De Montfort University

Efficient Enforcement of Security Policies in Distributed Systems

by

Ali Mousa G. Alzahrani

PhD Thesis

This thesis is submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy

Supervised by

Dr. Helge Janicke and Dr. Antonio Cau

in the

Faculty of Technology

Software Technology Research Laboratory (STRL)

April 2013

Abstract

Policy-based management (PBM) is an adaptable security policy mechanism in information systems (IS) that confirm only authorised users can access resources. A few decades ago, the traditional PBM has focused on closed systems, where enforcement mechanisms are trusted by system administrators who define access control policies. Most of current work on the PBM systems focuses on designing a centralised policy decision point (PDP), the component that evaluates an access request against a policy and reports the decision back, which can have performance and resilience drawbacks.

Performance and resilience are a major concern for applications in military, health and national security domains where the performance is desirable to increase situational awareness through collaboration and to decrease the length of the decision making cycle. The centralised PDP also represents a single point of failure. In case of the failure of the centralised PDP, all resources in the system may cease to function. The efficient distribution of enforcement mechanisms is therefore key in building large scale policy managed distributed systems.

Moving from the traditional PBM systems to dynamic PBM systems supports dynamic adaptability of behaviour by changing policy without recoding or stopping the system. The SANTA history-based dynamic PBM system has a formal underpinning in Interval Temporal Logic (ITL) allowing for formal analysis and verification to take place. The main aim of the research to automatically distribute enforcement mechanisms in the distributed system in order to provide resilience against network failure whilst preserving efficiency of policy decision making. The policy formalisation is based on SANTA policy model to provide a high level of assurance.

The contribution of this work addresses the challenge of performance, manageability and security, by designing a Decentralised PBM framework and a corresponding Distributed Enforcements Architecture (DENAR). The ability of enforcing static and dynamic security policies in DENAR is the prime research issue, which balances the desire to distribute systems for flexibility whilst maintaining sufficient security over operations. Our research developed mechanisms to improve the efficiency of the enforcement of security policy mechanisms and their resilience against network failures in distributed information systems.

Acknowledgements

In the name of Allah, the Most Merciful and the Most Gracious, I give praise and thanks to Him for supporting me with the strength to complete this thesis, and for providing me with knowledgeable and caring individuals during the study process. This thesis could not have been completed without the recommendations, advice and encouragements of many people. It may not be possible to mention all of them here, but their contributions, guidance and support are extremely appreciated.

I would like to thank both Dr. Helge Janicke, who was my first supervisor, he was generous with me, provided freely of his knowledge and made many suggestions, and Dr. Antonio Cau who was my second supervisor and has encouraged my autonomy and given me invaluable instruction.

Dr. Turki Alghamdi is more than a friend. Without his help, encouragement, and academic discussion, the pace of my work would be slower. I will never forget the nice times we spent working together at the university .

I would like to pay high regards to my dear father and mother, my dear children (Almuthana and Lana), my brothers and my sisters for their sincere encouragement and inspiration throughout my research work and lifting me uphill this phase of life. I owe everything to them. Besides this, several people have knowingly and unknowingly helped me in the successful completion of this research.

Also, my wife (Um Almuthana) has been, always, my pillar, my joy and my guiding light, and I thank her.

Finally, I would like to thank University of Hail, not only for providing the scholarship which allowed me to undertake this research, but also for giving me the opportunity meet so many nice people, places and cultures.

Publications

1. Alzaharani A., Janicke H. and Abubaker S. (2010). Decentralized XACML Overlay Network. *In proceedings of the Third IEEE International Symposium on Trust, Security and Privacy for Emerging Applications* (TSP-10). Bradford, UK.

2. Alzaharani A., Janicke H. (2010). Decentralized Policy Based Management. *In Proceedings of the Saudi International Conference* (SIC2010). Manchester, UK.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Objectives

- Motivate the needs of security policy enforcement in distributed systems.
- Highlight the original contribution and identify the research question.
- Provide the research methodology and define the success criteria.
- Provide the thesis organisation.

1.1 Problem Statement

A distributed system consists of a large number of heterogeneous networks that interact with each other in order to achieve a common goal. Managing distributed systems results in high administrative costs and long deployment cycles for business initiatives. Security and management of distributed systems are mutually dependent and each needs the services of the other [46]. Managing security in heterogeneous, distributed systems can become expensive and prone to error because security administration must be distributed to multiple policy administrators, which makes it difficult to provide consistent security policies across the entire system. It is thus necessary to analyse security policies for conflicts and inconsistencies that may lead the system to insecure states. Analysis of policy specification is also important in order to ensure that enterprise security goals are met. In large-scale systems with large numbers of both users and policies, it must be possible to analyse the policies to check for the existence of policies that implement the high-level security goals of the organisation.

Policy Based Management (PBM) is an evolutionary process. Policy-based resource allocation, the association between policy and the devices/entities on which it must be implemented, and even the policies themselves are subject to frequent review and change. Decentralised management of large and complex organisational structures is both difficult and error-prone and administrators must be isolated from the details of the underlying implementations and policy representations. This can be accomplished with Decentralised Policy Based Management (PBM) that permits integrated enforcements and hides the heterogeneity of policies and the complexity of policy deployment.

1.2 Motivation

With the growth of the World Wide Web (WWW), the need for distributed systems has increased extraordinarily in the last decade. Accompanying this growth has been the increased need for distributed access control architectures, where communication and the sharing of information to provide services with limited resources is essential. In addition, providing services to consumers or users via new technologies (e.g., mobile agents, active networks) and the use of the Internet increase the security concerns associated with today's networked environments.

The management of these systems must be distributed in order to be scalable with the size of enterprise networks, and management procedures must be automated to reduce administrative costs, which can be facilitated with Policy Based Management approaches. Policy Based Management (PBM) can be defined as an adaptable security policy mechanism in information systems (IS) that confirm only authorised users can access information [19, 88]. A few decades ago, the PBM focused on a closed system, where enforcement mechanisms were trusted by system administrators who defined access control policies i.e. static policies. Building on the PBM approach by involving dynamic policies defines decisions on the basis of the system state or changes in the system state. Sometimes access control requirements refer to the execution history [3], this means that a policy decision can depend on the system behaviour that was observed in the past i.e. dynamic policy. For example, the access requirement is being "if a user at some point in the past has read file (A) that is secret then the same user cannot write to file (B), otherwise the user can write on file (B)".

Sharing information in a secure manner, but keeping information available, is a major concern for distributed systems in military, health and national security domains where it is desirable to increase situational awareness through collaboration.

Decentralised Policy Based Management (PBM) avoids a single point of failure for security policy enforcement where it involves distributed enforcements. In case of centralised enforcement, the failure of the enforcement would cease all the resources on the system to function. In addition, Decentralised Policy Based Management (PBM) would be more efficient in terms of resource utilisation.

Thus, based on the identified requirements, we concluded that the efficient distribution of enforcements mechanism and the automated decomposition of dynamic security policies without conflict among them are key in building large-scale policy managed distributed systems.

1.3 Research Question

The overall and the central research question investigated in this thesis is:

- **Q1.** How to improve the access control decision making with support for static and dynamic security policies in a distributed setting?
- **Q2.** To what extent can a Decentralised Policy Based Management system improve performance, security, manageability and resilience?

1.4 Original Contribution

The research improves the efficiency of security policy enforcement mechanisms and their resilience against policy enforcement failures in Policy Based Management system (PBM).

Performance, by the distribution of Policy Decision Points (PDPs), is based on a sound theory of policy decomposition, in which correctness is preserved to decrease the length of the authorisation decision to performing any service. Moreover, the Decentralised Policy Based Management (PBM) framework is a manageable system that can be readily understood and safely managed by administrators as well as analysing dynamic policies and refining them in our Distributed Policy Enforcements Architecture (DENAR) to result in fewer security breaches due to administrative and enforcement errors. Finally, the collaborative DENARs are designed to fulfill resilience in less enforcement failure and resource utilisation.

The objectives of the research are:

- **O1.** Providing a formal framework for Decentralised (PBM) that enforce security policies in a distributed setting. [See Chapter 4 and 6]
- **O2.** Improving the performance (network traffic) of security policy enforcement mechanisms and their resilience against unintentional and intentional enforcement failures in distributed information systems by designing Distributed Policy Enforcements Architecture (DENAR). [See Chapter 5]
- **O3.** Providing an automated policy decomposition of static and dynamic security policies to accomplish high assurance safety-critical information systems for Distributed Policy Enforcements Architecture (DENAR). [See Chapter 7]

- **O4.** Designing coordination and collaboration enforcement model, using the policy decomposition of (O3) and the Distributed Policy Enforcements Architecture (DENAR) of (O2), that enable Decentralised PBM in (O1) to deploy and enforce policies for large-scale systems. [See Chapter 8]

1.5 Research Methodology

The **'Constructive Research'** is a scientific research method that is detailed in [63] [29] is used as the research method in this research. The constructive research method is being used in the majority of the computer science approaches. It refers to knowledge contributions being developed as a new framework, theory, model or algorithm.

The methodology of the proposed research is made up of nine work packages which are adjusted with the list of the steps in constructive research [63]. The first work package addresses the research background and the research project requirements. Seven are scientific research work packages. The last work package concentrates on the thesis writing up. The investigation work packages are illustrated in Figure 1.1.

1.5.1 Research Work Packages and Milestones

The research is undertaken along a theoretical to an applied axis and is structured in work packages as follows.

Work package organisation

- **WP1: Literature Review.**

  An introduction and critical review of related work is conducted via the digital resources, including the following topics: Access control models, security policy specification languages and enforcement policy systems in centralised and distributed PDPs. Additionally, the literature review provides the technology that we used: Security Analysis Toolkit for Agents (SANTA).

- **WP2: Decentralised PBM Framework.**

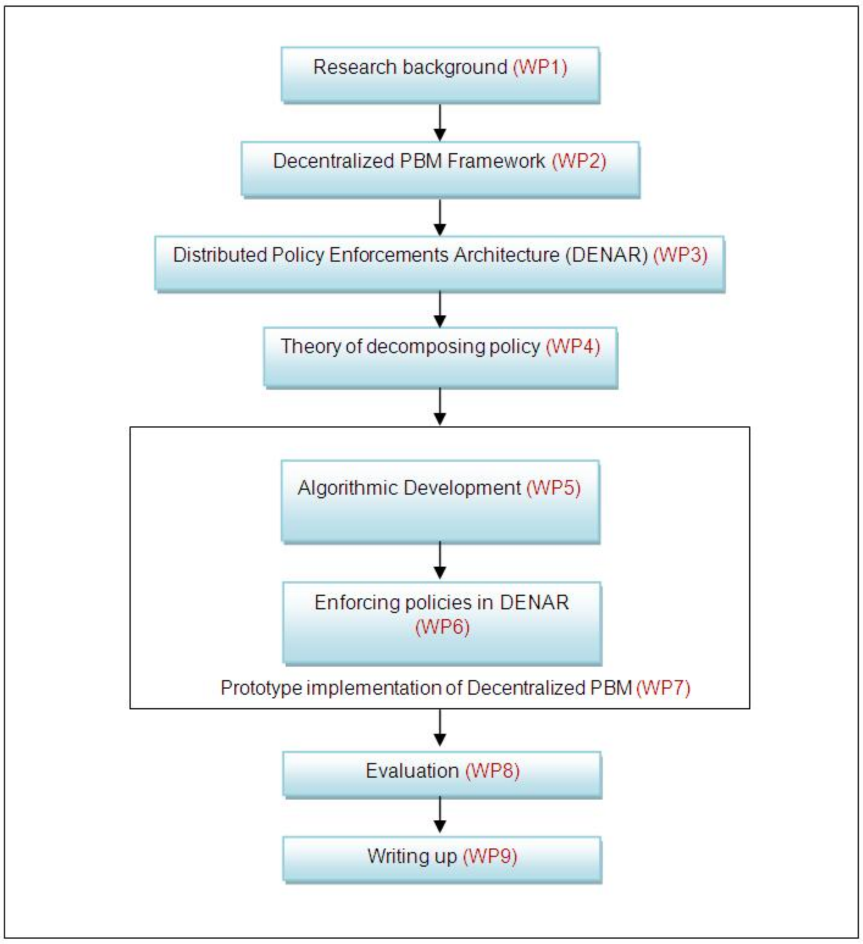

Research background (WP1)
Decentralized PBM Framework (WP2)
Distributed Policy Enforcements Architecture (DENAR) (WP3)
Theory of decomposing policy (WP4)
Algorithmic Development (WP5)
Enforcing policies in DENAR (WP6)
Prototype implementation of Decentralized PBM (WP7)
Evaluation (WP8)
Writing up (WP9)

FIGURE 1.1: Research work packages.

The framework that expresses compositional policies, and their decomposition and enforcement in the PDPs is designed to capture the research objectives as expressed in the research question.

- **Task 2.1:** Identifying requirements for a Decentralised PBM and designing a framework that show the methodology to achieve decentralised enforcement.

- **WP3: Distributed Policy Enforcements Architecture (DENAR).**

  This work package designs Distributed Policy Enforcements Architecture (DENAR). The research in this work package explicitly states how the architecture components interact to achieve the research objectives.

  - **Task 3.1:** Designing Distributed Policy Enforcements Architecture (DENAR) that fulfill the second research objective.

- **Task 3.2:** Describing the DENAR component interaction.

- **WP4: DENAR Analysis.**

  The research investigation in this stage focus on the development of a novel formal model for policy enforcement in DENAR (Analysis and Dependencies). This work package is concerned with the analysis of policies and dependencies on the requirements of the access control in Decentralised PBM.

  - **Task 4.1:** Syntactic and semantic analysis of policies to identify dependencies between policy and the access control in information systems.

  - **Task 4.2:** Analysis the enforcement of static and dynamic policies is being in DENAR to achieve a collaboration enforcements.

- **WP5: Policy Decomposition Model.**

  This work package addresses the policy decomposition model of re-specifying the policy into independent sub-policies that can be deployed and enforced in the (DENAR).

  - **Task 5.1:** Development of the policy decomposition model that can split a policy based on the rule object domain with the concern of policy dependencies that found in dynamic policy. e.g. Chinese Wall [25].
  - **Task 5.2:** Specification of measurements for efficiency and resilience based on the Distributed Policy Enforcements Architecture (DENAR) [See Task 3.1 and Task 4.2] and identification of the effect of the refinement rules on these measures.

- **WP6: DENAR Enforcement and Coordination.**

  The development of a behavioural model to express decomposed sub-policies and enforcement in the DENAR including coordination and concurrency between DENARs.

  - **Task 6.1:** Behavioral semantics of DENAR capturing decision making and their interaction with PEPs.
  - **Task 6.2:** Modelling Push and Pull Models to achieve coordination and concurrency between DENARs .

- **WP7: DENAR Prototype**

  This work package describes the design and implementation of our prototype for DENAR.

- **WP8: Evaluation through case study.**

  The Decentralised PBM performance, security, manageability and resilience against the centralised PBM approach through representative case studies.

  - **Task 8.1:** Network simulators of the Centralised PBM and Decentralised PBM systems to case studies, which are developed. the case studies show the expressiveness of the dynamic and static policies and exemplify the enforcement and coordination in the Distributed Policy Enforcements Architecture (DENAR) for a realistic system scenario.
  - **Task 8.2:** Validation of the results against the objectives of the approach: i) Performance: by the network traffic for DENAR in comparison with the centralised PDP results; ii) Security: by comparative discussion of the ability of enforcing the case studies developed in Task 8.1 using the centralised PDP and the DENAR; iii) Manageability: by discussing the resource utilisation and administration in the DENAR; iiii) Resilience: by simulating both the centralised PDP and the (DENAR) and varying the number of failures are discussed.

- **WP9:** Write up. Based on results of work packages.

1.5.2 Deliverables/Milestones

The following deliverables are produced as part of the work program:

- **D1** Problem, objectives, proposed framework and solution and literature review. From reports on WP1 and WP2 and reports, we produced a first published paper [6].
- **D2** Distributed Policy Enforcements Architecture (DENAR) model (WP5) and tool-support. This delivery includes a software prototype that demonstrates the feasibility of automation. Reported on the a second published paper [7].
- **D3** PhD Theses. Reports on the case study and validation of the approach.

1.6 Success Criteria

In today's critical communication and information-sharing age, healthcare, commercial and governmental organisations require to protect the information assets within their organisations' networks as well as the security requirements, which can be mutable and frequently dependent on the history of the system by involving the dynamic policy (e.g., Chinese-Wall policies) or require dynamic reconfiguration triggered by events. Therefore, policy decisions cannot be made in isolation of the system. However, research on the impact that history-based and dynamic policies have on the distribution of PDPs in a system is still in its infancy. PDPs are very much linked to the system and are required to have the ability to observe events in order to make decisions. The proposed framework exploits these dependencies to identify an efficient and resilient deployment of PDPs in our Distributed Policy Enforcements Architecture (DENAR), whilst ensuring that static and dynamic policies are correctly enforced in DENAR.

The measure of success is that both the Decentralised PBM framework and Distributed Policy Enforcements Architecture (DENAR) and their supporting algorithm indeed resolve the proposed research question and demonstrate it by experiments through the implementation prototype. The DENAR prototype demonstrates that the research output results match the research objective factors as follows:

- **Performance,** a discussion of improving the enforcement will be provided and some experiments will be used to measure the network traffic in both centralised PDP and a network of DENARs.
- **Security,** some experiments will be used to discuss the ability of enforcing both static and dynamic policies in both centralised PDP and the network of DENARs.
- **Manageability,** a discussion of the resource utilisation and administration in the DENAR will be provided.
- **Resilience,** Network simulators for both centralised PDP and the network of DENARs will be used to discuss the resilience in some potential failures scenarios.

1.7 Thesis Outline

This section provides the proposed structure of the authors' PhD thesis.

Chapter 1: Introduction.

This Chapter gives an overview of the problem, motivation and approach to the solution.

Chapter 2: Related Work.

Access control models, security policy specification languages, enforcement policy systems in centralised and distributed PDPs are introduced. Additionally, the security policy concepts are presented.

Chapter 3: Preliminaries.

The Security Analysis Toolkit for Agents (SANTA) technology is described in this Chapter.

Chapter 4: Decentralised Policy Based Management (PBM).

The Decentralised Policy Based Management (PBM) framework that expresses the policy decomposition and enforcement in the DENAR is described. This Chapter identifies the methodology used in the proposed framework. This Chapter fulfils the first objective in the research.

Chapter 5: Distributed Enforcements Architecture (DENAR).

The Distributed Policy Enforcements Architecture (DENAR) components are detailed in structural and behavioural diagrams with their interaction. The network of DENARs is described in scope of administration task. This Chapter fulfils the second objective in the research.

Chapter 6: DENAR Analysis.

This Chapter is concerned with the analysis of security policies and dependencies on the structure of information systems. Formal model of policies and enforcement is provided (Analysis and Dependencies). Clearly, syntactic and semantic analysis of policies to identify dependencies between policy and the access control requirements in information systems (particularly in dynamic policies). Moreover, it identifies and analyses the distributed policy enforcements challenges to meet the enforcement in collaborative DENARs. This Chapter fulfils the second and third objectives in the research.

Chapter 7: Policy Decomposition and Deployment.

This Chapter models the policy decomposition model that fragment the policy into sub-policies. The refinement method is then implemented to relate sub-policies to each other so those can be deployed and enforced in the network of DENARs. This Chapter fulfils the third objective in the research.

Chapter 8: DENAR Enforcement and Coordination.

In this Chapter, the network of DENARs is presented in structural and behavioural diagrams to describe the enforcement mechanism in DENAR. In addition, DENARs coordination mechanism is provided and related with the result the produced in Chapter 7. This Chapter fulfils the fourth objective in the research.

Chapter 9: DENAR Prototype.

This Chapter designs the Distributed Enforcements Architecture (DENAR) prototype to evaluate the research objectives. DENARs prototype implements for DENAR network and DENARs enforcement and coordination software.

Chapter 10: Case Study and Evaluation.

This Chapter introduces the Centralised PBM and Decentralised PBM simulators. In addition, the Chapter provides three case studies to illustrate the practical applicability for both static and dynamic policies those enforced in Centralised PBM and Decentralised PBM simulators. The performance, security, manageability and resilience factors are evaluated through the case studies in Decentralised PBM simulator against the Centralised PBM simulator.

Chapter 11: Conclusion and Future Work.

This Chapter includes the final results, conclusion of our research and our views for future work.

Chapter 2

Related Work

Objectives

- Provide an overview of security policy and highlight the current policy specification languages.
- Give an overview of the access control models.
- Provide an overview of the Policy Based Management (PBM).
- Identify the difficulties and problem of the related research in both centralised and decentralised policy enforcement.

2.1 Introduction

In this Chapter we critically review related work in the area of security with the emphasis on Policy-based languages and access control mechanisms in general using both centralised and distributed approaches. Firstly, we survey some of the well known access control models (Section 2.3) and critically discuss some of the formal and informal policy languages that have been proposed in this context (Section 2.4). Following the review of policy models and languages, we review the Policy Based Management (PBM) framework. Subsequently, the centralised and distributed enforcement approaches are discussed in Section 2.6. The Chapter concludes with a short summary in Section 2.7.

2.2 Security Policy Overview

Security in the context of distributed systems is identified as consisting of four primary requirements: confidentiality or secrecy, integrity, availability and accountability. Confidentiality requires that the information or resources in distributed systems only be disclosed to authorised parties. Integrity is associated with the protection of information against improper or unauthorised modification. Availability refers to the provision of prompt access to information and resources by those authorised. Finally, accountability indicates knowing who has had access to information or resources.

Authentication, access control and audit are the mutually supportive technologies to accomplish the above requirements. Authentication deals with identification of parties and access control is concerned with limiting the activity of legitimate parties who have been successfully authenticated by ensuring that every access to a system and its information or resources is controlled and that only accesses that are authorised can take place.

An access control system has three primary components: the subjects whose active entities cause information to flow among objects or change the system state, e.g. users, agents running on behalf of users, groups, compounds of subjects or roles; the objects (targets) whose entities contain or receive information; and the rules which specify the ways in which the subjects can access the targets [75]. However, in this project we also suppose that the confidentiality and integrity of messages that are exchanged between parties in the system should given on the notation of specification and enforcement of authorisation, obligation, delegation and integrity constraints on interactions where they are defined according to [51] as:

- **Authorisation.** Authorisation policies (often called permissions or privileges) define the conditions under which an authenticated subject (a user or an agent acting on behalf of a user) is allowed to perform a specific action on an object. In some languages, a positive authorisation policy defines under which conditions the subjects are permitted to execute a specific action on objects. On the other hand, a negative authorisation policy specifies under which conditions the subjects are forbidden to execute a specific action on objects.

- **Obligation.** Obligation policies define the conditions or events (sometimes also called triggers) under which a subject must perform a specific action on an object, therefore, when certain events occur and provide the ability to respond to changing circumstances.

- **Delegation** Delegation policies define the conditions under which a subject can permit a specific access right (i.e. the right to perform an action on an object) which they possess, to another subject to perform an action on their behalf. Integrity policies define constraints on the execution of an action on an object by a specific subject.

A security policy is defined as a specification of the security requirements of a system. Access control models provide a formal representation of security policies such as confidentiality policies or integrity policies or a mixture of both. A variety of useful access control models are given in (Section 2.3). Additionally, a considerable body of work is devoted to developing languages for expressing security policies. A review of these works is given in (Section 2.4).

2.3 Access Control Models

In this section we overview different approaches to the specification of security policies, where the main focus is on access control. Access control is described in [12] as follows: "The function of access-control is to control which principals (subjects) have access to which resources (objects) in the system". Based on this definition, we give an overview of access control as the mechanisms that enforce authorisation and delegation policies. Access control models have been traditionally divided into Mandatory Access Control (MAC), which is mostly concerned with controlling information flow between the objects of a system, and Discretionary Access Control (DAC), which is concerned with the specification of authorisation rules to govern the access of users to the information.

2.3.1 Discretionary Policies

The Discretionary Access Control (DAC) model was developed by the TCSEC [36] and is derived from the Discretionary Security Property of the Bell-LaPadula Model [See Section 2.3.3.1].

In DAC each object has a possessor who exercises primary control over the object. The controls are discretionary in the sense that a subject with possession of access permission on an object can delegate that permission to any other subject [36]. DAC policies exploit the access matrix model as a framework for reasoning about the permitted accesses. [See Section 2.3.4 for the access matrix model]

However, DAC policies do not enforce any control on the information flow once this information has been acquired by a process. As a matter of fact, DAC does not distinguish between the user and subjects operating on behalf of the user. As a result, if the ownership of an object is obtained by a malicious or incompetent subject, nothing can prevent that subject from destroying all discretionary protections.

2.3.2 Non-Discretionary Policies

Non-discretionary policies establish controls that cannot be changed by users directly, but only through administrative action. In this case, an administrator determines which subjects can have access to certain objects based on the organisation's security policy. In this context administrative policies that control who is authorised to modify access rights are of grater importance.

Thus, any global and persistent access control policy that depends on access control decision information not directly controlled by the security administrator is non-discretionary, and in NDAC, the policy for delegating authority must be explicit [4].

2.3.3 Mandatory Policies

In mandatory access control (MAC), both principals and target objects are tagged with security labels. The enforcement of these policies essentially matches the principal's security clearance against the security label of the resource. Typically this form of access control model is used for military and governmental policies. Examples of MAC policies are Bell-LaPadula or the Chinese Wall Models.

2.3.3.1 Bell-LaPadula Model

The Bell-LaPadula Model (Multilevel Security) [66] provides a means to prove the security of time-sharing mainframe systems. In the model, information is tagged with a label that indicates the sensitivity of the information. Traditionally the military classification scheme contains the levels Unclassified, Confidential, Secret and Top Secret, but changes to this classification may occur over time and often lead to incompatibilities between different systems [12]. Principals hold a specific security clearance corresponding to the classification. An individual is only allowed to access documents (objects) that he is cleared for, or documents that require a lower clearance level. The model relies on three basic rules; the *-property, the simple property and the tranquillity property.

- *-property (star property): An individual is only allowed to access an object if the security level of the object is the same or greater than the security level of the individual. Thus information cannot leak out to individuals with lower security clearances. Therefore, this property is also sometimes referred to as the confinement property.

- Simple property: An individual must have a security label more privileged than the labels on the objects they may access.

- Tranquillity property: The tranquillity property requires that when a system is currently accessing an object, the security label of the individual and object involved must not be changed.

Thus, it demonstrates that the current state of the access control system guarantees a correct behaviour.

2.3.3.2 Chinese Wall Model

The Chinese Wall model [25] was stimulated by business operations where it introduces dynamic compartments to capture the concept of separation of duty. It comes from the financial background, where service firms have internal rules that prevent conflicting interests. An informal example is given by [25] which explains the Chinese Wall requirements. This example cannot be expressed using other MAC models. The key contrast between the Bell-LaPadula and Chinese Wall models is that the Chinese Wall Model leaves a subject initially the free choice, which information to access, but restricts further access to compartments, that are different to the previous one. Two properties then ensure valid access control:

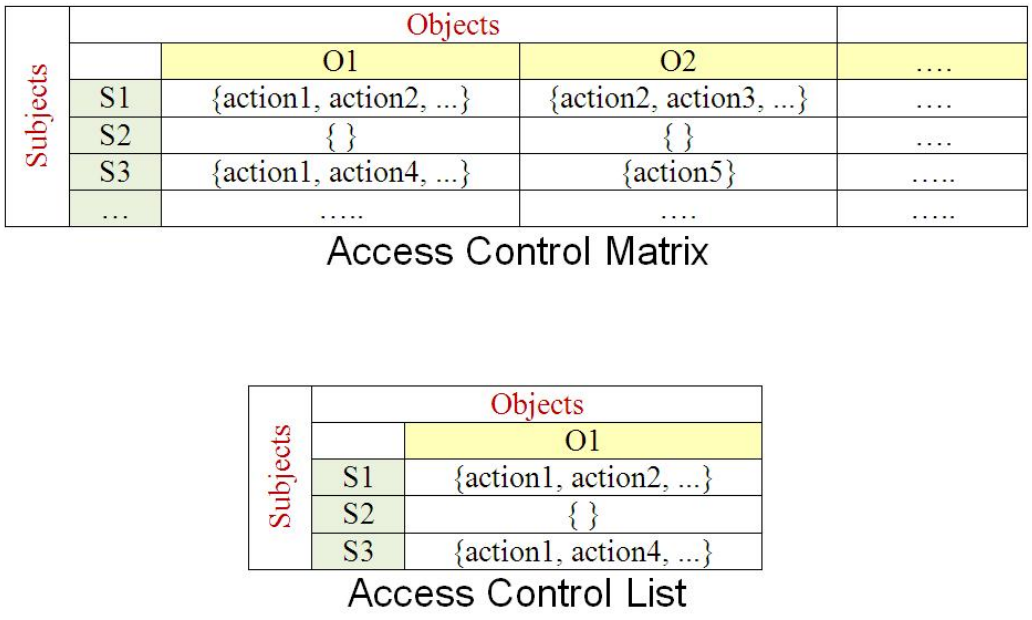
- ss-property: This property states that access is only granted if the requested object is in the same company dataset as an object already accessed by that subject, i.e. within the Wall, or belongs to an entirely different conflict of interest class.

- *-property. This property states that a write access is only permitted if access is permitted by the simple security property and no object can be read which is in a different company dataset to the one for which write access is requested and contains unsanitised information.

The Chinese Wall model is a primitive history-based (or audit-based) access-control model [1] and as such requires more sophisticated enforcement mechanisms than the Bell-LaPadula model.

2.3.4 Access Control Matrix

A good way of representing access-control specifications is in the form of an access-control matrix. The access control matrix of access control was proposed by Lampson [65] and extended in HRU model [2, 18, 22]. The HRU extension defines the issue of subject/object creation and deletion rights. The matrix addresses that subjects are allowed to perform actions on objects in the system. In the access matrix model, the state of the system is defined by a triple (S,O,A), where S is the set of subjects, O is the set of objects and A is the access matrix where rows correspond to subjects, columns correspond to objects and entry A[S,O] reports the privileges of subjects on objects.



Subjects	Objects			
		O1	O2	....
	S1	{action1, action2, ...}	{action2, action3, ...}	....
	S2	{ }	{ }	....
	S3	{action1, action4, ...}	{action5}	.....
	...	.....	....	.....

Access Control Matrix

Subjects	Objects	
		O1
	S1	{action1, action2, ...}
	S2	{ }
	S3	{action1, action4, ...}

Access Control List

FIGURE 2.1: Access Control Matrix.

A reference monitor can then directly base the authorisation decision on the matrix entries. However, in other enforcement mechanisms, the reference monitor needs to satisfy this specification by other means. The access control matrix can be distributed to be enforced. The most common form is to distribute the matrix to each protected object. Figure 2.1 shows the access control matrix and the access control list which is based on the distribution to objects. The access control matrix stores the actions, that a subject can perform on an object (resource). However, in large-scale systems, the access matrix model is insufficient because increasing the number of entries in the matrix leads to more complexity, and it becomes less verifiable.

2.3.5 Role-based Policies

The main motivation for the Role Based Access Control ($RBAC$) model was to combine the flexibility of explicit access control along with additionally imposed enterprise-specific constraints. In addition, although different research has been done so far, the main notion of $RBAC$ remains the same [30]. The $RBAC$ security system adjusts the access to objects based on organisational activities and responsibilities called roles which are assigned to subjects within the system. Instead of specifying all the accesses that each subject is permitted to execute, a security policy in $RBAC$ is specified for roles, while subjects are granted to adopt roles. The subject playing a role is allowed to execute all privileges for which the role is authorised. This significantly simplifies the security administration in $RBAC$. For example, if a user moves to a new function within the organisation, the user can simply be assigned to the new role and removed from the old role. An additional motivation for $RBAC$ has been to reuse role specifications by a form of inheritance whereby one role (often a manager in the organisation) can inherit the privileges of another role and therefore avoid the need to recreate the specification of permissions.

Sandhu et. al [40] defined four conceptual models in an effort to standardise $RBAC$. We briefly summarise these models in order to present an overview of the features supported by $RBAC$ implementations. $RBAC_0$ contains users, roles, permissions and sessions. Permissions are assigned to roles and users can be assigned to roles to assume those permissions. A user can start a session to activate a subset of the roles to which the user is assigned. $RBAC_1$ includes $RBAC_0$ and initiates role hierarchies [84]. Hierarchies are a way of structuring roles to reflect an organisation's lines of authority and responsibility, and are specified using inheritance between roles. Role inheritance allows the reuse of permissions by allowing a senior role to inherit permissions from a junior role. $RBAC_0$ example is given in [84] and shown in Figure 6.1.

$RBAC_2$ includes $RBAC_0$ and initiates constraints to restrict the activation of roles in sessions or the assignment of users or permissions to roles. Constraints are used to specify application-dependent conditions, and satisfy well-defined control

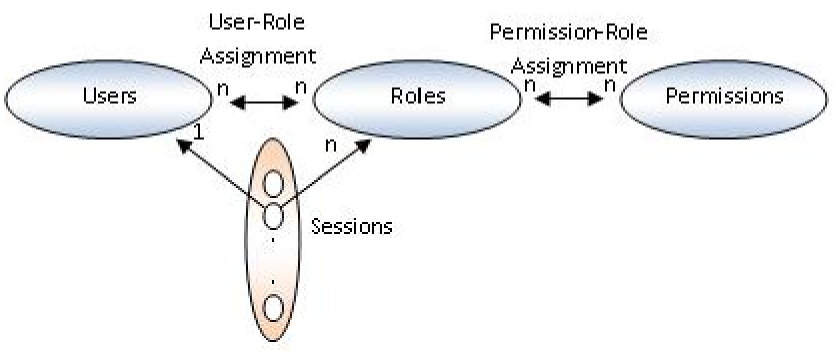


FIGURE 2.2: RBAC0 Example.

principles such as the principles of least-privilege and separation of duties. Finally, $RBAC_3$ combines both $RBAC_1$ and $RBAC_2$, and provides both role hierarchies and constraints.

The use of $RBAC$ models to manage access to computational resources and digital information within a closed computer system is widely accepted as a best practice for commercial applications. Systems such as Microsoft Active Directory, SELinux, Solaris, Oracle DBMS and PostgreSQL effectively have utilized some form of $RBAC$ to guarantee the confidentiality of digital resources [82].

Role-based access control models have been still in challenges of researchers because the simple hierarchical relationship is insufficient to model various sorts of relationships that may occur. Team-based Access Control [94] is the extension of $RBAC$ that allows for the specification of temporal constraints on role assignment and activation and was proposed with Temporal Role-based Access Control by Bertino et.al. [20, 21]. Both $RBAC$ and $TRBAC$ have been modelled by [16] using Constraint Logic Programming. With the aim of expressing dynamic separations of duty in $RBAC$, [73] extending the traditional set-oriented specification of $RBAC$ with a version that is based on first order temporal logic. $RBAC$ has also been standardised by the National Institute of Standards and Technology (NIST) [8]. $RBAC_3$ models can be expressed using Siewe's [87] basic authorisation framework, by introducing specialised actions for role-assignment and (de-) activation, together with data structures that represent the current role-assignments and activations.

2.4 Policy Specification Languages

Having reviewed the foundations of security models and policies, we will now review the state of the art in policy specification languages. Figure 2.3 summarises a policy hierarchy (according to [30]), and represents different views on policies and abstractions of policies for the purpose of refining high-level management goals into low-level policy rules whose enforcement can be fully automated. A policy language is expected to include (according to [30]):

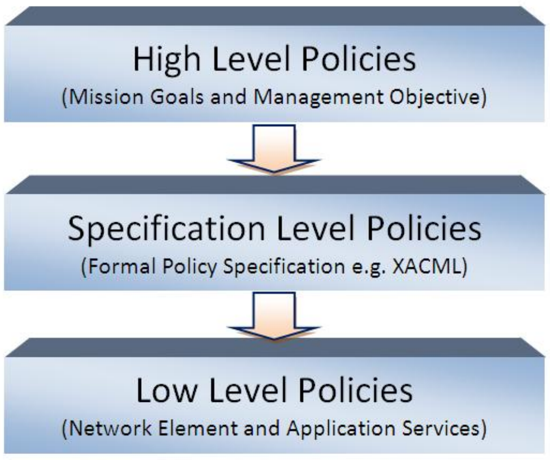

High Level Policies
(Mission Goals and Management Objective)
Specification Level Policies
(Formal Policy Specification e.g. XACML)
Low Level Policies
(Network Element and Application Services)

Figure 2.3: Policy Representation Levels.

- High level policies, which can be business or management goals, or even natural language statements. High-level abstract policies are not enforceable and their recognition involves refining them into one of the other two levels below.

- Specification level policies (called high level policies by some researchers) refer to those policies specified by a human administrator to supply abstractions for low-level policies in a specific format. These policies relate to objects or specific services, and their interpretation can be automated.

- Low-level policies or configurations include security mechanism configurations, device configurations and directory schema entries.

2.4.1 Authorisation Specification Language (ASL)

The first work investigating logic-based languages for the specification of security policies was the work by Woo and Lam [98], but this was generally not intuitive and did not easily map onto implementation mechanisms. Their language has a strong mathematical background, which can make it complex to use. ASL [48, 49] is an example of a formal logic language for specifying authorisation policies. They proposed the use of positive and negative authorisation rules and showed how conflicts are resolved by decision rules. The example below is an authorisation rule in ASL, which states that all subjects belonging to the group Employees but not to Soft-Developers are authorised to read file1.

$$\boxed{cando(file1, s, +read) < -in(s, Employees)\ \&\ -in(s, Soft - Developers)}$$

The "cando" predicate indicated a specification of positive authorisations; the sign in front of the action in the "cando" predicate indicates the modality of the authorisation e.g. negative authorisation would be denoted by a (-) sign in front of the read action. A "dercando" predicate is defined in the language to specify derived authorisations based on previous access using so-called "done" rules. These are essentially facts, which are created by the system during runtime and reflect the access executed by a user. Furthermore, the "do" and "done" predicates, can be used to specify history-dependent authorisations based on actions previously executed by a subject. The final decision, whether to grant access or deny a request is then resolved by a so-called decision rule. For example the following:

$$\boxed{do(file, s, +a) < -dercando(file, s, +a)\& - dercando(file, s, -a)}$$

This rule indicates that if it can be derived that s is authorised to perform action a on file, and it cannot be derived that s is denied to perform action a on file, then s is effectively authorised to perform a on file.

However, temporal dependencies among authorisations are not compositional. Thus, Siewe's [87] extends the language to allow temporal dependency of authorisations and caters for the compositionality of security policies, but the major points for criticism are that it is not possible to express obligation policies in ASL.

In addition, the approach of derived authorisations based on the system state or the occurrence of an event is still ambiguous. The Flexible Authorization Framework (FAF) [50] extends the approach in [49] to handle dynamic authorizations and is also based on stratified clausal-form logic. It includes support for *RBAC* and discusses propagation rules (for example the propagation of access rights from groups to group-members) as well as having an emphasis on integrity rules. However, the integrity rules in their model cannot take into account the result of the execution, as integrity constraints are checked prior to the decision whether the access is granted or denied - they are mainly concerned with the integrity of the access control policy specification itself.

2.4.2 LaSCO

The Language for Security Constraints on Objects (LaSCO) [45]is a graphical approach for specifying security constraints on objects. A policy graph is an annotated directed graph where nodes represent system objects and edges represent system events. Nodes and edges are decorated using domain predicates and requirement predicates. Domain predicates restrict what can match a node or edge while requirement predicates are constraints to be met on domain match.

Damianou [31] states some of the drawbacks of LaSCO. Firstly, it cannot express any form of obligation (which is also the case for ASL), secondly it is not compositional and thirdly there is no textual representation of LaSCO graphs, as is found in other graph-oriented languages.

The main advantage of LaSCO is that a graphical representation is more accessible to the human user and aids in the specification of security policies. We also believe that the representation of access rights in the form of a graph is intuitive for the analysis of permissible information flows in policies.

2.4.3 eXtensible Access Control Markup Language (XACML)

XACML [76] is a technology developed as a research project by SUN Microsystems. It is a standard language (XACML version 2.0 has been accepted by the Organisation for the Advancement of Structured Information Standards (OASIS)) for specifying access control policy, the structure of XML messages that request access to resources, and the structure of the messages responding to these requests. The language is based on eXtensible Markup Language (XML) and specifies a subject, object, action and condition policy in the context of XML documents. The language supports role based access control, where roles, the same as groups, are defined as collections of attributes relevant to a principal [19].

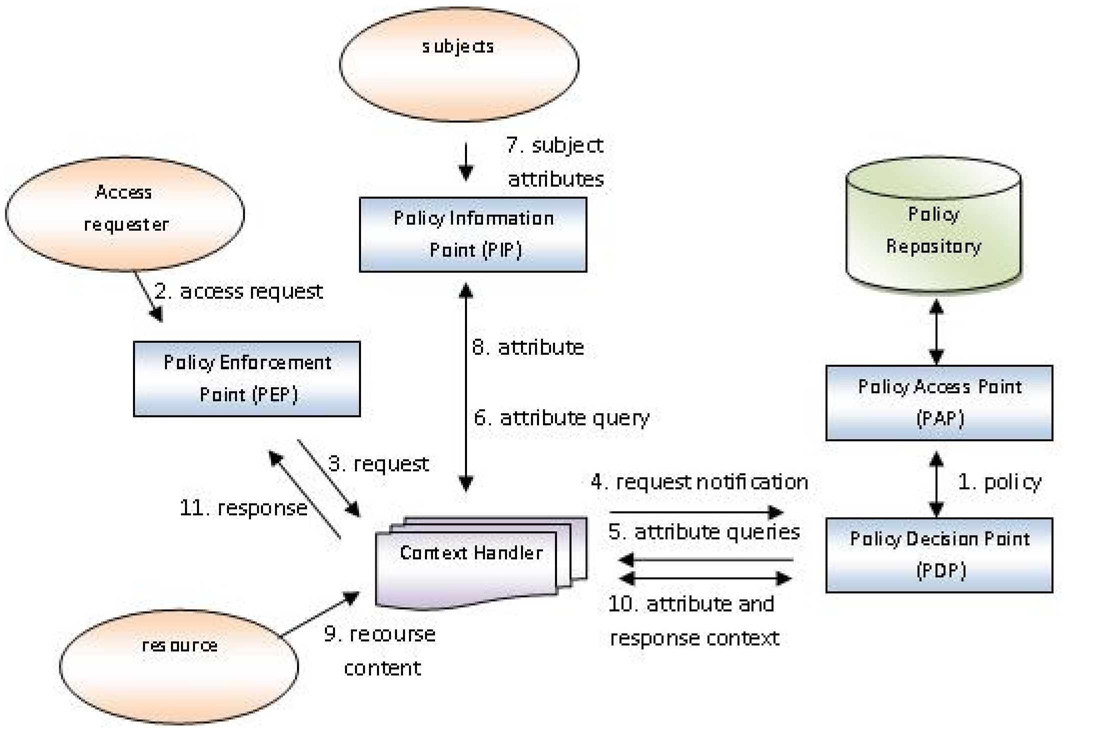


FIGURE 2.4: XACML Representation.

XACML's standardised architecture for decision making exploits two components: the Policy Enforcement Point (PEP) and the Policy Decision Point (PDP). The PEP constructs the request based on the requester's attributes, the resource requested, the action specified, and other situation-dependent information through PIP. The PDP receives the constructed request, evaluates it with the applicable policy and system state through the Policy Access Point (PAP), and then returns permit, deny, indeterminate or not-applicable to the PEP. The PEP then allows or denies access to the resource. The PEP and PDP components can be embedded within a single application or distributed across a network. Figure 2.4 illustrates the XACML standardised architecture as described above. In [34] XACML obligation is presented but the attribute coordination is not addressed between multiple PDPs. Moreover, in [60, 61], the description logic reason for the language is provided.

However, in XACML the underlying representation is XML; therefore, the policy is verbose and not really aimed at human interpretation. On the other hand, XACML comes with an open source reference implementation of a Policy Decision Point (PDP).

2.4.4 Ponder

The Ponder policy specification system [30, 33, 37], is an object-oriented, declarative policy language to express security and management policies that was created at Imperial College.

Ponder policies have been implemented for controlling policy-based networking (PBN) equipment or software written in the Java language. Moreover, it provides a common language abstraction over the heterogeneous components in such a security network. Additionally, the language supports native authorisation, delegation obligation and, unlike the previously discussed policy models [5], obligation policies [32]. Ponder also provides the grouping of objects in domains and roles and groups give structure to the system's subjects. The Ponder authorisation policy syntax is:

```
inst ( auth+ | auth-) policyName "{"
subject [<type>] domain-Scope-Expression ;
target  [<type>] domain-Scope-Expression ;
action   action-list;
 [ when  constraint-Expression ; ]"}"
```

Positive (auth+) and negative (auth-) authorisation policy defines which subject (group or role) can perform what actions (activities) on a domain of objects, where the former is indicative of a permission, and the latter a prohibition. As we mentioned, Ponder's roles can have attached obligation policies, which is the strength of Ponder. The syntax for such policies is:

```
inst oblig policyName "{"
on  event-specification ;
subject [<type>] domain-Scope-Expression ;
[ target  [<type>] domain-Scope-Expression ; ]
do   obligation-action-list ;
[ catec   Expression-specification ; ]
[ when  constraint-Expression ; ]"}"
```

Ponder is an attractive policy specification language, since it has tool-support which allows it to have a graphical view, it is suitable for large system policies and defines the domains and the Ponder compiler. It translates high-level Ponder specifications into low level policy languages, for instance the Windows 2000 security templates, or policies for the Java security model. The three policy languages, ACL, LaSCO and Ponder are discussed in the comparative study [5]. It critically compares the expressiveness of the BMA security model for Electronic Patient Records system developed in [10, 11]. It conclude that because Ponder allows the specification of obligation policies Ponder gives the most flexibility rather than ACL and LaSCO.

2.4.5 Usage Control (UCON)

$UCON$ is a relatively new approach that aims to unify prior models at a high abstraction level over the usage of digital objects and to combine Access Control, Trust Management and Digital Rights Management in one more expressive framework. The authors of $UCON$, Park and Sandhu [77, 85] address concepts behind $UCON$ and describe the different forms of usage in terms of rights mixture. The $UCON$ model abstracts a system, as previous models, into subjects, objects and rights. Furthermore, it obtains authorisation rules, obligations, and conditions. The authors in [78]introduced the ($UCON_{ABC}$) core models that particularly focus on the notion of mutable and immutable attributes that are associated with objects or subjects. Mutable attributes can change during the execution of the system, whereas immutable attributes can only change by administrative action. Before, ongoing and post usage are phases addressed in this model that satisfy denying a resource whilst the resource is already in use. Furthermore, they address pre-update, ongoing and post-update functions to comply with obligations and maintain mutable attributes. For example on classification of attributes we refer to [79]. Figures 2.5 and 2.6 illustrate the model.

While covering many aspects of former models, like access-history tracking or auditing, the definitions that are given in the paper are mostly informal and are

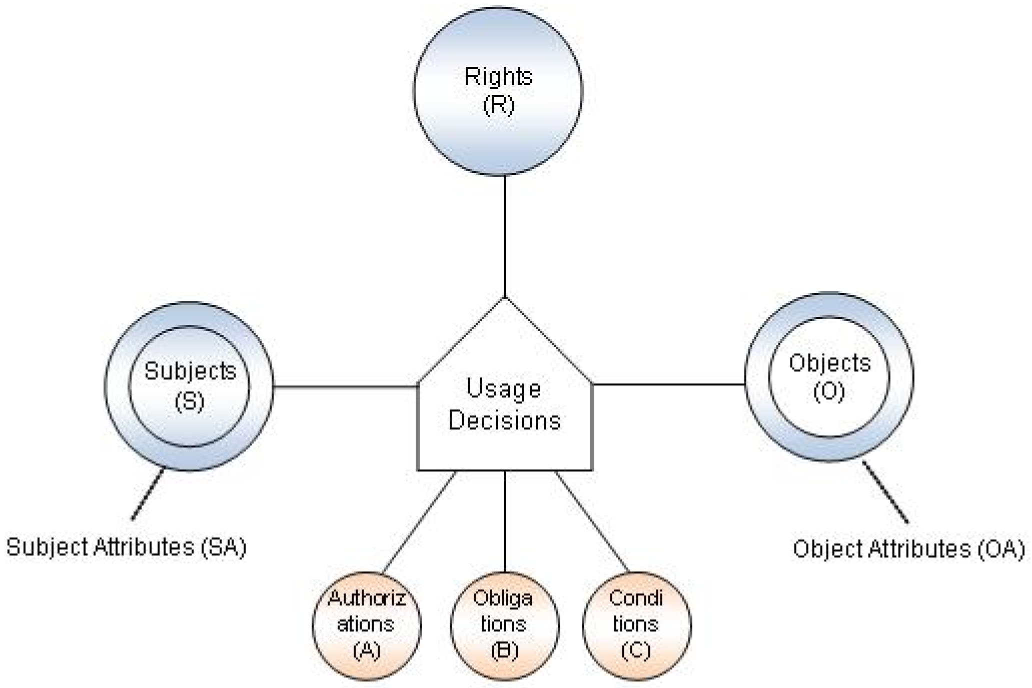


Figure 2.5: Continuity and Mutability Properties of UCON.

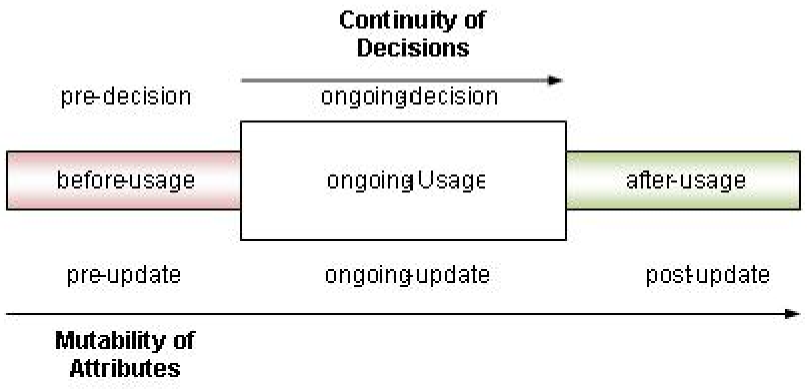


Figure 2.6: UCON Components.

not suited for the analysis of $UCON$ policies. However, it is a powerful abstract model that covers some aspects which are not covered by former frameworks. Later on, Zhang [101, 102, 103]formalised the different ($UCON_{ABC}$) models using Lamport's temporal logic of action (TLA) [64]for providing a formal definition of the ($UCON_{ABC}$) core models and their functioning. Obligations in $UCON$ are presented in [59].

In [56], an alternative approach to formalise $UCON$ model is presented which is based on ITL. The approach deals mainly with scenarios of continuous on-going usage. It redefines a concept of a usage process and request. Usage process is formed by a sequence of intervals constructed using a set of operators. Interval is a (in)finite sequence of system states. In [53, 54], although the formal model significantly reduces the number of assumptions made in [3], it is being improved to specify the potential of usage control.

2.4.6 Security Analysis Toolkit for Agents (SANTA)

SANTA [51, 87] is a framework which integrates the specification of security, functional and temporal requirements for the development of distributed systems in a uniform and formal framework that was developed at The Software Technology Research Laboratory, DMU. The authors have chosen Multi-Agent Systems as a representative of distributed systems for the reasons which have been addressed in [57] by Janicke. Interval Temporal Logic (ITL) [74] is used as the underpinning logic for SANTA. Therefore, it shows that the system implementation satisfies the functional, security and temporal requirements. Figure 2.7 outlines the SANTA framework.

The foundation is a formal computational model and a compatible security model. It provides linguistic support, in the form of the SANTA Wide-Spectrum Language (SANTA-WSL), that supports the specification and design of a Secure Multi Agent System (SMAS). The language is agent-based and offers constructs for the specification of reactive agents, objects that represent shared resources, security policies and their enforcement mechanisms. The specification-oriented semantics of SANTA-WSL is given in ITL which is the basic logical underpinning of the SANTA framework. Policies are defined at a higher level of abstraction than the system implementation itself. For instance, an authorisation policy defines the conditions under which a user can access a specific resource in a declarative manner (without

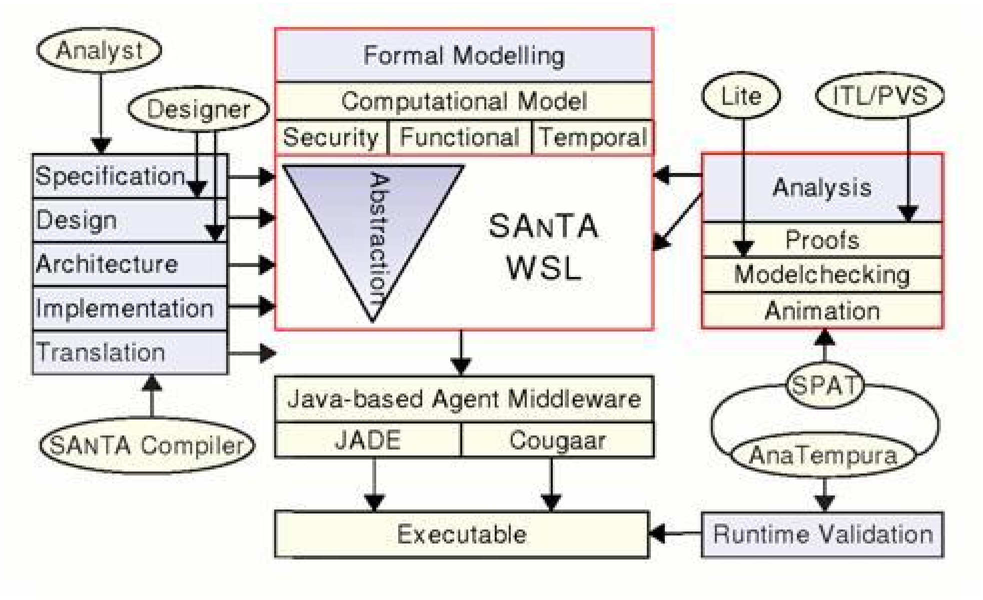


FIGURE 2.7: SANTA Components [51].

detailing how this constraint is actually implemented). SANTA has the ability to express behaviours as part of the specification removes the need to explicitly model state for the execution history as is, e.g. the case in *UCON* [78] or [17].

Additionally, enforcement mechanisms are defined at a high abstraction level. The system properties that must be satisfied to comply with a specific policy are stated by enforcement mechanisms. Then, during the development process, policies and enforcement mechanisms are refined into concrete and deterministic enforcement code that can be readily implemented in agent frameworks such as JADE [80]. Finally, SPAT [87]is tool-support for the specification and analysis of policies and is provided as part of the SANTA framework. The support for obligations specification, integrity constraints and scoping and parallel composition of policies is the strength of SANTA.

