec. 1 2010.

[34] M. Kaiiali, R. Wankar, C. Rao, and A. Agarwal, "Enhancing the hierarchical clustering mechanism of storing resources' security policies in a grid authorization system," in *ICDCIT*, pp. 134 –139, 2010.

[35] M. Kaiiali, R. Wankar, C. Rao, and A. Agarwal, "Concurrent hcm for authorizing grid resources," in *ICDCIT*, pp. 255 –256, 2012.

[36] W. U. the Group of Logic, Institute of Mathematics and U. o. R. P. the Group of Computer Science, Institute of Mathematics, "Rough set exploration system." `http://alfa.mimuw.edu.pl/~rses/start.html`, 2005.

[37] Q. Wang, H. Dai, and Y. Sun, "A rough set based clustering algorithm and the information theoretical approach to refine clusters," in *Intelligent Control and Automation, 2004. WCICA 2004. Fifth World Congress on*, vol. 5, pp. 4287 – 4291 Vol.5, june 2004.

[38] Y. Wei, J. Li, and R. Wang, "The algorithm of grid clustering based on fuzzy rough set & its application," in *Wireless Communications, Networking and Mobile Computing, 2008. WiCOM '08. 4th International Conference on*, pp. 1 –4, oct. 2008.

[39] P. Lingras and R. Yan, "Interval clustering using fuzzy and rough set theory," in *Fuzzy Information, 2004. Processing NAFIPS '04. IEEE Annual Meeting of the*, vol. 2, pp. 780 – 784 Vol.2, june 2004.

[40] W. T. F. Encyclopedia, "Breadth-first search." `http://en.wikipedia.org/wiki/Breadth-first_search`, 2011.

[41] M. Lorch, B. Cowles, R. Baker, L. Gommans, P. Madsen, A. McNab, L. Ramarkrishnan, K. Sankar, D. Skow, and M. Thompson, "Conceptual grid authorization framework and classification." `http://www.gridforum.org/documents/GFD.38.pdf`, 2004.

[42] S. Turner and S. Chokhani, "Clearance attribute and authority clearance constraints certificate extension," RFC 5913, Internet Engineering Task Force, jun. 2010.

[43] C. Adams and S. Farrell, "Internet x.509 public key infrastructure certificate management protocols," RFC 2510, Internet Engineering Task Force, mar. 1999.

[44] U. S. G. A. Office, *Full PKI Implementation Faces ManyFormidable Challenges*, pp. 42 – 59. feb. 2001.

[45] M. A. Bishop, *The Art and Science of Computer Security*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002.

[46] "The globus toolkit 4." `http://www.globus.org/toolkit/`.

[47] "Gt4 authorization framework." `http://www.globus.org/alliance/events/sc06/AuthZ.pdf`.

[48] "Oasis, extensible access control markup language (xacml), v2.0." `http://www.globus.org/alliance/events/sc06/AuthZ.pdf`, jan. 2005.

[49] S. Farrell and R. Housley, "An internet attribute certificate profile for authorization," RFC 3281, Internet Engineering Task Force, apr. 2002.

[50] R. Buyya and M. Murshed, "Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing," *CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE (CCPE*, vol. 14, no. 13, pp. 1175–1220, 2002.