2.4.7 Trust Specification

Web based labelling, signed email, active networks and e-commerce applications require connectivity between entities that do not know each other for establishing and enforcing access control [42]. Thus, there has been significant prior research into schemes which allow gradual exchange of digital certificates or credentials that represent statements certified by given entities, which can be used to establish properties of their holder (e.g. identity, accreditation). An access control decision of whether or not a party may execute an access is based on properties that the party may have, and can prove by presenting credentials. This approach is called certificate-based authorisation and is adopted for trust specification [30]. The access control decision is a complex process and its completion sometimes requires trust negotiations for privacy guaranteed, safe and fair credentials exchange between parties (subject and object providers) [67].

Yao [99] has described a number of existing trust management systems, in particular tracing PolicyMaker's [24] evolution into KeyNote [23] and the Internet Engineering Task Force's (IETF) Simple Public Key Infrastructure (SPKI) [39]. Most of these schemes are based on asymmetric digital cryptography as well as providing a universal solution to both access control and authentication which make the systems more complex.

2.5 Policy Based Management (PBM)

Policy Based Management (PBM) is a technology wherein various resources on a network are not individually configured but rather are dynamically configured by an overlay network [69] based on various policies that are defined by a network administrator. In this section, an overview is introduced as well as a current framework and protocol is described.

2.5.1 Overview

Present day computer networks are extremely complex both in terms of the technologies behind them as well as the variety of network hardware that they use. There are multiple types of servers, routers, switches and other network hardware supplied by multiple vendors. In addition, not all the systems on a network may use the same software platform. Due to this, a network administrator would typically be required to know about multiple operating systems, hardware and software and be continuously informed and updated about the various new features being continuously introduced. Policy Based Management (PBM) provides a way to overcome this problem by simplifying and largely automating the network administration process.

Ensuring that the many different devices on present day IP networks inter operate smoothly is not a trivial task. In addition, various new technologies that have emerged in order to overcome a few limitations of the traditional IP protocols, such as delivering Quality of Service, have only served to make present day networks more and more complex. In many cases, it is more economical to simply introduce excess capacity into existing networks rather than train the manpower on the different technologies that are continuously being introduced. Indeed this is often the route employed by many network administrators. However, with the advent of Policy Based Management, this is poised to become a thing of the past.

PBM is a technology wherein various resources on a network are not individually configured but rather are dynamically configured by an overlay network [69] based on various policies that are defined by a network administrator. For example, instead of configuring a hundred printers on a network independently to print documents only in black and white, a network administrator would simply define a single policy for the same and update it on a central policy server. This policy would then automatically be applied to all printers on the network by the network management software. The Internet Engineering Task Force (IETF) and the Distributed Management Task Force (DMTF) [100] are the chief bodies for the standardisation of this policy framework so that various resources and technologies can comply with it and it can become an accepted standard.

2.5.2 IETF Policy Based Admission Control Framework

The Internet Engineering Task Force (IETF) has proposed a framework for policy based admission control [100] in order to aid applications or end users to request specific quality or levels of service from an internetwork in addition to the IP best effort services. This framework addresses important aspects of admission control which were previously unresolved. As per RFC 2753, "network managers and service providers must be able to monitor, control, and enforce use of network resources and services based on policies derived from criteria such as the identity of users and applications, traffic/bandwidth requirements, security considerations and time of day/week."

The IETF/DMTF policy framework consists of four elements the Policy Management Tool, the Policy Repository (PR), the Policy Decision Point (PDP) and the Policy Enforcement Point (PEP).

The Policy Management Tool is the user interface of a Policy Based Management system. The network administrator uses the policy management tool to specify policies that are to be enforced on all devices in a network. These policies are stored in a Policy Repository (PR). The actual point at which the policies are enforced on the network is known as the Policy Enforcement Point (PEP). This Policy Enforcement Point gets its directions from the Policy Decision Point (PDP) which is responsible for interpreting the policies stored in the Policy Repository and communicating them to the PEP. The general layout of these elements in Policy Based Management is shown in Figure 2.8 below.

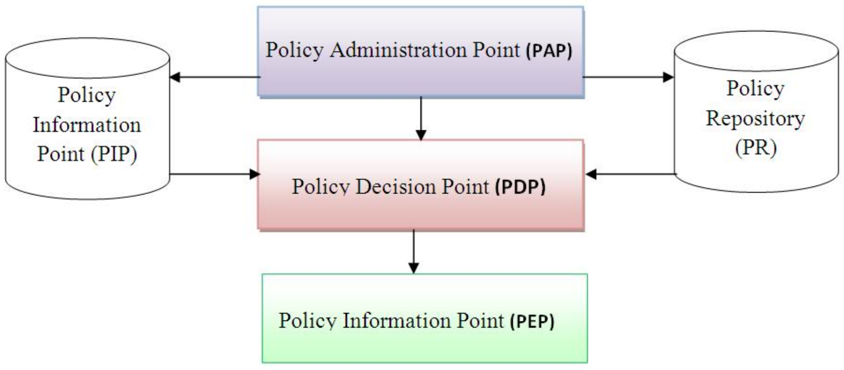

Policy Administration Point **(PAP)**
Policy Information Point (PIP)
Policy Repository (PR)
Policy Decision Point **(PDP)**
Policy Information Point **(PEP)**

Figure 2.8: IETF/DMTF policy framework.

The policies stored in the Policy Repository must correspond to the information model specified by the Policy Framework Working Group in order to allow various devices with various capabilities and from different vendors to inter operate with each other. Various standardised protocols are used by various components of the Policy Management System to communicate with each other. The Common Open Policy Service (COPS) [38] or Simple Network Management Protocol (SNMP) [26] is generally used for PDP and PEP communication. The Policy Repository may be accessed using the Lightweight Directory Access Protocol (LDAP) [43].

Effective implementation of Policy Based Management is dependent on two features of the management architecture which are centralisation and business level abstraction. Centralisation is the means by which all devices on a network can be configured from a single point which is the Policy Management Tool. Without the benefits of PBM an administrator would have to spend hours configuring and provisioning each device on a network manually. With a Policy Based Management system, however, he/she simply requires to spend a few minutes entering the desired policy into the Policy Management Tool and the system will take care of the rest and apply the policy to all the various devices automatically.

As we have seen, there are two basic elements for policy control the PEP (Policy Enforcement Point) and the PDP (Policy Decision Point). The Policy Enforcement point (PDP)-. is the point at which the policies for network management such as admission control policies based on factors such as the time of day, user identity, credentials, user groups, service agreements between various ISPs related to revenue sharing on the basis of bandwidth usage, priorities for services such as video conferencing, VoIP etc. are actually enforced. The Policy Decision Point is responsible for making decisions which the PEP enforces.

When the PEP requires to make a decision regarding access control for resources, bandwidth etc, it formulates a request for policy control which may contain one or more policy elements detailing the resource requested and sends it to the PDP. The PDP accesses the policy repository and returns the policy decision which the PEP then enforces by appropriately accepting or denying the request. The PDP may also return additional information to the PEP which may be used for formulating error messages or warnings based on pre defined criteria. The PDP may use additional mechanisms and protocols for providing accounting, authentication, policy storage and other functions. For this purpose, it may contact other external servers using protocols defined for network management and communication like SNMP (Simple Network Management Protocol) [26] or LDAP (Lightweight Directory Access Protocol) [43] among others. The detailed component framework for Policy Based Admission Control is shown in Figure 2.9.

We now come to the question of the format in which the policies should be entered into the policy management tool. Due to the large plethora of vendors and equipment available on the market today, one policy format may not be compatible between all the multiple equipment that the network supports. For this reason, policies can be defined at two levels a business level and a technology level. Business level policies are general policies that apply to all the various devices on the network whereas technology level policies are policies that are derived from the business level policies according to the technological requirements and instruction

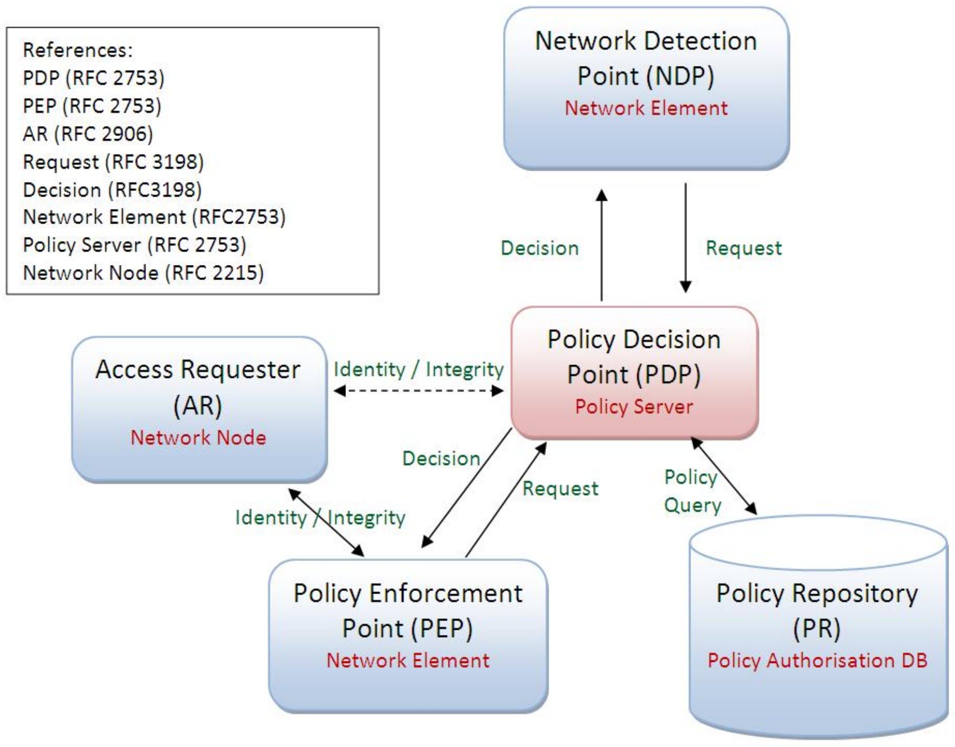


FIGURE 2.9: Components Architecture for Policy Based Admission Control.

formats of the various devices on the network. The process of separating the business level policies from the technology level policies is known as Business Level Abstraction. The network management system uses a policy transformation logic to transform the business level policies into technology level policies based on the various devices on the network and stores them in the policy repository from where they are accessed by the PDP in order to convey a decision to the PEP based on the policy.

2.5.3 Common Open Policy Service (COPS) Protocol

The COPS protocol is a simple query and response protocol between a PEP and a PDP. The basic model of this interaction is compatible with the framework document for policy based admission control [38]. As per RFC 2748 [38], Durham et. al describe the COPS protocol in detail, the main characteristics of the COPS protocol include:

- COPS utilizes a client-server model. It is the PEP that sends requests to the PDP which acts as the server and the PDP responds with its decisions.
- TCP is used as the transport protocol for exchange of messages and no additional mechanisms are specified for reliable communication between the PDP and the PEP.
- The protocol is designed for general configuration and enforcement of policies and uses self identifying objects that encapsulate all relevant information for decision making. As a result the protocol itself need not be modified for various usage scenarios.
- Though COPS provides for message level security for message integrity and protection, existing protocols for security such as IPSEC [14] or TLS [35] may optionally be utilized as a security measure to secure communications between the PDP and the PEP.
- The protocol is intended to be stateful. The PDP is expected to retain previous requests made by the PEP unless they are explicitly deleted by the PEP and may generate asynchronous decisions at any time to modify a previously installed request state. Also it may respond to a new request differently based on the decision made on a previous request. The PDP is also capable of transmitting configuration information about various devices to the PEP and modifying or removing them at a later stage as required.

2.6 Enforcement

The enforcement of policies refers to providing mechanisms in the system which can ensure that the policy specification is not violated by the system's execution [51]. A well known and widely used architecture for policy enforcement is the Reference Monitor (RM). The ISO standard (ISO/IEC 10181-3:1996) [47] defines the standard model for policy enforcement that is used in most centralised and distributed policy enforcement approaches. Figure 2.10 depicts this model. The model separates the enforcement and the policy evaluation into a Policy Enforcement Point (PEP) and the Policy Decision Point (PDP). PEP is the component intercepting the request, and the Policy Decision Point (PDP) is the component that evaluates an access request against a policy and reports the decision back to the PEP. Thus, it is beneficial as the implementation of the PEP is usually application or platform specific [9].

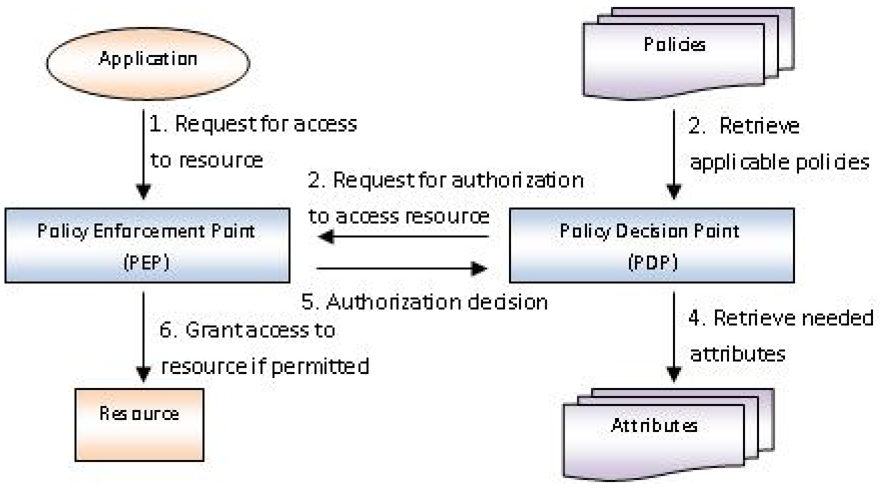

Application
Policies
1. Request for access to resource
2. Retrieve applicable policies
2. Request for authorization to access resource
Policy Enforcement Point (PEP)
Policy Decision Point (PDP)
5. Authorization decision
6. Grant access to resource if permitted
4. Retrieve needed attributes
Resource
Attributes

FIGURE 2.10: Policy Enforcement Model.

2.6.1 Centralised Policy Enforcement

The policy enforcement model in the ISO standard (ISO/IEC 10181-3:1996) [47] is based on a centralised enforcement decision mechanism that receives events from different components of the system. This means enforcing local security policies in which all the security relevant events are produced by a single node. Therefore, a centralised policy decision point in large scale computational systems is not sufficient due to the heavy congestion resulting from the very large number of events which becomes a single point of failure. A major challenge to shift the computational mechanism towards distributed systems is interpreting and efficiently enforcing security policies. Most approaches to policy enforcement such as [1] and [70] are based on the centralised enforcement decision mechanism.

The centralised Policy Based Management (PBM) is more efficient when the length of the authorisation decision and the authorisation traffic to performing any service are not critical and the service can be provided later, e.g. banking services. On the other hand, in such systems, when the authorisation decision response time and the traffic are being important, the Decentralised PBM becomes more efficient, e.g. Military, National Security.

2.6.2 Decentralised Policy Enforcement

The chief disadvantages of having a centralised PDP is that though coordination among various decision requests are easily achieved, it is a bottleneck to performance because all requests have to be routed through it.

The GlobusToolkit [86] have addressed distributed PDPs (Master/Slave PDPs model) and the coordination between them. The master PDP orchestrates the querying of a series of slave PDPs which each make their own (sub)-authorisation decisions whilst the master then determines whether the ultimate decision is granted or denied. They do not detail how their "master policies" are incorporated into the XACML-PDP policies. In addition, the model does not address coordination between multi-user access requests.

Another method to enhance the PDP's performance is to add LPDP (Local Policy Decision Point) as an assistant to the centralised PDP, such as the case in [38]. LPDP stays in the same application or network node with a PEP, rather than in the server as a centralised PDP does. It frequently backups the decisions of the PDP, so when connection is interrupted between the PDP and the PEP, the LPDP can substitute the PDP to guide the PEP temporarily. But centralised PDP has the superior authority. LGI [71] is based on controllers that moderate the network traffic going to and coming from observed nodes. Minsky et. al [71] have illustrated a conceptual mechanism of Distributed Chinese-Wall (an example of stateful policies) that can be enforced using LGI where it is a decentralised enforcement mechanism. LGI has provided the exchange of messages between targets, and does not deal with the behaviour of targets, or to changes in their internal state. The significant differences between LGI and our work is that the enforcement depends on agent cooperation in LGI where as in our work it is automatically enforced on objects-domain or in another domain that has been made decisions in the past by other PDPs.

A set of distributed PDP's co-located with each of the PEP's would solve this problem. One of the ways which have been proposed in order to achieve distributed PDP's is to break down a main policy into its component parts based on the target device to which it applies [90]. Though we will not delve into the mechanics of how exactly decomposition is done in this paper, it should be clear to us that when a main policy is broken down into its various component parts, coordination is required between the various PDP to whom it is distributed to ensure that the component policies produce the same effect as intended by the main policy. The sum of the parts should perform in exactly the same manner as the main policy performs.

In [90], [68], the notion of policy decomposition have been provided. Lin et. al. [68] identify that the policy decomposition is based on the sensitivity of attribute information necessary for access control and/or user defined constraints at each PDP. However, Chadwick et. al. [90] policy decomposition model is guided by the resource type hierarchy. Their model considers refining a high-level access control policy into sub-policies that is specific to a resource instance and then sub-policies are then deployed at the PDPs controlling each resource. In comparison, our policy decomposition model is not guided by the resource type hierarchy or the sensitivity of attribute information but object domains, thus, not every domain has its PDP where that is based on policy analysis.

Chadwick et. al. [27], have also provided a conceptual model for coordination by sharing and exchanging coordination data between one or more PDPs and have extended the policy specification to include coordination and obligation policies. Nevertheless, their work is informal and they do not show how new policies are deployed to the PDPs when the system configuration or policy changes. Their proposed coordination object is conceptually a repository which maintains a persistent record of the various coordination attributes each time a request to access a particular controlled resource is permitted. It may contain multiple dimensions for the subject, resource, action or the environment variables. It is persistent and stateful, much like the environment variables of date and time - the only difference being that in the latter, the system is only required to read the values whereas in the former, it is required to read as well as update the values on successful execution of the request. A case when the subject attribute may require coordination is in a situation where maybe only one of a group of users or user groups is allowed access to a particular resource. The resource attribute may require coordination in practical scenarios like limiting the total usage of bandwidth per user to say 3 GB. The action attribute may require to be coordinated in cases when the same user cannot both create and execute a given test and the environment attribute may require coordination in cases where no more than, say 250 Dollar is to be withdrawn from a given ATM in a given day. In this research, we study this interesting problem and propose an efficient decentralised enforcement mechanism of security policies in distributed systems while managing the coordination between them.

2.7 Summary

In this Chapter we reviewed the specification and policy enforcement, obligation and integrity constraints of access control. Particularly, we focussed on policy models and languages for the specification of these requirements in access control systems. Many languages to define security policies have been proposed. The more formal, logic-based approaches (e.g. ASL) in general lack the flexibility and scalability of more informal specifications (e.g. Ponder, XACML), but have the advantage, that properties of the specification can be proved. The advantage of the informal models is typically the increased scalability, due to concepts like inheritance or instantiation of policy classes as well as the generally better developed tool-support. As has been noted by others [1, 70], most languages that did not start with a formal model as the underlying foundation are prone to semantic ambiguities. Thus, a mathematical approach is essential to achieve a level of assurance in the policy. Additionally, Janicke [51]emphasises Becker's view, that the logical foundation should not hinder the understandability of the model and the language through its complexity and give the example that the algebra for policy composition presented in [96], whilst addressing an important subject, appears to introduce a level of complexity that is not justifiable. Finally, we concluded the discussion of related work with the Policy Based Management (PBM) systems in both centralised and decentralised approaches.

Unfortunately, no policy models and languages have defined the distributed enforcements mechanism of dynamic security policies that achieve the mean of Decentralised Policy Based Management systems.

Chapter 3

Preliminaries

Objectives

- Give an overview of the SANTA policy language formalisation that involved in the contribution.

3.1 Introduction

The Security Analysis Toolkit for Agents (SANTA) history-based dynamic PBM system [87] [51] has a formal underpinning in Interval Temporal Logic (ITL) [74] allowing formal authorisation and verification to take place [56]. We build our decomposition policy model on the formal policy model of SANTA to provide a high level of assurance. Consequently, our framework can automatically distribute PDPs in the system in order to provide resilience against network failure and efficacy of policy decision making.

The SANTA security policy framework involves three main components. Firstly, the policy model allows for the expression of security requirements such as authorisation, obligation and delegation (delegation is excluded in our work). ITL is appropriate to express temporal dependencies and the dynamic change of system requirements in a compositional manner for outcome of policy decisions. Moreover, ITL specification can link different granularity of time via the projection operator. In addition, ITL has an executable subset, Tempura, which means an interpreter for our decomposition policy is readily available, if expressible in this subset. Therefore, much of the proof of a system that is specified in ITL can be decomposed into proofs of its parts. Secondly, linguistic support is presented to abstract from the underlying mathematical description and appeal to a more business-oriented audience. However, the semantics of the language is unambiguous as it directly translates into the underlying model. Finally, the Security Policy Analysis Tool (SPAT) [87] is tool-support for the specification and analysis of policies that are provided as part of the SANTA framework. SPAT allows for the prototyping and validation of policies, and which assists policy developers to translate high-level requirements into concrete policies.

3.2 SANTA Policy Language

A security policy expresses protection requirements on the system in a precise and unambiguous form. Policies in SANTA are concerned with access control, obligations and integrity. They relate to the entities in the system, and define constraints on their interactions. Access control requirements in this model are authorisation requirements, viz. constraints on the actions that a subject can perform on objects. Obligation requirements express that subjects must perform specific actions.

The aim of policies is to express these requirements at a high level of abstraction, hiding the details of the implementation that is necessary for their enforcement. In SANTA, policy rules are used as the basis for policy specifications. Rule-based languages are well established and well suited because most of these requirements are already informally expressed in the form of conditions and consequences. Each rule is expressed in terms of subjects, objects and actions. Subjects are the actors in the system. They can request access to objects, that represent the available resources.

In SANTA two different types of policies are distinguished: environmental and behavioural policies. The former represents the more traditional view that the access to shared resources in the environment is protected. The latter addresses constraints on the behaviour of agents in the system which is not in our considered in work.

Policy Specification is the process of expressing informal protection requirements within the policy language. Policy rules form the basis of SANTA policy specifications and one of the major tasks during the specification process is the development of rules that adequately capture the informal requirements. The accurate specification of rules can be difficult when complex requirements need to be expressed. These are for example dependencies on the state of the system or dependencies on the history of the execution [3]. The SANTA policy language provides support for both state and history dependencies. Not all requirements can be easily expressed in form of rules. One example would be an electronic paper submission system that is staged in different phases, e.g. registration, submission, review, etc. During each phase a different policy applies, controlling the ability of users to submit, review and comment on a paper. This would require the policy to change dynamically at the transition from one phase to the next. This form of requirements is difficult and cumbersome to encode in rules, because the different phases have to be considered in each of the rules. The approach in SANTA is different in that these types of requirements are captured through policy composition. Thus, SANTA policy specifications is being more efficient to be involved for the policy decomposition to be deployed in distributed enforcements.

Policy Composition The advantage of SANTA policies over the majority of other policy languages is that policies can be specified in small units, which are composed using a rich set of operators. The provided operators allow for policies to be composed along a temporal and structural axis.

Temporal composition leads to policies that change dynamically over time or on the occurrence of events. This can be used to specify the transition from one policy to another, for example to capture protection requirements for different phases.

Structural composition is concerned with the separation of subjects, objects, actions and policies, that apply only to a certain subset of these entities. A typical example is a hierarchical organisational structure, where each department defines its own policies. The combination of all these policies, together with some general protection requirements, yield the overall policy that applies to the organisation as a whole. The concept of Dynamically Changing Access Control Policies has been investigated by Siewe [87]. Indeed, his work on temporal composition of authorisation policies forms the basis of the SANTA policy model. It has been significantly extended to capture other types of requirements, such as delegation, obligation and integrity. The structural composition and the unique problems that arise for the conflict resolution between two dynamically changing policies are addressed.

Using these new concepts, policies can be specified as compositions of smaller, simpler policies along both the temporal and structural axis. This compositionality is also of great advantage for the enforcement of policies. Whilst the overall composition allows for the analysis of the system wide effects that the enforcement of the policy has, it is also a great aid to decompose policies into units that are enforceable by the mechanisms present in the system. This makes it possible to drop the assumption of a centralised enforcement mechanism and replace it by the decentralised enforcements. Therefore, the policy composition is not an objective in this research where it is well described in [87] and we focus on both Access Control List (ACL) [2] and History Based Policy (HBP) [3] only.

3.2.1 Policy Syntax

Policies in SANTA are an integral part of the system specification. They capture protection requirements for the IS. The smallest unit of a policy specification is a policy rule. Rules capture individual requirements, such as: "allow students to submit their assignments." The rule syntax and the informal semantics is detailed in [51] where it provides examples and the informal meaning of authorisation, delegation, obligation and integrity rules. Rules are combined into larger units, named **simple policies**. Simple policies are a set of rules that are all enforced simultaneously. They define for example the protection requirements that apply in a specific situation or phase of the system execution.

3.2.1.1 Policy Rules

The rule-based approach to policy specification is advantageous because it provides a higher level of abstraction to the specification. An access control rule for example describes under what conditions a specific access control decision is taken it does not define the actual mechanism that is used to enforce this decision. In this sense the policy is more abstract than a concrete check that is implemented directly in the accessing code. Also, the syntax of policy rules contains some high level constructs to reference the past behaviour of the system. Examples are the temporal modalities **always** and **sometime**.

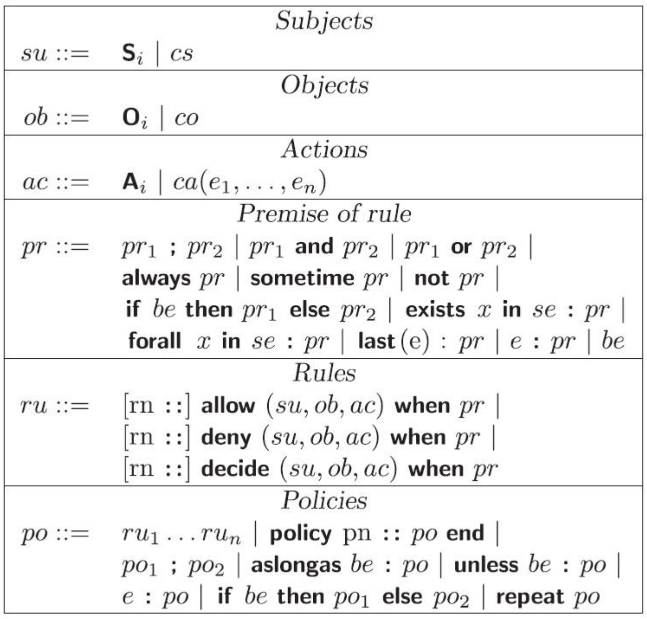
Rule Structure A simple policy (central policy) is a set of policy rules enclosed in parenthesis. The set may be empty. Every rule consists of a premise and a consequence. The premise describes the condition that when observed by the enforcement mechanism leads to he specified consequence. Figure 3.4 defines the rule structure which is presented in [54].

The general form of a rule is:

$$\boxed{\mathit{consequence}\ (x, y, z)\ \mathit{when\ premise}}$$

The consequence of a rule distinguishes the class of requirements that can be expressed in that rule. The triplet (x,y,z) references the subject, object and action to which the rule applies. Finally, the premise of the rule describes the condition under which the rule fires.

Policy Scope. Every policy has a scope, that defines to which subjects, objects and actions the policy applies to. The scope of a policy is accessible in the syntax using the keywords subjects, objects and actions, that represent the set of subjects, the set of objects and the set of actions in the scope of the policy. The scope of a policy affects the rules contained in that policy. The subject, object and action in each rule definition must be within the scope of the policy ($x \in \mathit{subjects}$ $o \in \mathit{objects}$ and $z \in \mathit{actions}$). Rules can also be defined in terms of all subjects, objects and actions in the scope using the keywords S, O and A. This provides a



	Subjects
$su ::=$	$\mathsf{S}_i \mid cs$
	Objects
$ob ::=$	$\mathsf{O}_i \mid co$
	Actions
$ac ::=$	$\mathsf{A}_i \mid ca(e_1, \ldots, e_n)$
	Premise of rule
$pr ::=$	pr_1 **;** $pr_2 \mid pr_1$ **and** $pr_2 \mid pr_1$ **or** $pr_2 \mid$ **always** $pr \mid$ **sometime** $pr \mid$ **not** $pr \mid$ **if** be **then** pr_1 **else** $pr_2 \mid$ **exists** x **in** se **:** $pr \mid$ **forall** x **in** se **:** $pr \mid$ **last** (e) **:** $pr \mid e$ **:** $pr \mid be$
	Rules
$ru ::=$	[rn **::**] **allow** (su, ob, ac) **when** $pr \mid$ [rn **::**] **deny** (su, ob, ac) **when** $pr \mid$ [rn **::**] **decide** (su, ob, ac) **when** pr
	Policies
$po ::=$	$ru_1 \ldots ru_n \mid$ **policy** pn **::** po **end** $\mid po_1$ **;** $po_2 \mid$ **aslongas** be **:** $po \mid$ **unless** be **:** $po \mid e$ **:** $po \mid$ **if** be **then** po_1 **else** $po_2 \mid$ **repeat** po

FIGURE 3.1: SANTA Rule Structure [54].

greater flexibility in the expression of rules. By default the scope is the universal scope, viz. the sets of all subjects, objects and actions in the system.

Using the keywords S, O and A. Often rules apply to more than one subject, object or action. To avoid the effort to duplicate the rules for the different subject, object and action pairings, the keywords S, O and A can be used to reference free variables in the rule definition. These free variables are bound by the scope of the policy in the semantics of each rule. For example, given that the scope of the policy is:

$$subjects = \{x_1, ..., x_n\} \quad objects = \{y_1\} \quad actions = \{read\}$$

The requirement to unconditionally grant all subjects in the scope of the policy the right to perform read on object o1, would require the definition of n rules:

$$\boxed{\begin{array}{c} allow\ (x_1, y_1, read)\ when\ true \\ allow\ (x_2, y_1, read)\ when\ true \\ \ldots \\ allow\ (x_n, y_1, read)\ when\ true \end{array}}$$

Using the keyword S this could be written more compact as:

$$\boxed{allow\ (S, y_1, read)\ when\ true}$$

Referencing state and history in rules. Rules in SANTA can express state and history-based dependencies. This is achieved by allowing the premise of a rule to reference the current state of the system or the behaviour of the system in the past. The referenced state and behaviour is restricted to the part of the system that is observable by the mechanism enforcing the policy. This means that policies and the mechanisms enforcing them cannot be seen in separation if the requirements are dependent on the system state or the history of the execution.

These constraints are not a limitation of the policy language itself. Policies can violate these restrictions. However, it is beneficial to be aware of the limitations of the enforcement mechanisms when writing the policy to ensure that it is enforceable.

Premise of Policy Rules.

The premise of a rule allows the expression of a set of behaviours, that when observed trigger (or fire) the rule. The syntax of the premise is restricted to ensure that the rules are implementable. Due to the expressiveness of the model care needs to be taken to ensure that the premise does capture the requirement correctly and does not define a too large (or too narrow) set of behaviours that trigger the rule.

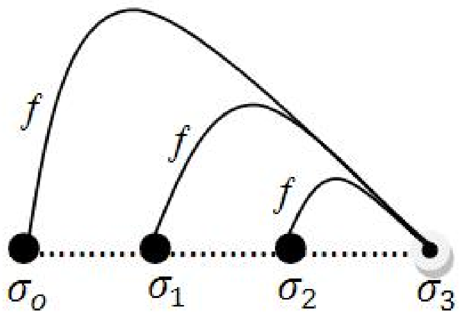


FIGURE 3.2: Informal Interpretation of a Policy Rule

Figure 3.5 depicts informally the relation between the set of behaviours expressed in the premise and the consequence.

Assume the policy containing the rule started to be enforced in state σ'_0 . Given that the premise of a rule is represented by the formula f and the consequence of the rule is evaluated in the state σ'_3 , then the consequence is *true* if the behaviour described by f is satisfied by at least one of the intervals depicted in Figure 3.5. For example the rule *consequence when* $x = 0$ would mean that if x has been zero in state σ'_0 , or σ'_1 , . . . , or σ'_3 then the consequence would be *true* in state σ'_3 . The expressiveness of the policy model is both a boon and a bane. The benefits are that many complex requirements can be captured in a concise and short form. The disadvantage is that some requirements, that are straightforward to express in other policy models, require more thought. This is especially the case when rules depend on the past behaviour of the system. The following illustrates the syntax that can be used to specify premises of policy rules that are state or history-dependent.

Constraining the length of the behaviour. Often rules are state-dependent, this means that f does not actually express a behaviour, but rather a predicate on the current state of the system. For example to express that the rule should only fire if x is now (in state σ'_3) equal to zero then the length of the formula f must be explicitly restricted by writing:

$$\boxed{\textit{consequence when } 0 \ : \ x = 0}$$

Sequence. It is also possible to specify a list to restrict the length of the behaviour, for example the rule:

$$\boxed{\textit{consequence when } [0\ .\ .\ 2] \ : \ x = 0}$$

would fire if $x = 0$ holds in σ'_2, σ'_1 or σ'_0 , viz. on an interval of length 0, 1 or 2 in the past. The concrete meaning of 2 states in the past depends on the concrete enforcement mechanism that is chosen for the enforcement of the policy. For the enforcement mechanisms described in SANTA the transition from one policy state to the next means that the enforcement mechanisms solicited (granted or denied) exactly one access. Figure 3.5 depicts informally the relation between the set of behaviours expressed in the premise and the consequence.

Temporal Modalities. Often the modalities **always** or **sometime** are used informally in requirements. These can be directly expressed in SANTA. Their scope is restricted by the length of the interval specification that contains the modality. For example writing:

$$\boxed{\textit{consequence when } 2 \ : \ \textit{sometime } x = 0}$$

Denotes within the past interval length two there is some suffix interval that satisfies $x = 0$. In other words it means that:

$$\boxed{\textit{consequence when } 2 : \ (x = 0 \textit{ or } (1 \ : \ \textit{true}; \ x = 0) \textit{ or } (2 \ : \ \textit{true}; \ x = 0))}$$

The global length specification is important, to limit the interpretation of the behaviour to the past interval of length 2. The case for always is similar, however all suffix intervals must satisfy $x = 0$, which is equivalent to replacing the or in the above rule with an and. The informal meaning of the conditional choice is self explanatory. The existential and universal quantification can be used to quantify over lists, e.g. the list subjects, objects and actions that can be used to access the policy scope.

Consequence of Policy Rules

Authorisation, Delegation, Obligation and Integrity rules are distinguished syntactically by their consequence. However, we are not interested in Delegation and Integrity rules in SANTA so refer the reader to [51] for more details. Table 7.1 provides an overview of the available consequences:

Authorisation (See Section 3.3.1.2)	
$allow(x, y, z)$	positive authorisation
$deny(x, y, z)$	negative authorisation
$decide(x, y, z)$	decision rule

Obligation (See Section 3.3.1.3)	
$oblige(x_1, x_2, z)$	obligation

Table 3.1: Consequences in Policy Rules

3.2.1.2 Authorisation Rules

Authorisation rules express access control requirements. Three different types of rules are concerned with authorisation: positive authorisation, negative authorization and decision rules.

Positive Authorisation. Positive authorisation rules are statements that indicate under which condition an access request should be granted. It is important to note that it is only an indication, which is taken into account for the final access decision of the policy.

Negative Authorisation. Negative authorization rules are statements that indicate under which condition an access request should be denied. Similarly to positive authorisations, they are only an indication, which is taken into account for the final access decision of the policy.

State-based Authorisation Rule. The condition in this rule must be fulfilled at the time of the access control check.

Example 3.1 (Positive Authorisation)

$$\boxed{allow\ (S, temp, clear)\ when\ true}$$

The rule states an unconditional positive authorisation for all subjects to perform action (*clear*) on file (*temp*).

Example 3.2 (Negative Authorisation)

$$\boxed{deny\ (S, marks, write)\ when\ 0:\ group(S, student)}$$

The rule states the negative authorisation for any subject in the policy scope that is a member of the group *student* to write in file *marks* where the predicate group(subject, group) denotes the group membership test. This relation is maintained by the system and must be accessible to the mechanisms that enforce the policy. This is an example of a The preceding 0: denotes that this condition must be fulfilled at the time of the access control check.

Example 3.3 (Negative Authorisation)

$$\boxed{deny\ (S, O, read)\ when\ 0 :\ level(O) > clearance(S)}$$

The second rule is the no read up rule from the Bell-LaPadula policy [66]. It states that no subject with a clearance level (denoted by clearance(S)) that is lower than the security level of the object (denoted by level(O)) is allowed to read information.

History-based Authorisation Rule. The condition in this rule must be fulfilled based on the past of the access control check.

Example 3.4 (Positive Authorisation)

$$\boxed{\begin{array}{l} allow\ (S, O, A)\ when\ true \\ allow\ (S, passwd, write)\ when\ 0\ :\ group\ (S, admin) \\ allow\ (S, O, A)\ when\ T :\ (\\ exists\ y\ in\ objects\ :\ exists\ z\ in\ actions\ : \\ ((dataset(O) = dataset(y))\ and \\ (sometime\ done\ (S, y, z))) \end{array}}$$

The first rule states an unconditional positive authorisation for all subjects, objects and actions in the scope of the containing policy. This is an example of an activity based authorisation rule. The second rule states the positive authorisation for any subject in the policy scope that is a member of the group *admin*. Here the predicate *group(subject, group)* denotes the group membership test. This relation is maintained by the system and must be accessible to the mechanisms that enforce the policy. The preceding 0: denotes that this condition must be fulfilled at the time of the access control check. This is an example of a state-based authorisation rule.

The third rule states a positive authorisation from the Chinese Wall policy [25]:

Once a subject [denoted by S] has accessed [denoted by z] an object [denoted by y], the only other objects [denoted by O] accessible by that subject are within the same company data-set [denoted by dataset].

The preceding T: denotes that the length of the subsequent behaviour is the time since the rule started being enforced. The rule is actually more precise, as it explicitly specifies that O and y must have been in the same data-set at the time the rule started being enforced. The sometime done(S,y,z) denotes that at some point in time since the enforcement of the rule the subject S successfully performed an action z on an object y in the same data-set. This rule is an example of a history-based authorisation rule.

Decision Rules and Conflict Resolution. Decision rules specify the final access control decision of a policy. Any policy should contain at least one decision rule, as otherwise no access will be granted by the policy. The alternative term "conflict resolution rule" originates from the fact that this rule de-conflicts the policy if a positive and negative authorisation is derived for a specific access. The term decision rule describes more accurately the fact that any access control decision is defined by the policy is decided by one or more of these rules not only decisions in the conflicting case. The examples below provide three widely used decision rules.

Example 3.5 (Decision Rule)

$$\boxed{\textit{decide}\ (S,O,A)\ \textit{when}\ 0:\ \textit{allow}\ (S,O,A)}$$

This rule states that access is granted if a positive authorisation can be derived from the policy. This rule is used in closed policies, where any access is denied, unless it is explicitly allowed. The rule ignores all negative authorisation rules, viz. negative authorisation rules in a policy with this decision rule are not having any effect on the policy decision and should be omitted.

Example 3.6 (Decision Rule)

$$\boxed{\textit{decide}\ (S,O,A)\ \textit{when}\ 0:\ \textit{not deny}\ (S,O,A)}$$

The access in this rule is granted if no negative authorisation can be derived from the policy. This rule is used in open policies, where any access is allowed, unless it is explicitly denied. Blacklists are typical examples of open policies. The rule ignores all positive authorisations, viz. positive authorisation rules in a policy with this decision rule are not having any effect.

Example 3.7 (Decision Rule)

$$\boxed{decide\ (S, O, A)\ when\ 0:\ allow\ (S, O, A)\ and\ not\ deny\ (S, O, A)}$$

The example states a rule that is used in hybrid policies. Hybrid policies are taking into account both, positive and negative authorisation. Hybrid policies are suitable to express more complex access control requirements. The difficulty with hybrid policies is that conflicts can occur, in the sense that a subject is at the same time allowed and denied to access a resource. In these cases, the decision rule does also resolve the conflict. The rule in the example states that access is only granted if explicitly allowed and not explicitly denied in the policy. It therefore gives precedence to denials. It is allowed to have more than one decision rule in the policy. However, it is important to note that it is sufficient to have one decision rule firing for the access to be granted. Like other rules, the decision rules can also contain state and history-dependent premises.

3.2.1.3 Obligation Rules

Obligation rules state under which condition a subject must perform a specific action. Consequently obligations can only be enforced if they are defined as behavioural policy rules (e.g. an e-learning system to notify students when their grades appear).

Example 3.8 (Obligation Rule)

$$\boxed{oblige\ (S, S, notify)\ when\ 0:\ done\ (S, O, submit)}$$

This assumes that the e-learning system defines the actions submit grades and notify students for their grades.

3.2.1.4 Simple Policies

Simple policies represent a collection of policy rules that all apply simultaneously. In the SANTA language this is currently represented by grouping a set of rules using parenthesis. Siewe showed in [87] that set theoretic operators can be used to compose simple policies. He also defined some operators to filter rules based on their type (e.g. positive authorisation, negative authorisation rules). These operators for the set theoretic composition of simple policies are currently not included in the SANTA syntax, but can be included without much difficulty.

Chapter 4

Decentralised Policy Based Management (PBM)

Objectives

- Describe the Decentralised Policy Based Management (PBM) framework.
- Identify the framework with their need.

4.1 Introduction

This Chapter provides an overview of proposed Decentralised Policy Based Management (PBM) framework. It identifies the methodologies for the Decentralised PBM with their need. Improvement of access control decision making is achieved by Decentralised Policy Based Management (PBM) framework and its required steps those described in this Chapter. The framework details the workflow to achieve the enforcement of static and dynamic policies in a distributed setting.

The remainder of this Chapter is structured as follows. Section 4.2 provides general overview of the framework with its description steps in subsections. Section 4.3 summarises the Chapter.

4.2 Decentralised Policy Based Management (PBM) Framework

The main aim of this research is the development of a high-level security framework for the adaptable management of security in large-scale safety-critical information systems. This section provides the framework definition and identifies the methodology to achieve the Decentralised PBM.

Figure 4.1 depicts the steps in the use of our proposed framework where administrators write policies that express the information system's security requirements at a high level of abstraction. The red rectangles are the outputs of each step where the outputs from each step being inputs for the next step. After our Decentralised PBM system is implemented, changing policies require to return to policy analysis step again. Moreover, reconfiguring the enforcement infrastructure back to the policy decomposition step.

The overall system policy is composed of several smaller policies that capture individual security requirements. The compositional approach allows for the structuring of the system policy into logical units, reducing the complexity. The decomposition of this policy is design by the network infrastructure and non-fundamental considerations such as efficiency and resilience.

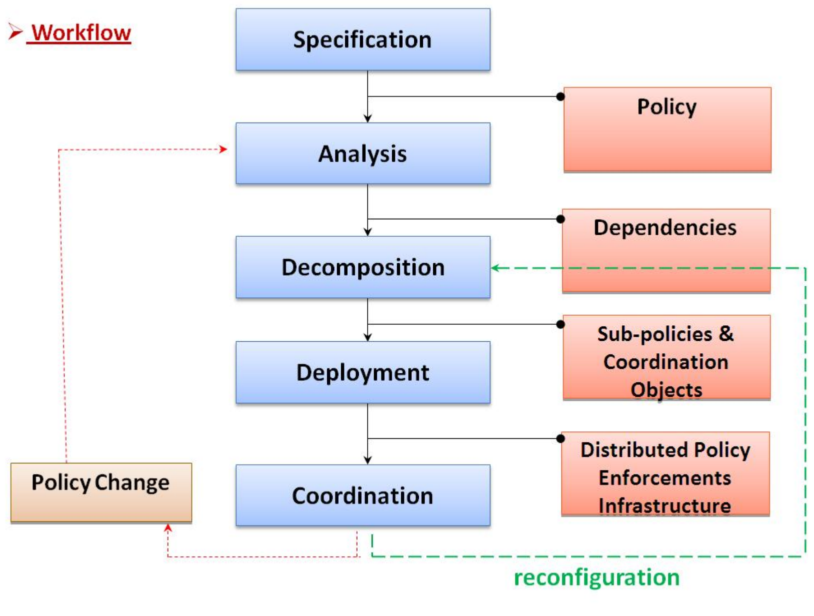


FIGURE 4.1: Decentralised Policy Based Management (PBM) Framework.

4.2.1 Policy Specification

The technologies we are using in our research is the SANTA [51] for the policy specification. The SANTA policy language provides support for both state and history dependencies. The main reason for our choice of policy language is that the reasoning about histories using temporal logic descriptions is more natural to support with tools than analysis models that incorporate history-dependencies by expressions on log-files or lists. [See Section 3.3 for the SANTA details].

4.2.2 Policy Analysis

The policy analysis and its dependencies on the structure of the access control requirements is a key for policy decomposition. This step addresses the identification of dependencies between the policy specification and the access control requirements.

In order to enforce a policy, all relevant observations must be available to all Policy Decision Points (PDPs) and PDPs must control the interaction with the protected resources and also between each other when they share a decision. For example, in Chinese-Wall policy [25] between two resources in such system requires PDPs to control interactions with both resources especially when these resources are located in different systems or locations (domains).

We are mainly interested in history-based policies [3] and build upon the work on the analysis of this class of policies those described in [58], [56] and [52]. The techniques presented in this work are as however independent of the concrete policy language chosen and can easily be adapted to other policy languages that allow for the expression of history-based policies (e.g. [51]). In Chapter 6, the policy analysis dependencies are described. [See Chapter 6 for more details]

4.2.3 Policy Decomposition

Policy decomposition addresses the decomposition of the policy into sub-policies with the consideration of the dependencies between their rules. To ensure correct enforcement, the decomposition must not violate any dependencies.

Often security requirements distinguish between different situations of the system in question. According to [55], policy composition befits the complicated requirements where smaller policies are combined to express more complex security requirements. On the other hand, the policy decomposition is breaking these composed policies into smaller sub-policies for the needs of distributing enforcements. In our contribution, the importance of both composition and the decomposition to fulfil these requirements is essential. We strongly believe that perform composition to produce a central policy that covers all requirements for the system and then decomposition model is efficient to guarantee the efficiency of the enforcements in distributed manner.

In our policy decomposition model, we use structural refinement to transform the central policy into several policies according to policy object (target) domains. The domain is a collection of objects governed by the same geographical boundaries, object type, responsibility or the level of security or authority.

Finally, these sub-polices can be deployed in a network of DENARs to meet the need of Decentralised PBM. In addition, collaborative DENARs network is modelled to make the shared decision when it's required.

Unfortunately, freedom in the choice of a policy language for the specification of policy can result in some advantages when specifying and modelling the policy for access control systems from early stage of designing a system. On the other hand, it results a disadvantages when the policy is specified in rules and then these rules composed in one simple (central) policy or decompose into sub-policies. (Specially when some policies are involved like the Bell-LaPadula [66] or the Chinese Wall Policy [25]). In turn, this would result in a much larger set of sub-policies, more complex composed and decomposed algorithms, as well as a higher probability of "orphan" enforcing policies that cannot be matched by requited existing policy. However, to solve these drawbacks, in [53], the authors show how dependencies between policy rules affect their enforcement. Moreover in [53], they propose a technique by UCON model to improve the enforcement mechanism for these policies. However, our decomposition model and algorithms that are detailed in Chapter 7 is a refinement of these dependencies where they are being enforced in centralised PDP and also in distributed PDPs.

The idea of policy decomposition model is based on object's domains when it offers less decision response time with the similar accuracy of decision. Additionally, efficiency in enforcement is satisfied by applying multiple PDPs those share and synchronise the decision between them. In addition, we choose objects in domains mechanism instead of subjects where subjects always are free movement in the network systems but objects rarely are in some systems e.g. AdHoc systems and that is being a hot topic for researchers and not covered in this work. Moreover, applying both object's and subject's mechanisms to make a preference to the system administrator is being more efficient and a good future work. Additionally, the accurate decision is satisfied when split all objects in the system into domains as mentioned above as well as that can involved the Bell-LaPadula [66] or the Chinese Wall [25] policies by our mechanisms. [See Chapter 7 for more details in how decomposition model involved]

An example of a structural decomposition is shown in Figure 4.2 where the policy is decomposed into three independent sub-policies. Then, sub-policies are deployed into corresponding DENARs (in particulate Policy Repository PR) in Deployment step. The deployment is detailed in Chapter 7.

4.2.4 Policy and PDPs Deployment

A network of DENARs in peer to peer fashion is include the PDPs as an overlay to the network. The DENARs network link the PEP's to the PDPs dynamically

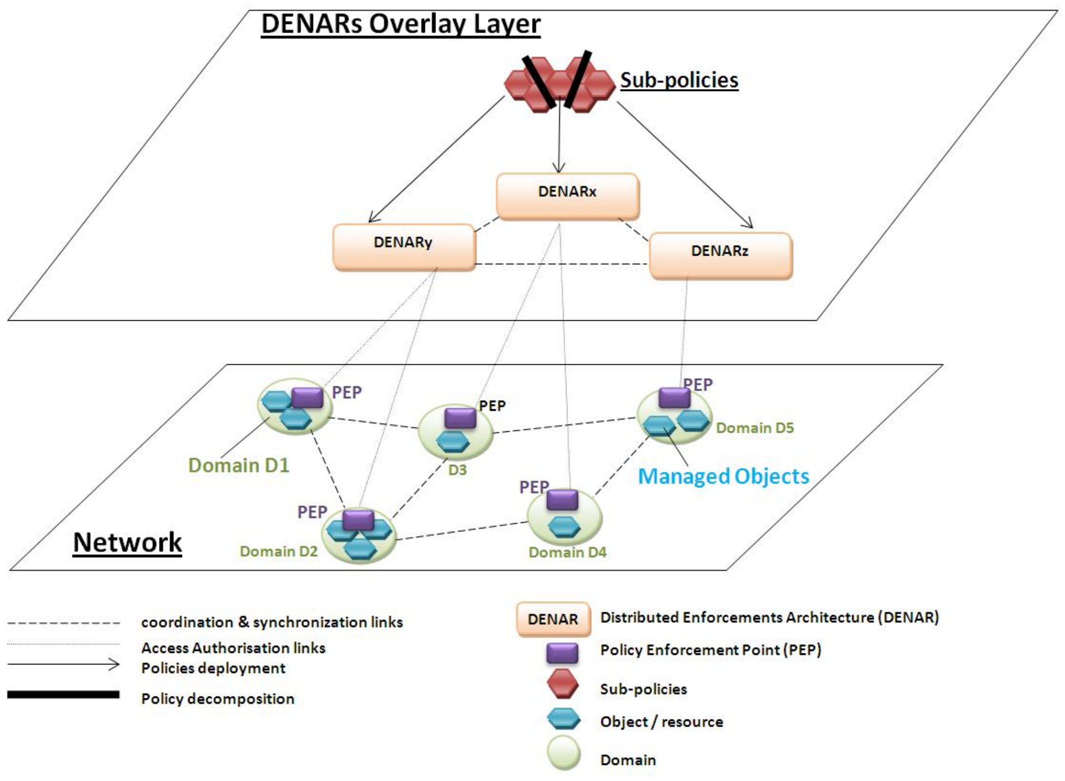


FIGURE 4.2: The Overlay Network of DENARs.

such that policies enforced at their DENAR' PDP as if they were enforced locally. Additionally, the policies are distributed among various PRs in DENARs where each sub-policies deployed in its domain policy repository PR. Having a system of DENARs removes the bottleneck, improves performance and also eliminates the single point of failure. Not only would decision making be much faster in such a setup, the system would be more efficient in terms of resource utilisation.

Though more efficient performance wise by implementing the network of DENARs has two serious drawbacks. Firstly, it is possible that enforcing these policies may suffer unexpected failures; perhaps because a PDP cannot retrieve a policy from its local policy repository (PR) or no backup DENAR is available. Secondly, coordination between the DENARs network is may lost in case of the policy that enforced is history-based policy (e.g. Chinese-Wall policy [25]) and requires to share decision between more than one PDP. Clearly, it is possible that enforcing this sort of policy may suffer unexpected failures, perhaps because a PDP cannot retrieve such conditional attribute from remote PIP or there is no collaboration between DENARs.

Ideally, policy enforcement must be resilient to such failures if no decision can be computed or no conditional attribute can be evaluated. This can be achieved in two steps. The first step would be to physically transmit the decomposed sub-policy to a particular policy repository PR and connect all DENARs in the network immediately with it. Once the decomposed policies are deployed to all the connected DENARs, all of them can take local decisions when a policy request is received in case of the independent policy is enforced, thus overcoming the problem of immediate network overload due to multiple possible concurrent decision requests. The second step would be to incorporate a mechanism by which all the enforcements query each one of the other connected DENARs in the network for a decision and allow the operation only if all of them either respond with a '*allow*' or a '*deny*' response. However, in a network where there could potentially be thousands of PDPs connected to each other, this in itself would be a serious bottleneck to performance because of the tremendous amount of bandwidth it would consume. Therefore, we address this step only for dependent policies with rich the efficiency as well by propose Pull and Push Models. [See Section 8.3.1 for more details].

In summary, we combine both the above mentioned models to optimise the distributed functionality of the network of DENARs based on available the object-domain for the first issue and the rule dependency for the second one where those have been proposed and discussed in above units.

4.2.5 PDPs Enforcement and Coordination

After the distribution of policies among various PDPs, the coordination and synchronisation between the distributed PDPs is declared and implemented. PEPs are automatically linked with DENARs. In addition, the DENARs are connected with each other to fulfil the need of coordination. In this manner, the PDP's can now arrive at decisions locally and thus save on time and network resources. Moreover, the connections between components of the DENAR's component are stated in this step. [See Chapter 5 for the DENAR architecture and Chapter 8 for the coordination mechanism details]

4.3 Summary

This Chapter introduced the Decentralised Policy Based Management (PBM) framework. The framework workflow from policy specification to the DENARs coordination is described. The workflow steps illustrated the methodology to achieve the efficiency in enforcing static and dynamic policies in distributed systems. However, ignoring any step in the framework affects the overall access control decision making or affects the original contribution of this research. Involving the framework output the performance, security, manageability and resilience factor that considered in the research.

The following chapters give description of each steps of the presented framework. In Chapter 5, the DENAR architecture is designed and detailed. The analysis of policy and enforcement in DENAR are provided in Chapter 6. The policy decomposition step and policy deployment are modelled in Chapter 7. Finally, the coordination step is detailed with the enforcement model for the collaborative DENARs are detailed in Chapter 8.

Chapter 5

Distributed Enforcements Architecture (DENAR)

Objectives

- Describe the DENAR architecture.
- Show how the DENAR components interact.
- Provide DENAR administration technique and configuration.

5.1 Introduction

The Distributed Enforcements Architecture (DENAR) is described focusing on component functionalities and interactions. In addition, the DENAR administration technique and configuration is provided.

Performance and manageability are contributing factors to improve access control decision making. Thus, DENAR is designed in this Chapter to fulfill these factors.

The remainder of this Chapter is structured as follows. Section 5.2.1 provides an overview of the architecture. Section 5.2.2 provides a description of the DENAR's components functionality where the component interactions are detailed in section 5.2.3. The DENAR administration is described in task of configuration, deployment, re-propagation and recovery techniques in section 5.3. Finally, section 5.4 summarises the Chapter.

5.2 Distributed Enforcements Architecture (DENAR)

According to the IETF and DMTF model [100], the policy enforcement architecture consists of three components which are Policy Repository (PR), Policy Enforcement Point (PEP) and Policy Decision Point (PDP). Moreover, it may include Policy Administration Point (PAP) that formulation, analysis and verification of policies on the part of humans. In addition, there may involve Policy Information Point (PIP) that can provide information against which policy conditions (such as subject, object, environment or past decision access) are evaluated in a PDP. The PR provides mechanism for storing policies and retrieving them as required by the decision points. The responsibility of PEPs is enforcing the outcome of those policy decisions. The PDP is a module in the system that is responsible for making policy decisions. They evaluate authorisation requests coming from PEP against the policies stored in the repository that would be applicable under a given circumstance and determine what needs to be done to comply with those policies.

The below Figure 5.1 shows the Policy Enforcement Architecture in DMTF model.

In such Centralised PBM system, policy makers pre-define a central policy through a PAP, then the PAP deposits them in a PR. Therefore, a PDP can monitor the

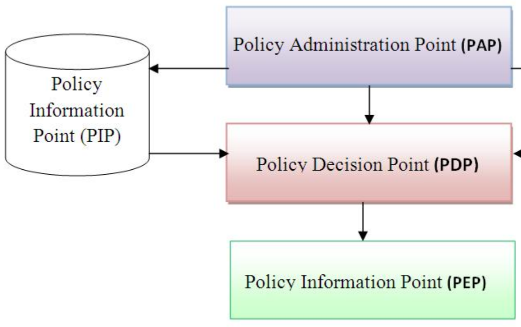

Policy Administration Point (PAP)
Policy Information Point (PIP)
Policy Repository (PR)
Policy Decision Point (PDP)
Policy Information Point (PEP)

FIGURE 5.1: Policy Enforcement Architecture in DMTF model.

access control. Once the specific access request occurs, PEPs trigger the PDP to retrieve the PR for applicable policies. According to these policies, when the specific conditions are met, the corresponding actions should be enforced by the related PEP. For the communication protocol, Simple Network Management Protocol (SNMP) [26] or Common Open Policy Service (COPS) [38] may be involved for the communication between the PDP and the PEP. In addition, Lightweight Directory Access Protocol (LDAP) [43] is involved for the communication in the PDP to retrieve a policy from PR.

Having a centralised PDP has a number of benefits. The most important advantage to having a centralised PDP is that the activities of the various PEP's can be coordinated amongst each other. Resources that are being accessed by multiple PEP's can be synchronised and the overall usage limits for a specified resource across the entire distributed system can be set. Let us illustrate with an example.

Assume that each user in a university network has access to the printers on the network but can print no more than 100 pages per day.

In case of a centralised PDP, this is easy to accomplish given that the PDP is stateful and stores access request and decision states for at least a day for later reference. In case of such a setup, the PDP will check how many pages the user has already printed before allowing access to print further pages and if the limit is exceeded, will deny the request. However, in a network which has hundreds of network nodes and devices, having a centralised PDP can be a huge bottleneck to performance if the system is busy. Access requests may flood the PDP and there may be unacceptable delays in getting a response. Another disadvantage to having a centralised PDP is that it presents a single point of failure. In case of the failure of the PDP, all the devices on the network may cease to function. In practice, this is rarely the case as there is at least one alternate backup provided in case of the failure of the main PDP but the fact still remains that decision making is centralised and thus tends to decrease the performance and efficiency of the system.

The Policy Framework proposed by the IETF, RFC 2753 [100] mentions a local Policy Decision point in addition to the remote PDP in order to facilitate decision making. However, according to their specifications, even if a decision is arrived at by the local PDP, the details of such a decision as well as the original request needs to be forwarded to the remote PDP for final ratification. Their measure, in essence is designed simply to ensure minimal disruption in service in case of the failure of the remote PDP by allowing the local PDP to take decisions for a limited time frame when the main PDP is not functioning. It is not meant to be a proper distributed system per se. As a result, most of the earlier implementations of policy based control as well as a majority of current ones, rely on a centralised PDP for their decision making.

In such a scenario, having a system of distributed PDPs would remove the bottleneck, improve performance and also eliminate the problem of a single point of failure. The distributed PDP's may either be co-located with the PEP or may service a specified area such as an administrative domain. Not only would decision making be much faster in such a setup, the system would be more efficient in terms of resource utilisation. An accepted method which has been proposed in order to achieve this distribution is to break up high level policies into smaller component parts specific to each resource being controlled [90] . This process is known as the decomposition of policies. The decomposed components of the policy can then be distributed to the respective PDP's in charge of the specified resource for which the policy has been decomposed. In this manner, the PDP's can now arrive at decisions locally and thus save on time and network resources. Under such an arrangement, the role of the centralised PDP would simply be to decompose the policies into component parts and distribute them to the various distributed PDPs.

However the above arrangement, though more efficient performance wise, has one serious drawback. All coordination between the distributed PDPs is now lost. If we consider the example mentioned above, this would mean that a user could technically print out 99 pages from a printer in the network which is under the control of a particular PDP and then simply move to another part of the network and print out another 99 pages from another printer which is controlled by a second PDP. As far as both the PDP's individually are concerned, the user has not exceeded the limit specified for him in the policy but as far as the entire distributed system is concerned, he has far exceeded his quota.

Then, it also could not count to 99 where this poses a serious problem. By stateless, what is meant is that they take access control decisions for every request in isolation to all previous requests which have been made or permitted. Given this scenario, coordination is required even in case of a centralised PDP in order to enforce usage limits on various resources which are being controlled. Indeed this is the main reason why most distributed systems, like the ATM systems in banks, rely on centralised PDP's with custom built software to enforce usage limits in order to enforce their policies. Chadwick et. al [27] suggested a way to overcome this problem. They proposed the introduction of a coordination object to enable otherwise stateless PDP's to coordinate their decision making.

The coordination is required between all PDPs to share information that is required in access decision making. Therefore, there is an evident need for adding new components into IETF and DMTF model [100] for coordination. We design our Distributed Policy Enforcements Architecture (DENAR) based on this needs.

5.2.1 The Architecture

The Peer-to-Peer (P2P) networking approach, at least the one characterised by Overlay Network [69], appears to be a much more effective architecture on which to build the distributed PDPs for the Decentralised PBM system.

Since each peer can communicate with any other peer in the network to provide a service, P2P networks support a more distributed architecture. By involving an Overlay Network to create a group of DENARs for the Decentralised PBM system, there are less central components in the DENARs network operations. Additionally, the PDP peers in the DENARs network can communicate with each other (in particular, through PIPcoordinator and PDPcoordinator components) and propagate the decision information throughout the DENARs overlay network. Thus, this approach provides a fully distributed architecture.

Obviously, a simple policy (central policy) is decomposed into sub-policies to be deployed in multiple PRs, so that every PDP peer would only need to communicate directly with its DENARs' PR for a policy and PIP for conditional attributes.

A benefit to decompose the policy into sub-policies and deploy them into PRs is that the network traffic load can be shared through the local domain for local DENAR or through remote DENAR by PDPs peers, instead of through select routes to centralised PR. Consequently, it avoids single points of failure and efficiently utilizes the available network bandwidth, therefore reducing congestion [97].

P2P protocols provide a discovery mechanism. The discovery mechanism allows PDP peer to dynamically discover (in the same DENAR) the PR for the policy that would be enforced and the PIP for the required attributes for the local enforcement. In the discovery mechanism, each PDP peer can discover other PDP peers for remote DENAR within the network. We believe discovery to be a useful feature for the Decentralised PBM system, since we expect there to be situations where network domains topology may change. Thus, it would be inefficient to structure the network of DENARs with predefined PDPs, PRs and PIPs.

Therefore, instead of directly implementing the Internet Engineering Task Force (IETF) model [100] [95] to provide a more suitable policy-based system for the network, we decided to modify the IETF and DMTF model [100] to be built using P2P protocols (Overlay Network) and adding new components which are PDPcoordinator and PIPcoordinator with the other components in the model to play the role of coordination and concurrency between DENARs.

The below Figure 5.2 shows the DENARs network where the DENAR's component and interactions are detailed in the following subsections.

5.2.2 Component Functionalities

Since each peer can communicate with any other peer in the network to provide a service, P2P networks support a more distributed architecture. By involving Overlay Network [69] to create distributed PDPs for the Decentralised PBM system, distributed PRs and PIPs are involved. Thus, there is no centralisation in the enforcement operations but each Policy Decision Point (PDP) peer communicates directly with its PR, PIP, PDPcoordinator and PIPcoordinator in local DENAR.

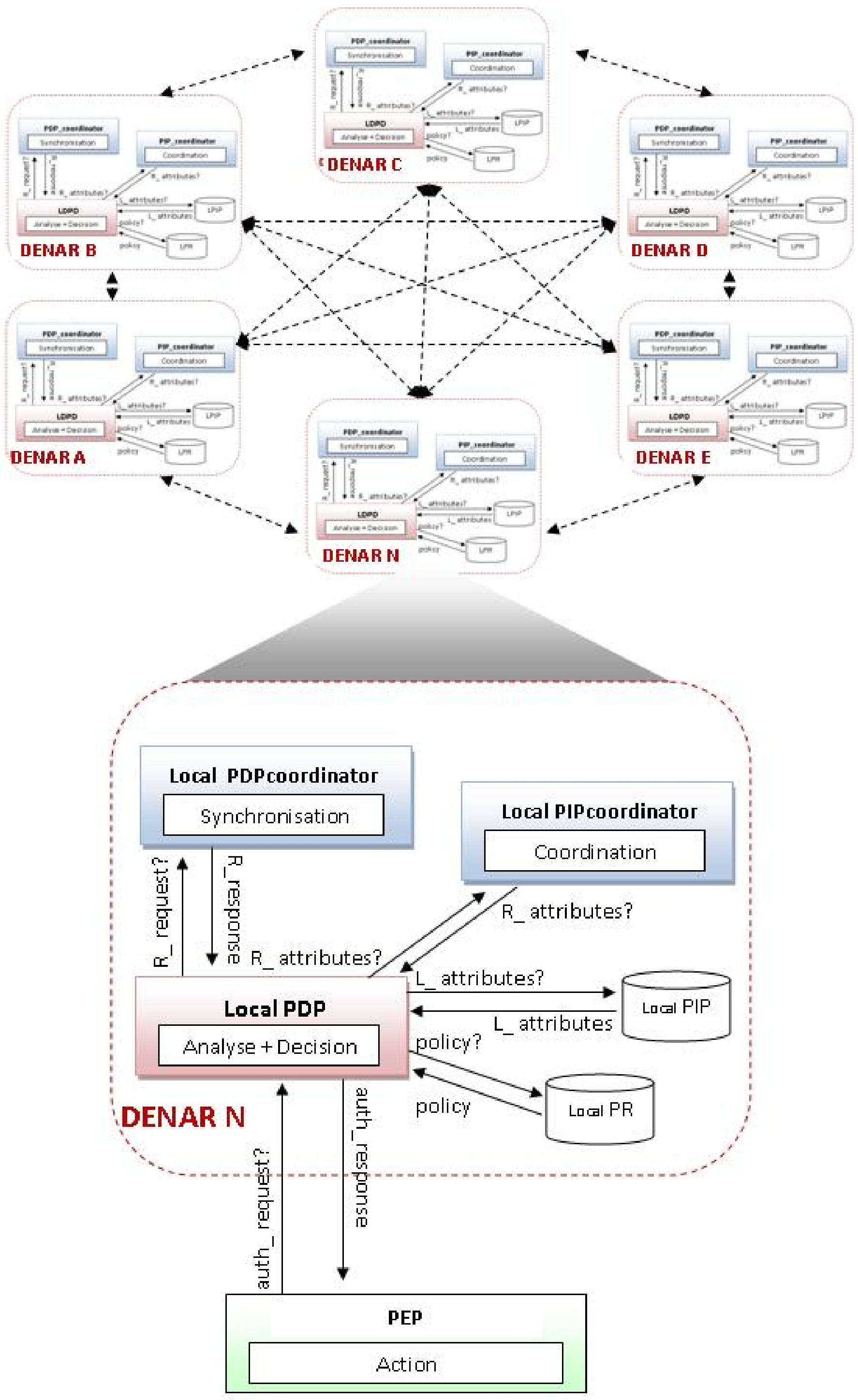


FIGURE 5.2: The DENARs network.

However, for the remote (collaborative) decision, the PDP peers can communicate with each other and propagate the decision information throughout the overlay network of DENAR (in particular, through PDPcoordinator and PIPcoordinator components) [See Chapter 5, Section 5.2.2.5 and 5.2.2.6 for more details]. This approach provides a fully distributed architecture. Obviously, a simple policy (central policy) is decomposed into sub-policies to be deployed in multiple PRs, so that every PDP peer would only need to communicate directly with its DENARs' PR for a policy and PIP for conditional attributes. Figure 5.3 shows DENAR architecture components where they are detailed in the following subsections.

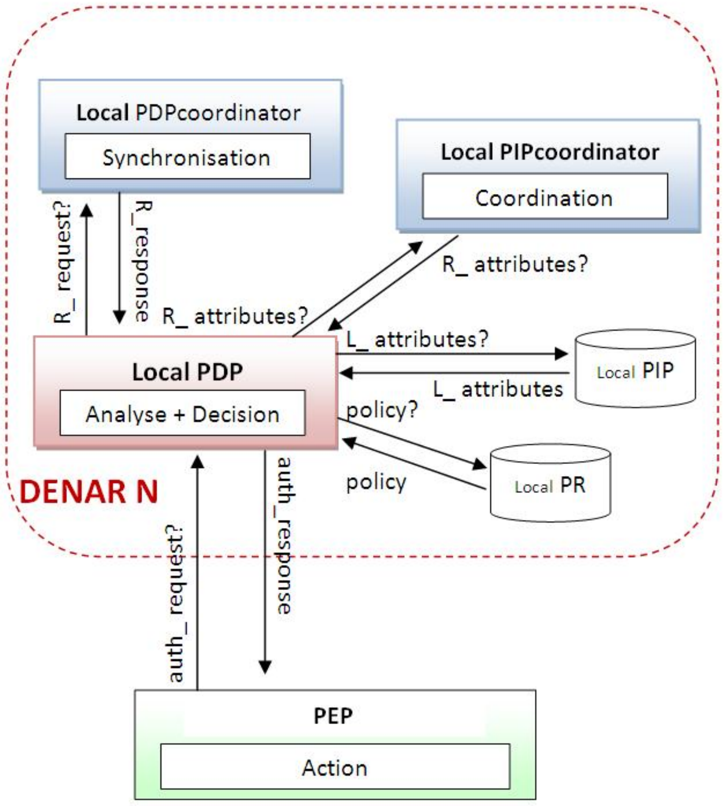


FIGURE 5.3: The Distributed Policy Enforcements Architecture (DENAR).

5.2.2.1 Policy Enforcement Point (PEP) Functionality

A Policy Enforcement Point (PEP) is a core component in our architecture. It is responsible for sending requests to a PDP, and enforcing decisions made by the PDP, thus acting as an authorisation link between the PDP and system elements (subjects and objects) for the authorisation purpose.

To perform its task the PEP collects access requests from various system elements, and forwards them to the PDP for decision making. Moreover, the PEP enforces the actions determined by the PDP. However, in some cases, when enforcement task is complicated the PEP must obligate the PDP about the action execution, and this is done by obligation policies where this value is required in a future decision, e.g. a bank could define an obligation to notify its customers whenever anyone withdrew money from their bank account, thus, the process performing access first and then notifying the action.

In DENAR, a PEP discovers a PDP that it should communicate with as well as other PEPs in the network. After a PDP has been found, the PEP sends an authorisation request messages to the PDP. If the PEP fails to communicate with the PDP, the PEP returns discovery and waits until it has found the PDP otherwise communicates with backup DENAR. In particular, the PDP in backup DENAR that configured as backup enforcement by a system administrator, i.e when any of these PDPs is disconnected, this often causes accessing problems which lead to security violations. [See Chapter 5, Section 5.3.3 for more details for resilience]

The PEP has an initialization phase and a run phase. The main purpose of the initialization phase is, firstly, to find other PEPs in the network and connect with them. Secondly, it has to find a proper PDP for this domain and connect with it. Initialization is completed after the PEP has created its authorisation message pipe with the proper PDP. After initialization, the PEP uses the authorisation message pipe to periodically send authorisation request messages to the PDP and check the response to those messages. If the PEP fails to receive a response from the PDP after time has elapsed for sending another authorisation request, the connection to the PDP is gone. When the connection to the PDP is lost the PEP will return to the initialization phase to find another PDP link otherwise communicates with backup DENAR (in particular the PDP in backup DENAR).

Once the PEP is up, it uses the pipe advertisement (i.e., the same one that the PDP used) to set-up a connection with the PDP. Once connectivity between the PDP and the PEPs has been established, the PEP can send an authorisation request packet to PDP and receive the decision. The PEP creates a directory with the configuration file and the cached service advertisements from other peers. Once the PEP receives the decision, it may execute the obligation that was sent with the decision.

5.2.2.2 Policy Repository (PR) Functionality

In each DENAR, there is one PR that stores the sub-policies for a domain or domains (based on the network topology configuration). System administrators configure these PRs in their DENARs. The PDP in the DENAR is responsible to retrieve proper policy from DENARs' PR.

5.2.2.3 Policy Information Point (PIP) Functionality

An authorisation request is evaluated against a policy and if the policy has some access restrictions (conditions), a PIP in the DENAR provides local conditional attributes (those are stored in the same domain) those are evaluated by a PDP. On the other hand, the PDP contacts a local PIPcoordinator for the remote conditional attributes (those are stored in the different PIPs in other DENARs) [See Chapter 5, Section 5.2.1.5].

5.2.2.4 Policy Decision Point (PDP) Functionality

In general, a PDP retrieves a policy from a PR. The request is evaluated against the policy and if the policy has some access restrictions (conditions), the PDP contacts the PIP for attributes those would evaluated in the conditions.

In DENAR, the local PDP in the DENAR is responsible for receiving an authorisation request that is sent by a PEP and retrieves a policy from a local PR. The request is evaluated against the policy and if the policy has access restrictions (conditions), the PDP contacts the local PIP for the local conditional attributes (those are stored in the local PIP) those would evaluated in the conditions.

Alternatively, it contacts a local DENARs' PIPcoordinator for the remote conditional attributes (those are stored in the different PIPs in other DENARs in the network) those will be evaluated in the conditions where PIPcoordinator can communicate with other PIPcoordinators in the DENARs network to find the value for those attributes. In addition, the local PDP contacts a local PDPcoordinator DENARs' if the decision is shared by other PDP in the DENARs network. In this case, the local PDPcoordinator communicates with other PDPcoordinators in the network to ask their PDPs to share decision.

For configuration, the PDP creates the default PDPs group. After that, it uses a server pipe to accept connections from the other peers in that group. There is only one active PDP within a DENAR. Once the PDP is up and running, it applies discovery service. Subsequently, a PDP searches for the PR, PIP, PIPCoordinator and PDPCoordinator within the same DENAR as well as PEPs.

5.2.2.5 PIPcoordinator Functionality

The PIPcoordinator is a new component in Distributed Enforcements Architecture (DENAR). The main function of the PIPcoordinator is to act as bridge for sharing conditional attributes between two DENARs. A PDP in the DENAR can communicate with other PIPs in the DENAR network only via PIPcoordinators. In case of enforcing dynamic policies, the PDP communicates with other PIPs to evaluate attributes those question in the policy condition and cannot be found in the local PIP. The PIPcoordinator is being the coordinator between the PDP and other remote PIPs to perform sending those attributes (push) or request those attributes (pull). [See Section for details 8.2.1]

The PIPcoordinator discovers other remote PIPcoordinators those should communicate with in different DENARs. In running phase, after PIPcoordinators have been found, the PIPcoordinator sends (push) attribute request to other PIPcoordinators in other DENARs if and only if the attribute is located in remote PIP or PIPcoordinator request (pull) attributes from other PIPcoordinators (produced from / required for) the current decision (e.g., in case of enforcing history-based policy). If the PIPcoordinator fails to communicate with another PIPcoordinator in the DENARs network, the PIPcoordinator returns discovery and waits until it has found it otherwise communicates with backup DENAR (in particular the PIPcoordinator in backup DENAR).

The PIPcoordinator has an initialization phase and a run phase. The purpose of the initialization phase is to find other PIPcoordinators in the DENARs network and connected with them. Initialization is completed after the PIPcoordinator has created its shared attribute exchange pipe with other PIPcoordinators. After initialization, the PIPcoordinator uses the shared attribute exchange pipe to periodically send (push) or request (pull) attributes to proper PIPcoordinator in the DENARs network and check the response to those messages in case of the Pull or

Push Models. If the PIPcoordinator fails to receive a response from other PIP-coordinators after time has elapsed for sending another shared attribute request, the connection to the PIPcoordinator is gone. When the connection to the other PIPcoordinators are lost, it will communicate with backup DENAR (in particular the PIPcoordinator in backup DENAR).

5.2.2.6 PDPcoordinator Functionality

The PDPcoordinator is a new component in the DENAR. The main function of the PDPcoordinator is to act as bridge for sharing a decision between two PDPs. A PDP in the DENAR communicates with other PDPs in the DENAR network only via PDPcoordinators. In case of enforcing hypothetical policy where a rule depends on another authorisation rule, multiple PDPs communicate to make the final decision where one of these PDPs is being the coordinator PDP that received the authorisation request from a PEP.

The PDPcoordinator discovers other remote PDPcoordinators those should communicate with in the DENARs network. After PDPcoordinators have been found, the PDPcoordinator sends an authorisation request messages to other PDPcoordinators in the DENARs network if and only if authorisation decision will be shared with multiple PDPs (e.g., in case of enforcing hypothetical rule) . If the PDPcoordinator fails to communicate with another PDPcoordinator, the PDPcoordinator returns discovery and waits until it has found it otherwise communicates with backup DENAR (in particular the PDPcoordinator in backup DENAR).

The PDPcoordinator has an initialization phase and a run phase. The purpose of the initialization phase is to find other PDPcoordinators in the DENARs network and connected with them. Initialization is completed after the PDPcoordinator has created its shared authorisation decision pipe with other PDPcoordinators. After initialization, the PDPcoordinator uses the shared authorisation decision pipe to periodically send a request for shared authorisation decision to proper PDP-coordinator in the DENARs network and check the response to those messages. If the PDPcoordinator fails to receive a response from other PDPcoordinators after time has elapsed for sending another shared authorisation decision request, the connection to the PDPcoordinator is gone. When the connection to the other PDPcoordinators are lost, it will communicate with backup DENAR (in particular the PDPcoordinator in backup DENAR).

5.2.3 Component Interactions

Interactions in DENAR are between the component themselves and also between DENARs in the DENARs overlay network. The component interactions make the access control decision and interactions between DENARs to support the co-ordination and synchronisation requirements in access control decision making. Figure 5.4 illustrates messages exchange pipes in DENARs network to achieve the above requirements where they are detailed in the below subsections.

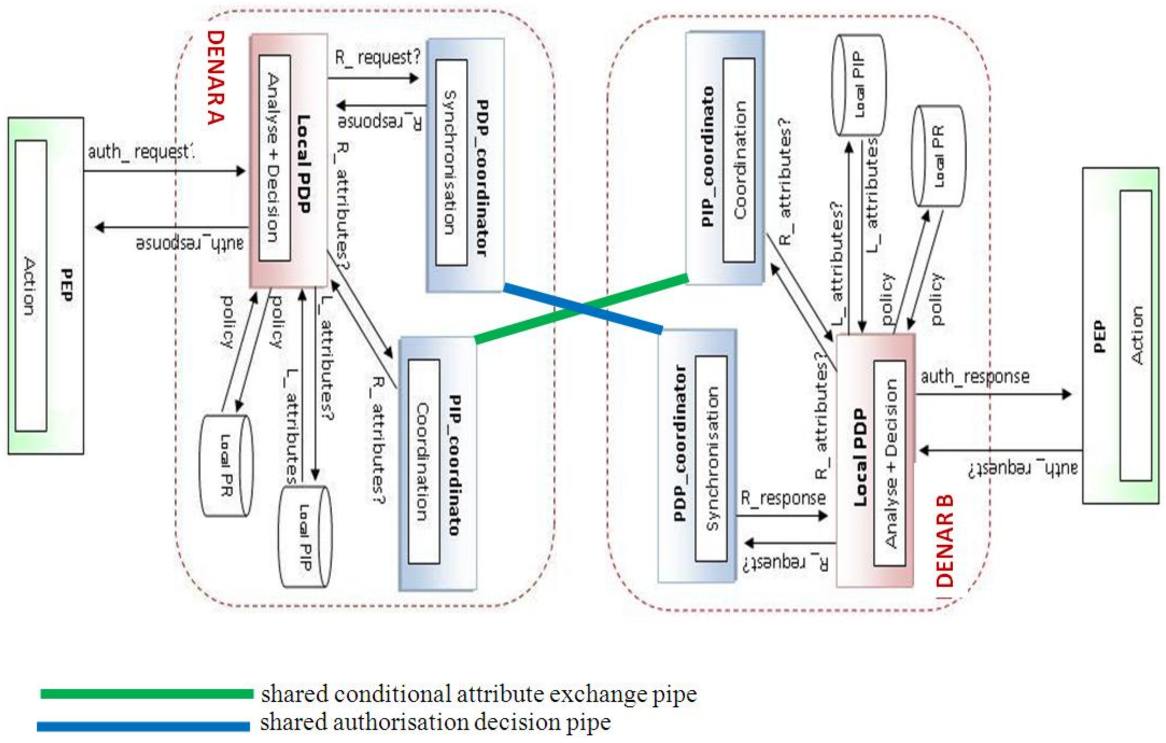


Figure 5.4: Messages Exchange in DENARs network.

5.2.3.1 PEP and DENAR Interaction

There are three types of messages for the PEP and the PDP:-

1. **Discovery Configuration:** to establish a communication between a PEP and a PDP, the PEP will create its authorisation message pipe with all PDPs in the DENARs network to find all active PDPs in the network, thus, a system administrator has the choice to configure a proper PDP for the PEP and the backup PDP as well. The administrator can also setup the secure

message exchange protocol between the PEP and PDP. For the communication protocol, SNMP (Simple Network Management Protocol) [26] or COPS (Common Open Policy Service) [38] may be involved for the communication between a PDP and a PEP.

2. **Peer Authorisation Request:** after initialization, the PEP uses the authorisation message pipe to periodically send authorisation request messages to the DENAR (in particular the PDP) and check the response to those messages. Transition between the PDP and its PEPs is also logged to a file for obligation needs.

3. **The Decision Information:** if there is an obligation policy that must be enforced, the PEP can respond to the PDP after the action is executed so the PDP can log that into its local DENARs' PIP or remote PIPs in different DENARs(in Push Model).

The diagrams below in Figure 5.5 and Figure 5.6 show the PEP and DENAR for the activity and sequence interaction.

5.2.3.2 DENAR Components Interaction (PDP, PR, PIP, PDPcoordinator and PDPcoordinator)

A PDP in the DENAR discovers one PR and PIP that are located in the same DENAR otherwise discover the backup PR or PIP those configured in DENAR backup. There are messages exchanges between these components as:-

1. **Discovery Configuration:** to establish a communication between a PDP and a PR and PIP, the PDP finds the active PR and PIP in the DENAR. Then, a system administrator can configure the backup PR and PIP in DENAR backup. The administrator can also setup the secure message exchange protocol between the PDP and local PR and PIP as well as the message exchange protocol for the backup PR and PIP. For the communication protocol, LDAP (Lightweight Directory Access Protocol) [43] is involved in the PDP to retrieve a PR or PIP.

2. **Policy Request:** after initialization, the PDP uses the configured protocol to retrieve a proper policy from the local PR or backup PR that enforced against the authorisation request to make the decision.

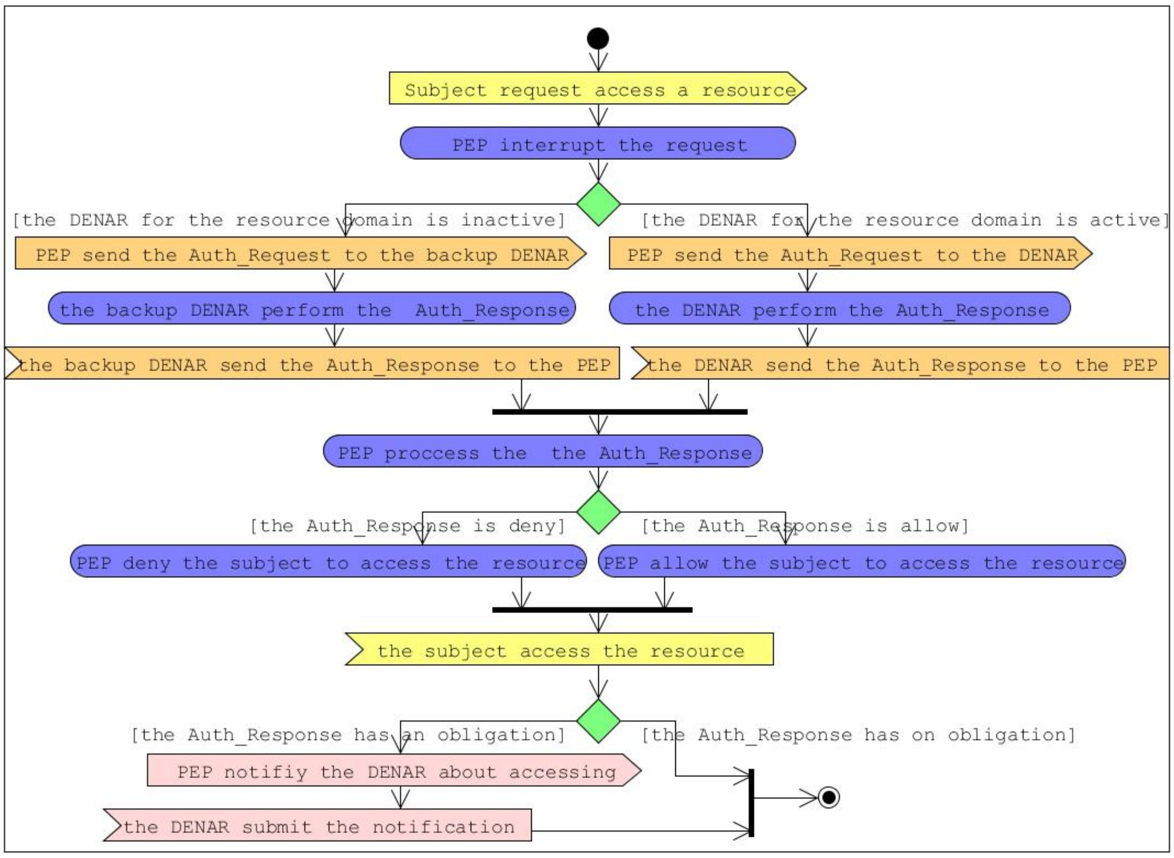

Subject request access a resource
PEP interrupt the request
[the DENAR for the resource domain is inactive]
[the DENAR for the resource domain is active]
PEP send the Auth_Request to the backup DENAR
PEP send the Auth_Request to the DENAR
the backup DENAR perform the Auth_Response
the DENAR perform the Auth_Response
the backup DENAR send the Auth_Response to the PEP
the DENAR send the Auth_Response to the PEP
PEP proccess the the Auth_Response
[the Auth_Response is deny]
[the Auth_Response is allow]
PEP deny the subject to access the resource
PEP allow the subject to access the resource
the subject access the resource
[the Auth_Response has an obligation]
[the Auth_Response has on obligation]
PEP notifiy the DENAR about accessing
the DENAR submit the notification

FIGURE 5.5: PEP and DENAR Interaction Activity Diagram

3. **Shared Decision Request:** the PDP contacts its local PDPcoordinator in the DENAR if the decision is shared with other PDPs in the DENARs network. In this case, the local PDPcoordinator communicates other PDP-coordinators in the DENARs network to ask their PDPs to make decision. [See Chapter 8, Section 8.2 for more details].

4. **Local Conditional Attribute Request:** after initialization, the PDP uses the configured protocol to retrieve particular conditional attributes those required in the policy conditions if there are restrictions in the policy. The PDP retrieves local attributes from local DENARs' PIP otherwise from backup DENAR (in particular PIP backup).

5. **Remote Conditional Attribute Request:** the PDP uses PIPcoordinator component to retrieve particular conditional attributes as required in the policy conditions if there are restrictions in the policy. The PDP retrieves via this component only remote conditional attributes from remote PIPs those

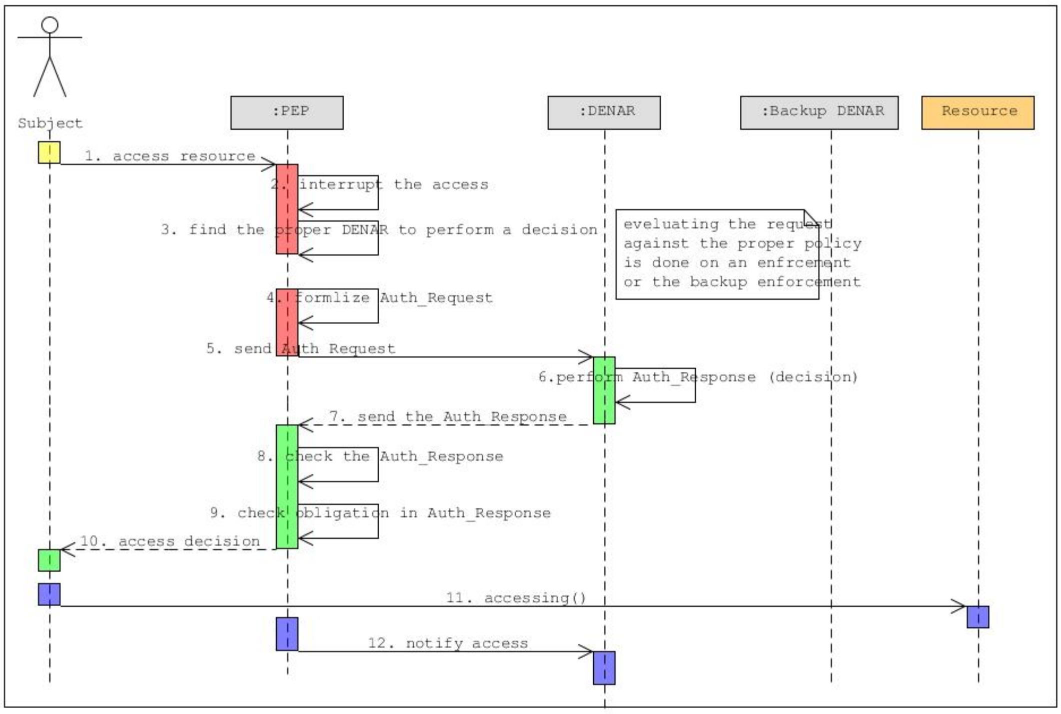

Subject
:PEP
:DENAR
:Backup DENAR
Resource
1. access resource
2. interrupt the access
3. find the proper DENAR to perform a decision
eveluating the request against the proper policy is done on an enfrcement or the backup enforcement
4. formlize Auth_Request
5. send Auth Request
6.perform Auth_Response (decision)
7. send the Auth Response
8. check the Auth_Response
9. check obligation in Auth_Response
10. access decision
11. accessing()
12. notify access

FIGURE 5.6: PEP and DENAR Interaction Sequence Diagram.

are located in different DENARs. The PIPcoordinator can communicate only with PIPcoordinators in the DENARs network to do this task. However, we have modelled this task as Pull Model in our coordination mechanism. [See Chapter 8, Section 8.3]

6. **Submission Decision Locally:** if there is an obligation policy that must be enforced, after the PEP sends the confirmation of the action execution. The PDP after that sends that value into its DENARs' PIP.

7. **Submission Decision Remotely:** if there is an obligation policy that must be enforced, after the PEP sends the conformation of the action execution. The PDP after that sends that value into remote PIPs those located in different DENARs. The PDP uses PIPcoordinator component to store the value into those remote PIPs. The PIPcoordinator can communicate only with PIPcoordinators in the DENARs network to do this task. However, we have model this task as Push Model in our coordination mechanism. [See Chapter 8, Section 8.3]

The diagrams below in Figure 5.7 and Figure 5.8 illustrate the DENAR components for the activity interaction.

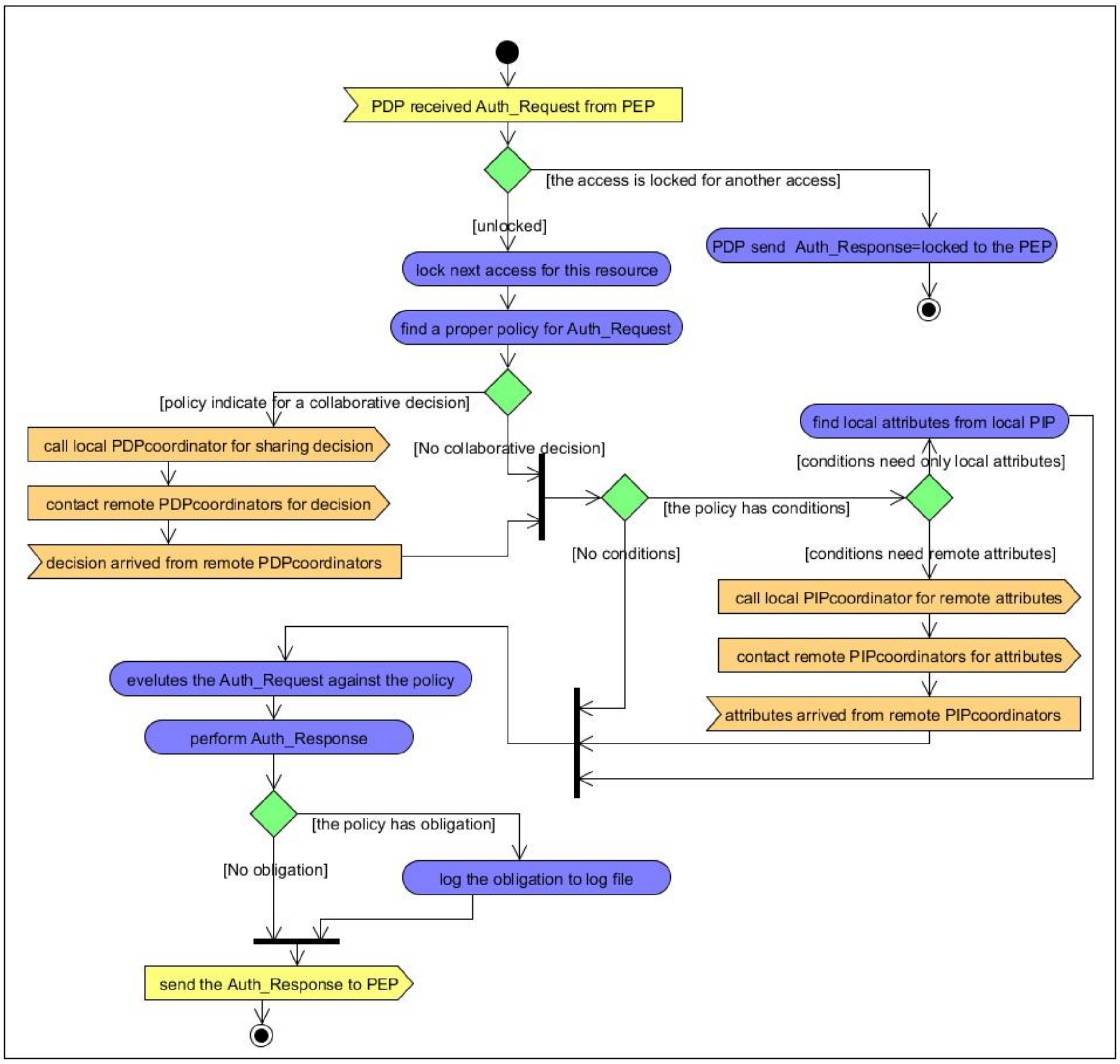

PDP received Auth_Request from PEP
[the access is locked for another access]
[unlocked]
PDP send Auth_Response=locked to the PEP
lock next access for this resource
find a proper policy for Auth_Request
[policy indicate for a collaborative decision]
call local PDPcoordinator for sharing decision
[No collaborative decision]
contact remote PDPcoordinators for decision
decision arrived from remote PDPcoordinators
find local attributes from local PIP
[conditions need only local attributes]
[the policy has conditions]
[No conditions]
[conditions need remote attributes]
call local PIPcoordinator for remote attributes
contact remote PIPcoordinators for attributes
attributes arrived from remote PIPcoordinators
evelutes the Auth_Request against the policy
perform Auth_Response
[the policy has obligation]
[No obligation]
log the obligation to log file
send the Auth_Response to PEP

FIGURE 5.7: DENAR Components Interaction Activity Diagram 1.

Figure 5.7 shows the activity after receiving an authorisation request for a PEP until the DENAR makes its decision and Figure 5.8 shows the activity after receiving an access notification from the PEP until the DENAR end the enforcement process.

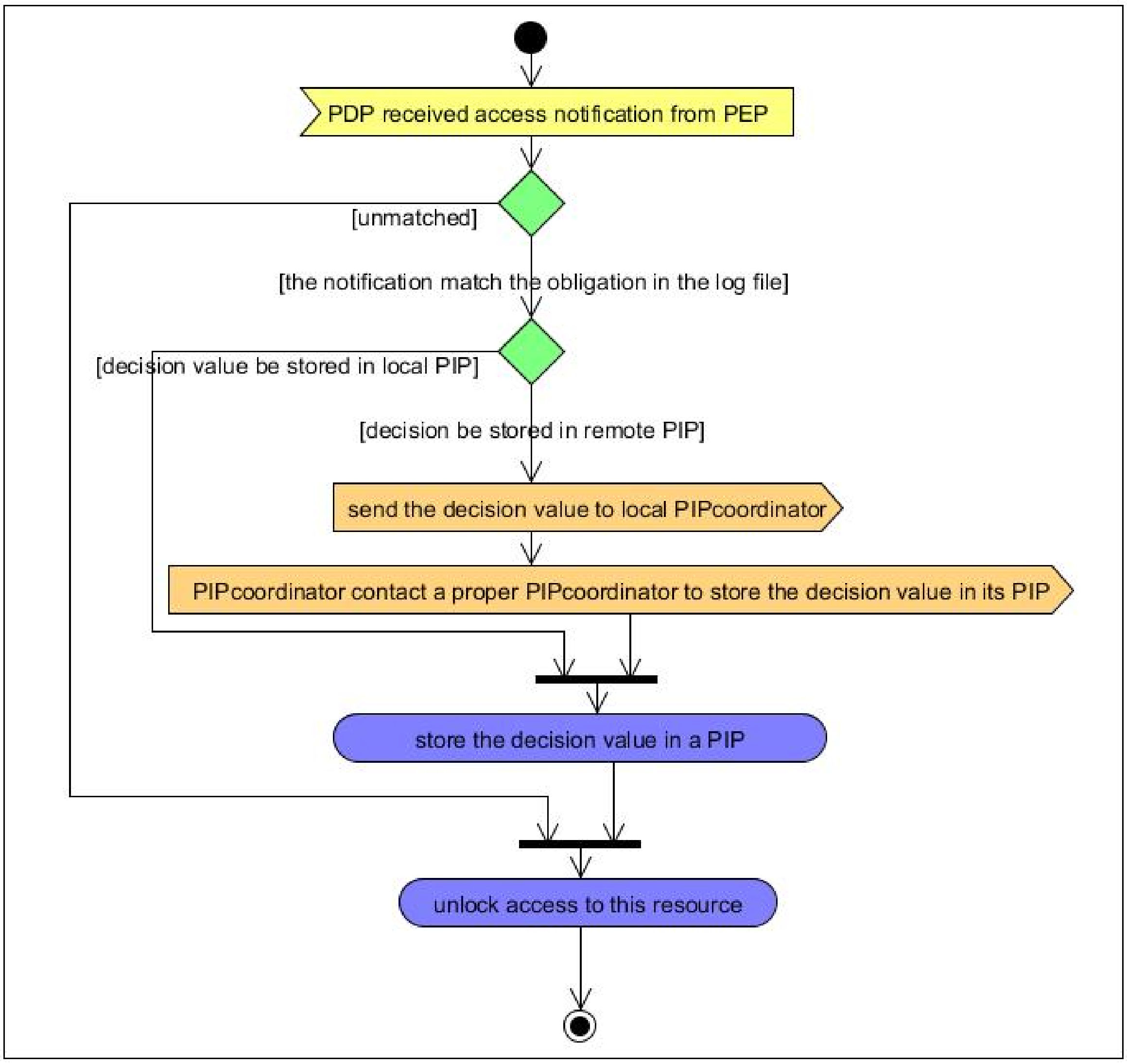

PDP received access notification from PEP
[unmatched]
[the notification match the obligation in the log file]
[decision value be stored in local PIP]
[decision be stored in remote PIP]
send the decision value to local PIPcoordinator
PIPcoordinator contact a proper PIPcoordinator to store the decision value in its PIP
store the decision value in a PIP
unlock access to this resource

FIGURE 5.8: DENAR Components Interaction Activity Diagram 2.

5.3 DENARs Administration

In the Decentralised PBM, a central policy (a simple policy) is formalised, analysed, decomposed and then deployed into a network of DENARs by centralised PAP (Policy Administration Point) into distributed PRs to satisfy and meet all distributed environments requirements. [See details in the Chapter 7] .

The complexity here would be how policies or domains topology could be changed on-the-fly. Thus, distributed enforcements topology, decomposed policies structure, etc., which is beyond our scope at this section.

Scenario: A network administrator configured a university network to contain four network domains (D1, D2, D3 and D4) where domains held resources. DENARs overlay network are configured on top of the university network. The network administrator decided that each domain has its local DENAR to make the access decisions for its resources. The access control policy had specified as central policy that contain all access rules for this network. Let us assume a case where there are four DENARs which are distributed and connected to each other. Each of these DENARs may be contacted in order for various groups of subjects to gain access to protected resources inside the university.

The administrator deploys the decomposed sub-policies to their policy repositories PRs according to their DENARs based on our policy decomposition model and deployment model in Chapter 7. Each decomposed sub-policy would only be available to the local PDP for the DENAR in which it has been deployed.

In order to assume truly distributed systems, these systems should share some information and share their decision making as well. The DENARs in the network are able to exchange the information and collaborate to make access control decision. Each sub-policy is deployed on a certain PR where only the local PDP should have access to it.

Figure 5.9 below illustrates the DENARs network configuration for the above scenario. The red dotted lines indicate to shared authorisation decision pipes between PDPcoordinators. The green dotted lines show shared conditional attribute exchange pipes between PIPcoordinators. The solid black lines show the authorisation message pipes between PEPs and PDPs.

5.3.1 DENARs Configuration

In general our DENAR architecture provides method to implement Decentralised PBM system via an Overlay Network [69] architectures to offer a distributed enforcements model.

DENAR is based on the IETF model [100] where it use COPS protocol [38] as the authorisation mechanism between PEPs and peer PDP and P2P protocol (e.g., JXTA protocols [97]) for the message exchange between PIPcoordinator peers and also PDPcoordinators in the DENARs network.

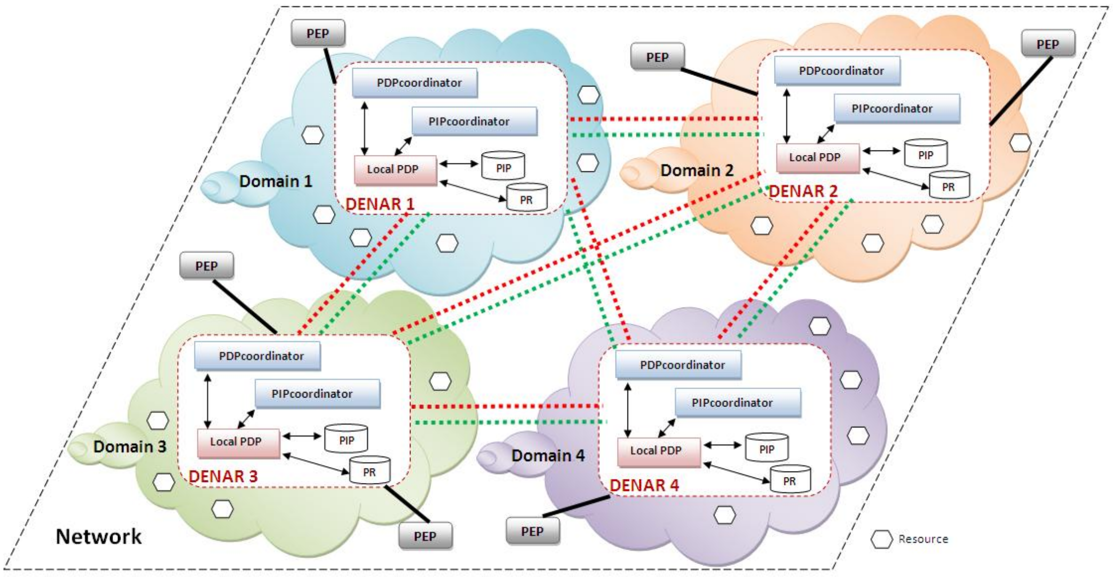


Figure 5.9: The DENARs network for the Scenario.

The PR and PIP are assumed to be within the DENAR as the PDP. The PDP and the PEPs will use the same pre-defined COPS protocol to establish a connection.

Clearly, the main motivation for our DENAR architecture is that chosen predefined PEPs can communicate with the active PDP(s) in domain, otherwise in case of no PDP is active, PEPs in that domain can communicate to the backup PDP. We assume that the pipe for a message exchange between PDPs peers defined where it provides major security concerning for applications in such military, health and national security domains. The PDPs peers must know the “secret keys” before two-way communication can be established. The architecture also assumes that an authorisation request is routed between PEPs in the based in known routing table tell a particular PEP picks the request to forward to a the PDP that connected with the PEP. In this case, the object (target) that is need for perform an action on being within the same domain of the PDP responsible for make the authorisation decision.

Each PR has a different policy file where it is accessible by the PDP that is responsible for the authorisation decision for those objects (targets) are located in the same DENAR. In this case, in the backup DENAR, the PR has a copy of all decomposed policies and also PIP has a copy of all conditional attributes for all targets in the system. Additionally, in case of any policy change or domain topology change, our decomposition mechanism will restart again. However, in case of reconfiguring the policy (changing) or in the network reconfiguration the system can relay on the backup DENAR only to offer the accessibility and availability.

To meet the efficiency target for enforcing the history-based policy [3], we assume that, any PDP can "push" a new value of conditional attribute for the current decision to other PIPs those required for future decision [this is done by obligation, see Chapter 3, Section 3.3.1.3] . On the other hand, the PDP in such cases can "pull" any conditional attribute value that is located in remote PIPs that required in the PDP for the current decision. The decomposition attempts to minimise the lookup and update access control attributes and policies.

Finally, in the DENAR, a central policy (a simple policy) is formalised, analysed decomposed and deployed by centralised PAP. In fact, distributed enforcements implementation for distributed environments that often built upon a physically large and topologically complex network must be reliability and efficiency of administration into consideration. Large distributed systems are usually divided into various and separate domains. As a consequence, policies also are administered in centralised PAP and then decomposed and deployed into distributed PRs to satisfy and meet the distributed environments requirements.

5.3.2 DENARs Deployment

A library can be designed to easily create and synchronise a network of DENARs in a peer to peer fashion based on the network domains topology. A network administrator is able to configure the DENARs as an overlay to the network and also link the PEP's to the DENARs components dynamically. In the case of any DENARs failing, the library is dynamically reconfigured to connect PEPs or any components in the DENAR to the backup DENAR components in the network. The PEP will be capable of connecting to a one active DENAR or if it fails, it will automatically connect to the backup DENARs for a decision.

For the PEP, the DENARs appear to be one coherent system and the distribution and coordination between the various DENARs are transparent. The chief characteristic of the overlay network is that it can be used to easily convert existing standalone, stateless enforcements into a network of stateful networked enforcements with minimal alteration in the existing enforcements code. Having a system of DENARs network improves performance in access control decision making and also eliminates the problem of a single point of failure.

5.3.3 DENARs Re-propagation and Recovery Technique

The distributed policy enforcements that comprise Decentralised Policy Based Management (PBM) system are viewed as independent pieces and are managed as such. Administrators have to configure a number of DENARs based on the network topology. The problems originate from the independent nature of each DENAR. Every DENAR has private sub-policies those decomposed from the central policy, performs access control according to that specification, and is oblivious to the sub-policies those deployed into other DENARs.

This often causes accessing problems which lead to security violations. Firstly, when any of these DENARs is disconnected. Secondly, the network administrators are trying to reconfigure the overlay DENARs topology (e.g. make more DENARs than the actual the overlay DENARs topology exist) in some distribution network view. Finally, the system administrators are trying to reconfigure the Decentralised PBM system with new policies adding or changing (only with new dependent policy).

For the three cases, we design a backup DENAR that is configured by the system administrator. The backup DENAR is utilized in case of any failure communication between any PEP, PIPcoordinator or PDPcoordinator and DENARs. In addition, it is used in case of reconfiguring the Decentralised PBM system with new policies. Clearly, decentralised policy-based management system is become centralised policy-based management system only during the reconfiguration which relay on the backup DENAR for access control. The backup DENAR has its PR that store the central policy (all rules before the decomposition into sub-policies), its PIP that stores all conditional attributes.

However, according to our Decentralised Policy Based Management (PBM) framework we detailed in Chapter 4 and the Distributed Policy Enforcements Architecture (DENAR) four benefits are accrued by re-propagation and recovery technique:

Scalability: A network of DENARs is distributed to improve the performance of access control decision making, thus, DENARs re-propagation can lead to avoiding the performance bottlenecks of a particular DENAR that has access control requests.

Flexibility: Adding or removing resources for the system does not affect the DENARs topology. Therefore, only adding a new policy for that resource and its attribute can be added without reconfiguring the DENARs topology.

Simplicity: Individualized administration of DENARs is eliminated, simplifying management.

Consistency: DENARs topology remains consistent in case a new policy is added or changed. The policy may be analysed and decomposed without changing of DENARs topology.

5.4 Summary

This Chapter introduced the Distributed Enforcements Architecture (DENAR). The components, their function and interaction are described. Details of their design and implementation, together with algorithm are given. Moreover, the network of DENARs administration and configuration are detailed. The deployment of DENARs in Decentralised PBM is introduced where the collaboration between them are involved through the two new component we added, which are PIPcoordinator and PDPcoordinator. In addition, re-propagation and recovery techniques are described for DENARs network.

The novelty of the DENAR architecture is the inclusion of coordination and collaboration mechanisms that allow for distributed PDPs of dynamic policies e.g. history based policies. This improves the standard IETF framework [100] that cannot coordinate distributed PDPs and PIPs and adds a formal underpinning to similar approaches to coordination [27] by using the history-based policy language SANTA [51]. The network of DENARs which include multiple PDPs and PIPs remove the single point of failure that is common in centralised enforcement approaches i.e. that rely on the IETF model. The choice of a distributed PIP overlay in the architecture provides a key contribution, which makes the decision making more resilient to network failure.

In Chapter 9 and 10, the implementation and evaluation showed that the Distributed Enforcements Architecture (DENAR) is feasible to enforce both static and dynamic policies. DENARs' manageability is discussed in Chapter 10, section 10.4.3.

The following Chapter gives an analysis of enforcement static and dynamic policies in the DENAR.

Chapter 6

DENAR Analysis

Objectives

- Identify the security policy challenges in DENAR.
- Analyse the security policy dependencies.
- Identify the distributed policy enforcements challenges.
- Analyse the distributed policy enforcements collaboration.

6.1 Introduction

The distribution of enforcement mechanism in the Decentralised PBM and the conception and development of static and dynamic policies are an essential part of DENAR's design.

This Chapter addresses the analysis of dependencies in security policy to meet policy decomposition requirements. In addition, the analysis of distributed policy enforcements resilience significantly extends this approach, allowing the collaborative DENARs to enforce static and dynamic policies in a decentralised manner.

The remainder of this Chapter is structured as follows. Section 6.2 analyses the security policy enforcement challenges and dependencies in DENAR. Section 6.3 identifies the domain scope for DENARs in subsection 6.3.1 and analyses the collaborative DENARs decision in subsection 6.3.2. Finally, section 6.4 summarises the Chapter.

6.2 Security Policy Analysis in DENAR

In this section, we identify policy dependency challenges facing distributed policy enforcements needs. The dependency between policies in dynamic policy (History Based Policy (HBP)) [3] is analysed and classified to be involved in the decomposition model in Chapter 7.

6.2.1 Security Policy Challenges

A common approach is to distribute the policy to localised policy decision points (LPDPs) that may well form a part of the PEP itself [38] [62] [100]. However, this is not always possible, depending on the policy that is to be enforced by the PDP. When policy is static (Access Control List (ACL)) [2], the distribution of this sort of policy is straightforward and represents essentially the decomposition of an access control matrix into a set of access control lists (object-based distribution) or capability lists (subject-based distribution) [13]. This decomposition is well understood and implemented in many systems [2]. On the other hand, it is impossible with dynamic policies where there is no collaborative enforcement between those LPDPs, i.e. one of LPDPs need for its current enforcement a previous decision result of the other LPDPs.

More recently the decomposition of dynamic policies (History Based Policy (HBP)) [3] have received attention, paying attention to the subject or object attributes that are required for the policy decision making. The decomposition then does entail an analysis of the attributes that are referenced in the policy and distributing these attributes to various localised Policy Information Points (PIPs) that store the attribute information.

In [68] and [91], the notion of policy decomposition is provided. In [68], the policy decomposition is based on the sensitivity of attribute information necessary for access control and/or user defined constraints at each PDP. In [91], however, policy decomposition is guided by the resource type hierarchy. Their model considers refining a high-level access control policy into sub-policies that are specific to a resource instance and then sub-policies are deployed at the PDPs controlling each resource. In comparison, our policy decomposition approach is not guided by the resource type hierarchy or the sensitivity of attribute information but object domains, thus, not every domain has its PDP where that is based on policy dependency analysis.

A major challenge is how a central (simple) policy that includes both static policy (e.g. ACL [2]) and dynamic policy (e.g HBP [3]) are deployed into collaborative enforcements and then interpreting and efficiently enforcing these policies. To shift the computational mechanism towards distributed systems and enforcements, we propose a policy decomposition that finds the dependency between rules in the central policy and then decomposes them into sub-policies based on their object-domains and dependency if they have to be ready for the deployment into enforcements.

Our policy analysis and dependencies address the analysis of the policy specification into static policy (independent policy) and dynamic policy (dependent), in which policy decisions are based on the history of events. To ensure correct enforcement, the policy decomposition is designed to take into account enforcing a policy into a single PDP or collaborative enforcements (PDPs). We propose that the policy is decomposed into sub-policies according to rules' object domains and then send these sub-policies to corresponding DENARs which are deployed in deployment mechanism (described in this Chapter, section 6.3).

In the following example, we illustrate how our policy decomposition for static policy (independent policy) (e.g Access Control List (ACL) [2]) is being simple. Nevertheless, the situation becomes complicated when dynamic policy (dependent policy) (e.g History based policy(HBP) [3]) has to decompose into sub-policies.

An illustrative example: Suppose a company has two categories of employees (*fulltime* staff and *agency* workers). Further, we distinguish two types of resources representing documents available in the system requirements; these are (*operational* and *strategic*). For this example, we concern ourselves with an action (*read*), representing all types of access that allow subjects to retrieve information from the documents.

The policy denotes that only *fulltime* staff can *read strategic* documents whereas *operational* documents are *readable* by both *fulltime* staff and *agency* workers. Also, the *fulltime* staff can *delete* the *strategic* and *operational* documents with a condition of (*if and only if any of fulltime members has read the strategic document*).

We here have the set of subjects ($S =$ *fulltime, agency*); the set of objects ($O =$ *operational, strategic*). Let *fulltime* staff be (*Fiona and Fred*), agency workers be (*Alice and Adam*). *Operational* documents are (*open.txt, transfer.txt and close.txt*) and *strategic* document is (*regulation.txt*). These requirements can be formalised in a central policy that captures these requirements as rules.

As we illustrated before, the general form of a rule in SANTA is:

$$\boxed{consequence\ (S, O, A)\ when\ premise}$$

The consequence of a rule distinguishes the class of requirements that can be expressed in that rule. The triplet *(S,O,A)* references the subjects *(S)*, objects *(O)* and action *(A)* to which the rule applies. Finally, the premise of the rule describes the condition under which the rule fires.

The formalisation for the example requirements above in SANTA is given below. There is no condition in *rule1* and *rule2*, thus, the premise is stated as *True* but in *rule3* where there is condition is stated in:

$$\boxed{sometime\ done\ (S, O, read)}$$

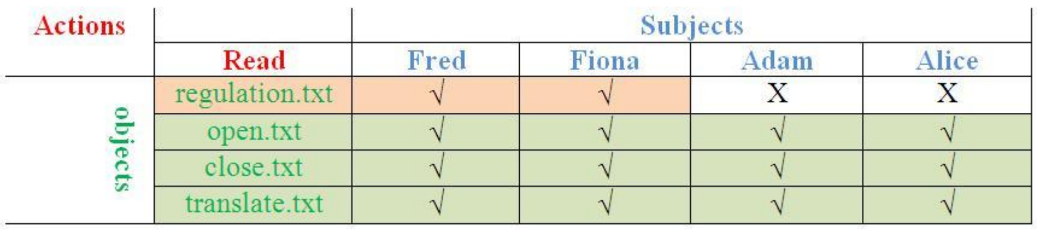
<u>Example 6.1</u>

<u>*rule1 :*</u>
allow $(S, O, read)$ *when true where* $S := \{Fiona, Fride\}$ *and*
$O := \{regulation.txt\}$

<u>*rule2 :*</u>
allow $(S, O, read)$ *when true*
where $S := \{Fiona, Fride, Adam, Alice\}$ *and*
$O := \{open.txt, close.txt, transfer.txt\}$

<u>*rule3 :*</u>
allow $(S, O, delete)$ *when sometime done* $(S, regulation.txt, read)$
where $S := \{Fiona, Fride\}$ *and*
$O := \{open.txt, close.txt, transfer.txt, regulation.txt\}$

The central policy above has three rules where *rule1* and *rule2* show the static (independent) policy and the *rule3* illustrates the dynamic (dependent) policy. *Rule1* and *rule2* can be implemented in matrix or (ACL) as shown in Figure 6.1 but *rule3* cannot for the reason that the decision for the *rule3* is based on the past decision of the *rule1*. Therefore, decomposing the central policy into sub-polices is being complicated for the (History Based Policy) reason.



Actions		Subjects			
	Read	Fred	Fiona	Adam	Alice
objects	regulation.txt	√	√	X	X
	open.txt	√	√	√	√
	close.txt	√	√	√	√
	translate.txt	√	√	√	√

FIGURE 6.1: Access Control Matrix.

The orange colour shows the *rule1* and the green colour shows the *rule2*. However, the *rule3* cannot be implemented in this matrix where the decision for the *rule3* is based on the past decision of the *rule1* that indicated in its condition.

The enforcement execution for *rule1* and *rule2* are shown the Figure 6.2 where no conditions are included. The event means that subject(s) perform a particular action on object(s). The enforcement decision for *rule1* and *rule2* are fully independent of each other, thus, no matter if each of them are in a different system to where the rule exists.

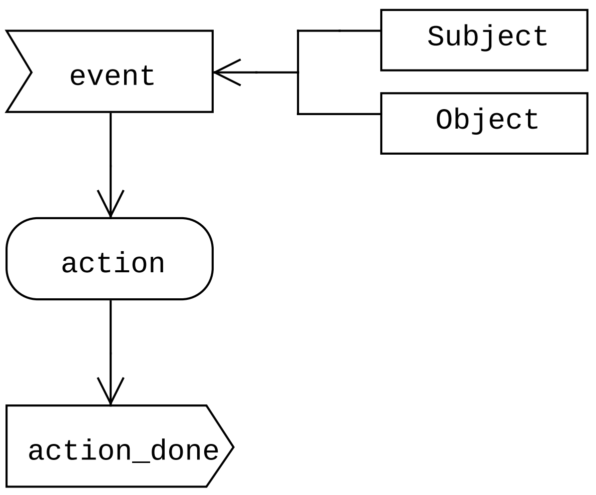


FIGURE 6.2: The Enforcement Decision for no Condition Rule.

However, conditions' branches in the rule are conceivable to question for subject attributes', object attributes' or environment attributes. In addition, they may question for values of decisions that are taken at some point in the past as it is shown in *rule3*. Clearly, Figure 6.3 shows the dependency between *rule1* and *rule3* where the enforcement execution for *rule3* that restricts performing its action on the past decision of the *rule1*.

At the end of this Chapter, the result of the our policy decomposition for the above policy is shown where the *rule1* remains as it is and the *rule2* is decomposed into sub-policies based on fragmenting the object scope and then are deployed and enforced in their domains' enforcements. Finally, *rule3* is decomposed into sub-policies based on the rule's object-domains and its dependency with *rule1* and then are deployed and enforced in their domains' enforcements. Therefore, the challenge

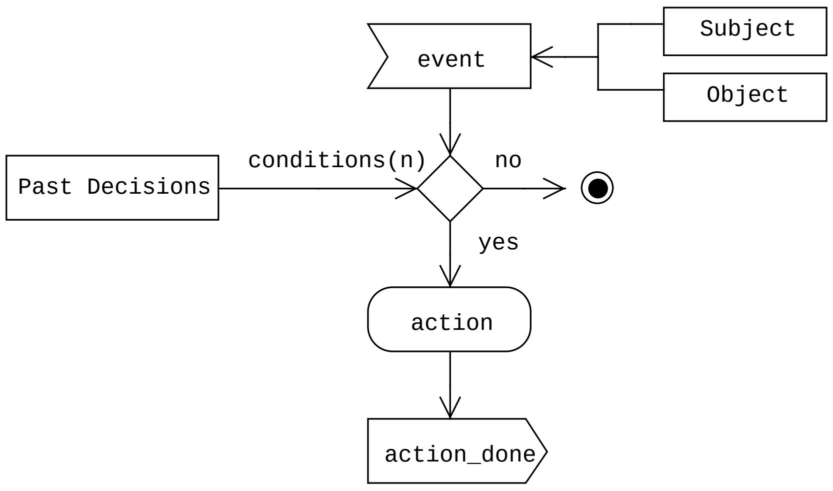


Figure 6.3: The Enforcement Decision with Past Decision Condition.

that is illustrated in this Chapter is showing how these rules are decomposed appropriately to be enforced and result in the same decision by distributed PDPs in comparison with a centralised PDP.

6.2.2 Policy Dependency Scope

A policy analysis and dependencies on the structure of the system requirements are the key for Distributed Enforcements Architecture (DENAR). Policy analysis identifies dependencies between policy and the access control in information systems (particularly, for dynamic policies). The second aspect is the analysis of the enforcement for both static and dynamic policies in DENAR to meet collaboration enforcement requirements.

In order to enforce a policy, all relevant observations must be available to a PDP and the PDP must control the interaction with the protected resources. In dynamic policies (History Based Policy (HBP)) [3], for example, the Chinese-Wall policy [25] between two resources in a system requires the PDP to control interactions with both resources. The techniques presented in this section are however independent of the policy language chosen and can easily be adapted to other policy languages that allow for the expression of history-based policies (e.g. [51]).

In some cases, an access to a resource must be restricted to satisfy the IT management requirements. On the other hand, no restrictions are required as well to meet the IT requirements.

The general form of a rule is:

$$\boxed{\textit{consequence } (s, o, a) \textit{ when premise}}$$

The consequence of a rule distinguishes the class of requirements that can be expressed in that rule. The triplet (s,o,a) references the subject, object and action to which the rule applies. Finally, the premise of the rule describes the condition under which the rule fires.

Premise in a rule describes a set of conditions that presents these restrictions. Empty or none existed conditions in premise expresses no restrictions for subject(s) to access resource(s) where it is the simplest rule in ACL model [2]. However, when the access to resource(s) must be restricted, this could be expressed in premise with one or multiple conditions.

Regarding access control requirements, we can classify the premise into basic premise (no condition) or conditional premise (has non empty condition) where we classify it as independent or dependent premise based on a conditions' structure. The following subsection shows these classifications.

Basic Premise (No condition)

Definition 6.2:- an event (either an explicit message being passed between two points, or a change of state) triggers the execution of an action.

The basic premise can be authorising for the event to trigger the action without any restrictions. Figure 6.2 depicts this process. Moreover, a system may allow different events to trigger the action. Based on a policy language that used to support subject scope or objects scope, viz. only one subject or a set of subjects and a one or a set of objects can be used in the event.

The basic premise can be formalised in as follows:

$$\boxed{\textit{consequence } (s, o, a) \textit{ when true}}$$

Conditional Premise

The conditional premise is supporting set conditions to be evaluated before the action is taken. The conditional premise is more complicated in that it can be nested and iterated. In addition, the action is executed only when the event occurs and all condition branches hold true. Figure 6.4 shows the control flow of the condition premise.

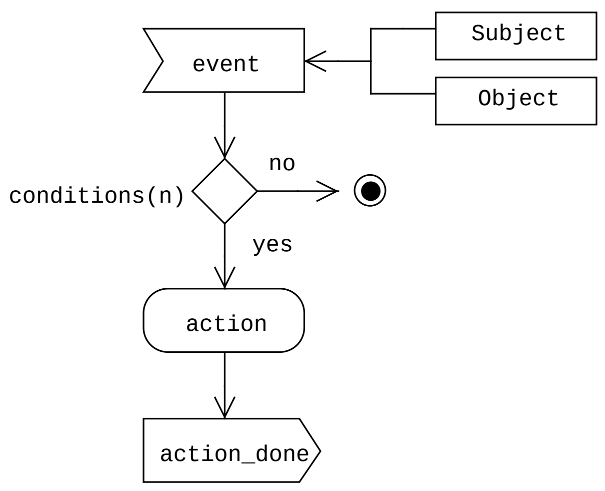


FIGURE 6.4: Conditional Premise Rule Decision.

The different decision of a rule can be determined by only the conditions they evaluated. If none of the conditions evaluate to true, no action is performed.

The conditional premise is conceivable to hold one or more of subjects' attribute, objects' attributes, environments' attributes, values of decision that are taken at some point in the past (past events) or hypothetical rule condition. Subjects', objects' and environments' attributes are involved in the conditional premise to be evaluated to true where they question for subject, object or environment attributes' themselves. The examples 6.2, 6.3, 6.4, 6.5 and 6.6 describe these sort of conditional premise sequentially.

Example 6.2 (Subjects' attributes condition) A subject can be any actor that deal with the system (e.g. user, software. etc.) Each subject in a system has some identified attributes those required to verify and recognise a particulate subject rather than other subjects in the system. The below rule shows the subjects' attribute condition when the age of a subject that perform an action on an object must be greater than 30 years.

$$\boxed{consequence\ (s, o, a)\ when\ (s.age > 30)}$$

Example 6.3 (Objects' attributes condition) A object can be any resource (target) that a subject deal with (e.g. file, door. etc.). Each object in a system has some identified attributes those required to verify and recognise a particulate object rather than other objects in the system. The below rule shows the Objects' attribute condition when the size of an object that perform an action on by a subject must be greater than 10kb.

$$\boxed{consequence\ (s, o, a)\ when\ (o.size > 10000)}$$

Example 6.4 (Environments' attributes condition) The environments' attributes indicate to the system attributes, e.g. the current date. The below rule shows the date of the event must be at 01.01.2012 to perform an action.

$$\boxed{consequence\ (s, o, a)\ when\ (system.date = 01.01.2012)}$$

Example 6.5 (Event condition) Another important IT management requirement is a consideration of the specification of history-based policies [51]. For example, the requirement "if a user at some point in the past has read file (A) that is secret then the same user cannot write to file (B), otherwise the user can write on file (B)" can be expressed by the use of a conditional in premise. This is considered in the expression of a large class of security requirements, including dynamic policies such as the Chinese Wall Policy [25]. These sorts of conditions express as the past event conditions. Thus, considering event condition on the state of the system or on the history of the execution is essential (e.g. in military system). The below rule shows the event condition when the subject (Alice) had read the object (open.txt), a subject (s) in the rule can then perform an action (a) on an object (o).

$$\boxed{consequence\ (s, o, a)\ when\ (done(Alice, open.txt, read))}$$

Example 6.6 (hypothetical rule condition) Hypothetical access is also important IT management requirement. For example, the requirement "if a user can read file (A) that is secret then the same user cannot write to file (B), otherwise if not can write on file (B)" can be expressed by the use of a condition in premise. This is considered in the expression of a large class of security requirements, including dynamic policies such as the Chinese Wall Policy [25]. These sorts of conditions express as the hypothetical rule conditions. Thus, considering hypothetical rule condition is essential in such systems. e.g. military system. The below rule shows hypothetical rule condition when the subject (Alice) can read the object (open.txt), a subject (s) in the rule can then perform an action (a) on an object (o).

$$\boxed{consequence\ (s, o, a)\ when\ (allow\ (Alice, open.txt, read))}$$

Independent and Dependent Condition

The premise context is conceivable for the classifying independent or dependent rules. The premise classification is involved to be used for the policy decomposition [See Chapter 7, Section 7.2]. We classify a rule to be independent, when no condition is involved in the rule or all conditions in condition set are being independent conditions. On the other hand, when one dependent condition in premise is found the rule is classified as dependent rule. The subject and environment attribute conditions are always classified in our analysis as independent conditions where these attributes can be found internally (in the same domain). In this case, the local PIP in a DENAR can provide these attributes and then make the authorisation decision. However, in case of object and event attribute and hypothetical rule condition's evaluation, these attribute values can be located in the same or different domains thus, the conditions evaluation may involve two or more DENARs in the network. [For more detailed see Section 6.3.3]

6.3 Collaborative DENAR Analysis

In the centralised enforcement approach all interactions between users and system resources are controlled and observable by the centralised PDP where there is no coordination. However, in the network of DENARs where multiple PDPs are involved, the coordination is a key to collaborate PDPs for an enforcement decision. The collaborative DENARs can be synchronised which is important when controlling usage limits for resources across the distributed system, or dynamic constraints, such as the mutual exclusion requirements found in the well known Chinese Wall policy [25] and dynamic separation of duty constraints [83]. In the following, we addresses how a domain scope can indicate to subsystems and logically how the collaborative DENARs is achieved.

6.3.1 Domain Scope

In large-scale systems, domains provide a practical solution for PBM system where specifying a policy for individual object is difficult to be managed in PBM system [41]. Sloman et. al [41] define a domain as a group of objects in a large system based on geographical boundaries, object type, responsibility or authority. For large-scale systems, therefore, specifying a policy for a particular domain (a group of objects that share the same interest) propagates scalability for managing PBM.

In decentralised access control when multiple domains are involved, the access control in a domain is independently maintained from other domains (one PDP is involved for each domain but there is no collaboration between PDPs). Specifying policies based on this technique results in difficulties for solving a policy conflict between domains especially when Bell-LaPadula Model (Multilevel Security) [66] or Chinese-Wall [25] policies are involved and are not enforced correctly. Ideally, a PBM system may involve multiple PDPs to improve the access control decision making performance.

Definition 6.1. A domain is a collection of objects governed by the same geographical boundaries, object type, responsibility or the level of security or authority those may grouped from different systems.

Grouping objects in a large system into domains is useful. Unlike Ponder policy language, in our Decentralised PBM, we use the domain scoping in the policy decomposition where it being after all policies are specified and all dependency between policies are detected as well and then the policy decomposition that based on the objects domains is performed. In Ponder policy language, the specification is based on the objects domains.

In addition, we can configure one PDP in DENAR to be responsible for enforcing the policies for one or more domains. The Decentralised PBM that has collaborative DENARs can share a single decision by multiple PDPs in the system. Therefore, according to our Decentralised PBM framework, policy decomposition step is after the policy specification to solve the policy conflicts from policy abstraction level and then the domain consideration being in policy decomposition. Clearly, our Decentralised PBM framework does not solve the policy conflation where it must be solved in the policy specification step.

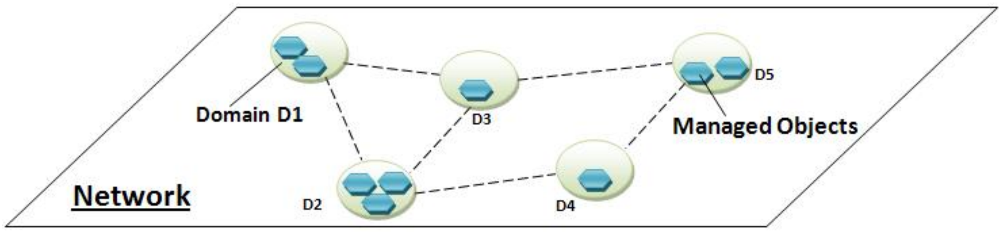


Figure 6.5: Network Domains.

The Figure 6.5 shows distributed domains in a network by circles where each circle contains one or more objects those grouped in same interest. Network administrators manage a distributed hierarchy of domain objects and support the efficient of controlling access by involving DENAR. The domain service can be implemented in our Decentralised PBM as in Ponder [32] using Lightweight Directory Access Protocol [43] .

6.3.2 Collaborative DENAR Decision

In this section, we show how the enforcement of static and dynamic policies can impact the potential problems that can occur when multiple PDPs are implemented. Also, we extend the approach in [53] to define sufficient constraints on the concurrency of the distributed PDPs to avoid conflicts.

In order to coordinate DENARs, the access control policy firstly defines the conditional attributes which need to be coordinated. Finally, it specifies the parameters for the conditional attribute updation - that specifies when the conditional attribute should be updated. The above be achieved by rules refinement to complement each rule that needs coordination in this manner. We introduce Pull Model for retrieving the conditional attribute and Push Model for updating the conditional attribute by an obligation. [See Chapter 7, Section 7.2. for policy refinement]

We base DENAR coordination on the UCON model [78], allowing for per, ongoing and post authorisation control of long standing interactions. In addition, we provide a semantics in Interval Temporal Logic [74] for the dynamic policies in a motivation example [See Chapter 6, Section 6.4] for the Chinese Wall [25] policy. We use the controller that presented in [53] for the static policy to avoid the conflicts with the respect of DENAR' decision. Additionally, we involve the concept of obligations discussed earlier wherein a PDP sends a PEP instructions on obligations that need to be necessarily fulfilled on execution of the request. The obligation element specifies the actions to be taken by the PEP, rather than the PDP, since it is at the PEP that the action is actually executed.

The development of a semantic model to express dynamic policies, viz. policy that can change over time and on the occurrence of event (e.g the Chinese Wall Policy [25]), and their distributed enforcements in distributed systems are the underpinnings of our research. Our collaborative DENARs is built on the formal policy model investigated in [53] and [55]. To meet the challenges, a sound theory of policy decomposition is developed, taking into account the limitation of involving multiple PDPs without considering the coordination for the dynamic policy enforcement where a PDP in DENAR may require conditional attributes from other DENARs in different domain.

The enforcement in access control system must to be stateful and maintain attributes that influence future control decisions to meet the requirements especially when some policies are involved, such as the Chinese Wall Policy [25] or Clark-Wilson [28]. Park et.al. [78] provide in Usage Control (UCON) model far reaching flexibility for access control specifications where the notions of mutable attributes are proposed. Using mutable attributes for these policies can be expressed, or the number of users concurrently accessing a resource can be limited where attributes can change their values based on the previous accesses that are initiated by users [53]. [See Chapter 2, Section 2.4.5 Usage Control (UCON) for more details].

6.3.2.1 Independent Rule Decision

In $UCON_A$, $preA_0$ (pre-authorisation) policy is specified to be enforced without pre-updates attribute. Therefore, we can classify this sort of policy as the independent policy where immutable attributes are evaluated. Therefore, the enforcement will be stated as the Figure 6.6 below illustrated.

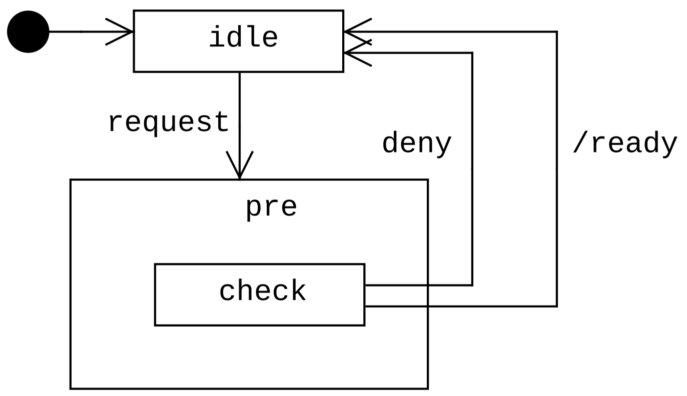


FIGURE 6.6: Static PDPcontroller.

The state *check* evaluates the request against a particular policy.

6.3.2.2 Dependent Rule Decision

$preA_1$ (pre-authorisation) policy is specified to be enforced with attribute on-updates (mutable attributes) before the access takes place. In addition, $preA_3$ (pre-authorisation) policy is specified to be enforced with attribute post-updates (mutable attributes) after the access is taking place. We design Pull Model to retrieve these attributes to the PDP [See Chapter 8, Section 8.3.1 for Pull Model]. Therefore, we can classify this sort of policy as the dependent policy. The PDP that enforces the dependent policy can also perform update action for these attributes to the same PDP or other PDPs for the access decision in Push Model [See Chapter 8, Section 8.3.2 for Push Model]. Therefore, the PDP will be stated as the Figure 6.7 below illustrate.

The states of the PDP are states PDPcontroller = idle, check, allowed. The behaviour of a PDPcontroller is then defined as:

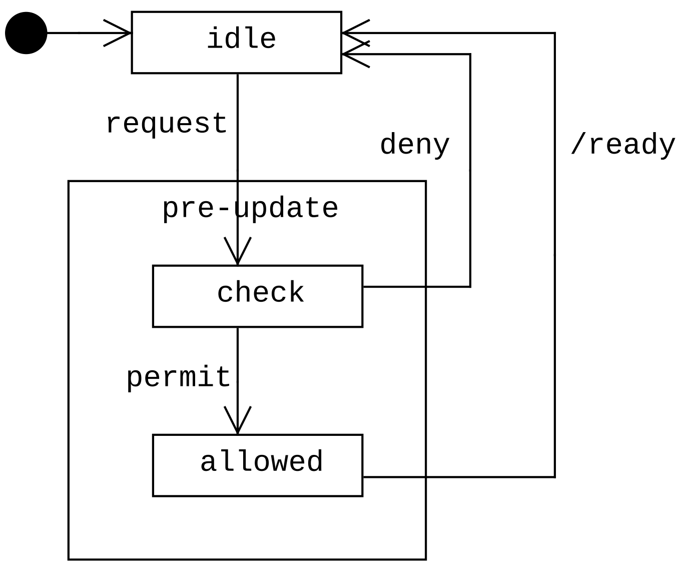


FIGURE 6.7: Dynamic PDPcontroller.

$$\varphi\ idle_{,p} \widehat{=} PDPcontroller_p(idle,\ \{E_{init_p},\ E_{deny_p},\ E_{rdy_p}\},\ \{E_{req_p}\}) \quad (1)$$

$$\varphi\ check_{,p} \widehat{=} PDPcontroller_p(check,\ \{E_{req_p}\}, \{E_{pull_p}\}^*, \{E_{push_p}\}^*, \{E_{permit_p},\ E_{deny_p}\}) \quad (2)$$

$$\varphi\ allowed_{,p} \widehat{=} PDPcontroller_p(allowed,\ \{E_{push_p}\}^*, \{\ E_{rdy_p}\}) \quad (3)$$

$$\varphi\ PDPcontroller_p \widehat{=} States_{PDPcontroller_p} = idle \wedge (\varphi idle_{,p};\ \varphi check_{,p}; \varphi(allowed_{,p}, \oplus(empty\ \wedge\ E_{deny_p})))^* \quad (4)$$

$$\varphi\ Control \widehat{=} \bigwedge_{P\in p}(\varphi PDPcontroller_p) \quad (5)$$

In equation 2, the state *check* does explicitly perform pre-update actions for $preA_1$ to modify the mutable attributes of the policy in the push event where these may located in same domain or different domains. These update actions are part of UCON policies and are defined as an obligation event on the state *check*. The example of this case is a check of bank account balance before withdrawing.

However, in equation 3, the state *allowed* does explicitly perform post-update actions for $preA_3$ to modify the mutable attributes of the policy in the push event where these may located in same domain or different domains. The example of this case is watching movies online where the access control is based on the usage time against the credit balance.

The choice of involving push event in *check* or *allowed* state is based on the system requirement where the push event does not occur in both states.

However, we take the view that the mutable attributes updates can be expressed as the assignment for a set of attributes to new values. More complex update activities can be introduced, which however complicates the model and its analysis when attributes can be shared between the DENARs for the various usage processes.

6.4 Summary

This Chapter presented the DENAR analysis which is based on ITL. The analysis presented in this work addresses these issues by providing analysis policy dependency and the semantics of distributed PDPs. The DENAR analysis presented here significantly extends approach [51], allowing for the enforcement of static and dynamic policies in a decentralised manner where multiple PDPs can be coordinated. We illustrated the syntax of ITL in terms of collaborative decisions for dynamic policy, we explained the semantics, and demonstrated some examples.

As part of our contribution, we demonstrate how collaborative DENARs enforce $UCON$ ($preA_0$, $preA_1$ and $preA_3$) pre-authorisation policies in a decentralised manner where those policies may decompose into DENARs.

In Chapter 10, the evaluation shows that the policy classification into independent and dependent policy rules and implementing the domain scope to enforce security policies results in more efficient enforcement.

In the next Chapter, we propose a novel policy decomposition model for both independent and dependent policies. Also, a policy deployment in DENARs is discussed.

Chapter 7

Policy Decomposition and Deployment

Objectives

- Describe the policy decomposition.
- Provide a policy deployment model.
- Present algorithms for policy decomposition and deployment.

7.1 Introduction

The policy decomposition and deployment are described in this Chapter. The decomposition will make use of the policy analysis results (Chapter 6) and describe the method to decompose a central policy into sub-policies and refine them to be enforced in the collaborative DENARs. The policy deployment provides a way of distributing sub-policies to different policy repositories based on the domains scope.

The decomposition will make use of the policy analysis results (Chapter 6) and describe

The remainder of this Chapter is organised as follows. Section 7.2 describes the policy decomposition. Subsection 7.2.1 explains a rule fragmentation and subsection 7.2.2 rule refinement. Section 7.3 describes the policy deployment to distributed DENARs in the Decentralised PBM system. Finally, section 7.4 summarises the Chapter.

7.2 Policy Decomposition

This section describes the method to decompose a central policy into sub-policies. The policy decomposition is achieved by the rule fragmentation phase and the rule refinement phase. In the rule fragmentation phase, the fragmentation is processed which output the fragmented-rule where each rule controls only one object. Moreover,in the rule refinement phase, the dependent and independent rules are recognised and refined to be enforced in DENAR where the DENARs collaboration are required in dependent rules enforcing. The rule refinement phase outputs the sub-policies.

Once the decomposition is complete, the sub-policies are deployed into their corresponding DENARs. In Figure 7.1 illustrates the central policy is being input in the policy decomposition. The green rectangles show the output of each phase.

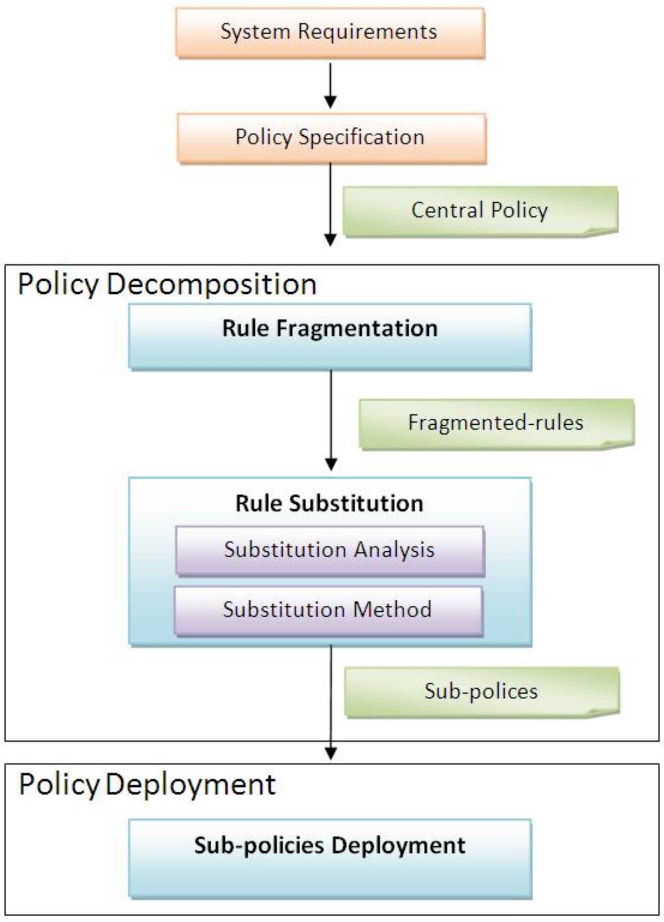


Figure 7.1: Policy Decomposition Phase and Policy Deployment Phase.

7.2.1 Rule Fragmentation

The specification of policy is being a prerequisite and input for the proposed rule fragmentation. The rule is fragmented according to object set that is reference in a policy scope. The fragmentation method is achieved by the idea of controlling only one object in a fragmented-rule that is generated from the rule.

According to the SANTA policy language we are used, the policy scope in defines to which subjects, objects and actions the policy applies to. The scope of a policy is presented in the syntax using the keywords S, O and A, that represent the set of subjects, the set of objects and the set of actions of a policy. The scope of a policy affects the rules contained in that policy. The subject, object and action in each rule definition must be within the scope of the policy (s ∈ S, o ∈ O and a ∈ A).

Often rules apply to more than one subject, object or action. To achieve the objective of the policy decomposition, fragmenting the rules for the different objects in the O can be used to reference only one object per rule. For example, given that the scope of the policy is:

$$\boxed{S = \{s\} \ and \ O = \{o_1, o_2, ..., o_n\} \ and \ A = \{read\}}$$

The requirement to unconditionally grant subject(s) in the scope of the policy the right to perform read on all objects , would require the definition of n rules:

$$\underline{rule:}$$
$$\forall o \in O \ [[\ allow \ (s, O, read) \ when \ true]]$$

Using the keyword O is instantiated to mean the conjunction of:

$$\underline{rule:}$$
$$[[\ allow \ (s, o_1, read) \ when \ true]] \wedge$$
$$[[\ allow \ (s, o_2, read) \ when \ true]] \wedge$$
$$/\ldots\ldots/ \qquad \wedge$$
$$[[\ allow \ (s, o_n, read) \ when \ true]]$$

on is the last object in the object set.

Also, the objects set may include all objects in the rule which are controlled or required. For example, given that a $CONTROLS(x)$ of the policy and a $REQUIRES(Y)$ are:

$$\underline{rule:}$$
$$allow \ (S, x, read) \ when \ done \ (S, y, read)$$
$$where \ S = \{s\} \ and \ O = \{x, y\}$$

The object x is controlled and object y is required in this rule where no fragmentation is applied here. The use of the rule fragmentation has the advantage that it allows the decomposition for the same rule into sub-policies based on the object domain.

Case1, A one object is isolated in the object set and therefore no fragmentation for this case. The rule is ready for the next phase (refinement phase) in the policy decomposition. Thus, according to early example 6.1 [in Chapter 6, section 6.2.1], *rule1* can be expressed in SANTA without fragmentation as:

$\underline{rule1:}$
$allow\ (S, O, read)\ when\ true$
$where\ S = \{Fiona, Fride\}\ and\ O = \{regulation.txt\}$

$\underline{The\ fragmentation\ result\ for\ rule1:}$
$allow\ (S, regulation.txt, read)\ when\ true$
$where\ S = \{Fiona, Fride\}$

No fragmentation just matching object O with its object set in the policy scope.

Case2, for the set of objects, the set of objects is defined and includes a number of objects those located in a same domain or different domains in the system. Thus, the object set can be fragmented based on their domains. In following, we eliminate the object set by the fragmentation method which generates new rule for each object in the object set for the policy scope.

$\underline{rule2:}$
$allow\ (S, O, read)\ when\ true$
$where\ S = \{Fiona, Fride, Adam, Alice\}\ and$
$O = \{open.txt, close.txt, transfer.txt\}$

The fragmentation result for the *rule2* is:

$\underline{rule2.1:}$
$allow\ (S, open.txt, read)\ when\ true$
$where\ S = \{Fiona, Fride, Adam, Alice\}$

$\underline{rule2.2:}$
$allow\ (S, close.txt, read)\ when\ true$
$where\ S = \{Fiona, Fride, Adam, Alice\}$

$\underline{rule2.3:}$
$allow\ (S, transfer.txt, read)\ when\ true$
$where\ S = \{Fiona, Fride, Adam, Alice\}$

Based on the object set in the policy scope for this rule that include three objects, the fragmentation method produces three rules where each rule controls only one object.

Case3, for the set of objects that includes CONTROLS (O) and REQUIRES (o) objects, the fragmentation method eliminates the object set by generates new rule for each controlled object in the object set for the policy scope.

$\underline{rule3:}$
$allow\ (S, O, delete)\ when\ sometime\ done\ (S, regulation.txt, read)$
$where\ S = \{Fiona, Fride\}\ and$
$O = \{open.txt, close.txt, transfer.txt, regulation.txt\}$

The fragmentation result for *rule3* is:

$\underline{rule3.1:}$
$allow\ (S, open.txt, delete)\ when\ sometime\ done\ (S, regulation.txt, read)$
$where\ S = \{Fiona, Fride\}$

$\underline{rule3.2:}$
$allow\ (S, close.txt, delete)\ when\ sometime\ done\ (S, regulation.txt, read)$
$where\ S = \{Fiona, Fride\}$

$\underline{rule3.3:}$
$allow\ (S, transfer.txt, delete)\ when\ sometime\ done\ (S, regulation.txt, read)$
$where\ S = \{Fiona, Fride\}$

$\underline{rule3.4:}$
$allow\ (S, regulation.txt, delete)\ when\ sometime\ done\ (S, regulation.txt, read)$
$where\ S = \{Fiona, Fride\}$

Algorithm for fragmentation based on the cases above, a simple fragmentation algorithm is given, as outlined by the following pseudo-code:

Algorithm 1 Rule Fragmentation

```
Require: central policy (P)
 1: Begin
 2: Load P
 3: for each r in P do
 4:    r' = r
 5:    for each o in {O} for r' do
 6:       obj = o
 7:       remove all objects in {O}
 8:       insert obj in {O}
 9:       RuleID = sequential number
10:       add RuleID into r'
11:       add r' into fragmented − rule repository
12:    end for
13: end for
14: End
```

The above algorithm performs the fragmentation of rules for the central policy (P). According to the number of objects in object-set (O), the fragmented rules are produced. Moreover, the *RuleID* is added as identifier of the new rules. The algorithm output copies the new rules into fragmented-rule repository where each rule controls only one object in its object-set to be ready for the next phase (Rule Refinement).

7.2.2 Rules Refinement

According to the discussion in Chapter 6, the IT requirements meet the importance of involving the dependent (dynamic) and independent (static) policies. The needs of dependency between rules is explored in Chinese-Wall [25] and highlighted in the example 6.1 (in particular *rule3*). The conditions' context (premise) is conceivable for the classifying independent or dependent rules. Clearly, refinement analysis classifies subject attributes' and environment attributes' that structured in a conditional premise into an independent condition. On the other hand, the rule that its conditional premise asking for object or past event attributes' is considered as the independent or dependent condition based on objects domain location. The conditional premise classifications are involved to be used for the refinement rule method [See Chapter 6, Section 6.2.2 for the conditional premise classification].

7.2.2.1 Refinement Analysis

The analysis in this section is expressed to only one rule that loaded from the fragmented policies sequentially that is going to be refined. Refinement algorithms follow two goals: first, to ascertain that the condition premise (condition context) does in fact match the condition premise classification in question. If it does, the algorithms proceed to refine the rule into independent or dependent rule. Consequently, each condition in the independent or dependent rule must show the type of signalling, viz. where the condition attribute values are founded and retrieved in local or remote domain. If the condition premise does not match the first condition premise class, the next condition premise class can then be tried. In the following subsection we show how our refinement analysis is processed in steps. [See Chapter 6, Section 6.2.2 for the independent or dependent rule].

- **Matching Condition Premise Class**

  The first step in the rule refinement method is to load the first rule from the fragmented-rule repository (the output of the rule fragmentation phase) to the refinement algorithm. [See Section 7.2.2.2 and Section 7.2.2.3]. Then, the matching of condition premise class (independent or dependent) is the recognition procedure that required for choosing appropriate refinement method.

  Basically, to identify the condition premise class matched for condition in a rule, the condition premise is compared to every the condition premise class till the match is found or until there are no further classes to compare to. Based on the policy language is used, the order of the condition premise in a rule may be presented differently.

  For all rules fragmented-rule repository, the condition premise must be matched. However, in the case of unmatched condition premise, human intervention becomes essential. The unmatched condition premise possibly may be caused by errors in the policy specification or a new condition premise class those not defined. The refinement algorithms show how condition premise can match its corresponding condition premise class. [See Section 7.2.2.2 and Section 7.2.2.3]

- **Mapping Objects' Domain**

  Mapping objects' domain is the process for pointing into the domain for a particular object by querying a reference table that reference objects' domains information. The domain can be a collection of objects governed by the same geographical boundaries, object type, responsibility or the level of security or authority. The query must only return one result domain or nothing. An object that does not belong to any domain in the system will result null for the query. Therefore, the null result drops the current rule and loads the next rule from the fragmented-rule repository and then notifies the system administrator about that rule.

  The assumption that every object is located in a domain where:

$$\boxed{\begin{array}{l}\underline{Mapping\ Objects'\ Domain}\\ Domainof : object(1) \mapsto domain(D1)\\ Domainof : object(6) \mapsto domain(D3)\end{array}}$$

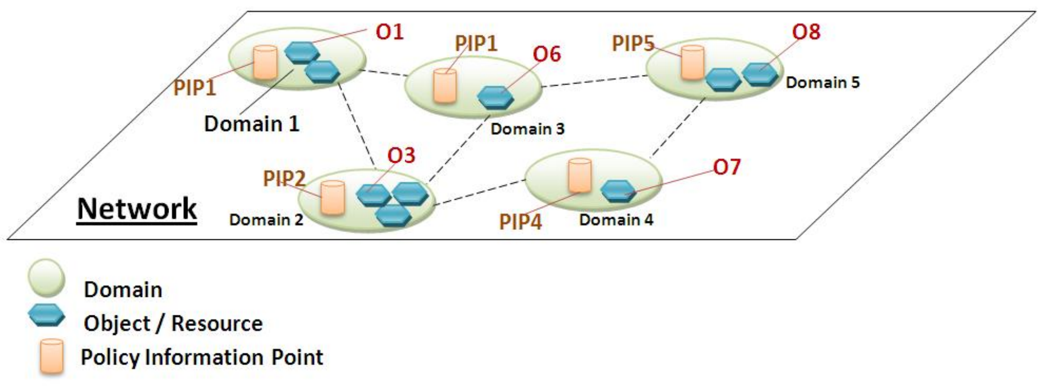


FIGURE 7.2: Network Domains.

$\underline{Domains'\ PIPs}$

$PIPof : domain(D1) \mapsto PIP(PIP1)$

$PIPof : domain(D3) \mapsto PIP(PIP3)$

Therefore, each rule in fragmented-rule repository has a single object and is therefore mapped to a single domain and also each domain maps a single Policy Information Point (PIP).

Thus, realising the object-domain for early stage is involved to determine if the condition is classified as independent or dependent condition and also recognise the Policy Information Point for a particular domain.

7.2.2.2 Refinement Independent Rule Method

In such cases, the conditions inside condition premise in question can split into branches those are based on the sort of the condition attributes. Without changing the execution flow, the subject and environment attribute conditions are represented by local condition signal, viz. finding these attributes internally (in the Policy Information Point PIP within same domain). Thus, in those cases, only enforcement process is obtained for the local condition signals and the decision, thus, these conditions referenced as default conditions in our method i.e. no coordination. The example 7.2 and 7.3 show this result.

Example 7.2: Suppose a rule that saying the subjects (*Adam* and *Alice*) can read the resource (*open.txt*) without any restrictions so the rule is expressed in SANTA as:

$$\boxed{\begin{array}{l} \mathit{allow}\ (S, \mathit{open.txt}, \mathit{read})\ \mathit{when}\ \mathit{true} \\ \mathit{where}\ S = \{\mathit{Adam}, \mathit{Alice}\} \end{array}}$$

Example 7.3: Suppose a rule that saying the subject (*Fiona*, *Fride*) can read the resource (*open.txt*) with two restrictions:- 1) *the subject age must be above 30 years.* 2) *The access must be at the first day of 2012.* So the rule is expressed in SANTA as:

$$\boxed{\begin{array}{l} \mathit{allow}\ (S, \mathit{open.txt}, \mathit{read})\ \mathit{when}\ ((S.\mathit{age} > 30)\ \mathit{and}\ (\mathit{system.date} ='' \ 01.01.2012'')) \\ \mathit{where}\ S = \{\mathit{Fiona}, \mathit{Fride}\} \end{array}}$$

Moreover, in some cases, an object that is controlled in a rule is located in the same or different domain location of the object attribute that is required in the condition. Thus, by mapping the object-domain for both controlled and required objects and comparing the domains, in one hand, if they are same, it is processed as independent condition; otherwise, it is processed as dependent condition.

Clearly, a condition that asking for the past event that happen on particular object where this object can in the same or different domain of the controlled object in the rule. Thus, by comparing the domain of the object for the past event and the domain for the object that is controlled in the rule, in one hand, if they are same, it is processed as independent condition. On the other hand, it is processed as dependent conditional.

Example 7.4: Suppose a rule that saying the subject (*Fiona*, *Fride*) can read the resource (*close.txt*) with two restrictions:- 1) *the subject has read (open.txt) resource in the past.* 2) *The (open.txt) resource size must be greater than 10000 bytes.* So the rule is expressed in SANTA as:

$$\begin{array}{l} allow\ (S, close.txt, clear)\ when \\ sometime\ done\ (S, open.txt, read)\ and\ ((open.txt).size > 10000)) \\ where\ S = \{Fiona, Fride\} \end{array}$$

We show in the example 7.4 the refinement of independent condition where the condition premise has two conditions. The first condition branch is match the event condition attribute where it is might be dependent or independent condition. In addition, the second condition branch is match the object condition attribute where it is might be dependent or independent condition. However, by comparing the object domain location for the object (*close.txt*) of the rule and the object domain location that required in the condition (*open.txt*), we assume these objects are located in a same domain, thus, the two conditions are classified as independent condition i.e. default condition.

Therefore, our model can rewrite the rule as shown below to illustrate the signaling. The symbol (@) indicates to the local signal and default condition where the PIP (*PIP5*) is presented to reference the located PIP for the attribute. The refinement of independent condition for above policy is:

$$\begin{array}{l} allow\ (S, close.txt, read)\ when \\ (@PIP5\ sometime\ done\ (S, open.txt, read))\ and \\ (@PIP5\ (open.txt).size > 10000)) \\ where\ S = \{Fiona, Fride\} \end{array}$$

Algorithm for refinement Independent Rule Method: Based on the distinction between condition premise classifications in a rule, refinement rule algorithm for independent rule is first match the condition premise (in line 6) and adds default indicator (@) for the subject and environment conditional (in line 7). Moreover, for object and past event conditional (in line 8), it checks the domain location for the object that controlled (o) in the rule and the object for each condition premise (*requiredObj*) (in line 13). Therefore, if object and past event conditional is independent, adding the default indicator into its condition with the PIP for the domain (in line 15). Finally, it copies the rule (r') into the sub-policies repository. The condition that does not include the indicator means that is dependent conditional which is refined in the next algorithm for the dependent rules. The algorithm is outlined by the following pseudo-code :

Algorithm 2 Rule Refinement (Independent Rule)

```
Require: $fragmented-rule\ repository$
 1: $Begin$
 2: $Load\ rules\ from\ fragmented-rule\ repository$
 3: for each $r\ in\ fragmented-rule\ repository$ do
 4:    $r' = r$
 5:    for each $condition\ branch\ c\ in\ r'$ do
 6:       if $c\ match\ subject\ or\ environment\ class$ then
 7:          $add\ @\ before\ c$
 8:       else
 9:          $o = the\ controlled\ object\ in\ r'$
10:          $controlledObjDomain = findDomain(o)$
11:          $requiredObj = the\ required\ object\ in\ c$
12:          $requiredObjDomain = findDomain(requiredObj)$
13:          if $controlledObjDomain = requiredObjDomain$ then
14:             $controlledObjPIP = findPIP(controlledObjDomain)$
15:             $add\ @\ controlledObjPIP\ before\ c$
16:          end if
17:       end if
18:    end for
19:    $copy\ r'\ into\ sub-policies\ repository$
20: end for
21: $End$
```

7.2.2.3 Refinement Dependent Rule Method

The object attribute condition is represented by local or remote condition signals. The remote condition signals signify to object attributes where object attribute condition is located in different domain from the object that controlled in the rule. The condition attribute values are retrieved by a coordination process, viz. communicated with other PIPs to get the values.

Moreover, according to the importance of the consideration of history-based policies [3] for the IT management in previous section, the past event conditional attribute is proposed as the object conditional attribute that can be independent or dependent based on the domain location for the object that controlled in the rule and the domain that the past event happened in.

Example 7.5: Suppose a rule that saying the subject (*Fiona*) can *read* the resource (*close.txt*) with two restrictions:- 1) *the subject has read (regulation.txt) resource in the past.* 2) *The (regulation.txt) resource size must be greater than 10000 bytes.* So the rule is expressed in SANTA as:

$$
\begin{array}{l}
\textit{allow } (S, close.txt, clear) \textit{ when} \\
(\textit{sometime done } (S, regulation.txt, read) \textit{ and } ((regulation.txt).size > 10000))) \\
\textit{where } S = \{Fiona\}
\end{array}
$$

We show in example 7.5 the refinement of dependent condition where the condition premise has two conditions. The first condition branch is match the past event condition attribute where it is can be dependent or independent condition. In addition, the second condition branch is match the object condition attribute where it is can be dependent or independent condition. However, by comparing the object domain location for the object (*close.txt*) of the rule and the object domain location that required in the condition (*regulation.txt*), we assume these objects are located in different domains, thus, the two conditions are classified as dependent condition.

Therefore, our method rewrites the rule as shown below to illustrate the signaling. The symbol (#@) indicates to the remote signal where the PIP (*PIP9*) is presented to reference the located PIP for its domain. The refinement of dependent condition for above policy is:

$$
\begin{array}{l}
\textit{allow } (S, close.txt, clear) \textit{ when} \\
(\#@PIP9 \textit{ sometime done } (S, regulation.txt, read))) \textit{ and} \\
(\#@PIP9\ (regulation.txt).size > 10000))) \\
\textit{where } S = \{Fiona\}
\end{array}
$$

The coordination process leads us to propose Push and Pull Models that involved to query the objects' or past events' attributes those are presented as remote signal in the condition premise. (Push and Pull Models are provided in Chapter 8, section 8.3)

However, the independent condition is classified as default condition where only enforcement process is obtained the local condition signal and the decision. In our decision making, the default conditions (local condition signal) are always being checked first before dependent conditions are evaluated, thus, to avoid the overhead communications. The dependent conditions attributes always be founded in different domains (different PIPs), thus, it requires coordination process to retrieve those attributes for the evaluation. Moreover, if any of independent condition attributes' are evaluated to false then no need for the coordination process (remote condition signals) to be run for evaluating dependent condition attributes'. The coordination process is run for both Pull and Push Models which are described in (Chapter 8, section 8.3, Coordination Mechanism).

Algorithm for Refinement Dependent Rule Method:

Based on the distinction between condition premise classifications in a rule, refinement rule algorithm for the dependent rule is first load rules from the sub-policies repository (in line 2). Then, it checks if the indicator (@) does not exist that means it is dependent so be refined in this algorithm (in line 5). Thus, it checks the domain location for the object that controlled in the rule and the object for each condition premise. If they are not similar (in line 10), the condition is dependent and then find a PIP for the domain (in line 11 and 12). Then, it provides the choice for the Pull or Push Models where in Pull adding remote indicator (#@) before the condition (in line 14), on the other hand , adding obligation in Push Model (in line 17). The algorithm is outlined by the following pseudo-code :

Algorithm 3 Rule Refinement (Dependent Rule)

```
Require: $sub-policies\ repository$
1:  $Begin$
2:  $Load\ rules\ from\ sub-policies\ repository$
3:  for each $r$ $in\ sub-policies\ repository$ do
4:      for each $condition\ branch\ c\ in\ r$ do
5:          if $c\ not\ include$ @ then
6:              $o = the\ controlled\ object\ in\ r$
7:              $controlledObjDomain = findDomain(o)$
8:              $requiredObj = the\ required\ object\ in\ c$
9:              $requiredObjDomain = findDomain(requiredObj)$
10:             if $controlledObjDomain\ != requiredObjDomain$ then
11:                 $controlledObjPIP = findPIP(controlledObjDomain)$
12:                 $requiredObjPIP = findPIP(requiredObjDomain)$
13:                 if $coordination\ mechanism\ is\ Pull$ then
14:                     $add\ \#@\ requiredObjPIP\ before\ c$
15:                 else
16:                     $requiredObjPDP = findPDP(requiredObjDomain)$
17:                     $add\ Oblig(requiredObjPDP, controlledObjPIP, c = true)$
18:                 end if
19:             end if
20:         end if
21:     end for
22: end for
23: $End$
```

7.3 Policy Deployment

Strictly speaking, deployment does not constitute a part of the decomposition method. However, deployment of the sub-policies is great importance to the approach as a whole. After policy decomposition has been completed, the sub-policies are distributed to their domains. Thus, for the Decentralised PBM, the generated sub-policies are being activated and the central policy is deactivated. On the other hand, for the other reason for switch to Centralised PBM approach, the central policy is activated for the DENAR that configured to be the central enforcement in case of reconfiguring a system or on longer need for Decentralised PBM. Therefore, re-propagation and recovery can be achieved. [See Chapter 5, Section 5.3.3].

However, rules deployment is based object domain location where the decomposition method eliminates the object set to control only one object in each rule. Consequently, finding the object domain location and deploying a rule in its domains' policy repository (PR) is become uncomplicated. Deployment algorithm is outlined by the following pseudo-code:

Algorithm 4 Policy Deployment

```
Require: $sub - polices\ repository$
1: $Begin$
2: $Load\ r\ from\ sub - polices\ repositry$
3: for each $r\ in\ sub - polices\ repositry$ do
4:    $obj = o\ of\ r$
5:    $domain = FindObjectLocation(obj)$ {f}inding the domain location for the object.
6:    $PR = FindPolicyRepository(domain)$ {f}inding the policy repository for the domain.
7:    if $PR = null$ then
8:       $create\ new\ Policy\ Repository\ for\ the\ domain\ and\ then\ store$ r $in\ Policy\ Repository$
9:    else
10:      $store\ r\ in\ PR$
11:   end if
12: end for
13: $End$
```

7.4 Summary

We proposed in this Chapter that the central policy is decomposed into sub-policies according to object domains and then distributed to corresponding DENARs in policy deployment. Fragmentation and refinement methods are described in SANTA syntax and algorithms for the policy decomposition. The fragmentation method described for decomposing the policy into sub-policies where each rule controls only one object. The refinement method refines the rule to indicate the location of PIPs those store conditional attributes. The local indicator shows the local condition signal to find these attributes internally in the same domain. The remote indicator shows the remote condition signal to find these attributes remote domains. In addition, the policy deployment is detailed in algorithm.

The implementation and evaluation of policy decomposition and deployment are not implemented in the DENAR prototype; therefore, not evaluated. The policy decomposition and deployment algorithms are easy to develop in any programming languages. Instead the algorithms where applied manually to decompose and refine the policies used for evaluation.

In the next Chapter, the DENAR enforcement and coordination mechanism involve the output of this Chapter where the sub-policies are decomposed and deployed in their DENARs. The next Chapter describes the DENAR enforcement and coordinate between DENARs.

Chapter 8

DENAR Enforcement and Coordination

Objectives

- Show how the network of DENARs interacts to enforce policies.
- Provide a coordination mechanism for DENAR enforcement.
- Identify the DENAR properties.

8.1 Introduction

This Chapter presents the DENAR enforcement and coordination models. Enforcing security policy in distributed PDPs can not meet the security access control requirements without considering the need of coordination between these distributed PDPs. The coordination need is clear in involving History Based Policy (HBP) in Decentralised PBM for access control decision. For enforcing History Based Policy (HBP), DENAR is required to have the ability to observe events and previous decisions in order to make a current decision which are in some cases being enforced by other DENARs. Thus, the access control decision making can be improved by identifying a local domain decision and remote domain decision.

The remainder of this Chapter is organised as follows. Section 8.2 describes the policy enforcement model in DENAR. Subsection 8.2.1 models the local domain decision where the subsection 8.2.2 models the remote domain decision. Section 8.3 describes DENAR coordination mechanisms. Subsection 8.3.1 describes a Pull Model decision where the subsection 8.3.2 describes Push Model those fulfill the coordination needs between collaborative DENARs). Section 8.4 highlights the DENAR's properties and show how these are achieved (in particular synchronisation, concurrency and security). Finally, section 8.5 summarises the Chapter.

8.2 Policy Enforcement in DENAR

In this section, we present a simple scenario that shows the relevant dependent (dynamic policy) and independent policy (static policy) enforcement in our Distributed Policy Enforcements Architecture (DENAR) and its components interactions to make a policy decision.

There are two types of policy decision which are local domain decision and remote domain decision. The local domain decision details the interaction between the DENAR components where no need for communicate other DENARs in the network. The remote domain decision details the interaction between the DENAR components themselves and other DENARs components where communicating other DENARs in the network is essential to make policy decisions. During the authorisation process, the DENAR receives an authorisation request from a PEP. Then, a PDP retrieves the policy for that request from the local PR. Thus, the local domain decision or remote domain decision can be made based on the rule type (dependent or independent rule).

Sequence diagrams are used to explain both local decision where one DENAR make the decision and also collaborative DENARs in the network those share the decision.

8.2.1 Local Domain Decision

There are four cases to make a decision locally based on the independent rules which are illustrated below.

Case1: when a rule has no restrictions (no condition premise). [See Chapter 6, Section 6.2.2 for more details]

Example 8.1: Suppose a rule that saying: subject (*Adam* and *Alice*) can *read* the resource (*open.txt*) without any restrictions. The rule is:

$$\begin{array}{l} allow\ (S, open.txt, read)\ when\ true \\ where\ S = \{Adam, Alice\} \end{array}$$

This rule is enforced locally by one DENAR because there is no condition premise.

Case2: a rule has restrictions (conditional premise) but the rule is an independent rule where restrictions on subjects and environments only. [See Chapter 6, Section 6.2.2 for more details].

Example 8.2: Suppose a rule that saying: subjects (*Fiona*, *Fride*) can *read* the resource (*open.txt*) with two restrictions:- 1) *the subject age must be above 30 years.* 2) *The access must be at the first day of 2012.* The rule is:

$$\begin{array}{l} allow\ (S, open.txt, read)\ when\ ((S.age > 30)\ and\ (system.date =' 01.01.2012')) \\ where\ S = \{Fiona, Fride\} \end{array}$$

Case3: a rule has restrictions (conditional premise) and the rule is independent rule where restrictions on objects, past events and hypothetical rule and those conditional attributes can be found in local PIP, i.e. in the same DENAR. [See Chapter 8, Section 8.3.2 for more details].

Example 8.3: Suppose a rule that saying the subject (*Fiona*, *Fride*) can *read* the resource (*close.txt*) with two restrictions:- 1) *the subject has read (open.txt) resource in the past.* 2) *The (open.txt) resource size must be greater than 10000 bytes.* The object domain location for the object (*close.txt*) of the rule and the object domain location that evaluated in the condition (*open.txt*) are located in a same domain, thus, these conditional attributes be founded in local PIP (let say *PIP1*). The rule is:

$$\begin{array}{l} allow\ (S, close.txt, read)\ when \\ (@PIP1\ sometime\ done\ (S, open.txt, read))\ and (@PIP1\ (open.txt).size > 10000) \\ where\ S = \{Fiona, Fride\} \end{array}$$

Case4: when a rule has no restriction but indicate to an obligation after the decision made and the action is taking place. The decision value must be stored in local PIP where this value is needed for other future decision in same DENAR (let say is enforced and needed for PDP1 so stored in PIP1). This sort of enforcement is detailed in Push Model. [See Chapter 6, Section 8.2.1.2 for more details]

Example 8.4: Suppose a rule that saying the subject (Fride) can read the resource (open.txt) without any restrictions. The rule is:

$$\begin{array}{l} \underline{rule1.1:} \\ allow\ (S, open.txt, read)\ when\ true \\ where\ S = \{Fride\} \end{array}$$

$$\begin{array}{l} \underline{rule1.2:} \\ oblig\ (PDP1, PIP1, S.read.open = true)\ when \\ always\ done\ (S, open.txt, read) \\ where\ S = \{Fride\} \end{array}$$

The below sequence diagram in Figure 8.1 is used to explain local decision where one DENAR can make the decision locally.

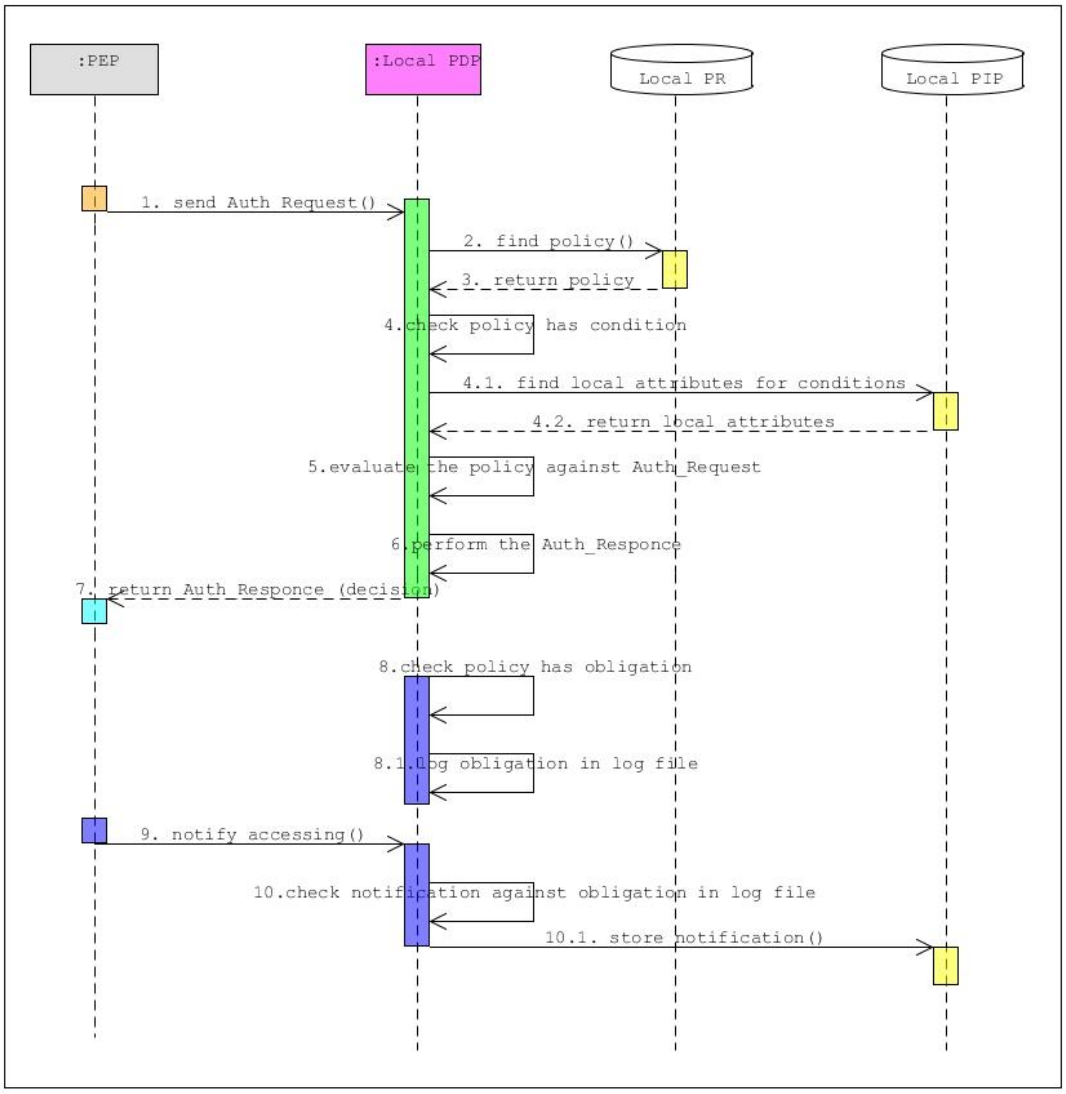

:PEP
:Local PDP
Local PR
Local PIP
1. send Auth Request()
2. find policy()
3. return policy
4.check policy has condition
4.1. find local attributes for conditions
4.2. return local attributes
5.evaluate the policy against Auth_Request
6.perform the Auth_Responce
7. return Auth_Responce (decision)
8.check policy has obligation
8.1.log obligation in log file
9. notify accessing()
10.check notification against obligation in log file
10.1. store notification()

FIGURE 8.1: Local Decision Sequence Diagram.

8.2.2 Remote Domain Decision

There are three cases to make a decision remotely based on the dependent rules which are illustrated below.

Case1: a rule has restrictions (conditional premise) where the main rule is based on another hypothetical rule that located in different DENAR (in a remote PR). [See Chapter 6, Section 6.2.2 for more details].

Example 8.5: Suppose a rule that saying the subject (*Fiona*, *Fride*) can read the resource (*open.txt*) with one restriction:-

the subject has authorisation to read the resource (*open.txt*) if and only if he/she has the authorisation to read the resource (*regulation.txt*). In this case, the resource (*regulation.txt*) is located in different domain and the hypothetical rule is deployed in remote PR and must evaluated by the remote PDP in different DENAR. The rule is:

$\underline{rule1:}$
$allow\ (S, open.txt, read)\ when\ allow\ (S, regulation.txt, read)$
$where\ S = \{Adam, Alice\}$

Case2: a rule has restrictions (conditional premise) and the rule is a dependent rule where restrictions on objects and past events and those conditional attributes can be found in Remote PIPs, i.e. in the different DENAR. This sort of enforcement is detailed in the Pull Model. [See Chapter 8, Section for more details 8.3.1].

Example 8.6: Suppose a rule that saying: subjects (*Fiona*, *Fride*) can *delete* the resource (*close.txt*) with two restrictions:- 1) *the subject has read (regulation.txt) resource in the past.* 2) *The (regulation.txt) resource size must be greater than 10000 bytes.* The object domain location for the object (*close.txt*) of the rule and the object domain location that evaluated in the condition (*regulation.txt*) are located in a different domain, thus, these conditional attributes be founded in remote PIPs (let say *PIP2*). The rule is:

$allow\ (S, close.txt, delete)\ when$
$(\#@PIP2\ sometime\ done\ (S, regulation.txt, read)))\ and$
$(\#@PIP2\ (regulation.txt).size > 10000)))$
$where\ S = \{Fiona, Fride\}$

Case3: when a rule has no restriction but indicate an obligation after the decision made and the action is taking plase. The decision must be stored in remote PIPs where this value is needed for other future decision in different DENARs (let say is enforced by *PDP2* and needed for *PDP1* so stored in *PIP1*). This sort of enforcement is detailed in Push Model. [See Section for more details 8.3.2]

Example 8.7: Suppose a rule that saying the subject (*Fride*) can read the resource (*regulation.txt*) without any restrictions. The rule is:

$\underline{rule1.1:}$
$allow\ (S, regulation.txt, read)\ when\ true$
$where\ S = \{Fride\}$

$\underline{rule1.2:}$
$oblig\ (PDP2, PIP1, S.read.regulation = true)\ when$
$always\ done\ (S, regulation.txt, read)$
$where\ S = \{Fride\}$

The below sequence diagram in Figure 8.2 is used to explain how collaborative DENARs share information to make a decision.

8.3 DENAR Coordination Mechanisms

Realisation of IT management processes by means of policy rules relies on local condition and coordination signals to acquire a particular decision. Hence, some features of the coordination signal that is implicit to the local condition signal definition need to be treated in a special manner. Thus, we propose Push and Pull Models to the coordination signal that run to check the value located in different DENARs. However, the Push and Pull Models are the coordination signals that run when the local condition signal is running to communicate with remote PIP(s) (in different DENARs) to check the value that needed to evaluate the condition.

However, to control rule enforcement for both techniques, the termination of the action must be detected, and continuation of this enforcement must be facilitated either the coordination signal is succeed to get the value or fail.

As these are not "local condition" signal but runtime-dependent messages originating in the policy enforcement infrastructure, they are called coordination signal.

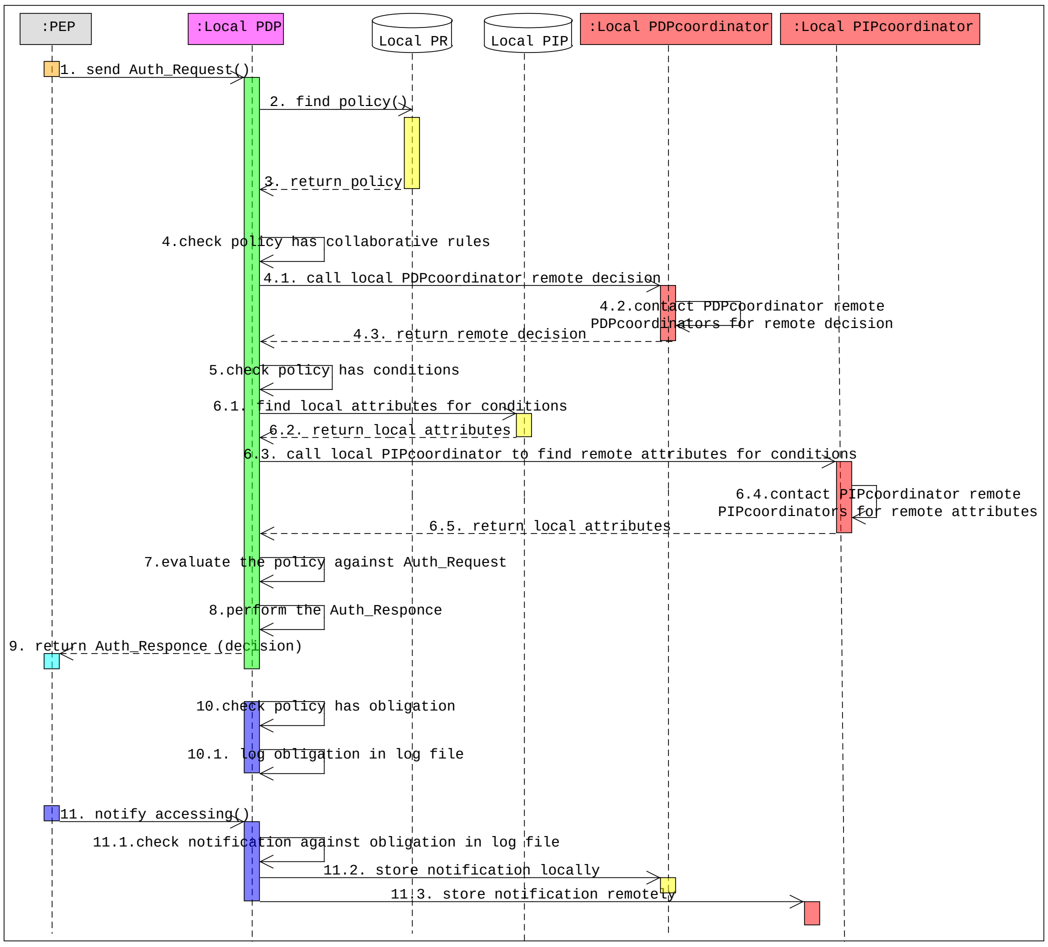

:PEP
:Local PDP
Local PR
Local PIP
:Local PDPcoordinator
:Local PIPcoordinator
1. send Auth_Request()
2. find policy()
3. return policy
4.check policy has collaborative rules
4.1. call local PDPcoordinator remote decision
4.2.contact PDPcoordinator remote PDPcoordinators for remote decision
4.3. return remote decision
5.check policy has conditions
6.1. find local attributes for conditions
6.2. return local attributes
6.3. call local PIPcoordinator to find remote attributes for conditions
6.4.contact PIPcoordinator remote PIPcoordinators for remote attributes
6.5. return local attributes
7.evaluate the policy against Auth_Request
8.perform the Auth_Responce
9. return Auth_Responce (decision)
10.check policy has obligation
10.1. log obligation in log file
11. notify accessing()
11.1.check notification against obligation in log file
11.2. store notification locally
11.3. store notification remotely

FIGURE 8.2: Remote Decision Sequence Diagram.

Hence, an remote signal is used to signify the dependent condition in a rule. The following Figure 8.3 and Figure 8.4 depict the difference between the local condition signal and coordination signal where the refinement rules involves remote signal to decouple signal partitions.

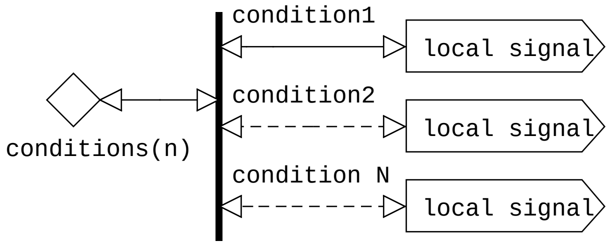


FIGURE 8.3: Local Condition Signal.

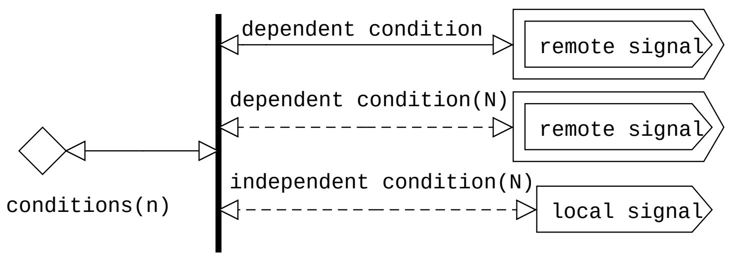


FIGURE 8.4: Coordination Signal.

This is achieved by allowing the conditional attributes to have reference that indicate to the local or remote signal where that based on the condition classifications and attribute in question. Moreover, the conditional attributes that have remote signal always use a Pull Model. However, the Push Model can be presented as obligation that is added in a rule.

The acknowledgement is required to provide coordination for either Push or Pull Model to request or provide some values those are evaluated in condition where may be distributed over several PIPs. In essence, in the Push Model, the coordination mechanism must be able to detect the execution of an action as well as sending the acknowledgement of this execution to DENARs that needs this value or associates with their enforcement. For the Pull Model, the coordination mechanism must be able to find the required attributes located in several PIPs, thus, one or more signals are run and then they are evaluated in the condition premise. Therefore, the execution of action is being made when these values are returned back and evaluated.

The policy decision and enforcement process are based on information that originates inside or outside the process by the local condition or coordination signal. Thus, this process and the policy action must be connected.

8.3.1 Pull Model

The Pull Model is used by a PDP to ask remote PIPs (those not located in the same DENAR of the PDP) about a value that required in dependent condition to complete the decision for a particular rule. By coordination mechanisms, remote signal must be introduced in a condition premise for dependent conditions where the remote signal is introduced in the rule refinement method (See Chapter 7, Section 7.2.2). This signal can be issued to mark the completion of requesting and receiving all dependent conditional attributes, i.e. all conditional attribute must be received and evaluated to true and then the action can be executed otherwise the action cannot be executed.

Example 8.8 Suppose that there are dependent rules:

The subject (*Fride*) can *read* the resource (*regulation.txt*). In addition, the subject (*Fride*) can *delete* the resource (*close.txt*) only if he has read the resource (*regulation.txt*). According our refinement method, for *rule1*, the object (*regulation.txt*) is located in (*domain1*), thus, *rule1* should deployed into the policy repository (*PR1*) for the (*DENAR1*) where it is enforced by (*PDP1*). Also, when the *rule1* enforced and the subject (*Fride*) read the resource (*regulation.txt*), the decision value is stored in (*PIP1*). In addition, for *rule2*, the object (*close.txt*) is located in (*domain2*), thus, *rule2* should deployed into the policy repository (*PR2*) for the (*DENAR2*) where it is enforced by (*PDP2*). However, *rule2* decision cannot made without evaluating its condition where the attribute value is located in different domain.

This *rule1* and *rule2* in Pull Model can be formalised in SANTA as:

$\underline{rule1:}$
$allow\ (S, regulation.txt, read)\ when\ true$
$where\ S = \{Fiona\}$

$\underline{rule2:}$
$allow\ (S, close.txt, delete)\ when\ sometime\ done\ (S, regulation.txt, read)$
$where\ S = \{Fiona\}$

In Figure 8.5, during the enforcement of *rule2* by (*PDP2*), the reference (indicator) in its condition premise shows that the value for condition context is existed in (*PIP1*). The coordination signal is run (in step 3) to request the past event attribute by ($@PIP1$) indicator. As the result of *rule1* enforcement in the past (in step 1), the *PIPcoordinator* for *DENAR2* runs remote coordination signal to return the decision value that made by (*PDP1*) in the past to (*PDP2*) in step 4.

Actually, (*PDP2*) hold the decision till the requested value is returned in step 2. Then, the action is executed if and only if the condition is evaluated to true.

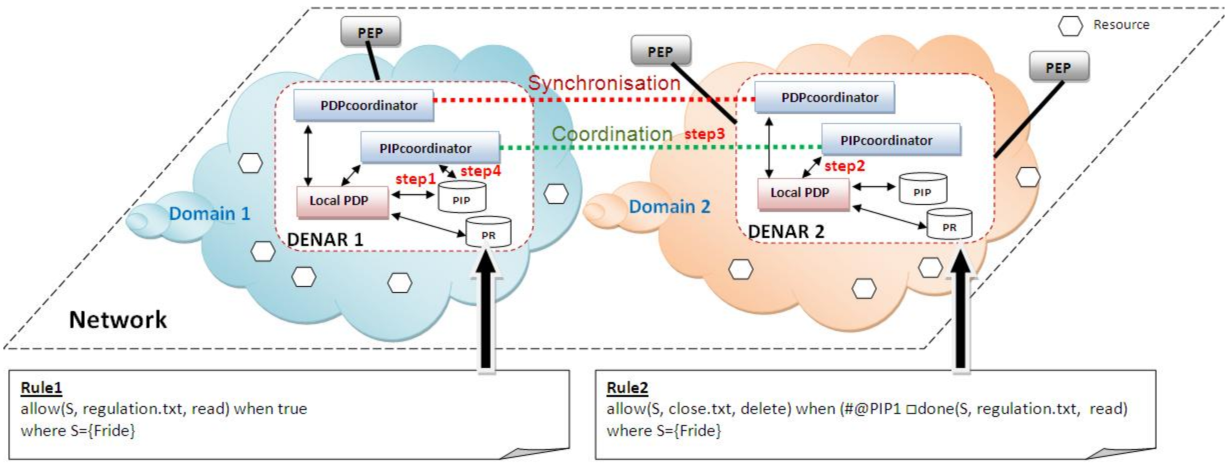


Figure 8.5: Pull Model.

8.3.2 Push Model

The Push Model is used by a PDP to send a decision value that might be requesting in future by other PDPs to save the overhead of communications between PDPs. Thus, the obligation should be concerned to all such rules to coordinate this value with other rules. To account for all remote signals in the Push communications technique with other PDPs, either any policy language can support the obligation be using (if available), or a synthetic, coordination signal must be introduced. This signal can be issued to mark the completion of enforcing a rule, i.e. the action has to be executed and then all values have to be sent to all PIPs those their PDPs need the result of this event.

<u>Example 8.9</u>

By going back to the same example 8.7 scenario, the formalisation in SANTA in the Push Model will be change to:

The *rule1* is remain as its in *rule1.1* and also a new obligation rule is produced in *rule1.2* that saying if the action of *rule1.1* is performed then notify *PIP2* about this decision where the *PDP2* will need it in the future.

$\underline{rule1.1:}$
$allow\ (S, regulation.txt, read)\ when\ true$
$where\ S = \{Fride\}$

$\underline{rule1.2:}$
$oblig\ (PDP1, PIP2, S.read.regulation = true)\ when$
$done\ (S, regulation.txt, read)$
$where\ S = \{Fride\}$

$\underline{rule2:}$
$allow\ (S, close.txt, delete)\ when$
$(@PIP2\ sometime\ done\ (S, regulation.txt, read))$
$where\ S = \{Fride\}$

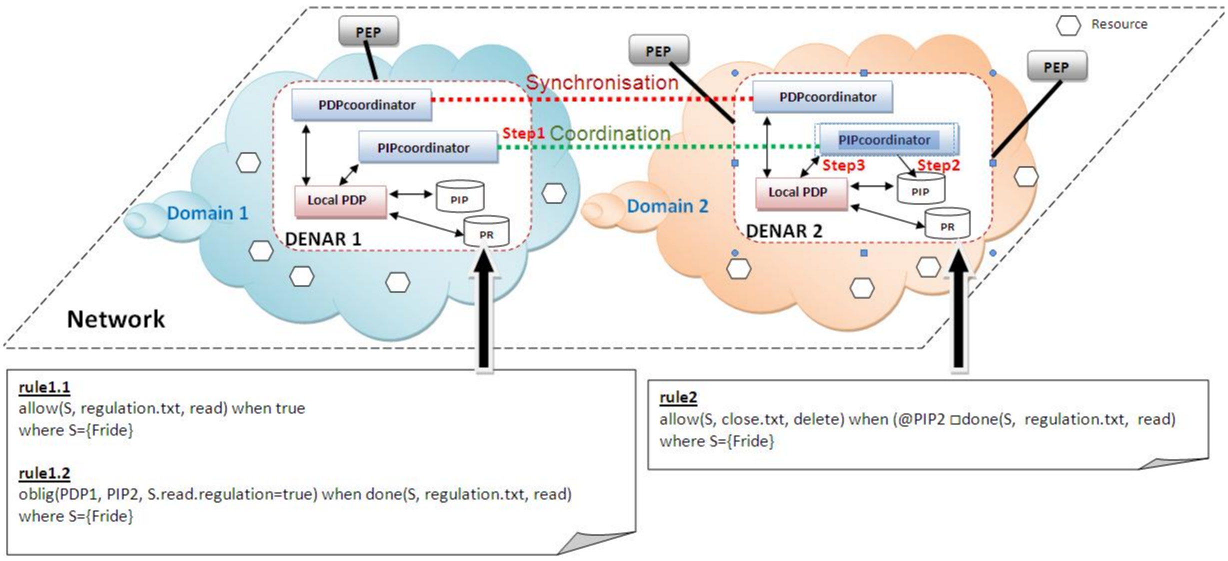
In Figure 8.6, *rule1* is enforced by *PDP1* and then *PDP1* obligates its decision to the *PIP2* by a coordination signal via PIPcoordinators (in step 1). The PIPcoordinator for *DENAR2* stores the value in *PIP2* (in step 2). Therefore, when *rule2* is enforcing, the *PDP2* can find the value locally in *PIP2* (in step 3) so that there is no need to communicate the *DENAR1* at the decision time of *rule2*.

8.4 DENARs Properties

DENAR's properties are introduced in this section. Synchronisation and concurrency in enforcement describe the coordination and collaboration between DENARs. Finally, the secrecy property describes the secure technologies that can be involved in the DENAR.

8.4.1 DENAR's Coordination and Synchronisation

As with all distributed systems, synchronisation and concurrency are major considerations in the distributed enforcements design. The two terms are overlapping and are almost synonymous. Synchronisation refers to the simultaneous access and



```
rule1.1
allow(S, regulation.txt, read) when true
where S={Fride}

rule1.2
oblig(PDP1, PIP2, S.read.regulation=true) when done(S, regulation.txt, read)
where S={Fride}
```

```
rule2
allow(S, close.txt, delete) when (@PIP2 □done(S,  regulation.txt,  read)
where S={Fride}
```

Figure 8.6: Push Model.

updating of shared data between two threads or processes. Concurrency on the other hand, refers to the efficiency with which a thread or process handles multiple simultaneous requests made from one access control request. The Distributed Policy Enforcements Architecture (DENAR) is designed to be synchronised and concurrent. [See Section 8.4.2 for concurrency]

When an access to a specified resource is requested, a policy that retrieved by the PDP may first consult evaluation for object attributes or past decision against the current values before make an access decision as restrictions (conditions).

Object and past decision attributes may stored in centralised repository (PIP), thus, all PDPs access the repository to retrieve the conditional attributes value but this idea in fact make the enforcement still centralised. On the other hand, distributed repositories (PIPs) may be designed. In the first case, the conditional attributes may be replicated across all PIPs on the network which would necessitate the PDPs to continuously coordinate the values of their resources and past decision attributes with each other. In other case, the conditional attributes may be only deployed into a particular PIP that being configured for a particulate domain. The last case is being more efficient for less communication and resources capacity.

However at this point, the conditional attributes those located in the local PIP (at DENARs' PDP) are available to the PDP but other conditional attributes are not. Therefore, we design PIPcoordinator component that instruct a PDP to retrieve the remote conditional attributes from other PIPs. Moreover, the component coordinates these values, where a decision is based on originating of these values between PIPcoordinators in DENARs, viz. a sub-policy can be located in a domain for which decision are being derived from a subject rights on other objects located in another domain or decisions that have been made in the past by other PDPs. Thus, the coordination can be done only by the PIPcoordinator component. Remotely retrieving or coordinate conditional attribute values for PDPs proposed to be done through Push and Pull Models those are designed in (Section 8.3.1). Additionally, locally and remotely storing and updating of these values in PIPs are proposed to be done through an obligation. Obligations are the set of actions which a PEP is required to perform either before it has executed a request, after it has executed a request or along with request execution. In this case, an obligation updates a proper conditional attribute value once access has been granted to a resource.

8.4.2 DENAR's Concurrency

As we mentioned in previous section that concurrency refers to the efficiency with which a thread or process handles multiple simultaneous requests made from one access control request. Our Distributed Policy Enforcements Architecture (DENAR) is designed to be concurrent.

When an access to a specified resource is requested, a policy that is retrieved by the PDP may consult for collaborative decision (e.g. hypothetical rule, see Chapter 6, Section 6.2.2) for more than one decisions those made in multiple PDPs. In this case, the PDP that received the request from a PEP is being the main decision point to make the final decision for the main request and other PDPs those involved make other requests. Therefore, we design PDPcoordinator component that instruct a PDP to communicate the remote PDPs for shared decision. Moreover, the component act as requester for access decision where a main decision is based on multiple decisions those made in remote PDPs and originating to the PDPcoordinator component. Thus, the decision concurrency can be done only by the PDPcoordinator component.

8.4.3 DENAR's Security

Each DENAR must be secure and also the communication between DENARs is the network must be secure. The sub-policies in all PRs and conditional attributes in PIPs may be encrypted. In addition, the communication between DENAR components can be secured (e.g. using LDAP, Lightweight Directory Access Protocol for retrieving policy from PRs and retrieving conditional attributes from PIPs). Finally, for the communication between PEPs and PDPs, the network administrator can implement a secure tunnel (e.g. using COPS protocol). Though COPS provides for message level security for message integrity and protection, existing protocols for security such as IPSEC [14] or TLS [35] may optionally be utilised as a security measure to secure communications between the PDP and the PEP.

The COPS protocol also specifies certain requirements for fault tolerance which is essential from a distributed application point of view. In order to achieve this, both the PEP and the PDP are expected to communicate with each other periodically using connection Keep Alive messages. If the PEP detects failure in the PDP it should try to reconnect to a backup PDP and in the absence of a connection, should revert to local decision making in order to minimise disruption. Once a connection is re-established with a PDP, the PEP should synchronise all decisions taken in its absence with it. Alternatively, the PDP may request this synchronisation. It is desirable for the PEP to cache details of previous request decisions so that in the event of the PDP failing it may continue to use these previous decisions for a limited period of time till the PDP becomes active again.

However, hacking any DENAR does not affect other DENARs where each DENAR access to its PR and its PIP.

8.5 Summary

This Chapter presented the enforcement and coordination models in DENAR. Designing the enforcement and coordination models, using the policy decomposition and the Distributed Policy Enforcements Architecture (DENAR), that enable enforcement of static and dynamic policies is one of the research objectives. Details of their design and implementation, together with algorithm are given.

Push and Pull models are proposed and detailed that provide the coordination and collaboration between DENARs. The access control decision making is improved by identifying local domain decision and remote domain decision. In addition, this Chapter showed DENAR's properties. Performance is improved by localised and remote decision making, saving time during enforcement.

In Chapter 9 (section 9.3.3), the implementation of the enforcement and coordination models is provided. In Chapter 10, the evaluation showed that the Distributed Enforcements Architecture (DENAR) is feasible to the enforcement of both static and dynamic policies. The Push model was not evaluated in this research because it would required design of the Policy Enforcement Point (PEP) to notify the PDP for enforces an obligation which was not presented in the scope of this thesis.

The following Chapter shows the Distributed Enforcements Architecture (DENAR) prototype to evaluate the research objectives.

Chapter 9

DENAR Prototype

Objectives

- Develop a network of DENARs.
- Implement the DENAR prototype.
- Support the evaluation of the research.

9.1 Introduction

The Distributed Enforcements Architecture (DENAR) prototype is developed to test the security, performance, manageability and resilience of DENAR to enforce access control policy. It consists of two parts; the DENAR network lab, software design and implementation modules. It also describes how these interact to achieve the research objectives.

The DENARs network lab at a technical level describes how the DENARs network is created and configured. The second part describes how the XACML architecture and the Sun XACML library are implemented in DENAR. In addition, it describes how the server-client application that is simulating usage scenarios works and how the DENAR (server) deals with the users (clients) request. Finally, enforcement and coordination are implemented where DENARs can seamlessly handle all communications and synchronisation among the various DENAR components connected in the network.

The rest of this Chapter is structured as follows. Section 9.2 introduces DENARs network lab. Section 9.2.1 describes the design of DENARs network topologies in Netkit and DENARs network domains and configuration is described in section 9.2.2. Section 9.3 describes the software design and implementation of DENAR. Subsection 9.3.1 provides XACML architecture. In subsection 9.3.2, the server-client application describes how the DENAR (server) deals with the users request (client). Finally, subsection 9.3.3 describes all classes in the DENAR for the enforcement and coordination in DENARs network. Finally, section 9.4 summarises the Chapter.

9.2 DENARs Network Lab

Setting up a network requires various equipment such as routers, switches, cables, and machines. Facilities and utilities, installing and maintaining all these devices in one lab is costly.

To avoid this expenditure and to create additional flexibility in the set up, this research opted to simulate these devices and save time, effort and money. Network simulation software is one solution to imitate the working of computers network.

Netkit [81] is free and open source networks simulator. Netkit [81] was developed by Roma Tre University and the Linux User Group LUG Roma. In Netkit, a virtual network lab can be created by writing simple configuration files that describe the topology and configurations of that network to be emulated on a single host machine. The topology is the layout pattern that shows how the machines are connected with each other in that network.

The DENARs network lab is created using Netkit package 2.8. This software consists of; Netkit core package, Netkit file system package and Netkit kernel package. These packages were obtained from (wiki.netkit.org/index.php/DownloadOfficial) and installed on Ubuntu (Debian GUN/Linux) operating system. The Installation instructions can be found in: (wiki.netkit.org/download/netkit/INSTALL).

We choose Netkit as it meets the requirements for our prototype implementation and can model the network at various levels of detail, allowing to run the components of DAENAR implemented in Java.

9.2.1 Building DENAR Topologies in Netkit

DENARs network is divided into sub-networks or LANs (Local Area Network). LAN refers to a group of computers connected together so that users can share some resources such as files, printers etc. In this network, each LAN comprises one DENAR (server), multiple resources and client machines. When a client requests accessing to a resource, a (DENAR) enforces a authorisation policy for the authorisation decision to the resource before performing the access action.

Figure 9.1 illustrates LAN1 in DENARs network. In this LAN, DENAR1 is the server machine and pc1-1 is a client machine. The machines are connected with each other using central router. This type layout represents a star topology. In this topology, each machine is connected with the router with a cable. Failure in any cable or any machine except the main router will not take down the entire LAN [89].

In Netkit simulation software, configuring DENARs network topology requires creating a file that describes the topology. The file that describes the topology of the entire network lab is called (lab.conf) and it is provided in (Appendix A.1.2). For each virtual machine in Netkit, there is an entry of two lines written in that

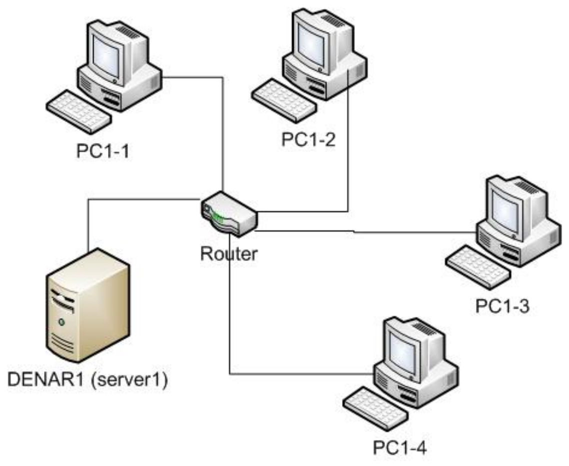


FIGURE 9.1: LAN1 (Local Area Network) in DENARs Network.

file. For example, entry;

```
PC1-1[mem]=128
PC1-1 [0]=lan1
```

LISTING 9.1: Client (PC1-1) and LAN (LAN1) Configuration in Netkit

The entry creates a virtual machine PC1-1 with network interface eth0, and allocates 128MB of memory for that machine to operate. Netkit also creates a virtual router for that LAN and connects that machine to one port on that router.

In the DENARs network, the set up of virtual machines are provided in the "virtual machine".stratup files that echo the virtual machines and connect them to the network.

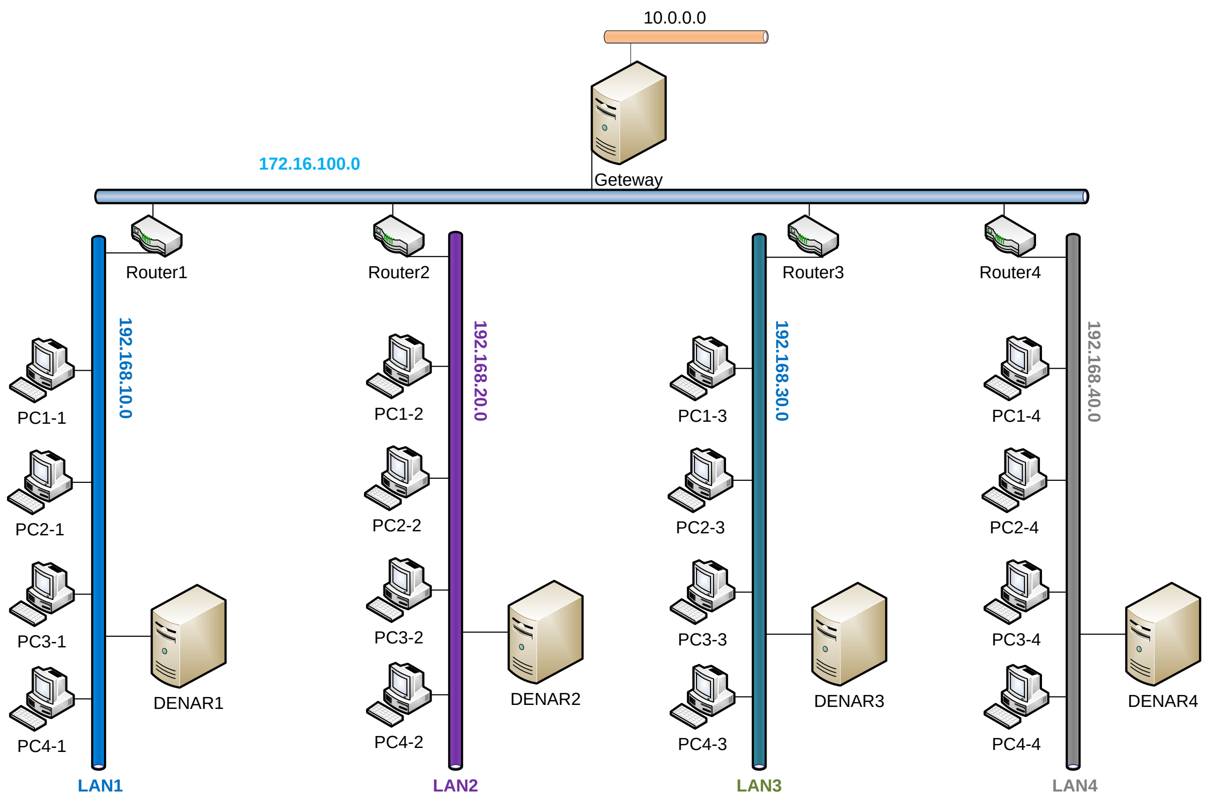
```
pc1-1.startup file
--------------------
echo 'send host-name "pc1-1";' >> /etc/dhcp3/dhclient.conf
dhclient eth

and so on for other PCs
```

LISTING 9.2: Virtual machine (PC1-1) set up in Netkit

9.2.2 DENARs Network Configurations

The client machines IP address can be configured manually. For each machine, IP address is written in machines' (.startup) file. This is not the best solution as it takes time. The network lab is designed to be scalable so that it can be expanded with the most minimum configuration files editing. Using Dynamic Host Configuration Protocol (DHCP) can achieve that goal and provides more realistic and professionalism to the network lab. It automatically assigns dynamic IP address to the client machines whenever they connect to a DENAR (server).

For each DENAR machine, the DHCP configuration file is (dhcpd.conf). The DHCP configures to allocates IP addresses within the range of IP addresses in a LAN. These address ranges can be modified at any time to accommodate any increase in the lab clients. Along with IP address, DHCP informs the clients about their default gateway and name server. It offers an IP address for any machine requesting IP address. If the machine accepts that offer, DHCP becomes responsible for maintain that IP address for that machine as long as it is connected to it [93]. Figure 9.2 illustrates the DENARs network that will be used in Chapter 10 to evaluate the efficiency of DENAR.



—

FIGURE 9.2: The DENARs Network.

9.2.2.1 Domain Configuration

Domain Name System (DNS) provides naming service that translates IP addresses to names, and vice versa [72]. For each LAN, there is a Name Server (NS) that runs name series. Each LAN represents a domain or zone that can be represented hierarchly. Figure 9.3 illustrates the hierarchy domain structure of the DENARs network lab:

- .project represents the top domain which is served by the Gateway.
- .d1.project represents the subdomain (domain1) which is served by Router1.
- .d2.project represents the subdomain (domain2) which is served by Router2.
- .d3.project represents the subdomain (domain3) which is served by Router3.
- .d4.project represents the subdomain (domain4) which is served by Router4.

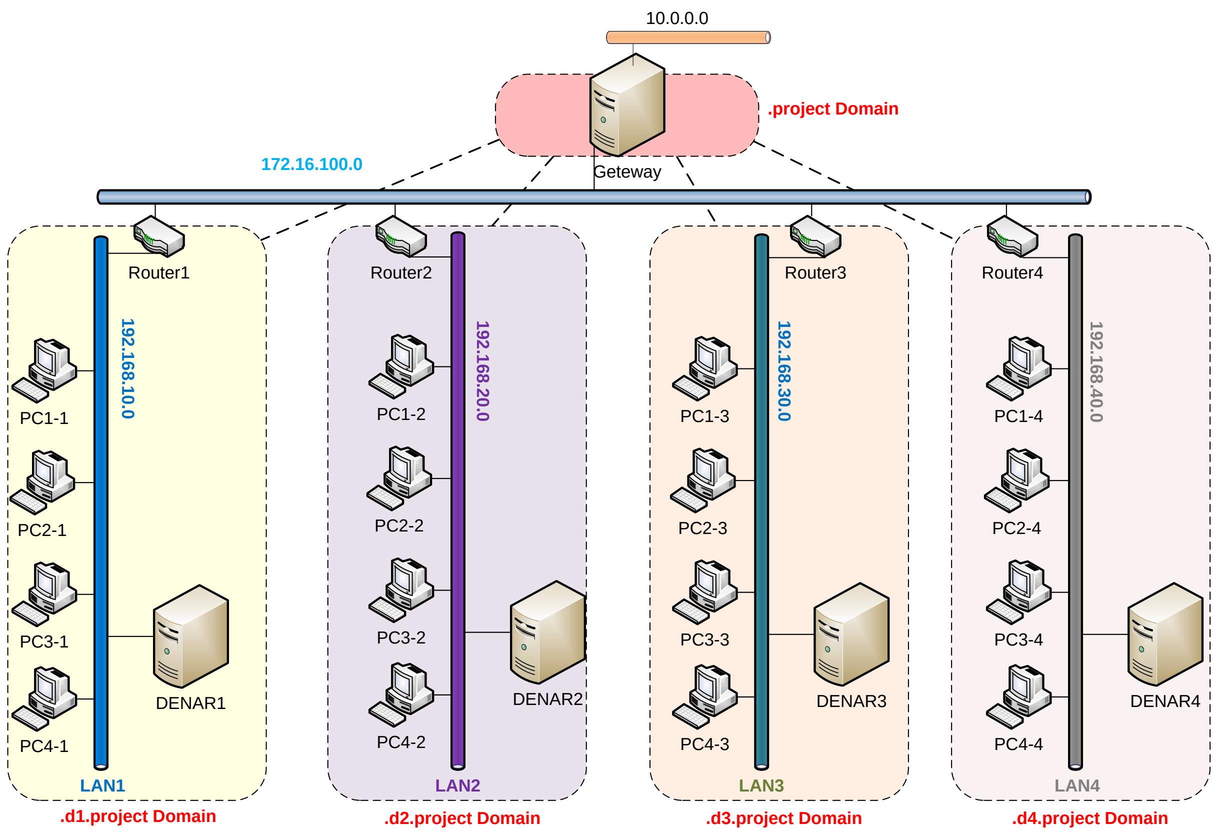


FIGURE 9.3: The DENARs Network domains.

Netkit configures and manages network domains by three files which are named.conf.local, named.conf.option and db.root. [See Appendix A.1.1 for those files.]

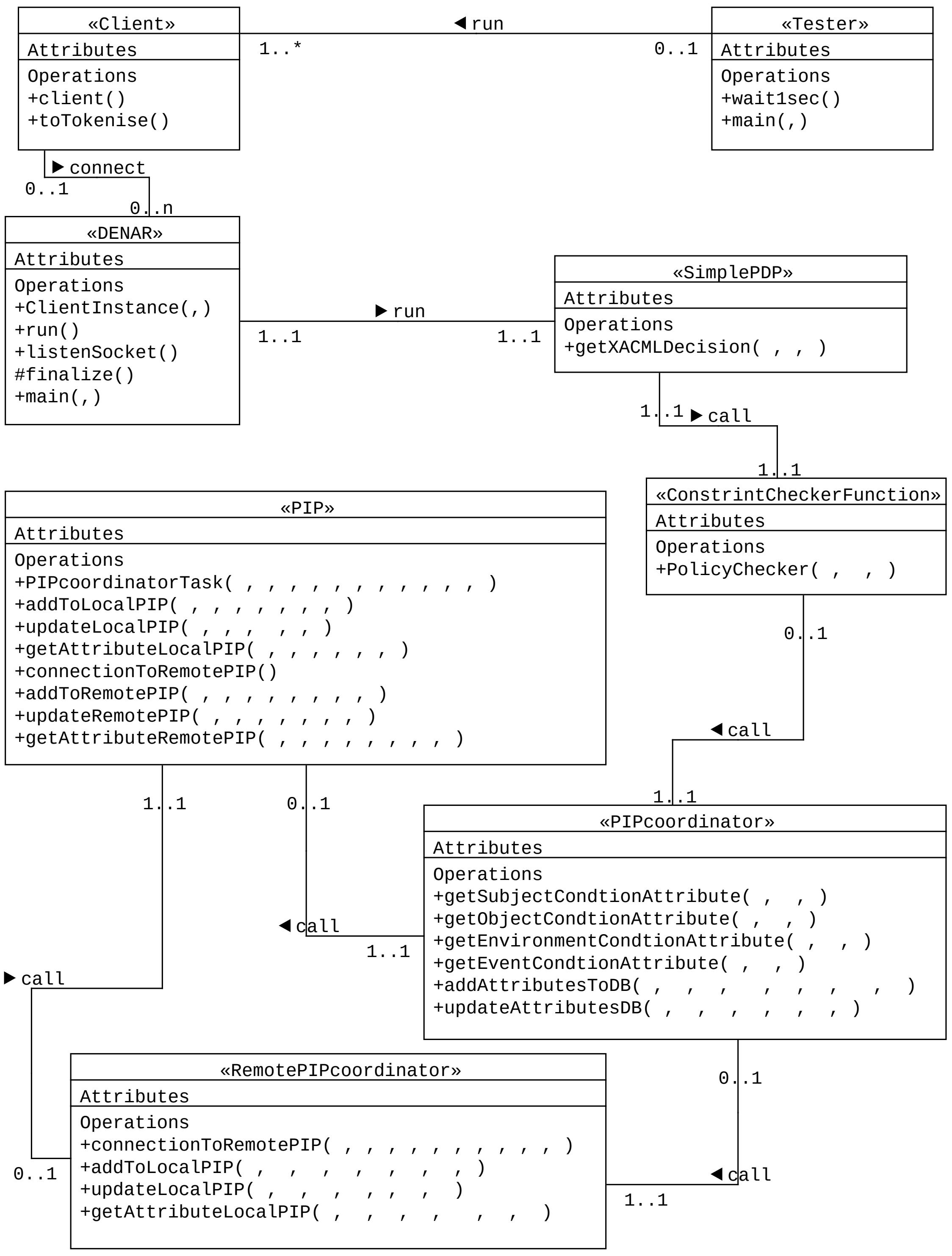
9.3 Software Design and Implementation of DENAR

In this section, DENAR software is presented which involves the XACML framework and policy language [76] in order to integrate a network of stateful, connected DENARs which involve multiple PDPs, with XACML policy language [76] to enforce a security policy.

While the network lab represents the infrastructure of DENARs network, we choose XACML policy language for our implementation. The Sun XACML implementation is well known in business and widely used. Given the extensibility of the DENARs prototype, it can possibly be implemented in far more complex and industry relevant applications. The XACML policy is a stateful policy language that is a subset of the SANTA language in terms of expressiveness.

The DENARs software is based on Sun's XACML implementation [92] of the Oasis Markup language (OASIS) [76]. Sun's XACML Implementation [92] in Java library of the OASIS standard allows a user to easily program a stateless standalone PDP which accepts requests encoded in the XML language and provides a response based on the conditions specified in a relevant policy. This library implements all the standard XACML features specified in the OASIS specification. The programmers guide to the XACML implementation [92] provides an in-depth account of how to work with and set up the XACML library. The core PDP used in DENAR implementation is adapted from Sun's sample implementation package (which also contains the sun.xacml library) and adapted for use with the other DENARs classes.

The class diagram in Figure 9.4 shows all classes in the system and the relationships between these classes where all classes are presented and described in below subsections.



«Client»
Attributes
Operations
+client()
+toTokenise()
◀ run
1..*
0..1
«Tester»
Attributes
Operations
+wait1sec()
+main(,)
▶ connect
0..1
0..n
«DENAR»
Attributes
Operations
+ClientInstance(,)
+run()
+listenSocket()
#finalize()
+main(,)
▶ run
1..1
1..1
«SimplePDP»
Attributes
Operations
+getXACMLDecision(, ,)
1..1
▶ call
1..1
«ConstrintCheckerFunction»
Attributes
Operations
+PolicyChecker(, ,)
0..1
◀ call
1..1
«PIP»
Attributes
Operations
+PIPcoordinatorTask(, , , , , , , , , , ,)
+addToLocalPIP(, , , , , , ,)
+updateLocalPIP(, , , , ,)
+getAttributeLocalPIP(, , , , , ,)
+connectionToRemotePIP()
+addToRemotePIP(, , , , , , , ,)
+updateRemotePIP(, , , , , , ,)
+getAttributeRemotePIP(, , , , , , , ,)
1..1
0..1
◀ call
1..1
▶ call
«PIPcoordinator»
Attributes
Operations
+getSubjectCondtionAttribute(, ,)
+getObjectCondtionAttribute(, ,)
+getEnvironmentCondtionAttribute(, ,)
+getEventCondtionAttribute(, ,)
+addAttributesToDB(, , , , , , ,)
+updateAttributesDB(, , , , , ,)
«RemotePIPcoordinator»
Attributes
Operations
+connectionToRemotePIP(, , , , , , , , , ,)
+addToLocalPIP(, , , , , , ,)
+updateLocalPIP(, , , , , ,)
+getAttributeLocalPIP(, , , , , ,)
0..1
0..1
◀ call
1..1

FIGURE 9.4: DENAR prototype class diagram.

9.3.1 XACML Architecture

OASIS standard XACML architecture (See Chapter 2, Section 2.4.3) was modified by [15] in a way that PDPs become easy to be deployed in the DENARs network lab. Figure 9.5 depicts the DSP XACML framework that presented in [15]. In Figure 9.5, the PDP becomes at the heart of the architecture instead of context handler, context handler is moved inside the PDP [15]. This, on one hand, increases the performance of handling authorisation requests, querying the attributes, evaluating decisions, and providing authorisation responses. On the other hand, this makes PDP to become portable and easy to be moved from one location to another across the network [15]. Moreover,in [7], we modified XACML architecture to provide synchronisation between multiple PDPs those enforce dynamic policies.

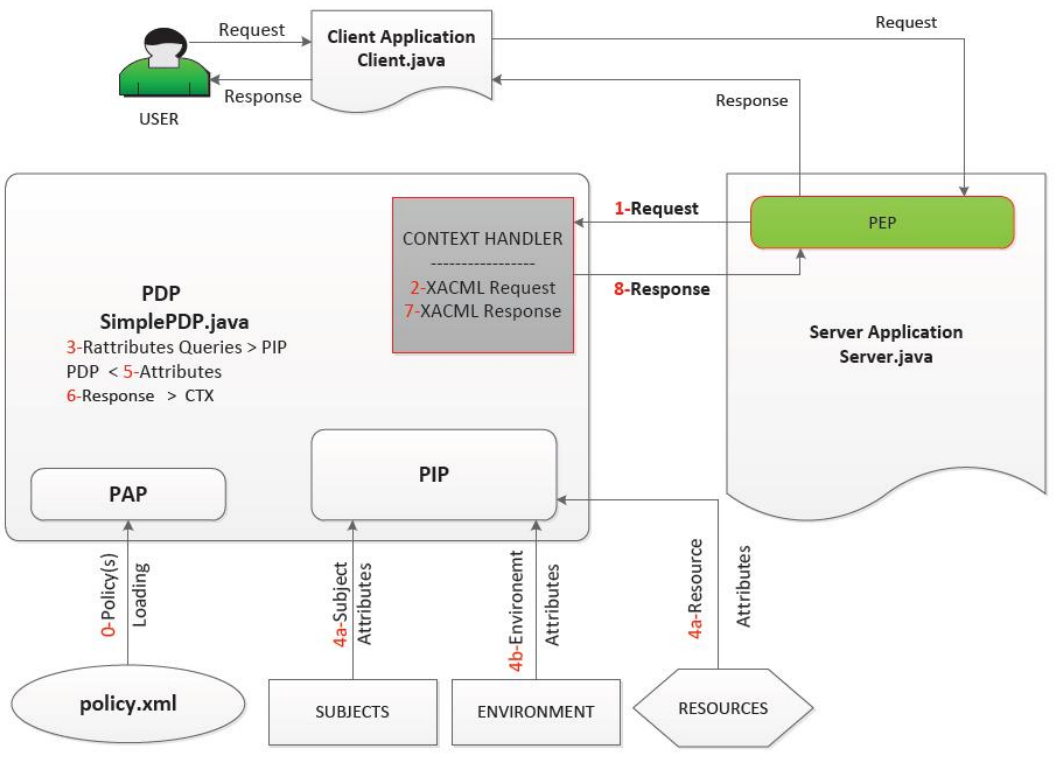


FIGURE 9.5: DSP XACML framework [15].

The PEP, in Figure 9.5, appears in the server application to be as close as possible to the resources. User requests initially come from client to the server application. PEP picks up the requests, sends them to the context handler in the PDP, and waits for the decisions. Decisions are sent from PDP to PEP wrapped inside XACML response.

Sun XACML implementation consists of several packages for implementing PDP, PEPs and any other XACML related component. The most important packages are:

- **com.sun.xacml** is the core package. It contains PDP class, policy and policy set handling, rule evaluation and target matching.
- **com.sun.xacml.ctx** is the package that defines XACML context schema types such as request and response. It contains classes for encoding and parsing XML context. The PEP can be built based on these classes.
- **com.sun.xacml.combine** is the package that supports all of the standard combining algorithms. It also contains standard interfaces and abstract classes for creating new combining algorithms. The combining algorithm specifies the final access control decision of a policy. In SANTA, the combining algorithm is provided in term of decision rule. [See Chapter 3, Section 3.2.1.2]
- **com.sun.xacml.attr** is the package that supports all the standard XACML attribute data types. It also contains standard interfaces and abstract classes to define new attribute types.
- **com.sun.xacml.finder** represents the PAP and PIP. It contains classes for retrieving policies, attributes not provided in the request for PDP.

DENAR software is implemented using Sun XACML implementation [92] and the DSP XACML framework [15] and additional classes we built. However, DSP XACML framework implements multiple PDPs but does not involve the PIPs where as the DENAR prototype does.

9.3.1.1 Policy File, Request and Response

In order for PDP to evaluate authorisation decisions, policies must be available in policy repository all the time for the PDP. DENAR enforces only one rule per request from policy files, i.e. those stored in policy repository (PR). According to our decomposition model in Chapter 7, for each object in the system, there is a rule that represents its authorisation statement. Rules in the policy file that are implemented in our prototype are very simple. That is, each rule allows a user to perform download action on the directory that he owns only. For example, user (ali) can perform download action on (/hosthome/XACML-Project/users/ali) file.

Listing 9.2 shows the rule corresponding to user (ali). That rule has an ID (in first line), which is an identifier used by PAP and PDP to classify the rules. It also has an effect, which is Permit. The rules' effect is the authorisation decision for the requested authorisation access. The subject attribute, (in line 6), represents the user name. The resource attributes, (in line 16 and 17), represents the resource to be accessed. The actions attribute, (in line 26), specifies action.

```
1   <Rule RuleId="Rule0001" Effect="Permit">
2    <Target>
3    <Subjects>
4     <Subject>
5      <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
6       <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">ali
7       </AttributeValue>
8       <SubjectAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema#string"
9        AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id" />
10     </SubjectMatch>
11    </Subject>
12   </Subjects>
13   <Resources>
14    <Resource>
15     <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
16     <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
17     /hosthome/XACML-Project/users/ali</AttributeValue>
18     <ResourceAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema#string"
19      AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id" />
20     </ResourceMatch>
21    </Resource>
22   </Resources>
23   <Actions>
24    <Action>
25     <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
26     <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">Download
27     </AttributeValue>
28     <ActionAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema#string"
29      AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id" />
30     </ActionMatch>
31    </Action>
32   </Actions>
33   </Target>
34  </Rule>
35 </Policy>
```

Listing 9.3: Policy File in XACML

In a request, (*ali*) is the subject, (*/hosthome/XACML-Project/users/ali*) is the resource, i.e. the object to be accessed, and (*download*) is the action to be performed on that resource. The PEP passes these three values to the context handler in the PDP.

The context handler generates a standard XACML request similar to that one in Listing 9.3. The request must contain subject, resource and action attributes. Then, it forwards the request to the PDP. The PDP uses subject attribute to search for any applicable rule. If any applicable rule is found, the PDP compares the resource and action attribute values of the applicable rule with the resource and action attribute values of the request. If all values are matched, the request is evaluated to true and the rule effect is returned. Otherwise, the PDP found no applicable rule and in this case it returns NotApplicable.

```
<Request xmlns="urn:oasis:names:tc:xacml:1.0:context"
mlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <Subject>
  <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
  DataType="urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name">
  <AttributeValue>ali</AttributeValue>
  </Attribute>
 </Subject>
 <Resource>
  <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
  DataType="http://www.w3.org/2001/XMLSchema#string">
  <AttributeValue>/hosthome/XACML-Project/users/ali</AttributeValue>
  </Attribute>
 </Resource>
 <Action>
  <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
  DataType="http://www.w3.org/2001/XMLSchema#string">
  <AttributeValue>Download</AttributeValue>
  </Attribute>
 </Action>
</Request>
```

Listing 9.4: Request File in XACML

According to the rule above, this request will be evaluated to true. The rule will return its effect to the policy. The policy will return that value to the PDP which in turn wraps it into an XACML response and sends it to the context handler.

Since the applicable rule effect is Permit, this value will be placed in the decision attribute of the XACML response and returned to the PEP to be enforced. Listing 9.4 shows the standard XACML response. In that response the decision attribute which contains Permit appears in line 3. If no applicable rule is found, that value will be NotApplicable.

```
1  <Response>
2    <Result>
3      <Decision>Permit</Decision>
4      <Status>
5        <StatusCode Value="urn:oasis:names:tc:xacml:1.0:status:ok"/>
6      </Status>
7    </Result>
8  </Response>
```

Listing 9.5: Response File in XACML

9.3.2 Client-Server Application

Client-Server application is a simple distributed Java application that allows users to download files. It consists of two parts; server part (DENAR.java file, Appendix A.2.3) and client part (client.java file , Appendix A.2.2).

The implementation in the paper [7] provides a library in Java which creates and links a network of PDP's in a peer to peer implementation. The library is provided for the maintenance of state even for PDP's that are stateless. The library also contains classes for the generation of PEP nodes that automatically link with the Overlay Network created and manage communication to and from the PDP. The PEP is capable of connecting to more than one PDP if required and automatically queries the backup PDP's for a decision if the main active PDP fails. For the PEP, the PDP's appear to be one coherent system and the distribution and coordination between the various PDP's are completely hidden from it by the Overlay Network. The chief characteristic of the overlay network is that it can be used to easily convert existing standalone, stateless PDP's into a network of stateful networked PDP's with minimal alteration in the existing PDP code.

By the same technique in [7], the network of DENAR can be implemented for the event of any DENAR failing, thus, the library will dynamically reconfigure itself to distribute the load among the other DENAR's in the network. The reason for not undertaking the peer to peer implementation in this work is that we have done that in [7]. However, in this work, we implements the network of DENAR by client-server technique to evaluate the efficiency of enforcing static and dynamic policy in distributed setting where integrating this work with the work in [7] would be useful for the future work to offer more resilience in policy enforcement.

9.3.2.1 Client Class

The client part enables users to request (downloading) files after being authenticated. The client application asks for username and password before allowing users to make request. If the user is authenticated; he/she can perform the action (download) to resources from the server (DENAR).

The client class variables, operations and responsibilities are illustrated in Figure 9.6.

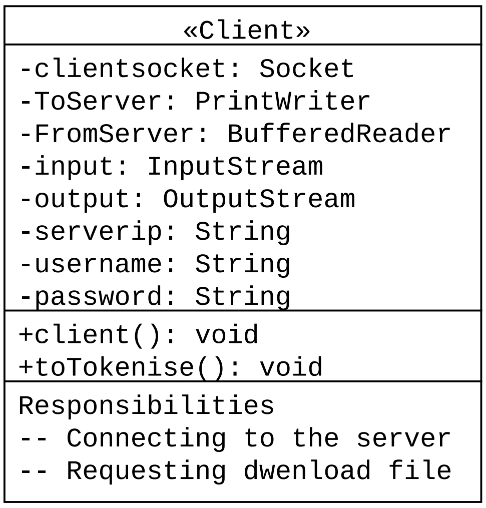


FIGURE 9.6: Client class.

Figure 9.7 illustrates flow chart process in the client class for the authorisation request to download a file.

The client application requires a username or subject, action and resource which is the destination file. It sends all these data to the DENAR (server) and waits for authorisation decision. If NotApplicable or deny is returned, this means that the user has no permission to perform the action on the destination file. Otherwise,

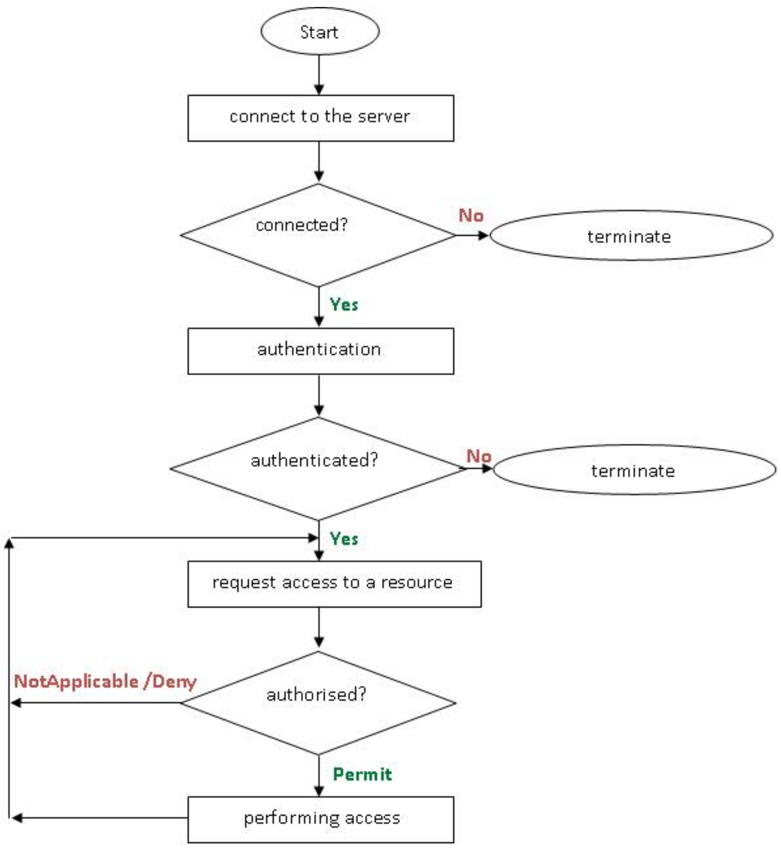


FIGURE 9.7: Client class flow chart.

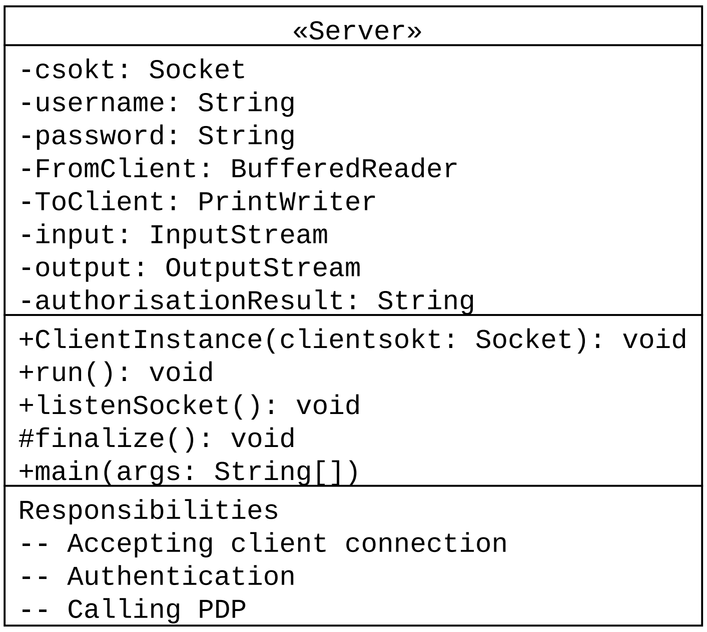
Permit will be returned and the user will be able to perform the action on that file.

9.3.2.2 DENAR Class

The DENAR part, on the other hand, is responsible for; authenticating users, authorising their request and enforcing authorisation decisions. This part is designed to be a multi-threading application. Because it is acting as a server, it has to be able to serve clients concurrently.

Using multi-threading, the server application is designed to create a connection with each client, and keeps that connection alive as long as the client remains connected. These connections must be reliable. That is, data must not be dropped and must not arrive late on the client or out of order. This can be achieved using sockets. A Socket is a bidirectional end point between two applications on the network. Each program binds a socket to its end of the connection using local and remote IP address and port numbers [44].

The DENAR class variables, operations and responsibilities are illustrated in Figure 9.8.



```
«Server»
-csokt: Socket
-username: String
-password: String
-FromClient: BufferedReader
-ToClient: PrintWriter
-input: InputStream
-output: OutputStream
-authorisationResult: String
+ClientInstance(clientsokt: Socket): void
+run(): void
+listenSocket(): void
#finalize(): void
+main(args: String[])
Responsibilities
-- Accepting client connection
-- Authentication
-- Calling PDP
```

FIGURE 9.8: DENAR Class.

Figure 9.9 illustrates a flow chart process in the DENAR class for the authorisation response to download request. In order to connect the client to DENAR, a user has to type the DENARs IP address (i.e. 192.168.20.1). The client application asks the user to type his username and password. If this authentication information was entered correctly, the user can now request access his files. (See Appendix A.2.4 for tester class)

Instead of manually generating requests from each user, tester class is an automatic code used to generate 50 requests from each user machine. This, on one hand, saves time and efforts. On the other hand make the test more effective by sending requests from each user to the PDP at the same time. It must be started in conjunction with client application as follow:

```
java tester [server address] [username] [password] [seed] | java client
```

LISTING 9.6: Connecting the client to DENAR

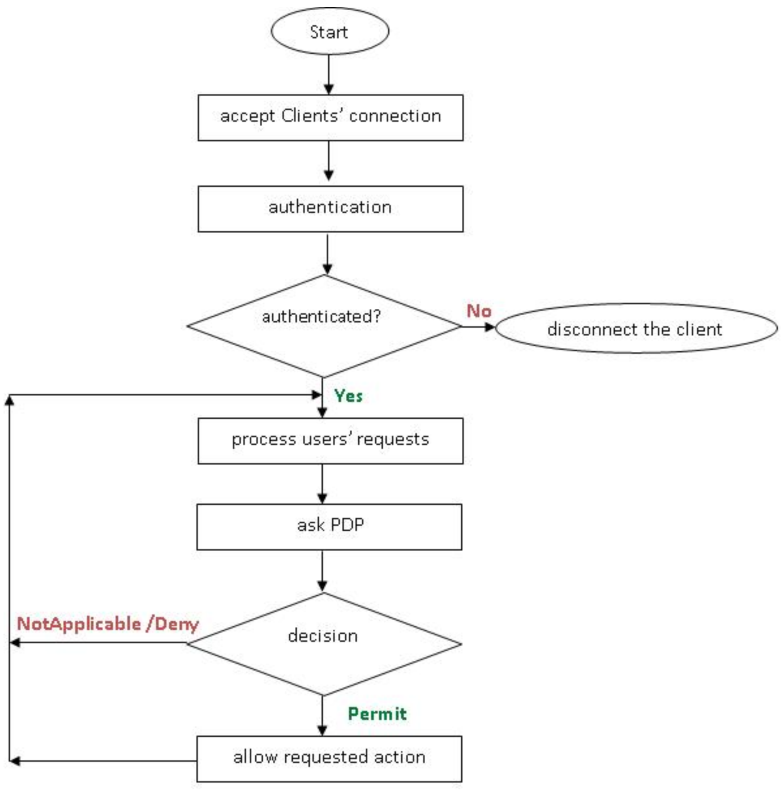


FIGURE 9.9: DENAR class flow chart.

This command makes tester class to start the client application, and supply connection, authentication and requests data to the client application on behalf of the user. Since this application generates requests randomly, the seed value makes it generate the same set of request by generating the same set of random numbers.

9.3.3 DENARs Enforcement and Coordination

In this subsection, the SimplePDP, ConstraintCheckerFunction, PIPcoordinator, RemotePIPcoordinator and PIP classes are described where they provide the DENARs policy enforcement and coordination.

9.3.3.1 SimplePDP Class

The SimplePDP class is the main class in the DENAR enforcement implementation. It extends the PDPConfig class. The SimplePDP class variables, operations and responsibilities are shown in Figure 9.10.

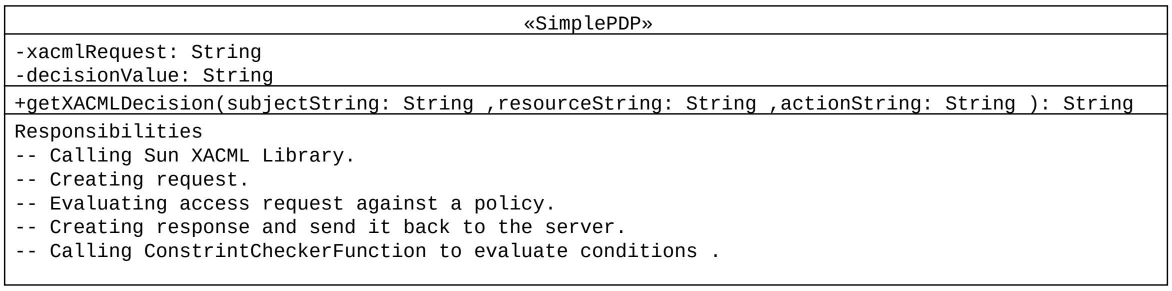

«SimplePDP»
-xacmlRequest: String
-decisionValue: String
+getXACMLDecision(subjectString: String ,resourceString: String ,actionString: String): String
Responsibilities
-- Calling Sun XACML Library.
-- Creating request.
-- Evaluating access request against a policy.
-- Creating response and send it back to the server.
-- Calling ConstrintCheckerFunction to evaluate conditions .

FIGURE 9.10: SimplePDP class.

SimplePDP.java file (Appendix A.2.5) represents the PDP of DENAR. Policy Repository (PR) is the first component of the PDP to be created.

The SimplePDP extends the original PDPConfig class in order to implement a custom function getXACMLDecision(). This function is required to setup a policy and attribute finders where they are imported from (com.sun.xacml.finder). Also, it creates a PDP, receives a request file and creates a response file where being the start task of the policy enforcement in DENAR. Since PDP understands XACML requests in XML format only, the context handler is implemented to format such requests. It is designed to formulate XACML request. It uses a template made of string with three spaces; subject, resource and action. The context handler fills these spaces with the values that come from the PEP to generate the XACML request in XML format. An alternative way is by an XML file represents the standard XACML request. The context handler reads that file and replaces subject, resource and action attributes with the same values come from the PEP. Instead of receiving a request file, the function receives request attributes (subject, resource and action) and creates the request which save a network bandwidth.

Sun XACML [92] defines a finder module (i.e. FilePolicyModule) to implement the repository. This module specifies the location of policies which is usually XML file. Instead of that, module is created and loaded with the DENAR policy file Policy.xml.

The PolicyFinder represents the PAP. It is loaded by FilePolicyModule which represents the repository of policy. Effectively, it loads DENAR policy file. A brief description of SimplePDP functions are as follows in below listing:

```
//Initialise and start PDP
```

```
PDP pdp = new PDP(new PDPConfig(attrFinder, policyFinder, null));
// load the policies
FilePolicyModule policyModule = new FilePolicyModule();
```

Listing 9.7: Starting PDP and loading policy in DENAR

The SimplePDP class also calls the specially designed ConstraintCheckerFunction class to evaluate conditional attributes from a policy against their values from distributed PIPs.

```
FunctionFactoryProxy proxy = StandardFunctionFactory.getNewFactoryProxy();
FunctionFactory factory = proxy.getConditionFactory();
factory.addFunction(new ConstraintCheckerFunction());
FunctionFactory.setDefaultFactory(proxy);
decisionValue = ConstraintCheckerFunction.PolicyChecker(subjectString, resourceString,
actionString);
//create xacml request and response
RequestCtx request = RequestCtx.getInstance(requestNode);
ResponseCtx response = pdp.evaluate(request);
```

Listing 9.8: Enforcing a policy In DENAR

Once the request is ready, the PDP uses the method evaluate(); to evaluate the request, and return XACML response which contains the result of evaluation to the context handler. The response consists mainly of decision attribute which is the actual value that the PEP uses to enforce the decision. The context handler is also developed to process the XACML response instantly in the memory instead of writing it to XML file so that the PEP read it. The context handler returns the decision value which is either Permit for authorisation or NotApplicable or Deny for no authorisation.

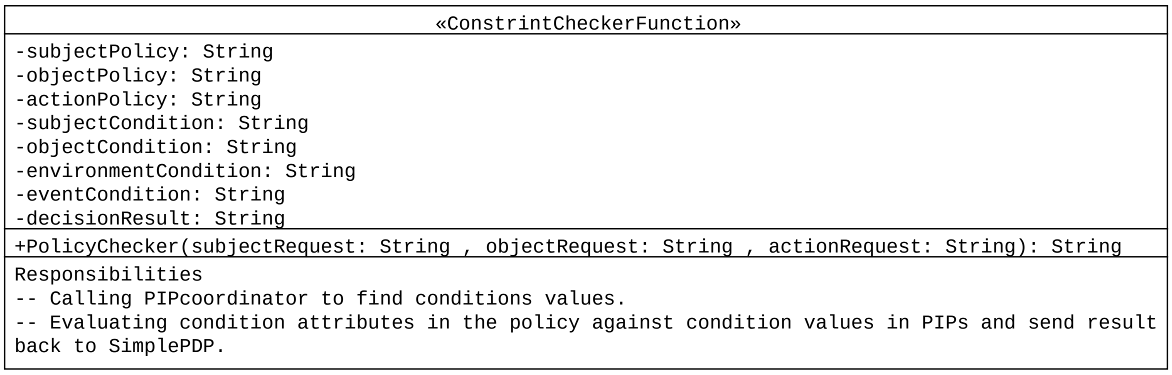
```
//create xacml request and response
RequestCtx request = RequestCtx.getInstance(requestNode);
ResponseCtx response = pdp.evaluate(request);
```

Listing 9.9: Creating authorisation response In DENAR

9.3.3.2 ConstraintCheckerFunction Class

The ConstraintCheckerFunction is a special class (Appendix A.2.6) designed exclusively to evaluate the conditional attributes when a policy is enforced by a PDP.

The ConstraintCheckerFunction class variables, operations and responsibilities are shown in Figure 9.11 below.



```
«ConstrintCheckerFunction»
-subjectPolicy: String
-objectPolicy: String
-actionPolicy: String
-subjectCondition: String
-objectCondition: String
-environmentCondition: String
-eventCondition: String
-decisionResult: String
+PolicyChecker(subjectRequest: String , objectRequest: String , actionRequest: String): String
Responsibilities
-- Calling PIPcoordinator to find conditions values.
-- Evaluating condition attributes in the policy against condition values in PIPs and send result
back to SimplePDP.
```

FIGURE 9.11: ConstraintCheckerFunction class.

The ConstraintCheckerFunction is invoked by the SimplePDP whenever there is a call to evaluate condition in a policy file. It uses one main method to arrive at a decision for any request.

```
String PolicyChecker(String  subjectRequest, String  objectRequest,
String  actionRequest )
```

LISTING 9.10: Evaluating conditions In DENAR

The syntax and semantics of the PolicyChecker, we are more concerned with, are the list of inputs. The ConstraintCheckerFunction retrieves these inputs from the actual policy file those meet the same attributes from request. It has to be supplied with all the parameters required by the coordination PIP of the DENAR. These parameters as declared in the ConstraintCheckerFunction are:

```
String  subjectPolicy = "";
String  objectPolicy = "";
String  actionPolicy = "";
String  conditionPolicy = "";
String  subjectCondition = "";
String  objectCondition = "";
String  environmentCondition = "";
String  eventCondition = "";
```

LISTING 9.11: Main conditional attributes in a policy

Examples of policy files are provided in Appendix A.2.1. Of special interest is the 'LimitedAccessPolicy.xml' (in Listing A.18) policy which is designed specially to demonstrate the capabilities of the DENAR coordination PIP using the custom ConstraintCheckerFunction. The ConstraintCheckerFunction is invoked by the 'LimitedAccessPolicy' policy. Listing 9.11 shows the rules' conditions those being evaluated before making a decision as defined in the policy.

```
<Condition FunctionId="PDPOverlay_Constraint_Checker">
  <SubjectCondition DataType="http://www.w3.org/2001/XMLSchema#string">Research
  </SubjectCondition>
  <ObjectCondition DataType="http://www.w3.org/2001/XMLSchema#string">Secret
  </ObjectCondition>
  <EnvironmentConditionDataType="http://www.w3.org/2001/XMLSchema#string">01Jan2012
  </EnvironmentCondition>
  <EventCondition DataType="http://www.w3.org/2001/XMLSchema#string">has allowed
  </EventCondition>
</Condition>
```

Listing 9.12: Example of conditional attributes in a policy

As can be seen, all the conditional attributes required by the policy to be evaluated first against condition values in PIPs. Calling the PIPcoordinator class is designed to access PIPs in DENARs using the PIPcoordinatorTask function. The PolicyChecker in the ConstraintCheckerFunction class returns the condition evaluation result for those conditional attributes in the policy against those condition values in PIPs if they are evaluation result equal to true where the result return back to the SimplePDP class.

```
String conditionEvaluationResult = "";
PIPcoordinator = new PIPcoordinator();
conditionEvaluationResult = PIPcoordinator.PIPcoordinatorTask(PIPfind, job,
subjectPolicy, objectPolicy, actionPolicy, subjectCondition, objectCondition,
environmentCondition, eventCondition, nodeAttribute, oldAttribute, newAttribute);
```

Listing 9.13: Calling PIPcoordinator class for add/update/find condition values

Once the access is done by receiving that for PEP, the PolicyChecker determines if new condition values need to be added or current condition values need to be updated where that enforced by obligation context in a policy. Thus, it simply creates a new entry for a new condition value in a PIP. Adding and updating condition values in PIPs is performed by the PolicyChecker if the obligation context indicates that in the policy, the DENAR ensures that all the coordinated condition values across the DENARs network are updated.

However, unlike the implementation in [27] where the coordinated condition values are updated before the access. In the DENAR, PDP updates the coordinated condition values after permitting access to the PEP. The updating can easily be incorporated by changing the ConstraintCheckerFunction and implementing a custom method in the PEP for acknowledging execution of the request. This could potentially be a consideration in future revisions of the prototype to achieve the objective of the Push Model in the coordination mechanism.

9.3.3.3 PIPcoordinator Class

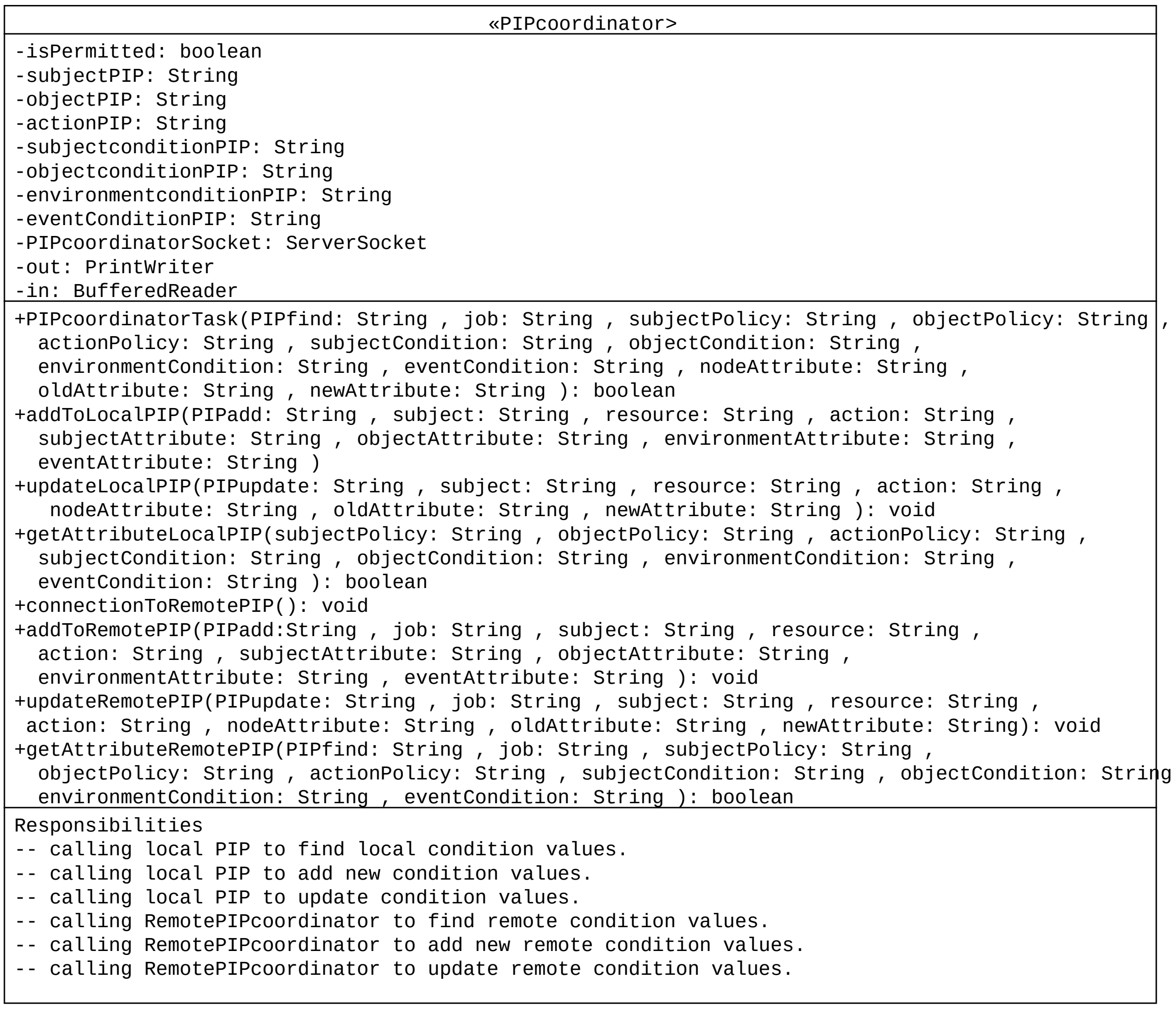
The main function of the PIPcoordinator class (Appendix A.2.7) is to act as bridge between DENARs for the coordination. The SimplePDP class in the DENAR can communicate with other PIPs in the DENARs network only via PIPcoordinators class. In case of enforcing policies, the PDP communicates with local PIP and remote PIPs in other DENARs to evaluate conditional attributes.

The PIPcoordinator is being the coordinator between the PDP and PIPs to add a new condition values or update condition values.

The PIPcoordinator class can determine if the condition values are located in the local PIP or remote PIPs from the condition context (condition attributes) in the policy. According to our decomposition model that is described in the Chapter 7, the conditional attributes that begin by indicator "#@" that indicates to the remote PIPs and followed by the PIP name so those condition values can be founded in the remote PIPs otherwise condition values can be founded in local PIP.

In addition, PIPcoordinator calls local PIP via PIP class (presented in Section 9.3.3.5) to add a new condition value, update condition value and fetch a required condition value for a decision. Also, the PIPcoordinator calls RemotePIPcoordinators class (presented in next section) to connect PIPs in different DENARs.

The PIPcoordinator class variables, operations and responsibilities are shown in Figure 9.12.



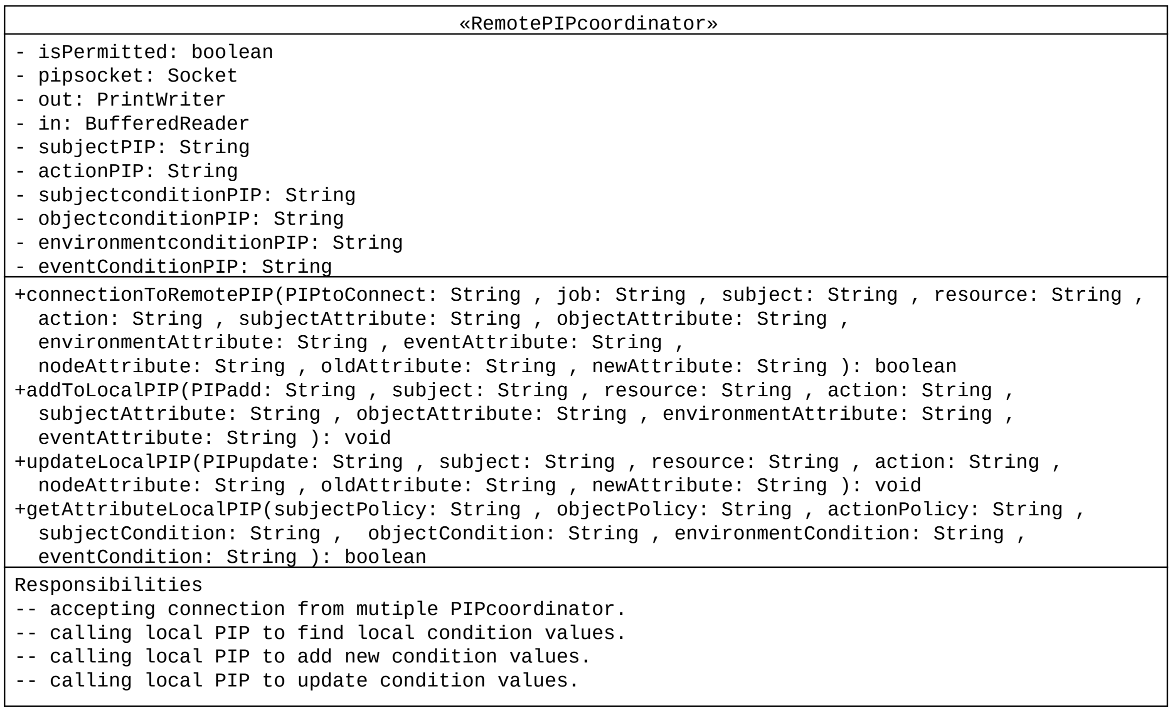
```
«PIPcoordinator>
-isPermitted: boolean
-subjectPIP: String
-objectPIP: String
-actionPIP: String
-subjectconditionPIP: String
-objectconditionPIP: String
-environmentconditionPIP: String
-eventConditionPIP: String
-PIPcoordinatorSocket: ServerSocket
-out: PrintWriter
-in: BufferedReader
+PIPcoordinatorTask(PIPfind: String , job: String , subjectPolicy: String , objectPolicy: String ,
  actionPolicy: String , subjectCondition: String , objectCondition: String ,
  environmentCondition: String , eventCondition: String , nodeAttribute: String ,
  oldAttribute: String , newAttribute: String ): boolean
+addToLocalPIP(PIPadd: String , subject: String , resource: String , action: String ,
  subjectAttribute: String , objectAttribute: String , environmentAttribute: String ,
  eventAttribute: String )
+updateLocalPIP(PIPupdate: String , subject: String , resource: String , action: String ,
   nodeAttribute: String , oldAttribute: String , newAttribute: String ): void
+getAttributeLocalPIP(subjectPolicy: String , objectPolicy: String , actionPolicy: String ,
  subjectCondition: String , objectCondition: String , environmentCondition: String ,
  eventCondition: String ): boolean
+connectionToRemotePIP(): void
+addToRemotePIP(PIPadd:String , job: String , subject: String , resource: String ,
  action: String , subjectAttribute: String , objectAttribute: String ,
  environmentAttribute: String , eventAttribute: String ): void
+updateRemotePIP(PIPupdate: String , job: String , subject: String , resource: String ,
 action: String , nodeAttribute: String , oldAttribute: String , newAttribute: String): void
+getAttributeRemotePIP(PIPfind: String , job: String , subjectPolicy: String ,
  objectPolicy: String , actionPolicy: String , subjectCondition: String , objectCondition: String
  environmentCondition: String , eventCondition: String ): boolean
Responsibilities
-- calling local PIP to find local condition values.
-- calling local PIP to add new condition values.
-- calling local PIP to update condition values.
-- calling RemotePIPcoordinator to find remote condition values.
-- calling RemotePIPcoordinator to add new remote condition values.
-- calling RemotePIPcoordinator to update remote condition values.
```

FIGURE 9.12: PIPcoordinator class.

9.3.3.4 RemotePIPcoordinator Class

The RemotePIPcoordinator class (Appendix A.2.8) acts as bridge for transmitting condition values between PIPcoordinator and remote PIPs, i.e. those located in different DENARs.

PDPs can communicate with other PIPs in the DENAR network only via RemotePIPcoordinator class which is invoked by PIPcoordinator class. The RemotePIPcoordinator is involved to add new condition values, update condition values and query a required condition value for a decision when those values are located in remote PIPs.



```
«RemotePIPcoordinator»
- isPermitted: boolean
- pipsocket: Socket
- out: PrintWriter
- in: BufferedReader
- subjectPIP: String
- actionPIP: String
- subjectconditionPIP: String
- objectconditionPIP: String
- environmentconditionPIP: String
- eventConditionPIP: String
+connectionToRemotePIP(PIPtoConnect: String , job: String , subject: String , resource: String ,
  action: String , subjectAttribute: String , objectAttribute: String ,
  environmentAttribute: String , eventAttribute: String ,
  nodeAttribute: String , oldAttribute: String , newAttribute: String ): boolean
+addToLocalPIP(PIPadd: String , subject: String , resource: String , action: String ,
  subjectAttribute: String , objectAttribute: String , environmentAttribute: String ,
  eventAttribute: String ): void
+updateLocalPIP(PIPupdate: String , subject: String , resource: String , action: String ,
  nodeAttribute: String , oldAttribute: String , newAttribute: String ): void
+getAttributeLocalPIP(subjectPolicy: String , objectPolicy: String , actionPolicy: String ,
  subjectCondition: String ,  objectCondition: String , environmentCondition: String ,
  eventCondition: String ): boolean
Responsibilities
-- accepting connection from mutiple PIPcoordinator.
-- calling local PIP to find local condition values.
-- calling local PIP to add new condition values.
-- calling local PIP to update condition values.
```

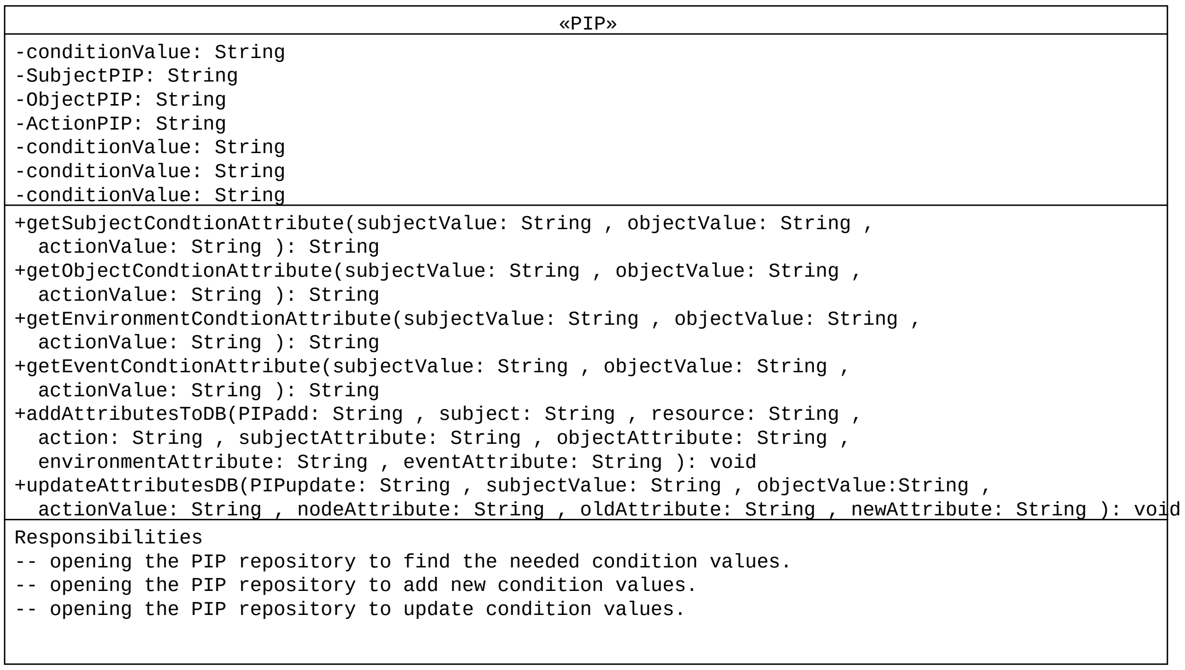
FIGURE 9.13: RemotePIPcoordinator class.

The RemotePIPcoordinator class variables, operations and responsibilities are shown in Figure 9.13.

9.3.3.5 PIP Class

The main function of the PIP class (Appendix A.2.9) is to open its defined repository and add, update and query condition values. The PIP class accepts modification and searching for condition values in a defined repository (PIP) either via PIPcoordinators class (locally) or via RemotePIPcoordinators class (remotely). The PIP class variables, operations and responsibilities are shown in Figure 9.14 below.

As we can see, the new implementations of the ConstraintCheckerFunction, the PIPcoordinators and the RemotePIPcoordinators classes can be supplied as parameters to the DENARs Network so that they are seamlessly integrated into the network. The coordination is achieved by PIPcoordinator and RemotePIPcoordinator classes where they can seamlessly handle all communications and synchronisation among the various DENARs connected in the network.



```
«PIP»
-conditionValue: String
-SubjectPIP: String
-ObjectPIP: String
-ActionPIP: String
-conditionValue: String
-conditionValue: String
-conditionValue: String
+getSubjectCondtionAttribute(subjectValue: String , objectValue: String ,
  actionValue: String ): String
+getObjectCondtionAttribute(subjectValue: String , objectValue: String ,
  actionValue: String ): String
+getEnvironmentCondtionAttribute(subjectValue: String , objectValue: String ,
  actionValue: String ): String
+getEventCondtionAttribute(subjectValue: String , objectValue: String ,
  actionValue: String ): String
+addAttributesToDB(PIPadd: String , subject: String , resource: String ,
  action: String , subjectAttribute: String , objectAttribute: String ,
  environmentAttribute: String , eventAttribute: String ): void
+updateAttributesDB(PIPupdate: String , subjectValue: String , objectValue:String ,
  actionValue: String , nodeAttribute: String , oldAttribute: String , newAttribute: String ): void
Responsibilities
-- opening the PIP repository to find the needed condition values.
-- opening the PIP repository to add new condition values.
-- opening the PIP repository to update condition values.
```

Figure 9.14: PIP class.

9.4 Summary

This Chapter introduced a high level design and some key aspects of the implementation prototype developed for policy enforcement in a network of DENARs and the coordination between DENARs.

The DENAR network is developed in the Netkit [81] simulation software. The DENAR software is implemented in Java where it is integrated with Sun XACML implementation library [92]. The Chapter also described how these parts work with each other to achieve the research objectives. In addition, The Chapter described how the access control request received, a policy enforced and the authorisation response is made in DENAR. Finally, the PIPcoordinator class is implemented to retrieve and update a local conditional attributes and the RemotePIPcoordinator class is implemented to retrieve and update a remote conditional attributes where these two classes are designed for the coordination task.

In Chapter 10, the evaluation showed that the implementation of Distributed Enforcements Architecture (DENAR) is feasible to the enforcement of both static and dynamic policies.

The following Chapter introduces the Centralised PBM and Decentralised PBM simulators. Moreover, the performance, security, manageability and resilience factors for this research are evaluated through the case studies in Decentralised PBM simulator against the Centralised PBM simulator.

Chapter 10

Case Study and Evaluation

Objectives

- Give appropriate network simulators that show how a security policy is enforced in both Centralised PBM and Decentralised PBM systems.
- Give appropriate case studies of static and dynamic policies showing policy enforcement in both Centralised PBM and Decentralised PBM simulators.
- Demonstrate the practical applicability of the presented research.
- Evaluate the Decentralised PBM approach against the Centralised PBM approach.

10.1 Introduction

This Chapter introduces the Decentralised Policy Based Management (PBM) simulator that allows for the enforcement of both static [65] and dynamic [3] policies. In addition, the Chapter provides three case studies to illustrate the practical applicability for both static and dynamic policies, i.e those enforced in Centralised PBM and Decentralised PBM simulators. Finally, this Chapter evaluates the Centralised PBM and Decentralised PBM approaches based on the research success criteria. [See Chapter 1, Section 1.6]

The Decentralised PBM simulator is designed to support the static and dynamic policy enforcement and is used to establish the enforcement functional behaviour of the Decentralised PBM approach. Enforcement functional behaviour is defined as the replication of policy decisions that would have been made by a Centralised PBM in the same scenario. Other success criteria such as performance, security, manageability and resilience are evaluated in this Chapter based on the case studies and in Centralised PBM and Decentralised PBM simulators to show the feasibility of the Decentralised PBM implementation.

The Centralised PBM and Decentralised PBM simulators are described in Section 10.2. Section 10.3 provides the three case studies for static and dynamic policies. Section 10.4 evaluates the Centralised PBM and Decentralised PBM approaches in the provided case studies and the simulators.

10.2 Policy Based Management (PBM) Simulation

The aim of this section is to describe the PBM simulation of the centralised and decentralised approaches. The two simulators are designed and configured in Netkit [81]. Both simulators were deployed on a host machine which has the following specifications:

- Processor: Intel(R) Core(TM) 2 Duo CPU E7500 @ 2.93GHz.
- Memory: 3GB.
- Operating System: Ubuntu GNU/Linux 3.20-35-generic kernel.

10.2.1 First Simulator: Centralised PBM

In Centralised PBM, the network includes four LANs with four routers and main getaway. Also, each LAN includes four client machines (PCs). One Policy Decision Point (PDP) is deployed on LAN2 server machine. The PDP is involved in the server application so that running the application will automatically run the PDP on the same machine (CentralPDP). Users can use the client application on client machines (PCs) to connect to the server and start performing requests to access their files.

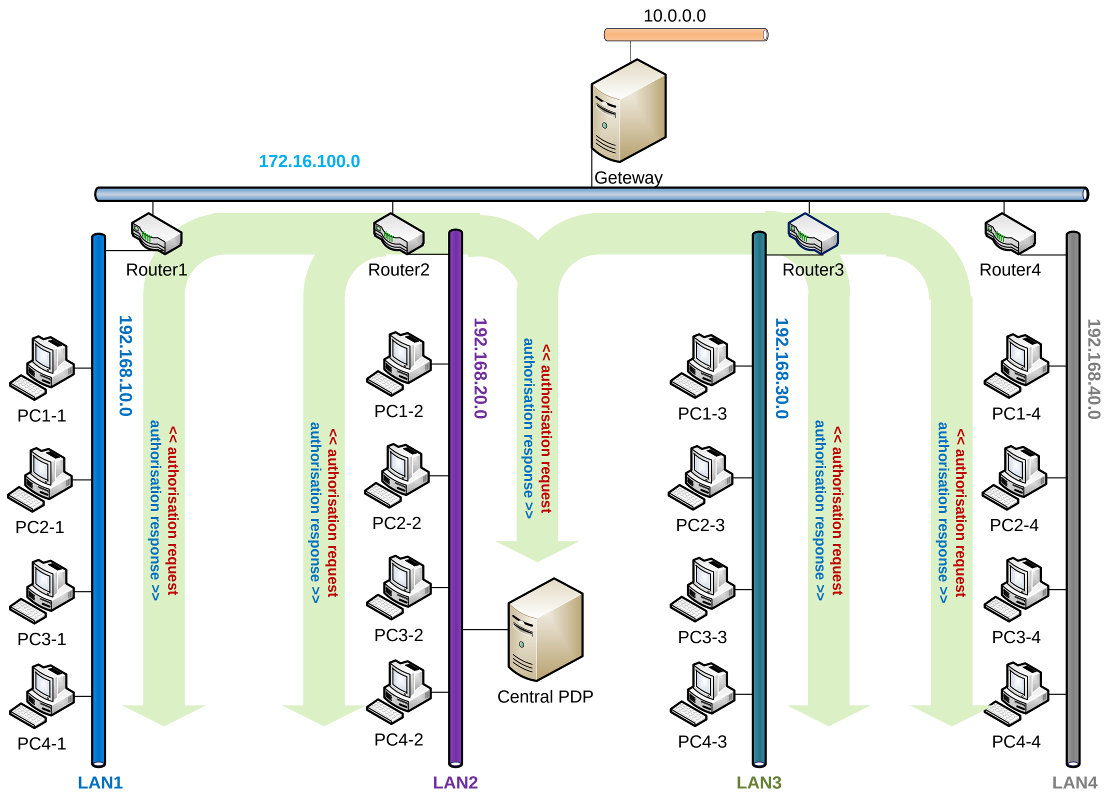
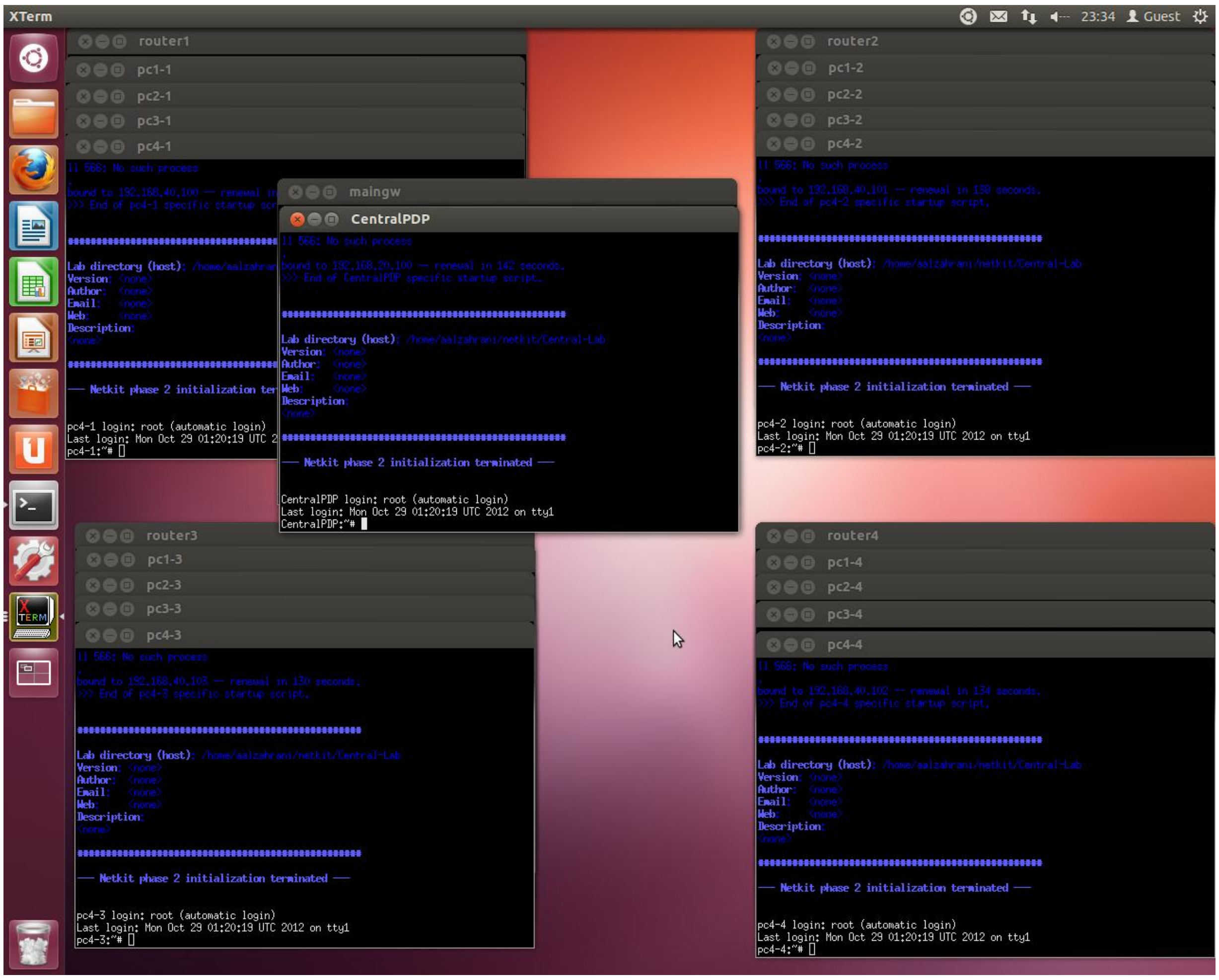


Figure 10.1: First Simulator: Centralised PBM.

Figure 10.1 shows the authorisation traffic flow, viz. authorisation requests and responses traffic, for the first simulator. The PDP is operating on LAN 2 server machine. Authorisation requests from all the clients in LAN1, LAN2, LAN3 and LAN4 are sent to the PDP where the authorisation response is sent back to the client that sent the request.

The Centralised PBM simulator screen is shown in Figure 10.2. Each LAN is shown in a corner of the screen with their client windows machines (PCs) and its window router. The centralPDP is shown in the middle of the screen.



```
ll 566: No such process
,
bound to 192.168.40.100 -- renewal in
>>> End of pc4-1 specific startup scr

##################################################

Lab directory (host): /home/aalzahran
Version: <none>
Author:  <none>
Email:   <none>
Web:     <none>
Description:
<none>

##################################################

--- Netkit phase 2 initialization ter

pc4-1 login: root (automatic login)
Last login: Mon Oct 29 01:20:19 UTC 2
pc4-1:~#
```

```
ll 566: No such process
,
bound to 192.168.20.100 -- renewal in 142 seconds.
>>> End of CentralPDP specific startup script.

##################################################

Lab directory (host): /home/aalzahrani/netkit/Central-Lab
Version: <none>
Author:  <none>
Email:   <none>
Web:     <none>
Description:
<none>

##################################################

--- Netkit phase 2 initialization terminated ---

CentralPDP login: root (automatic login)
Last login: Mon Oct 29 01:20:19 UTC 2012 on tty1
CentralPDP:~#
```

```
ll 566: No such process
,
bound to 192.168.40.101 -- renewal in 130 seconds.
>>> End of pc4-2 specific startup script.

##################################################

Lab directory (host): /home/aalzahrani/netkit/Central-Lab
Version: <none>
Author:  <none>
Email:   <none>
Web:     <none>
Description:
<none>

##################################################

--- Netkit phase 2 initialization terminated ---

pc4-2 login: root (automatic login)
Last login: Mon Oct 29 01:20:19 UTC 2012 on tty1
pc4-2:~#
```

```
ll 566: No such process
,
bound to 192.168.40.103 -- renewal in 130 seconds.
>>> End of pc4-3 specific startup script.

##################################################

Lab directory (host): /home/aalzahrani/netkit/Central-Lab
Version: <none>
Author:  <none>
Email:   <none>
Web:     <none>
Description:
<none>

##################################################

--- Netkit phase 2 initialization terminated ---

pc4-3 login: root (automatic login)
Last login: Mon Oct 29 01:20:19 UTC 2012 on tty1
pc4-3:~#
```

```
ll 566: No such process
,
bound to 192.168.40.102 -- renewal in 134 seconds.
>>> End of pc4-4 specific startup script.

##################################################

Lab directory (host): /home/aalzahrani/netkit/Central-Lab
Version: <none>
Author:  <none>
Email:   <none>
Web:     <none>
Description:
<none>

##################################################

--- Netkit phase 2 initialization terminated ---

pc4-4 login: root (automatic login)
Last login: Mon Oct 29 01:20:19 UTC 2012 on tty1
pc4-4:~#
```

FIGURE 10.2: Centralised PBM Simulator Screens.

10.2.2 Second Simulator: Decentralised PBM

In Decentralised PBM, four DENARs are deployed where each DENAR involves one PDP. The first PDP is deployed on LAN1 DENAR machine and similarly for other LANs.

Unlike the first simulator where all users (clients) are sending their requests to one PDP, requests are distributed across DENARs. In each LAN, one client sends its requests to the local DENAR and the other three clients send their requests to the remote DENARs which are on different LANs.

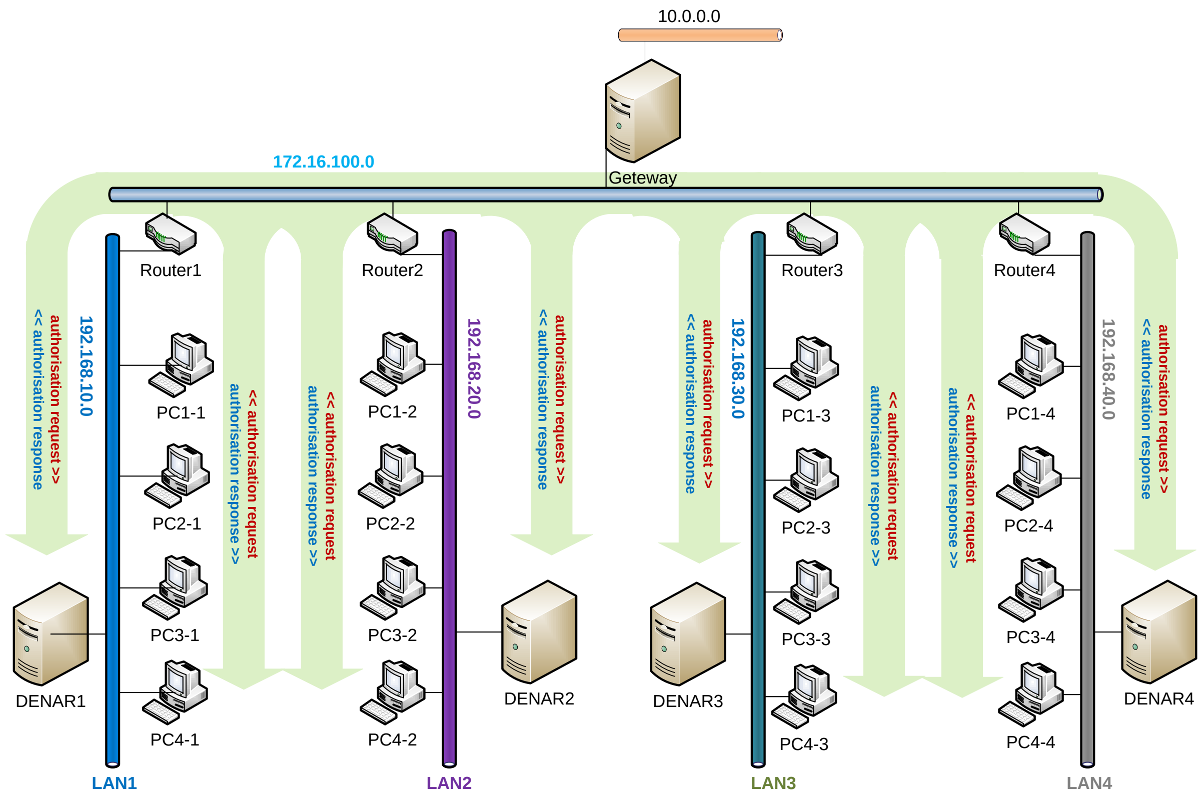


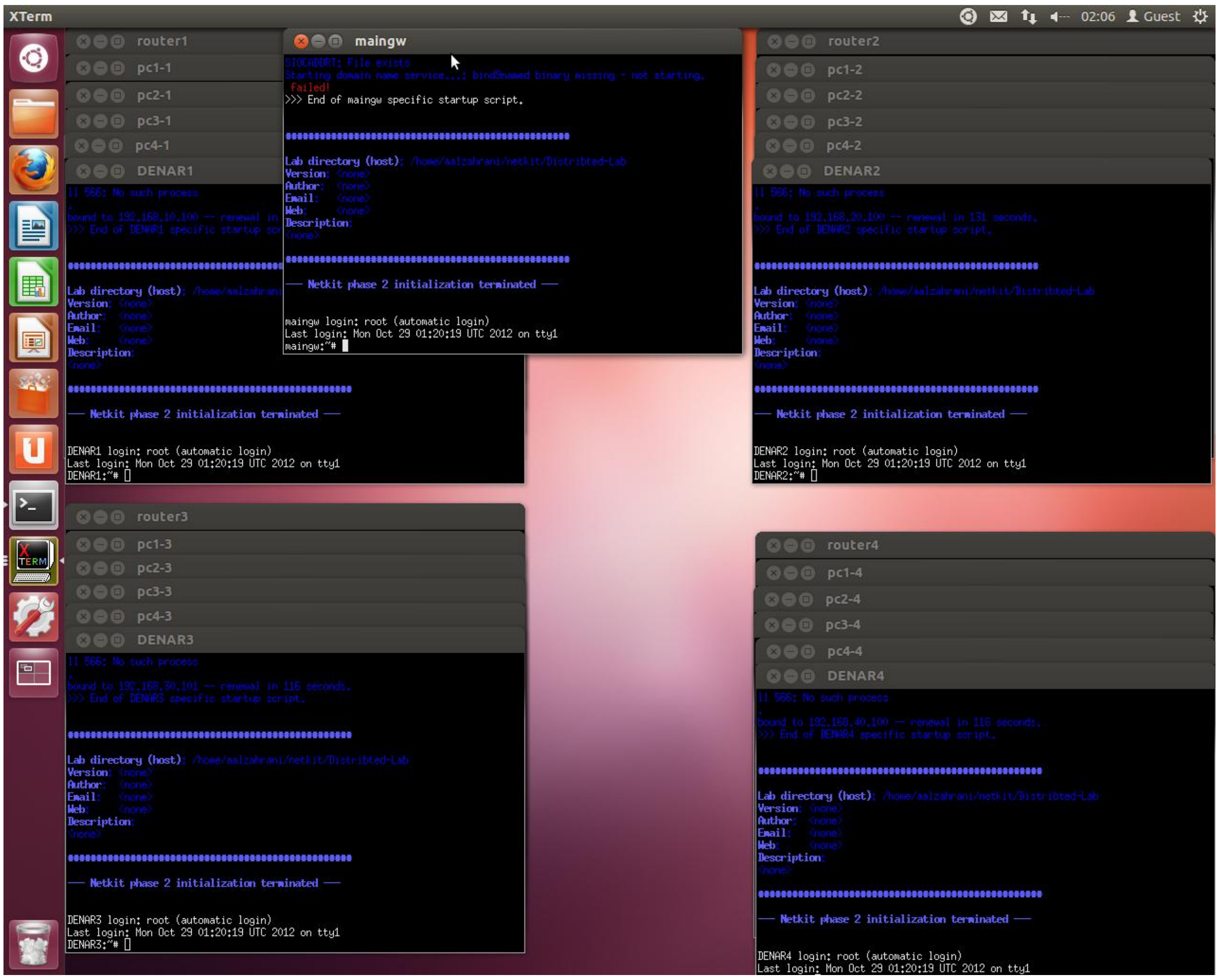
FIGURE 10.3: Second Simulator: Decentralised PBM for DENARs.

Figure 10.3 shows how the authorisation traffic flow is distributed over the DENARs network. one client (PC1-1) in LAN1 sends requests to the DENAR1 of LAN1, and the client (PC2-1) sends requests to DENAR2 of LAN2 and so on for the other clients in LAN1.

The Decentralised PBM simulator screen is shown in Figure 10.4 where each LAN is shown in a corner of the screen with client windows machines (PCs) and window DENAR for that LAN.

10.3 Case Study

Our case studies consider a university exam system. Modules' lecturer prepares a modules' exam in the online module directory. Student can download exams with no restriction or with some restriction based on policies in the case studies below.



```
SIOCADDRT: File exists
Starting domain name service...: bind9named binary missing - not starting.
 failed!
>>> End of maingw specific startup script.


################################################

Lab directory (host): /home/aalzahrani/netkit/Distribted-Lab
Version: <none>
Author:  <none>
Email:   <none>
Web:     <none>
Description:
<none>

################################################

--- Netkit phase 2 initialization terminated ---


maingw login: root (automatic login)
Last login: Mon Oct 29 01:20:19 UTC 2012 on tty1
maingw:~#
```

```
11 566: No such process
.
bound to 192.168.10.100 -- renewal in
>>> End of DENAR1 specific startup scr


################################################

Lab directory (host): /home/aalzahrani
Version: <none>
Author:  <none>
Email:   <none>
Web:     <none>
Description:
<none>

################################################

--- Netkit phase 2 initialization terminated ---


DENAR1 login: root (automatic login)
Last login: Mon Oct 29 01:20:19 UTC 2012 on tty1
DENAR1:~#
```

```
11 566: No such process
.
bound to 192.168.20.100 -- renewal in 131 seconds.
>>> End of DENAR2 specific startup script.


################################################

Lab directory (host): /home/aalzahrani/netkit/Distribted-Lab
Version: <none>
Author:  <none>
Email:   <none>
Web:     <none>
Description:
<none>

################################################

--- Netkit phase 2 initialization terminated ---


DENAR2 login: root (automatic login)
Last login: Mon Oct 29 01:20:19 UTC 2012 on tty1
DENAR2:~#
```

```
11 566: No such process
.
bound to 192.168.30.101 -- renewal in 116 seconds.
>>> End of DENAR3 specific startup script.


################################################

Lab directory (host): /home/aalzahrani/netkit/Distribted-Lab
Version: <none>
Author:  <none>
Email:   <none>
Web:     <none>
Description:
<none>

################################################

--- Netkit phase 2 initialization terminated ---


DENAR3 login: root (automatic login)
Last login: Mon Oct 29 01:20:19 UTC 2012 on tty1
DENAR3:~#
```

```
11 566: No such process
.
bound to 192.168.40.100 -- renewal in 116 seconds.
>>> End of DENAR4 specific startup script.


################################################

Lab directory (host): /home/aalzahrani/netkit/Distribted-Lab
Version: <none>
Author:  <none>
Email:   <none>
Web:     <none>
Description:
<none>

################################################

--- Netkit phase 2 initialization terminated ---


DENAR4 login: root (automatic login)
Last login: Mon Oct 29 01:20:19 UTC 2012 on tty1
```

FIGURE 10.4: Decentralised PBM for DENARs Simulator Screens.

Each student is doing four modules per semester where each module belongs to a different school. Figure 10.5 shows these requirements.

<u>Scenario:</u> Suppose students (*Ali* and *Adam*) are doing modules (*Math1*, *Computer1*, *Accounting1* and *Research1*). Each student can *download* modules' exams which is called (*exam1*) from the online module directory. The (*Math1*) module belongs to *Science* School and (*Computer1*) module belongs to *Technology* School. The (*Accounting1*) module belongs to *Management* School and (*Research1*) module belongs to *Art* School.

The above requirements are formalised in a central policy where it is suitable for the Centralised PBM system. On the other hand, the central policy is decomposed according our decomposition model [See Chapter 7] based on object domains. Suppose there are four domains where the domain means the school that the

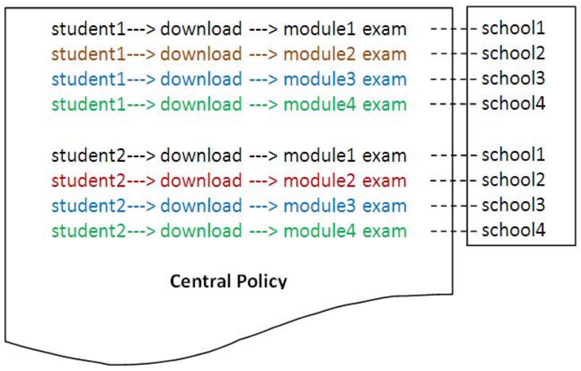

student1---> download ---> module1 exam
student1---> download ---> module2 exam
student1---> download ---> module3 exam
student1---> download ---> module4 exam
student2---> download ---> module1 exam
student2---> download ---> module2 exam
student2---> download ---> module3 exam
student2---> download ---> module4 exam
school1
school2
school3
school4
school1
school2
school3
school4
Central Policy

FIGURE 10.5: Central Policy.

module belongs to. According to the decomposition model the central policy is decomposed into four sub-policies as shown in Figure 10.6.

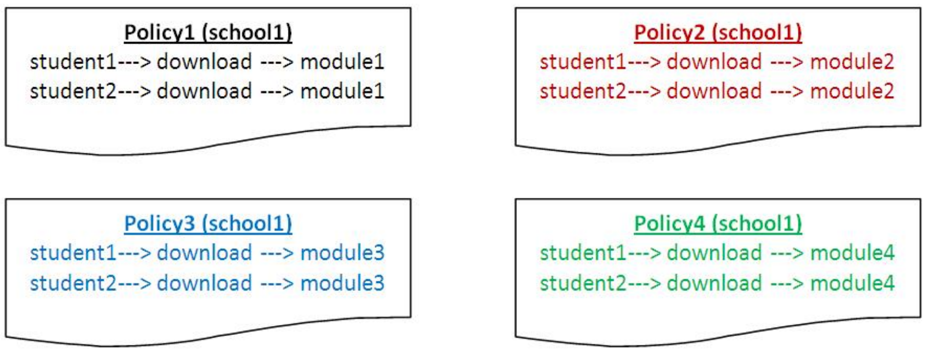

Policy1 (school1)
student1---> download ---> module1
student2---> download ---> module1
Policy2 (school1)
student1---> download ---> module2
student2---> download ---> module2
Policy3 (school1)
student1---> download ---> module3
student2---> download ---> module3
Policy4 (school1)
student1---> download ---> module4
student2---> download ---> module4

FIGURE 10.6: Decomposed sub-policies.

The below case studies show the different restrictions for the above central policy and sub-policies.

The policy :- Allowing (*ali*) to (*download*) resources ((*Math1/exam1*), (*Computer1/exam1*), (*Accounting1/exam1*) and (*Research/exam1*)) without restriction (condition). [See Appendix A.2.1, Listing A.10 for the central policy for Centralised PBM] [See Appendix A.2.1, Listing A.11, A.12, A.13 and A.14 for decomposed sub-policies for Decentralised PBM]

10.3.1 Case Study 1 (Static Policy)

Students (*ali*) is doing module (*Math1*). Student can *download* modules' exam (*exam1*) from the module directory. In addition, for this case there is no restriction for the student to download the exam.

Case study 1 policy :- Allowing student (*ali*) to (*download*) resource (*Math1/exam1*) without restriction (condition). [See Appendix A.2.1, Listing A.15 for the case study 1 policy]

In Centralised PBM, the policy is enforced in the centralised PDP in LAN2 where the resource (*Math1/exam1*) is located in a different domain to the client location domain (*Science School, LAN1*). Therefore, the traffics of authorisation request and authorisation response traffics crosses *LAN1* to *LAN2* as shown in Figure 10.7

.

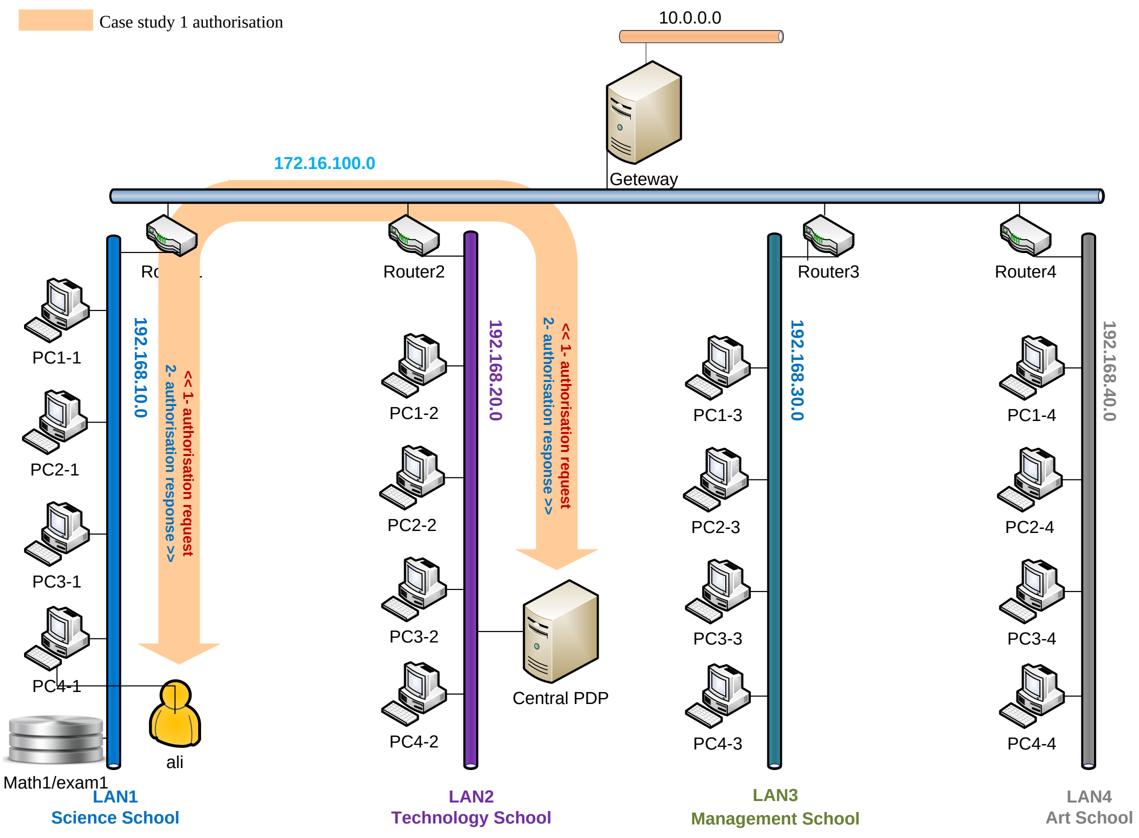


FIGURE 10.7: Case Study 1 (Static Policy) in Centralised PBM.

However, in Decentralised PBM, the policy is enforced in the *DENAR1* in *LAN1* where the resource (*/Math1/exam1*) is located in the same domain (*Science School, LAN1*). Therefore, the traffics of authorisation request and authorisation response traffics are being inside the same LAN or domain as shown in Figure 10.8 .

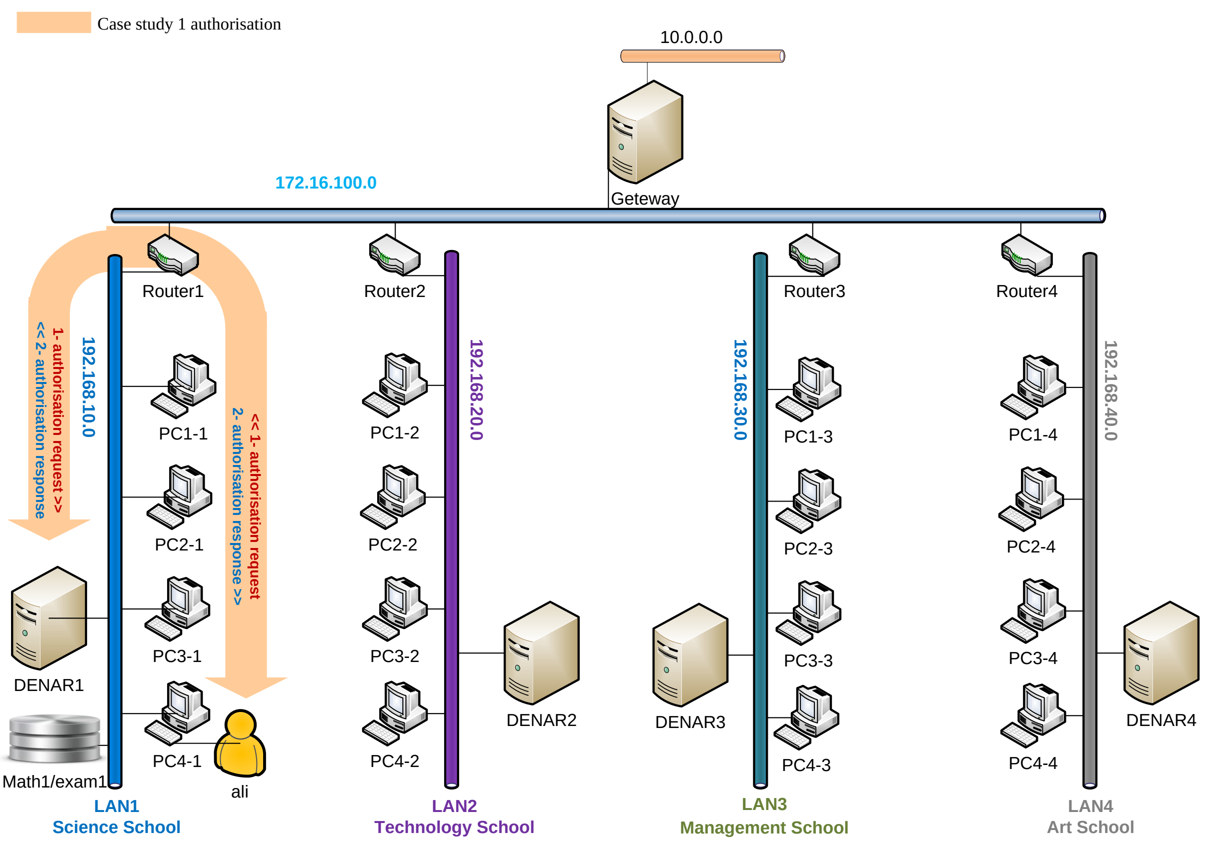


FIGURE 10.8: Case Study 1 (Static Policy) in Decentralised PBM.

10.3.2 Case Study 2 (Local Domains' Dynamic Policy)

In this case, a policy is enforced with condition of restricting access (*download*) to the (*Computer1*) exam. The condition shows the requirements that must evaluate to true before performing the action (*download*). The main objective restricting the access is that all conditional attributes must be evaluated in a PDP against their values which are located in a Policy Information Point (PIP).

Case study 2 policy :- Allowing (*ali*) to (*download*) resource (*Computer1/exam1*) with restriction (condition) where *(ali) belongs to Foundation group, the (Computer1/exam1) is classified as OpenBook, the download must be done before 01/Jan/13* *and the (Computer1/exam1) has not been downloaded before.* [See Appendix A.2.1, Listing A.16 for the case study 2 policy].

Listing 10.1 illustrates conditional attributes that are involved in the policy for this case study.

```
<Condition FunctionId="PDPOverlay_Constraint_Checker">
 <SubjectCondition DataType="http://www.w3.org/2001/XMLSchema#string">Foundation
 </SubjectCondition>
 <ObjectCondition DataType="http://www.w3.org/2001/XMLSchema#string">
 (/Computer1/exam1)=OpenBook</ObjectCondition>
 <EnvironmentCondition DataType="http://www.w3.org/2001/XMLSchema#string">
 01/Jan/2013</EnvironmentCondition>
 <EventCondition DataType="http://www.w3.org/2001/XMLSchema#string">
 NotDone(ali,(/Computer1/exam1), Download)</EventCondition>
</Condition>
```

Listing 10.1: Case study 2 policy in XACML

In Listing 10.1, the SubjectCondition attribute means that the student (*ali*) belongs to Foundation group and ObjectCondition attribute means that exam file (*/Computer1/exam1*) must be classified as OpenBook. Moreover, the EnvironmentCondition attribute means that exam must be downloaded before *01/Jan/2013* and EventCondition attribute means that exam (*/Computer1/exam1*) must not be done before by the same student.

In Centralised PBM for this case, the policy is enforced and also all conditional attributes in the policy are evaluated against their values in the centralised PIP by the CentralPDP. Figure 10.9 shows the authorisation traffic where it crosses LAN1 to LAN2 to be enforced in CentralPDP.

However, in Decentralised PBM for this case, the policy is enforced by the *DENAR1* and also all conditional attributes are evaluated against their values in the local PIP (*PIP1*) for domains' DENAR (*DENAR1*). In this case, the student (*ali*) requests downloading the exam (*/Computer1/exam1*) where it is located in (*Technology domain*), thus, this policy is enforced by *PDP2* in *DENAR2* that is configured for the Technology domain. The conditional attributes (ObjectCondition) and (EventCondition) values are retrieved from the local PIP for *DENAR2* because they are located in the same domain as the object (*/Computer1/exam1*). Figure 10.10 shows the authorisation traffic where it crosses *LAN1* to *LAN2* to be authorised in *DENAR2*.

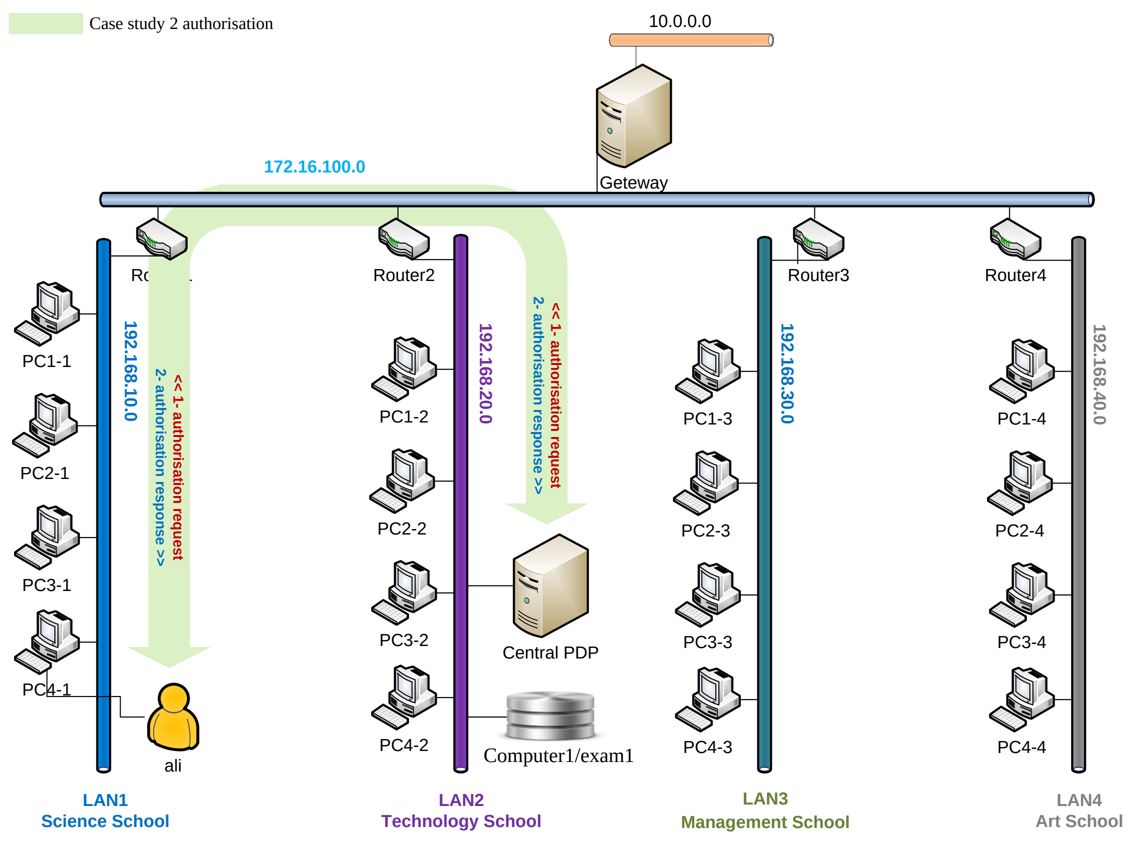


FIGURE 10.9: Case Study 2 (Dynamic Policy) in Centralised PBM.

10.3.3 Case Study 3 (Remote Domains' Dynamic Policy)

In this case, a policy is enforced with condition involving restricting the access (download) to the (*Accounting1*) exam. Listing 10.2 illustrates conditional attributes involved in a policy.

Case study 3 policy :-Allowing (*ali*) to (download) resource (*Accounting1/exam1*) with restriction (condition) where *(ali) belongs to Foundation group, the (Research1/exam1) is classified as MutipleChoice, the download must be done before 01/Jan/13 and the exam (Research1/exam1) is downloaded before by the same student.* [See Appendix A.2.1, Listing A.17 for the case study 3 policy].

```
<Condition FunctionId="PDPOverlay_Constraint_Checker">
 <SubjectCondition DataType="http://www.w3.org/2001/XMLSchema#string">Foundation
 </SubjectCondition>
 <ObjectCondition DataType="http://www.w3.org/2001/XMLSchema#string">
 (Research1/exam1)=MutipleChoice</ObjectCondition>
 <EnvironmentCondition DataType="http://www.w3.org/2001/XMLSchema#string">
```

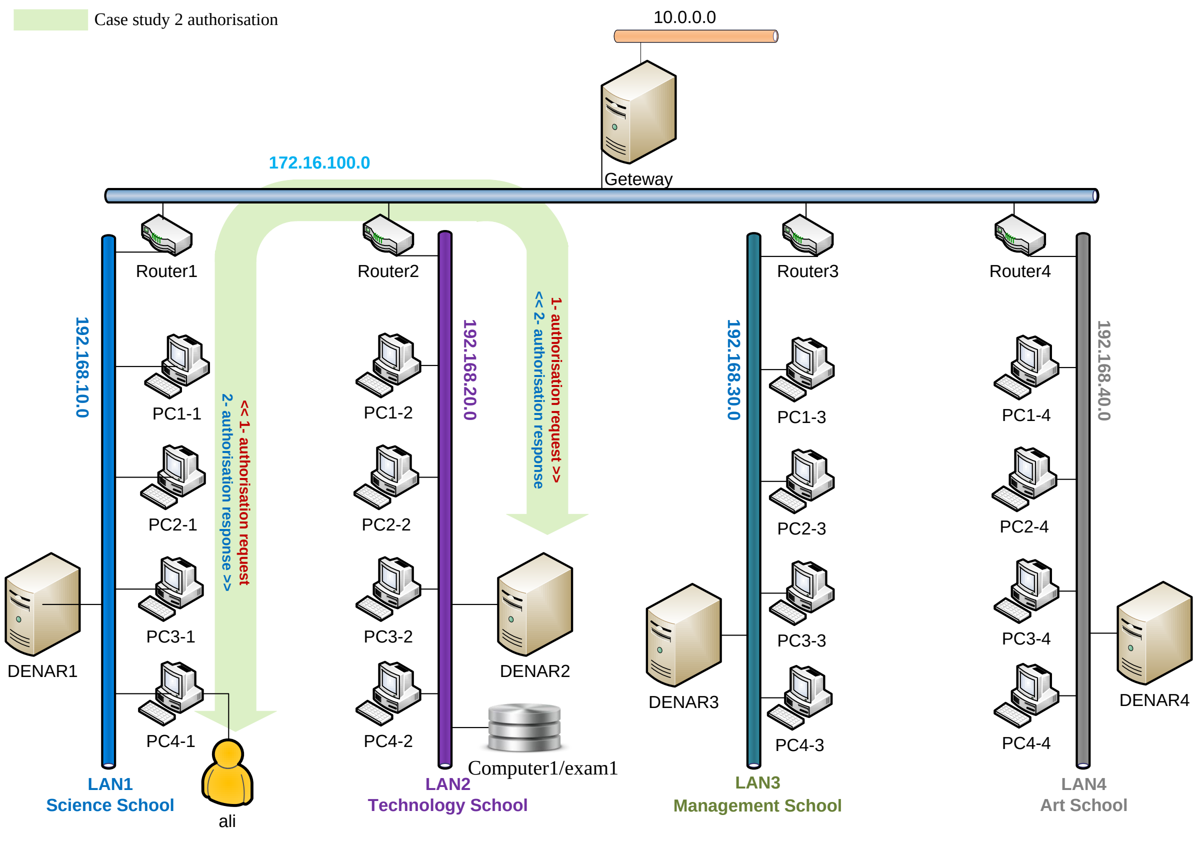


FIGURE 10.10: Case Study 2 (Dynamic Policy) in Decentralised PBM.

```
 01/Jan/2013</EnvironmentCondition>
 <EventCondition DataType="http://www.w3.org/2001/XMLSchema#string">
 Done(ali,(Research1/exam1), Download)</EventCondition>
</Condition>
```

LISTING 10.2: Case study 3 policy in XACML

In listing 10.2, the SubjectCondition attribute means that the student must belong to Foundation group and ObjectCondition attribute means that exam file (*Research1/exam1*) must be classified as *MutipleChoice*. Moreover, the Environment-Condition attribute means that exam must be downloaded before *01/Jan/2013* and EventCondition attribute means that exam (*Research1/exam1*) must be done before by the same student.

In Centralised PBM, the policy is enforced in the CentralPDP and all conditional attributes in the policy are evaluated against their values in the centralised PIP. Figure 10.11 shows these traffics.

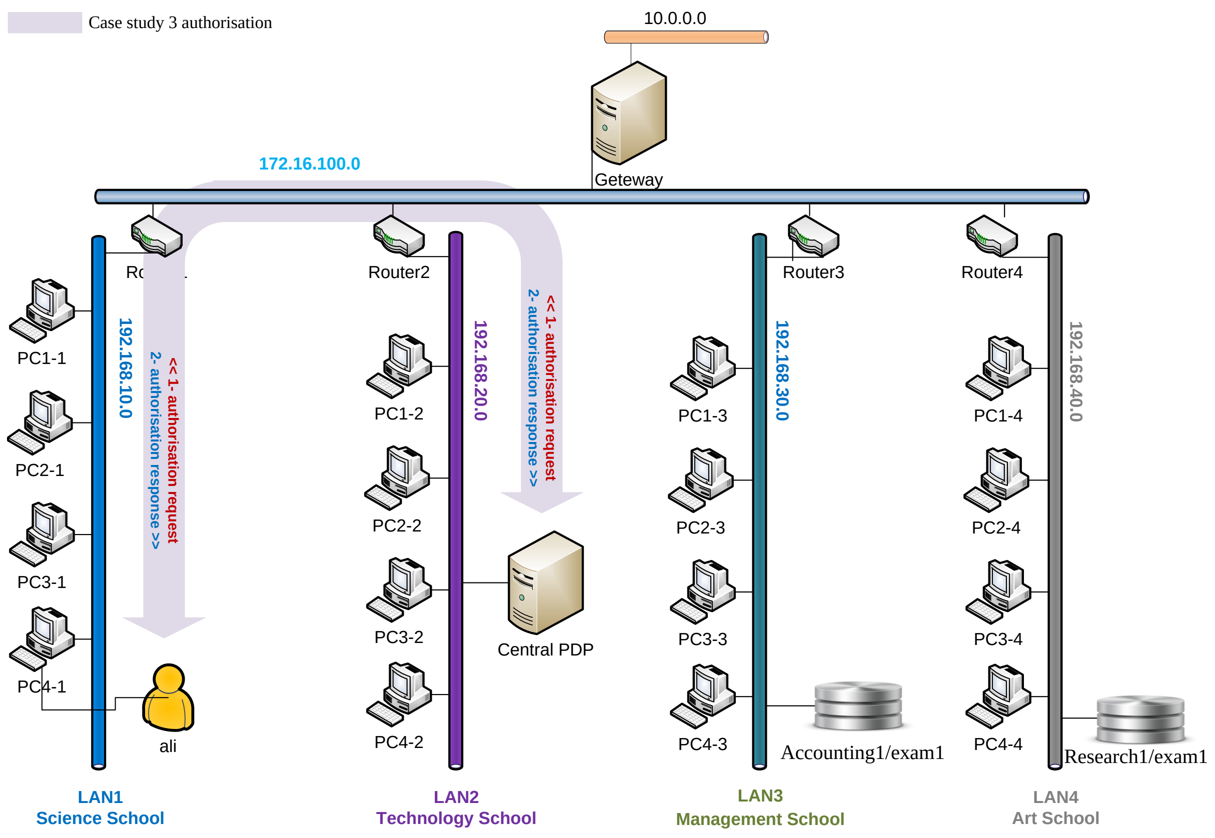


FIGURE 10.11: Case Study 3 (Remote Domains' Dynamic Policy) in Centralised PBM.

However, in Decentralised PBM, the policy is enforced by the *DENAR3* and all conditional attributes must be evaluated against their values in the remote PIP (*PIP4*) of *DENAR4*. In this case, the student (*ali*) requests downloading exam (*Accounting1/exam1*) where it is located in (*Management domain*), thus, this policy is enforced by the *PDP3* in *DENAR3* that is configured for the Management domain. The conditional attributes (ObjectCondition) and (EventCondition) values are retrieved from the remote *PIP4* for *DENAR4* because they are not located in the same domain of the object (*Research1/exam1*). Therefore, there are two traffic the first one (in green arrow) for authorisation request and response between the subject and the *PDP3* in *DENAR3*. The second traffic (in orange arrow) for the coordination for retrieving condition values where it is between the *PDP3* in *DENAR3* and the *PIP4* in *DENAR4*. Figure 10.12 shows the above traffics.

To meet the efficiency target for enforcing the history-based policy [3] (dynamic policy) which is involved in this case study, we design the Pull Model that is

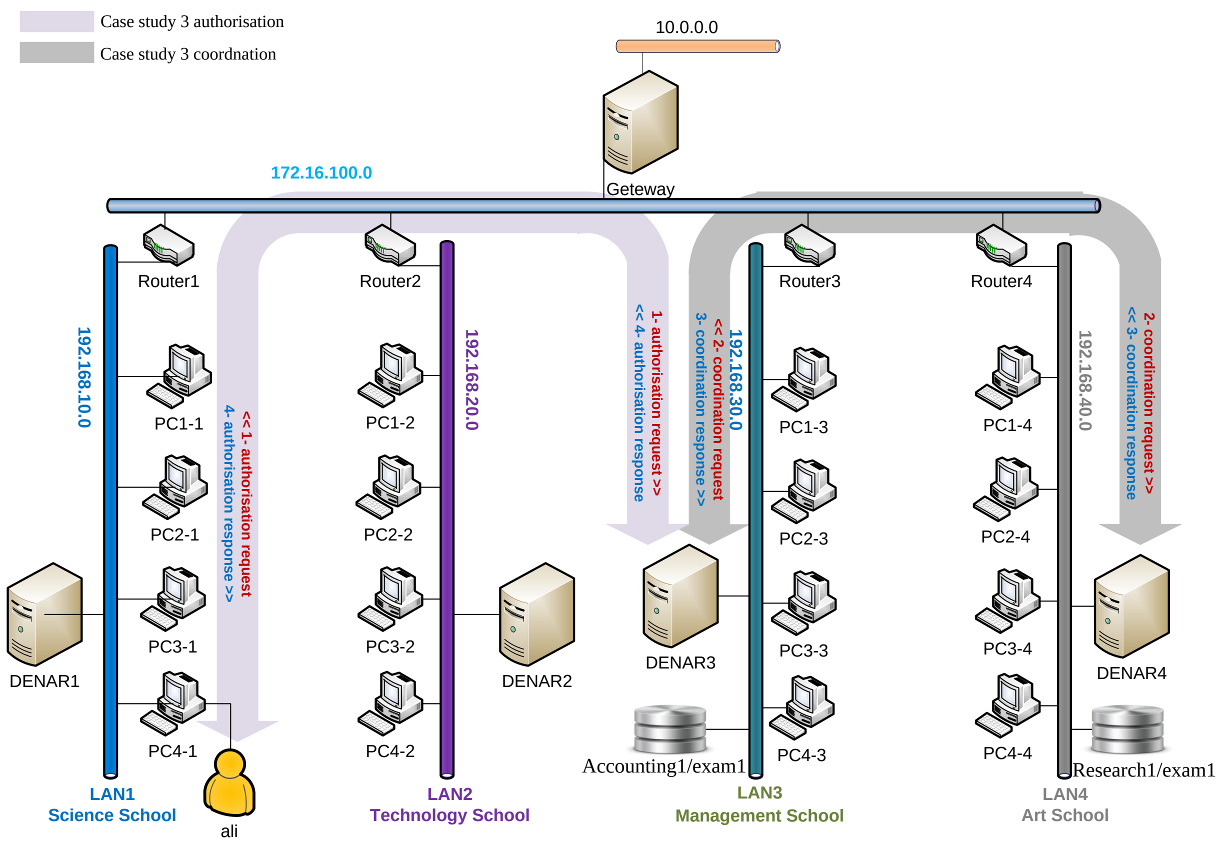


FIGURE 10.12: Case Study 3 (Remote Domains' Dynamic Policy) in Decentralised PBM.

designed for retrieving any condition value that is located in remote PIPs. [For Pull Model see Chapter 8, Section 8.3.1].

On the other hand, In Push Model, DENAR can "push" a new condition value for the current decision to other PIPs required for future decisions [this is done by obligation, for Push Model see Chapter 8, Section 8.3.2].

Decentralised PBM can involve Pull or Push Models in the policy for the coordination. Pull Model is designed for retrieving condition values where they are being in the *PIP4*. On the other hand, the Push Model is designed to push condition values to be stored in *PIP4* i.e. in this case study, the EventCondition value is updated and stored in *PIP4* after the subject (*ali*) downloads (*Research1/exam1*) and this is done by the obligation.

Moreover, for the Push Model in this case study, storing EventCondition value in *PIP3* saves authorisation time and traffic for the next authorisation decision. This technique is modelled in this research but not implemented and evaluated.

Therefore, this could potentially be a consideration in future work and revisions of the prototype.

10.4 Evaluation

This section evaluates the Centralised PBM and Decentralised PBM approaches in the provided case studies and the simulators. Following subsection discusses the evaluated factors for the DENARs' prototype that is designed for the Decentralised PBM approach. Performance, security, manageability and resilience are the factors in following subsections. The

10.4.1 DENARs' Performance

While the security in the access control systems is essential, the performance of enforcing security policy is also essential. The efficiency is currently based on the number of authorisation traffic. The authorisation traffic is discussed by a comparison between the Centralised PBM and Decentralised PBM simulators. The time of access control decision making is one of the important factors in critical systems (e.g. Military) where it is recommended as a future work. The reason for not measuring the response time and delays for the access control decision making is that the experiment simulated and evaluated in a single machine where virtual machines are simulated by Netkit.

10.4.1.1 Network Traffic

The network of DENARs reduces the network traffic bottlenecks for access control decision making in comparison with a centralised PDP. According to the case studies mentioned in section 10.3, this section discusses the network traffic in both Centralised PBM and the Decentralised PBM simulators identified at in section 10.2.

In the following we present two network traffic measurements for the first case study and the second case study mentioned at the beginning of this Chapter. Each client (PC) sends 50 requests to download a file with size (2 bytes) where each request is being authorised first before the action (download) is performed.

The network traffic measures the request, response hence small file transfer traffic between the client and its proper PDP. We show the network traffic comparison by the number of packets and the number of bytes for each client and PDP in both Centralised PBM and the Decentralised PBM simulators.

- **Static Policy (Case Study 1)**

  <u>Traffic for Centralised PBM:</u> In the Centralised PBM simulator, the enforcement of the first case study (Static Policy) is analysed where all authorisation requests from all clients in the network are sent to the centralised PDP to enforce the first case study policy and respond back with the authorisation decision. The static policy in this case study means there is no restriction for performing the action.

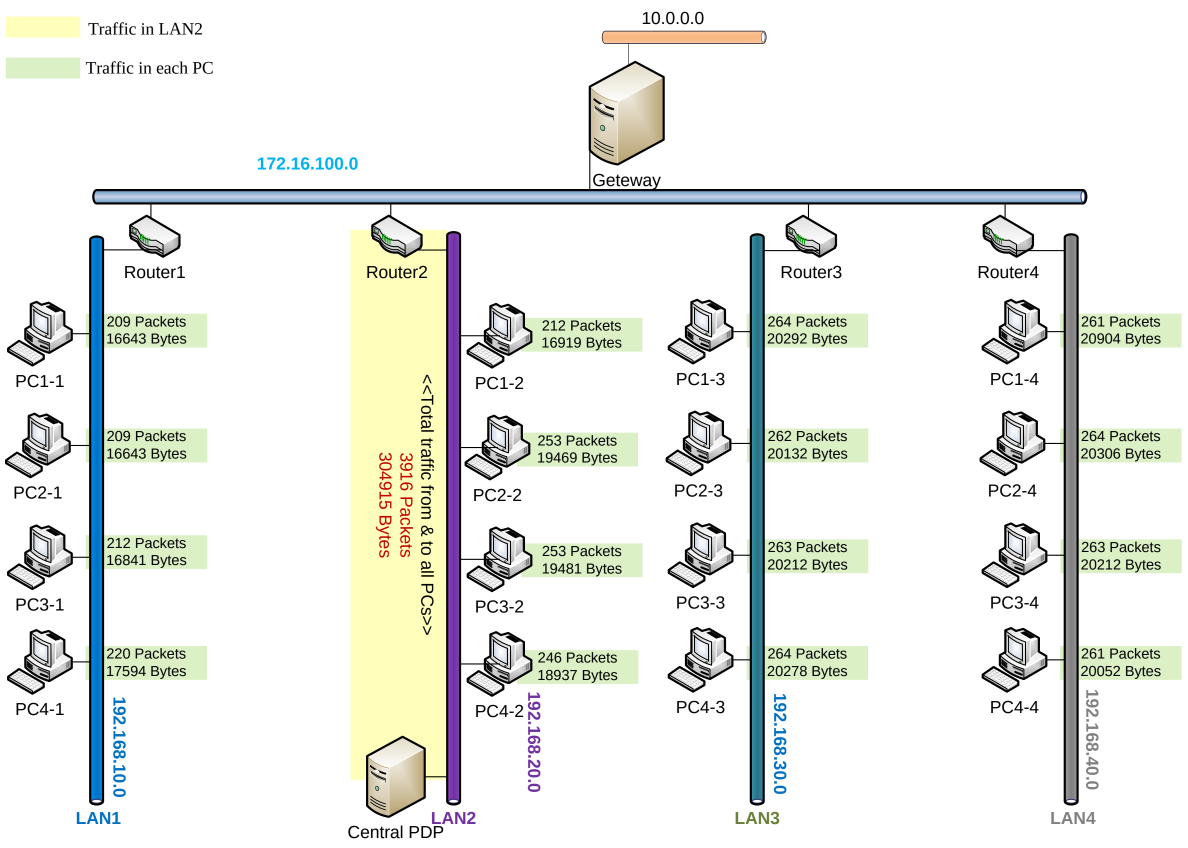


Figure 10.13: Centralised PBM Traffic for Case Study 1 (Static Policy).

<u>Traffic for Decentralised PBM:</u> In the Decentralised PBM simulator, the enforcement of the first case study (Static Policy) is analysed. The first clients (PC1) (those in the orange line) for each LAN send their requests to

DENAR1 and also the second clients (PC2) (those in the green line) for each LAN send their requests to the DENAR2. The third clients (PC3) (those in the blue line) for each LAN send their requests to the DENAR3 and also the fourth clients (PC4) (those in the pink line) for each LAN send their requests to the DENAR4.

Measurements Discussion:

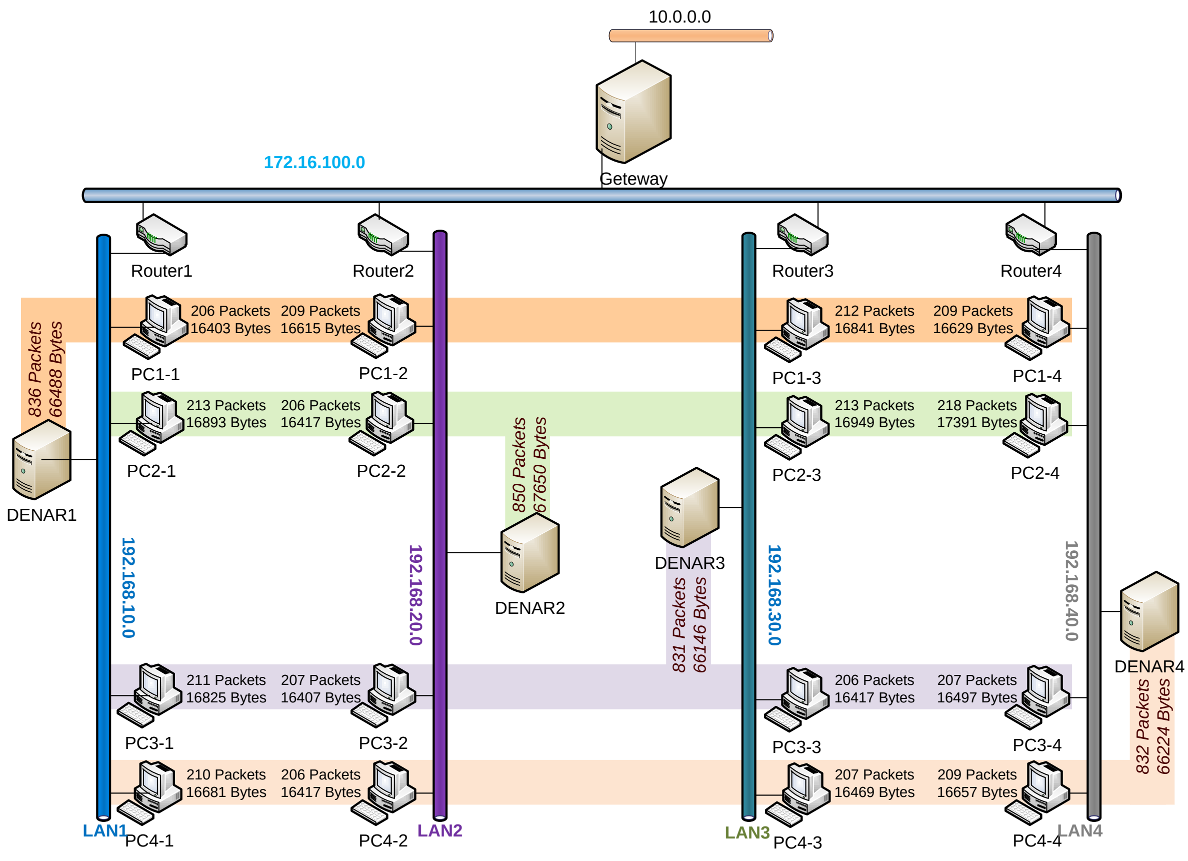


FIGURE 10.14: Decentralised PBM Traffic for Case Study 1 (Static Policy).

Figure 10.13 shows the traffic result on the CentralPDP machine in LAN 2 for the Centralised PBM similator. The clients (PCs) sent in total 800 requests to the CentralPDP machine which produce 3915 packets (304915 bytes) those are enter and exit to/from the CentralPDP machine. On the other hand, in Figure 10.14, the authorisation traffic is distributed between the DENARs which decrease the number of packets on LAN2 to 850 (67650 bytes). While each DENAR in the Decentralised PBM processes in total 200 requests, there is no traffic overload in a particular LAN or machine.

- **Dynamic Policy (Case Study 2)**

Traffic for Centralised PBM: In the Centralised PBM simulator, the enforcement of the second case study (Dynamic Policy) is analysed where all authorisation requests from all clients in the network are sent to the centralised PDP to enforce the second case study policy and response back the authorisation decision. The dynamic policy in this case study means there are restrictions (conditions) for performing the action. The condition shows that the requirements must be evaluated to true before performing the action. The main objective restricting the access is that all conditional attributes must be evaluated by a PDP against their values located in a PIP.

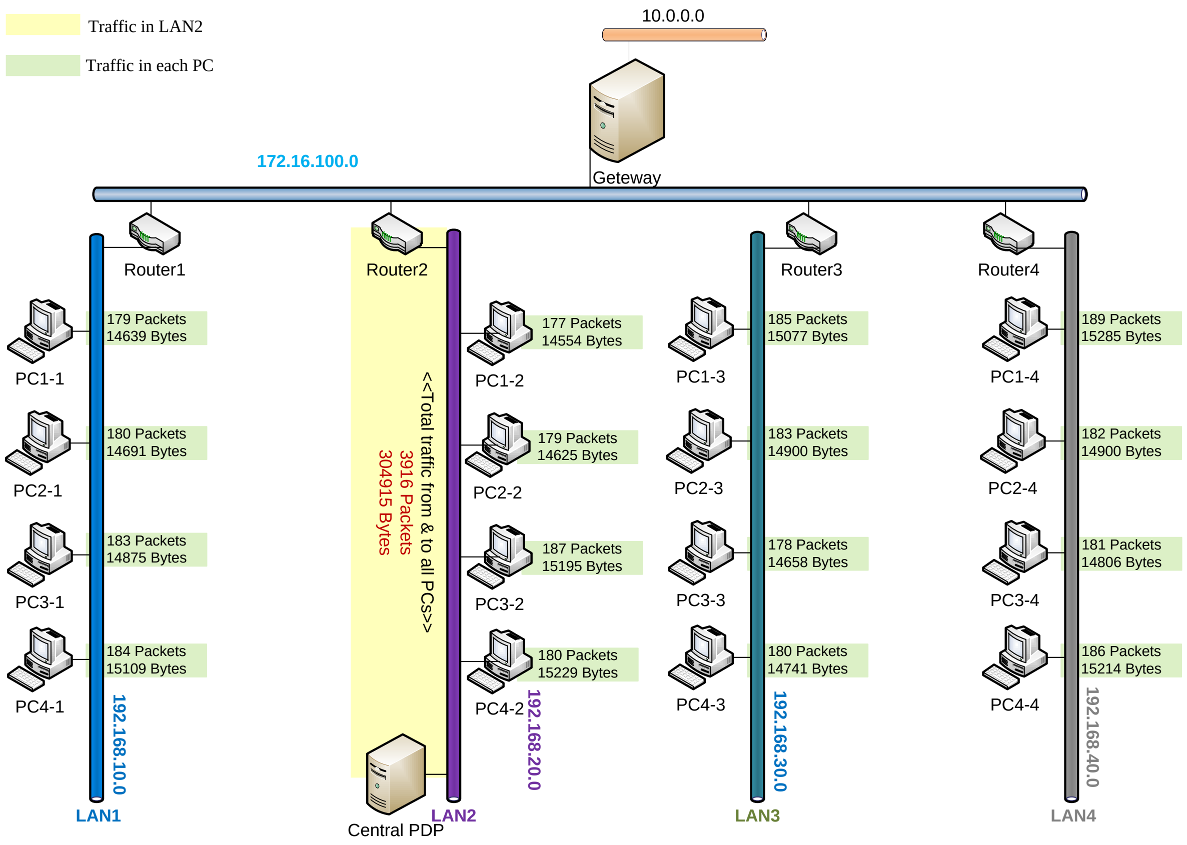


FIGURE 10.15: Centralised PBM Traffic for Case Study 2 (Dynamic Policy).

Traffic for Decentralised PBM: In the Decentralised PBM simulator, the enforcement of second case study (Dynamic Policy) is analysed. All conditional attributes must be evaluated by a PDP against their values which are located in the PIP of the same DENAR.

Measurements Discussion:

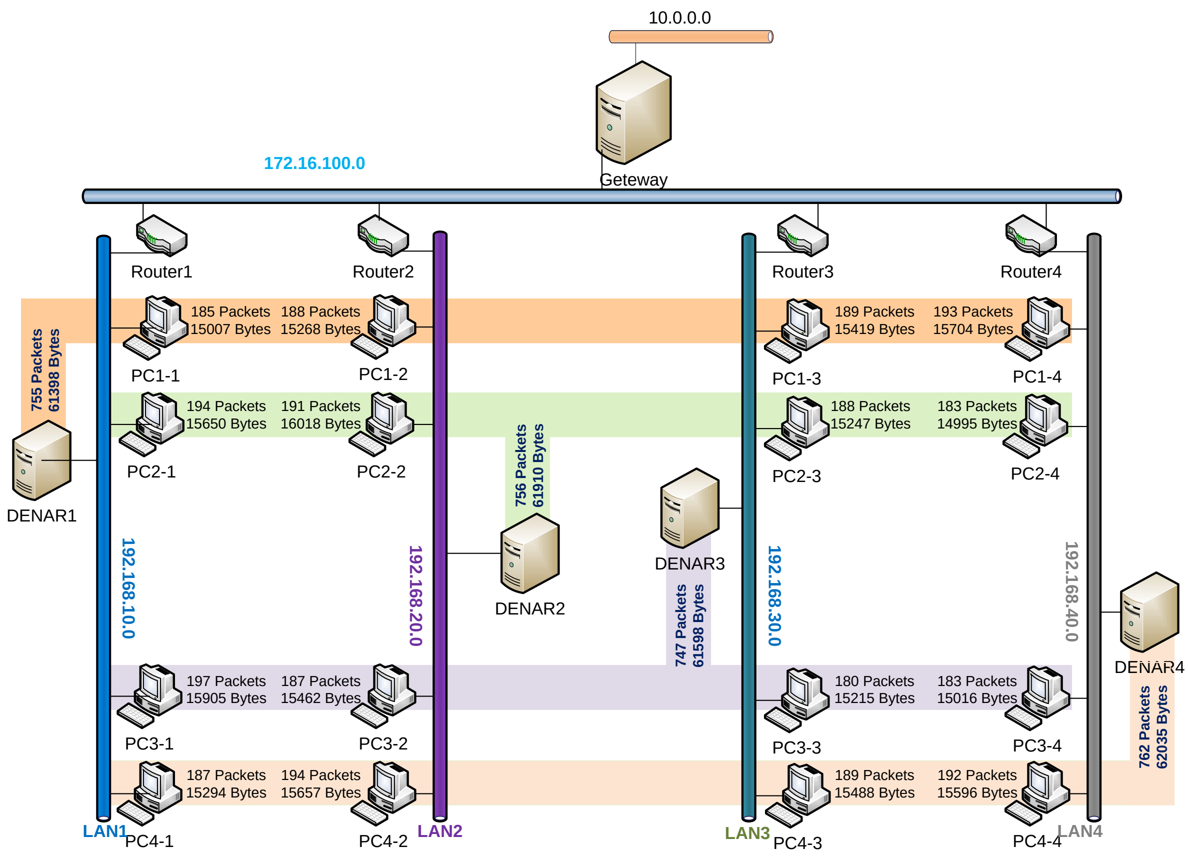


FIGURE 10.16: Decentralised PBM Traffic for Case Study 2 (Dynamic Policy).

Figure 10.15 shows the traffic result on the CentralPDP machine in LAN 2 for the Centralised PBM similator. The clients (PCs) sent in total 800 requests to the CentralPDP machine which produce 2913 packets (238498 bytes) those are enter and exit to/from the CentralPDP machine. On the other hand, in Figure 10.16, the authorisation traffic is distributed between the DENARs which decrease the number of packets on LAN2 to 756 (61910 bytes). While each DENAR in the Decentralised PBM is processing in total 200 requests, there is no traffic overload in a particular LAN or machine.

10.4.2 DENARs' Security

The main objective of this research is providing the Decentralised PBM approach the same security level as the Centralised PBM. The correct enforcement of security policy and involving the dynamic policy are essential in the current access control systems. The following discusses the enforcement functional behaviour and the ability of involving dynamic policy in our Decentralised PBM approach.

10.4.2.1 Enforcement Functional Behaviour

For Decentralised PBM, enforcement functional behaviour is defined as the replication of policy decisions that would have been made by a Centralised PBM in the same scenario.

During the access control testing of the three case studies (Section 10.3) in both Centralised PBM and Decentralised PBM network simulators (Section 10.2), the same tests were carried out.

Table 10.1: Enforcement Functional Behaviour of Enforcement for Centralised PBM and Decentralised PBM

Simulator	Case study	Result
Centralised PBM	1	* The policy is deployed in the centralised PR for the centralised PDP. * The policy is enforced by the centralised PDP. * The policy is enforced correctly.
	2	* The policy is deployed in the centralised PR for the centralised PDP. * The policy is enforced by the centralised PDP. * The condition values are retrieved from the centralised PIP for the centralised PDP. * The policy is enforced correctly.
	3	* The policy is deployed in the centralised PR for the centralised PDP. * The policy is enforced by the centralised PDP. * The condition values are retrieved from the centralised PIP for the centralised PDP. * The policy is enforced correctly.
Decentralised PBM	1	* The policy is deployed in PR1 for DENAR1. * The policy is enforced by PDP1 in DENAR1. * The policy is enforced correctly.
	2	* The policy is deployed in PR2 for DENAR2. * The policy is enforced by PDP2 in DENAR2. * The condition values are retrieved from PIP2 in DENAR2. * The policy is enforced correctly.
	3	* The policy is deployed in PR3 for DENAR3. * The policy is enforced by PDP3 in DENAR3. * The object and event condition values are retrieved from PIP4 in DENAR4 thus the coordination between DENAR3 and DENAR4 is required to evaluate object and event condition values. * The policy is enforced correctly.

All results satisfy the enforcement functional behaviour which mean both approaches result in the same enforcement decision for those policies in the case studies. The table 10.1 shows the functional behaviour of the policies enforcements in both simulators.

10.4.3 DENARs' Manageability

According to the Distributed Policy Enforcements Architecture (DENAR) (Chapter 5), we consider qualitatively administrative cost and resource utilisation properties of the DENAR architecture.

10.4.3.1 Resource Utilisation

DENARs in our project can arrive at decisions locally and thus save time and network resources.

However, given the number of policies that may be present in the Centralised PBM, the PDP, PIP, PR must have sufficient computing and storage resources to store and enforce these policies, which may become a bottleneck. In our Decentralised PBM, the distribution of PDPs, PIPs and PRs in a collaborative DENARs optimise the policy enforcement and network resource utilisation functionality based on available bandwidth and computing infrastructure.

10.4.3.2 Administrative Cost

- Simplicity: In DENAR, administration of Decentralised PBM policies is eliminated. According to our case studies above, changing policies over semesters is by school staff who are familiar with the school rules and regulation. Thus, managing policy repositories (PRs) individually by schools staff will save time and effort in comparison with the Centralised PBM system. Also, it provides privacy for a policy that located in a domain (school), thus, no disclosure from other administrators in other domains.

- Consistency: DENARs network topology remains consistent in case a new policy is added or changed in any particulate domain. The policy can be analysed and decomposed without changing of DENARs network topology.

A centralised backup DENAR can be configured by a system administrator in case of the reconfiguring the DENARs network with new policies adding or changing. Clearly, the DENARs network operates one centralised DENAR (backup DENAR) only during the reconfiguration and that for a short time. The backup DENAR has its PR that stores the central policy (all rules before the decomposition into sub-policies), its PIP that stores condition values. [See Chapter 5, Section 5.3.1]

- Flexibility: Adding or removing resources from the system does not affect the DENARs network topology. Therefore, only adding a new policy for that resource has been added and its attribute without reconfiguring the DENARs network topology.

10.4.4 DENARs' Resilience

Our research proposes mechanisms that improve the resilience against network failures in access control for distributed information systems.

In case of the failure of the policy enforcement in centralised PDP, all the devices on the network may cease to function. Clearly, an authorisation request to a particular resource or service does not respond. In practice, this is rarely the case as there is at least one alternate backup provided in . The problem is indeed if centralised PIP is involved where it remains the centralised authorisation decision making which still has the performance drawback. Moreover, conditional attributes are no longer observable due to failure of the centralised PIP. Finally, involving distributed PDPs and PIPs without coordination between PDPs to retrieve dynamic attributes from other PIPs where enforcing dynamic policies is a bottleneck.

In Figure 10.17 and table **??**, we discuss some potential failure scenarios in the centralised PDP network that may affect authorisation.

Table 10.2: Enforcement and Coordination Failures for Centralised PBM network

Failure No.	Case study	Enforcement and Coordination Affection

f1	1, 2 and 3	f1 affects only authorisations sent from LAN1 machines where the failure isolates all resources in LAN1 to reach the centralised PDP.
f2	1, 2 and 3	f2 affects authorisation in the entire network where there is no communication with the centralised PDP because all resources in LAN2 are unreachable. (LAN2 is down)
f3	1, 2 and 3	f3 affects authorisation in the entire network because the centralised PDP is down or unreachable.
f4	1, 2 and 3	f4 does not affect the authorisation for all mentioned case studies. However, f4 affects only authorisations sent from LAN4 because all resources in LAN2 are unreachable. (LAN4 down)

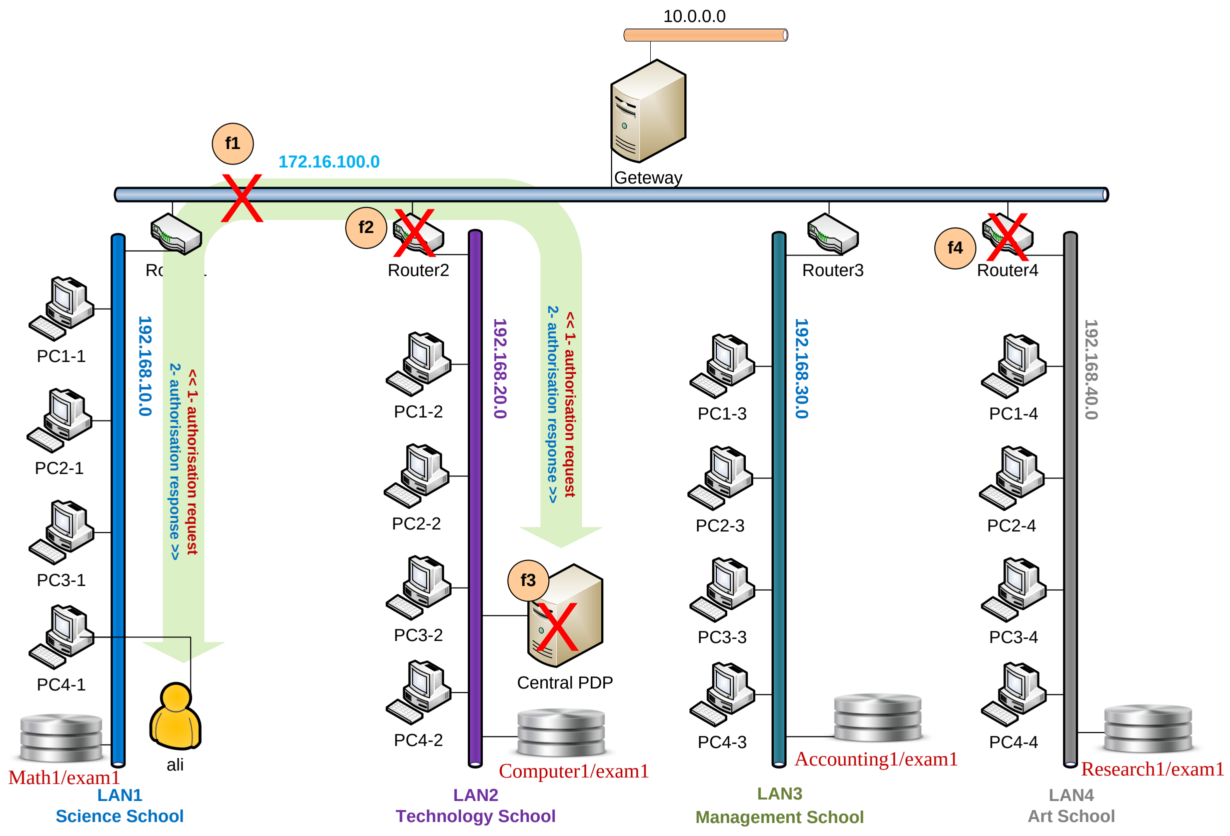


FIGURE 10.17: Failures in Centralised PBM.

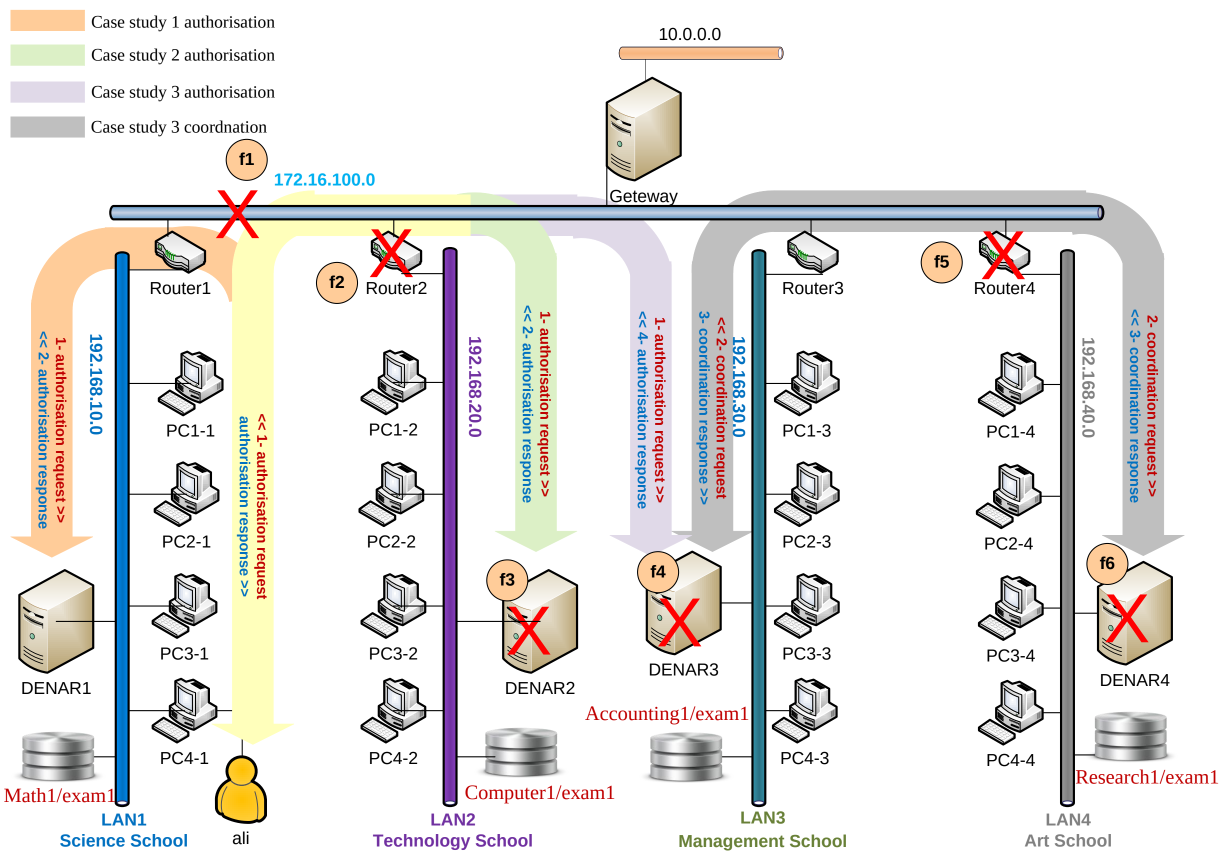

Case study 1 authorisation
Case study 2 authorisation
Case study 3 authorisation
Case study 3 coordnation
10.0.0.0
Geteway
172.16.100.0
f1
f2
f3
f4
f5
f6
Router1
Router2
Router3
Router4
1- authorisation request >>
<< 2- authorisation response
<< 1- authorisation request
authorisation response >>
1- authorisation request >>
<< 2- authorisation response
1- authorisation request >>
<< 4- authorisation response
<< 2- coordination request
3- coordination response >>
2- coordination request >>
<< 3- coordination response
192.168.10.0
192.168.20.0
192.168.30.0
192.168.40.0
PC1-1
PC2-1
PC3-1
PC4-1
PC1-2
PC2-2
PC3-2
PC4-2
PC1-3
PC2-3
PC3-3
PC4-3
PC1-4
PC2-4
PC3-4
PC4-4
DENAR1
DENAR2
DENAR3
DENAR4
ali
Math1/exam1
Computer1/exam1
Accounting1/exam1
Research1/exam1
LAN1
Science School
LAN2
Technology School
LAN3
Management School
LAN4
Art School

FIGURE 10.18: Failures in Decentralised PBM.

The collaborative DENARs are designed to fulfill resilience in less policy enforcement failure and resource utilisation. The collaborative DENARs means that DENARs query each other for authorisation decision purpose (coordination).

Figure 10.18 and table 10.3 show the supposed failures in the DENARs network that may effect the authorisation.

Table 10.3: Enforcement and Coordination Failures for Centralised PBM network

Failure No.	Case study	Enforcement and Coordination Affection
f1	1	f1 does not affect case study 1 authorisation because the DENAR1 responsible for the requested authorisation based on the location of the resource (Math1/eaxm1)
	2	f1 affects authorisation because the authorisation request is sent from LAN1 and the resource (Computer1/exam1) is located in LAN2. Therefore, the policy is enforced by DENAR2 where the DENAR2 is unreachable.
	3	f1 affects authorisation because the authorisation is sent from LAN1 and the resource (Accounting1/exam1) is located in LAN3. Therefore, the policy is enforced by DENAR3 where the DENAR3 is unreachable.
f2 and f3	1	f2 and f3 do not affect authorisation for the case study 1 because it is enforced locally by DENAR1.
	2	f2 and f3 affect authorisation for case study 2 because the authorisation is sent from LAN1 and the resource (Computer1/exam1) is located in LAN2. Therefore, the policy is enforced by DENAR2 where DENAR2 is unreachable.
	3	f2 and f3 do not affect authorisation for case study 3 because the policy is enforced by DENAR3 and the coordination values are being remotely in DENAR4.

f4	1	f4 does not affect authorisation for case study 1 because it is enforced locally in DENAR1.
	2	f4 does not affect authorisation for case study 2 because it is enforced by DENAR2.
	3	f4 affects authorisation for case study 3 because the policy is enforced by DENAR3 and the coordination values are requested by DENAR3 from DENAR4 where the DENAR3 is down.
f5 and f6	1	f5 and f6 do not affect authorisation for case study 1 because it is enforced locally by DENAR1.
	2	f5 and f6 do not affect authorisation for case study 2 because it is enforced by DENAR2.
	3	f5 and f6 affect authorisation for case study 3 because the policy is enforced by DENAR3 and the coordination values are requested by DENAR3 from DENAR4 where the DENAR4 is unreachable.

In DENARs network, a backup DENAR can be configured by the system administrator. The backup DENAR is utilised in case of DENARs failure. In addition, it can be used in case of reconfiguring the DENARs network with new adding or changing DENARs topology. Clearly, the network operates one centralised PDP only during the reconfiguration which is the backup DENAR for access control. The backup DENAR has its PR that stores the central policy (all rules before the decomposition into sub-policies), and its PIP that stores conditional attributes.

The centralised Policy Based Management (PBM) is more efficient when the length of the authorisation decision and the authorisation traffic to performing any service are not critical and the service can be provided later, e.g. banking services. On the other hand, in such systems, when the authorisation decision response time and the traffic are being important, the Decentralised PBM becomes more efficient, e.g. Military and National Security systems.

10.5 Summary

This Chapter described the simulators of the Centralised PBM and Decentralised PBM for both static and dynamic policies. The simulators are currently available as a prototype implementation that shows the feasibility of the enforcement involved.

The Chapter discussed three case studies that describe how static and dynamic policies are enforced in Centralised PBM and Decentralised PBM simulators. The first case study was provided to show how the static policy (independent policy) is enforced. The second case study showed the enforcement of the dynamic policy (locally domain dependent policy) where the coordination values are located in the same PIP of DENAR that enforce the policy. The third case study showed enforcement of dynamic policy (remote domain dependent policy) where the coordination values are located in different PIPs of the DENARs that enforce the policy.

The core aim of this work was to provide a Decentralised PBM framework for enforcing static and dynamic policies. This Chapter analysed and evaluated the proposed framework against the Centralised PBM approach.

The first factor, performance was evaluated by measuring the authorisation network traffic. The authorisation traffic is analysed to show how the centralised approach is affected by the traffic and can be a bottleneck for the authorisation response. Thus, we can observe that the decentralise approach is more efficient and does avoid a single point of failure.
The second evaluation factor discussed is the functional behaviour of enforcing the case studies policies in both centralised and decentralise approaches and the ability of involving the dependent policy in the decentralised approach.
Thirdly, the discussion of the administrative cost and resource utilisation are provided in the manageability factor. The Decentralised PBM is more efficient and manageable in contrast with the centralised approach. The Decentralised PBM

provides re-propagation and recovery techniques that improve the authorisation decision making to avoid the delay and failure.
Finally, the resilience of involving a network of DENARs in Decentralised PBM approach illustrated the effectiveness countermeasure of some network failures, i.e those facing the centralised approach.

The following Chapter introduces the final results, conclusion of our research and views for future work.

Chapter 11

Conclusion and Future Work

This chapter provides a summary of the work presented in this thesis, and identifies what has been achieved. We conclude with future work.

11.1 Summary of the Thesis

The thesis presented a new framework for Decentralised Policy Based Management (PBM) that supports the distribution of Policy Decision Points (PDPs) to enforce static and dynamic policies. The framework illustrated a practical workflow to achieve the enforcement of security policy enforcement for distributed systems (Chapter 4). The Distributed Enforcement Architecture (DENAR) is designed to include multiple PDPs and other DENAR components. The components, their function and interaction are described. The deployment of DENARs in Decentralised PBM is introduced where the collaboration between them is facilitated by the PIPcoordinator and PDPcoordinator. In addition, re-propagation and recovery techniques are described for DENARs network (Chapter 5).

Chapter 6 presented the DENAR analysis which is based upon ITL. The DENAR analysis provided policy dependency and the semantics of distributed policy enforcements, viz. allowing for the enforcement of static and dynamic policies in decentralised manner where multiple enforcements can be coordinated. As part of the research contribution, we demonstrated how DENARs collaborate to enforce static and dynamic policies in a decentralised manner to provide resilient enforcement.

Chapter 7 detailed the policy decomposition model that considers both static and dynamic policies. Fragmentation and refinement methods are described in SANTA syntax and the algorithms for the policy decomposition. The policy deployment model is detailed in algorithm. The advantage of this policy decomposition model is that it allows the coordination and collaboration between DENARs that provides enforcement functional behaviour of security policy with respect to centralised enforcement.

The DENAR enforcement and coordination models were presented Chapter 8. The coordination need is clear for dynamic policy in distributed DENARs for access control decision. Push and Pull models are proposed and detailed to implement the coordination and collaboration between DENARs. The access control decision making is improved by identifying and proposing local domain decision and remote domain decision.

The prototype implementation of the presented approach is based on XACML policy language and Netkit simulator. The Decentralised Policy Based Management (PBM) framework is however largely language independent and can potentially be applied to other policy languages.

Finally, the thesis demonstrated that distributed PDPs can be achieved by Decentralised PBM in which the presented framework supports the collaboration between DENARs as enforcement requirements (Chapter 8).

11.2 Revisiting Contributions

The main contribution of this thesis is a novel Decentralised Policy Based Management system (PBM) based on a distributed Policy Decision Points PDPs for enforcing static and dynamic policies in distributing setting, so the access control can be provided with efficient performance, security, manageability and resilience support. The performance, security, manageability and resilience are achieved by the Distributed Policy Enforcements Architecture (DENAR), where static and dynamic policies are enforced. This section revisits the four original contributions as follows:

1. **Decentralised PBM framework** has been developed. The Decentralised PBM framework has described the methodology of the framework workflow

to apply DENAR. The workflow illustrated how to achieve the enforcement of static and dynamic policies in distributed systems. In Chapter 6, the analysis of policies has provided to identify dependencies between policy and the access control requirements those involved in dynamic policy. Moreover, it identifies and analyses the enforcement challenges in distributed PDPs where a collaboration of PDPs is required.

2. **Distributed Policy Enforcements Architecture (DENAR)** has been designed. It has five components, including Local Policy Decision Point (PDP), Local Policy Repository (PR), Local Policy Information Point (PIP), and PIPcoordinator and PDPcoordinator components. The LPDP, LPIP and LPR aim to improve the performance (network traffic) of security policy enforcement and their resilience against enforcement failures when a decision is made locally. The PIPcoordinator and PDPcoordinator components are designed for the coordination mechanism.

3. **Policy Decomposition and Policy Deployment Techniqes** have been provided with the respect of independent and dependent rules. The policy decomposition results in sub-policies according to object domains that are then distributed to corresponding DENARs. In the policy decomposition, fragmentation and refinement methods have been developed. The fragmentation algorithm described for decomposes the policy into sub-policies. The refinement method refines the rule to indicate the location of PIPs that store conditional attributes where located in local or remote domain. In addition, the policy deployment is detailed algorithmically.

4. **Coordination and Collaboration Enforcement Model,** using the policy decomposition of and the Distributed Policy Enforcements Architecture (DENAR), those enable Decentralised PBM to deploy and enforce static and dynamic policies for large-scale systems. The PIPcoordinator component in the DENAR responsible of the Push and Pull models. The performance is achieved by identifying local domain decision (that done by one DENAR) and remote domain decision (that done by collaborative DENARs).

The proposed Decentralised PBM framework and the Distributed Policy Enforcements Architecture (DENAR) enforce static and dynamic policies in distributing setting. The approach presented here provides efficient performance, security, manageability and resilience to the enforcement of the policies. The DENAR was designed, implemented, and evaluated using a test-bed.

11.3 Achieving Success Criteria

To answer the research questions that were set in Section 1.3, the Decentralised PBM framework and the Distributed Policy Enforcements Architecture (DENAR) has been designed, implemented and evaluated throughout this thesis.

Involving the local domain decision and remote domain decision in DENARs within PBM framework allows static and dynamic policies to be enforced and coordinated. Enforcing independent policies for local domain decision using the one DENAR supports the performance aspect. The remote domain decision in collaborative DENARs provides the ability of the enforcing the dependent policies. A set of criteria of success and research objective factors have been defined in Section 1.3. These predefined research objective factors are revisited as follows:

- **Performance, a discussion of improving the enforcement will be provided and some experiments will be used to measure the network traffic in both centralised PDP and a network of DENARs.**

  The research has shown how performance can be improved effectively in terms of the network traffic. As described in Chapter 5 and Chapter 8, the research work showed how local domain decisions supports the performance by using the one DENAR to enforce independent rules. Moreover, in the network of DENARs, the authorisation traffic is distributed between DENARs which distributed the number of packets over the network instead of direct these traffic into a particular part of the network or machine.

- **Security, some experiments will be used to discuss the ability of enforcing both static and dynamic policies in both centralised PDP and the network of DENARs.**

  The evaluation of our approach, as presented in Chapter 10, has shown that the DENAR can enforce static and dynamic policies by the replication of policy decision that would have been made by a Centralised PDP in the same scenario. The coordination mechanism offers the Push and Pull models those are proposed for the collaborative DENARs to enforce dependent rules [See Chapter 8, Section 8.3].

- **Manageability, a discussion of the resource utilisation and administration in the DENAR will be provided.**

  Chapter 5 showed that the DENAR configuration, deployment, re-propagation and recovery techniques are described where the resource utilisation and administrative cost are discussed in Chapter 10, Section 10.4.3. The Decentralised PBM offers sufficient computing and storage resources to store and enforce policies. The administrative cost showed that DENAR is simple and flexible in its configuration and reconfiguration which can configure as Centralised PBM during the re-propagation and reconfiguration.

- **Resilience, Network simulators for both centralised PDP and the network of DENARs will be used to discuss the resilience in some potential failures scenarios.**

  Chapter 10 illustrated that the DENAR has resilience against network failures in access control for distributed information systems. The bottleneck is indeed if centralised PDP, PIP or PR is involved where it remains the authorisation decision making centralised which still has performance drawbacks and potential failures.

11.4 Future Work

Several issues for further work have been identified throughout the thesis. We list the most important of them in this section.

- **The Policy Decomposition and Deployment Prototype.**

  The most immediate need in DENAR prototype is the implementation of policy decomposition and deployment algorithm. The DENAR prototype implementation needs to concentrate on the enhancement of the analysis and decomposition of policy that can be enforced in collaborative DENARs.

- **Push Model Implementation for Usage Control.**

  Usage control is a largely open research area in computer security. There is still a lot of work to be done to cover usage control requirements. Usage control in PBM system has to be scrutinised by starting from specification and up to enforcement mechanisms and implementation. The Decentralised PBM which involves distributed enforcements must be able to coordinate mutable attributes that presented in [79] to be updated during the enforcement execution. For this purpose, we provided the Push Model that would require an implementation of the Policy Enforcement Point (PEP) to notify the distributed enforcements about the mutable attributes changes. Another not well understood aspect is the continuous monitoring of processes and the impact that UCON decisions have on the integrity of ongoing processes.

- **Sequential and Parallel Policies.**

  PDPcoodination component is deigned in DENAR to act as bridge for sharing collaborative decision between multiple DENARs. In case of enforcing sequential and parallel policies that presented in [53] multiple DENARs communicate to make the final decision where one of these DENARs is being the synchroniser DENAR that received the authorisation request from a PEP and make this final decision. This is not currently implemented in this work and represents an additional level of abstraction from the policy specification side. However, it is likely that any composed policy containing sequential and parallel policies can be expressed as a normal form compatible with the DENAR architecture. This requires further investigation and also has potential impacts on the coordination model.

- **Dynamic Reconfiguration Based on Traffic Mining.**

  Push and Pull models are proposed and detailed that provide the coordination and collaboration between DENARs. Pull model is designed for retrieving any conditional attribute value that is located in local or remote PIPs. [For Pull Model see Chapter 8, Section 8.3.1]. In Push model, DENAR can "push" a new condition value for the current decision to other PIPs those required for future decision [this is done by obligation, for Push Model see Chapter 8, Section 8.3.2]. However, refining a rule for the choice of Push or Pull model by mining the authorisation traffic saves authorisation time and traffic for the next authorisation decision e.g. PDPx requires a conditional attribute value (α) frequently from remote PIP (PIPy), thus, pushing this value to the PIPx will save authorisation time and traffic for the next authorisation decision.

- **Temporal Permissions.**

  For the temporal permissions, i.e. permissions that lasts for a certain amount of time, the enforcement can make use of DENARs coordination features. If the authorisation decision is occurred, the enforcement timer is reset to result the same authorisation decision for a defined period. This would allow authorisation rules depending on hypothetical decisions to cache some of the results from remote DENARs. Thus, caching the authorisation decision history for the current authorisation request without a need of enforcement process would improve the performance. Another aspect of this research avenue is the ability to identify fix-points in the enforcement of history-based access control rules [See [53]].

  For example:

Coordination Fixpoints	
sometime φ $@PDP_y$;	*always not* μ $@PDP_x$
remote	local

The coordination is only needed unless φ is observed once.

References

[1] Abadi, M. [2003], Logic in access control, *in* 'LICS '03: Proceedings of the 18th Annual IEEE Symposium on Logic in Computer Science', IEEE Computer Society, Washington, DC, USA, p. 228.

[2] Abadi, M., Burrows, M., Lampson, B. and Plotkin, G. [1991], 'A calculus for access control in distributed systems', *ACM Transactions on Programming Languages and Systems* **15**, 706–734.

[3] Abadi, M. and Fournet, C. [2003], Access control based on execution history, *in* 'Proc. 10th Annual Network and Distributed System Security Symposium'.

[4] Abrams, M. [1993], 'Renewed understanding of access control policies', *In Proceedings of the 16th National Computer Security Conference* pp. 87–96.

[5] Aljareh, S. and Rossiter, N. [2001], 'Towards security in multi-agency clinical information services', *Health Informatics Journal* **8**(2), 95–103.

[6] AlZahrani, A. and Janicke [2010], Decentralized policy based management., *in* 'The Saudi International Conference (SIC2010), 2010 SIC 4th International Conference on'.

[7] AlZahrani, A., Janicke, H. and Abubaker, S. [2010], Decentralized xacml overlay network, *in* 'Computer and Information Technology (CIT), 2010 IEEE 10th International Conference on', pp. 1032 –1037.

[8] American National Standards Institute [2004], Ansi incits 359-2004 role based access control, Technical report, American National Standards Institute, New York, NY. Retrieved in Jan 2009.

[9] Anderson, A. [2006], A comparison of two privacy policy languages: Epal and xacml, *in* 'SWS '06: Proceedings of the 3rd ACM workshop on Secure web services', ACM, New York, NY, USA, pp. 53–60.

[10] Anderson, R. [1996], A security policy model for clinical information systems, *in* 'Security and Privacy, 1996. Proceedings., 1996 IEEE Symposium on', IEEE, pp. 30–43.

[11] Anderson, R. [1999], Information technology in medical practice: Safety and privacy lessons from the united kingdom, *in* 'Med J Aust', Citeseer.

[12] Anderson., R. [2001], *Security Engineering, A Guide to Build Dependable Distributed Systems*, John Wiley & Sons, Inc., New York, NY, USA.

[13] Artz, D. and Gil, Y. [2007], 'A survey of trust in computer science and the semantic web', *Web Semantics: Science, Services and Agents on the World Wide Web* **5**(2).

[14] Atkinson, R. and Kent, S. [1998], 'Security architecture for the internet protocol'.

[15] Awad, S. [2011], The impact of using distributed xacml pdp on the network, Master's thesis, De Montfort University.

[16] Barker, S. and Stuckey, P. J. [2003], 'Flexible access control policy specification with constraint logic programming.', *ACM Trans. Inf. Syst. Secur.* **6**, 501–546.

[17] Becker, M. [2009], Specification and analysis of dynamic authorisation policies, *in* 'Computer Security Foundations Symposium, 2009. CSF'09. 22nd IEEE', IEEE, pp. 203–217.

[18] Benantar, M. [2005], *Access control systems: security, identity management and trust models*, Springer.

[19] Bergstra, J. and Burgess, M. [2007], *Handbook of Network and System Administration*, Elsevier Science.

[20] Bertino, E., Bettini, C., Ferrari, E. and Samarati, P. [1998], 'An access control model supporting periodicity constraints and temporal reasoning.', *ACM Trans. Database Syst.* **23**, 231–285.

[21] Bertino, E., Bonatti, P. and Ferrari, E. [2001], 'Trbac: A temporal role-based access control model.', *ACM Trans. Inf. Syst. Secur.* **4**, 191–233.

[22] Bishop, M. [2003], 'Computer security: Art and science.', *Westford, MA: Addison Wesley Professional* pp. 4–12.

[23] Blaze, M., Feigenbaum, J. and Keromytis, A. [1998], Keynote: Trust management for public-key infrastructures, *in* 'Infrastructures (Position Paper). Lecture Notes in Computer Science', pp. 59–63.

[24] Blaze, M., Feigenbaum, J. and Lacy, J. [1996], Decentralized trust management, *in* 'In Proceedings of the 1996 IEEE Symposium on Security and Privacy', IEEE Computer Society Press, pp. 164–173.

[25] Brewer, D. and Nash, M. [1989], The chinese wall security policy., *in* 'IEEE Symposium on Security and Privacy', pp. 206–214.

[26] Case, J., Fedor, M., Schoffstall, M. and Davin, J. [1989], *Simple Network Management Protocol (SNMP)*, Network Information Center, SRI International, United States.

[27] Chadwick, D., Su, L., Otenko, O. and Laborde, R. [2006], Coordination between distributed pdps, *in* 'POLICY '06: Proceedings of the Seventh IEEE International Workshop on Policies for Distributed Systems and Networks', IEEE Computer Society, Washington, DC, USA, pp. 163–172.

[28] Clark, D. and Wilson, D. [1987], A comparison of commercial and military computer security policies, *in* '1987 IEEE Symposium on Security and Privacy', IEEE Computer Society Press, pp. 184–194.

[29] Crnkovic, G. [2010], 'Constructive research and info-computational knowledge generation', *Model-Based Reasoning in Science and Technology* pp. 359–380.

[30] Damianou, N. [2002], A Policy Framework for Management of Distributed Systems, PhD thesis, University of London.

[31] Damianou, N., Bandara, A., Sloman, M. and Lupu, E. [2002], 'A survey of policy specification approaches', *Department of Computing, Imperial College of Science Technology and Medicine, London* .

[32] Damianou, N., Dulay, N., Lupu, E. and Sloman, M. [2000], 'Ponder: A language for specifying security and management policies for distributed systems', *London: Department of Computing, Imperial College, Tech. Rep* .

[33] Damianou, N., Dulay, N., Lupu, E. and Sloman, M. [2001], The ponder policy specification language., *in* M. Sloman, J. Lobo and E. Lupu, eds, 'POLICY', Vol. 1995 of *Lecture Notes in Computer Science*, Springer, pp. 18–38.

[34] Demchenko, Y., de Laat, C., Koeroo, O. and Sagehaug, H. [2008], Extending xacml authorisation model to support policy obligations handling in distributed applications, *in* 'Proceedings of the 6th International Workshop on Middleware for Grid Computing'.

[35] Dierks, T. and Allen, C. [1999], 'Rfc 2246: The tls protocol'.

[36] DoD Computer Security Center [1983], Trusted computer system evaluation criteria, Technical report, US Department of Defense, Washington, DC. Retrieved in Jan 2010.

[37] Dulay, N., Damianou, N., Lupu, E. and Sloman, M. [2001], A policy language for the management of distributed agents., *in* M. Wooldridge, G. Wei and P. Ciancarini, eds, 'AOSE', Vol. 2222 of *Lecture Notes in Computer Science*, Springer, pp. 84–100.

[38] Durham, D., Boyle, J., Cohen, R., Herzog, S., Rajan, R. and Sastry, A. [2000], RFC 2748: The COPS (Common Open Policy Service) Protocol, Technical report, IETF. Retrieved in Jan 2010.
URL: *www.ietf.org/rfc/rfc2748.txt*

[39] Ellison, C., Frantz, B., Lampson, B., Rivest, R., Thomas, B. and Ylonen., T. [1999], 'Spki certificate theory'. Retrieved in Jan 2012.
URL: *http://www.ietf.org/rfc/rfc2693*

[40] Feinsteink, H., Sandhu, R. and Youmank, C. [1996], 'Role-based access control models', *Computer* **29**(2), 38–47.

[41] Framework, A., Sloman, M. and Twidle, K. [1994], 'Domains: A framework for structuring management policy', *Network and Distributed Systems Management* pp. 433–453.

[42] Grandison, T. and Sloman, M. [2000], 'A survey of trust in internet applications.', *IEEE Communications Surveys and Tutorials* **3**.

[43] Group Network Working [2006], 'Comment on rfc 4516 - lightweight directory access protocol (ldap)'.

[44] Harold, E. [2006], *Java I/O*, 6 edition edn, O'REILLY, CA USA.

[45] Hoagland, J. [2000], 'Specifying and implementing security policies using lasco, the language for security constraints on objects', *CoRR* **cs.CR/0003066**.

[46] Hyland, P. and Sandhu, R. [1998], 'Management of network security applications', *The 21st National Information Systems Security Conference (NISSC),* .

[47] ITU-T, Recommendation [1996], 'Information technology-open systems interconnection-security frameworks in open systems', *Non-repudiation Framework* . Retrieved in May 2010.

[48] Jajodia, S., Samarati, P., Sapino, M. L. and Subrahmanian, V. [2001], 'Flexible support for multiple access control policies.', *ACM Trans. Database Syst.* **26**, 214–260.

[49] Jajodia, S., Samarati, P., Subrahmanian, V. and Bertino, E. [1997], A unified framework for enforcing multiple access control policies., *in* J. Peckham, ed., 'SIGMOD Conference', ACM Press, pp. 474–485.

[50] Jajodia, S. and Wijesekera, D. [2004], A flexible authorization framework for e-commerce., *in* R. K. Ghosh and H. Mohanty, eds, 'ICDCIT', Vol. 3347 of *Lecture Notes in Computer Science*, Springer, pp. 336–345.

[51] Janicke, H. [2007], The Development of Secure Multi-Agent Systems, PhD thesis, De Montfort University.

[52] Janicke, H., Cau, A., Siewe, F. and Zedan, H. [2007], Deriving enforcement mechanisms from policies, *in* 'POLICY 07: Proceedings of the Eighth IEEE International Workshop on Policies for Distributed Systems and Networks', IEEE Computer Society, Washington, DC, USA, pp. 161–172.

[53] Janicke, H., Cau, A., Siewe, F. and Zedan, H. [2008], Concurrent enforcement of usage control policies, *in* 'Proceedings of the 2008 IEEE Workshop on Policies for Distributed Systems and Networks', POLICY '08, IEEE Computer Society, Washington, DC, USA.

[54] Janicke, H., Cau, A., Siewe, F. and Zedan, H. [2012], 'Dynamic access control policies: Specification and verification', *The Computer Journal* .

[55] Janicke, H., Cau, A., Siewe, F., Zedan, H. and Jones, K. [2006], A compositional event & time-based policy model, *in* 'POLICY 06: Proceedings of the Seventh IEEE International Workshop on Policies for Distributed Systems and Networks', IEEE Computer Society, Washington, DC, USA, pp. 173–182.

[56] Janicke, H., Cau, A. and Zedan, H. [2007], A note on the formalisation of ucon, *in* 'SACMAT 07: Proceedings of the 12th ACM symposium on Access control models and technologies', ACM, New York, NY, USA, pp. 163–168.

[57] Janicke, H. and Finch., L. [2007], The role of dynamic security policy in military scenarios, *in* 'In Proceedings of the 6th European Conference on Information Warfare and Security', pp. 2007–9.

[58] Janicke, H., Siewe, F., Jones, K., Cau, A. and Zedan, H. [2005], Analysis and run-time verification of dynamic security policies., *in* S. G. Thompson and R. A. Ghanea-Hercock, eds, 'Defence Applications of Multi-Agent Systems', Lecture Notes in Computer Science, Springer, pp. 92–103.

[59] Katt, B., Zhang, X., Breu, R., Hafner, M. and Seifert, J. [2008], A general obligation model and continuity: Enhanced policy enforcement engine for usage control, *in* 'Proceedings of the 13th ACM Symposium on Access Control Models and Technologies', ACM, pp. 123–132.

[60] Kolovski, V. and Hendler, J. [2008], 'Xacml policy analysis using description logics', *Under submission* .

[61] Kolovski, V., Hendler, J. and Parsia, B. [2006], 'Formalizing xacml using defeasible description logics', *University of Maryland, USA, Tech. Rep. TR-233-11* .

[62] Krief, F. and Bouthinon, D. [2005], A learning and intentional local policy decision point for dynamic qos provisioning, *in* D. Gati, S. Galms and R. Puigjaner, eds, 'Network Control and Engineering for QoS, Security and Mobility, III', Vol. 165 of *IFIP International Federation for Information Processing*, Springer US, pp. 277–288.

[63] Labro, E. and Tuomela, T.-S. [2003], 'On bringing more action into management accounting research: Process considerations based on two constructive case studies', *European Accounting Review,* pp. 409–442.

[64] Lamport, L. [1994], 'The temporal logic of actions.', *ACM Trans. Program. Lang. Syst.* **16**, 872–923.

[65] Lampson, B. [1974], 'Protection', *ACM SIGOPS Operating Systems Review* **8**, 18–24.

[66] LaPadula, L. and Bell, D. [1973], Secure computer systems: Mathematical foundations, Technical report, MITRE Technical Report 2547. Retrieved in Jan 2009.

[67] Li, J., Li, N. and Winsborough, W. [2005], Automated trust negotiation using cryptographic credentials, *in* 'Proceedings of the 12th ACM Conference on Computer and Communications Security', ACM, pp. 46–57.

[68] Lin, D., Rao, P., Bertino, E., Li, N. and Lobo, J. [2008], Policy decomposition for collaborative access control, *in* 'Proceedings of the 13th ACM symposium on Access Control Models and Technologies', SACMAT '08, ACM, New York, NY, USA.

[69] Lua, E., Crowcroft, J., Pias, M., Sharma, R. and Lim, S. [2005], 'A survey and comparison of peer-to-peer overlay network schemes', *IEEE Communications Surveys and Tutorials* **7**, 72–93.

[70] M. Becker, C. F. and Gordon, A. [September 2006.], Secpal: Design and semantics of a decentralized authorisation language, Technical report, Microsoft ResearchResearch, Roger Needham Building 7 J.J. Thompson Avenue, Cambridge, CB3 0FB, UK,. Retrieved in Jan 2009.

[71] Minsky, N. and Ungureanu, V. [2000], 'Law-governed interaction: a coordination and control mechanism for heterogeneous distributed systems', *TOSEM, ACM Transactions on Software Engineering and Methodology* **9**, 273–305.

[72] Mockapetris, P. and Dunlap, K. [1988], *Development of the domain name system*, Vol. 18, ACM.

[73] Mossakowski, T., Drouineaud, M. and Sohr, K. [2003], A temporal-logic extension of role-based access control covering dynamic separation of duties., *in* 'TIME', IEEE Computer Society, pp. 83–90.

[74] Moszkowski, B. [1980], 'Interval temporal logic'. Retrieved in Jan 2009.
URL: *http://www.cse.dmu.ac.uk/STRL/ITL/*

[75] National, Computer, Security, Center and (NCSC) [1988], Glossary of computer security terms, Technical report, Report NSCD-TG-004, Fort Meade, Md.: NCSC. Retrieved in Feb 2010.

[76] OASIS [2005], 'extensible access control markup language (xacml) version 2.0,'. Retrieved in Sep 2009.
URL: *http://docs.oasis-open.org/xacml/2.0/accesscontrol-xacml-2.0-core-spec-os.pdf*

[77] Park, J. and Sandhu, R. [2002], Towards usage control models: Beyond traditional access control, *in* 'SACMAT '02: Proceedings of The Seventh ACM Symposium on Access Control Models and Technologies', ACM, New York, NY, USA, pp. 57–64.

[78] Park, J. and Sandhu, R. [2004], 'The uconabc usage control model', *ACM Trans. Inf. Syst. Secur.* **7**(1), 128–174.

[79] Park, J., Zhang, X. and Sandhu, R. [2004], 'Attribute mutability in usage control', *Research Directions in Data and Applications Security XVIII* pp. 15–29.

[80] Rafael ordini, Mehdi Dastani, J. D. and Seghrouchni, A. F. [2005], *Multi-Agent Programming*, Springer US, chapter Jade A Java Agent Development Framework, pp. 125–147.

[81] Roma, Tre, University, , the, Linux, User, Group, LUG and Roma [2006], 'Netkit'. accessed 12.06.2010.
URL: *http://wiki.netkit.org/index.php/MainPage*

[82] Samarati, P. and Vimercati, S. [2001], 'Access control: Policies, models, and mechanisms', *Foundations of Security Analysis and Design* pp. 137–196.

[83] Sandhu, R. [1988], Transaction control expressions for separation of duties, *in* 'Aerospace Computer Security Applications Conference, 1988., Fourth', IEEE, pp. 282–286.

[84] Sandhu, R. [1998], Role activation hierarchies., *in* 'ACM Workshop on Role-Based Access Control', pp. 33–40.

[85] Sandhu, R. and Park, J. [2003], 'Usage control: A vision for next generation access control', *Computer Network Security* pp. 17–31.

[86] Siebenlist, F. and Mori, T. [2005,], 'Globus toolkit: Authorization processing'. Retrieved in May 2010.
URL: *http://www.globus.org/alliance/events/gw06/gt-authz-gw06-v3.pdf*

[87] Siewe, F. [2005], A Compositional Framework for the Development of Secure Access Control Systems, PhD thesis, Software Technology Research Laboratory, Department of Computer Science and Engineering, De Montfort University, Leicester.

[88] Sloman, M. [1994], 'Policy driven management for distributed systems', *Journal of Network and Systems Management* **2**(4), 333–360.

[89] Stallings, W. [2010], *Data and Computer Communications*, 9 edition edn, Prentice Hall, NY USA.

[90] Su, L., Chadwick, D., Basden, A. and Cunningham, J. [2005*a*], Automated decomposition of access control policies, *in* 'Proc of 6 th IEEE International Workshop on Policies for Distributed Systems and Networks', pp. 6–8.

[91] Su, L., Chadwick, D., Basden, A. and Cunningham, J. [2005*b*], Automated decomposition of access control policies, *in* 'POLICY '05: Proceedings of the Sixth IEEE International Workshop on Policies for Distributed Systems and Networks', IEEE Computer Society, Washington, DC, USA, pp. 3–13.

[92] Sun Microsystems [2006], 'Sun's xacml implementation'. Retrieved in Jan 2010.
URL: *http://sunxacml.sourceforge.net/*

[93] Tanenbaum, A. and Wetherall, D. [2010], *Computer Networks.*, 5 edition edn, Prentice Hall, London.

[94] Thomas, R. [1997], Team-based access control (tmac): A primitive for applying role-based access controls in collaborative environments., *in* 'ACM Workshop on Role-Based Access Control', pp. 13–19.

[95] Verma, D. [2000], *Policy-Based Networking: Architecture and Algorithms*, New Riders Publishing, Thousand Oaks, CA, USA.

[96] Wijesekera, D. and Jajodia, S. [2003], A propositional policy algebra for access control, Vol. 6, ACM, New York, NY, USA, pp. 286–325.

[97] Wilson, B. [June 2002.], *JXTA*, first edition edn, New Riders Publishing, Indianapolis, IN.

[98] Woo, T. and Lam, S. [1993], 'Authorization in distributed systems: A new approach', *Journal of Computer Security* **2**(2), 107–136.

[99] Yao, W. [2002], Trust Management for Widely Distributed Systems., PhD thesis, University of Cambridge, Computer Laboratory.

[100] Yavatkar, R., Pendarakis, D. and Guerin, R. [2000], RFC 2753: A Framework for Policy-based Admission Control, Technical report, IETF. Retrieved in Jan 2011.
URL: *www.ietf.org/rfc/rfc2753.txt*

[101] Zhang, X., Parisi-Presicce, F., Sandhu, R. and Park, J. [2005], 'Formal model and policy specification of usage control', *ACM Trans. Inf. Syst. Secur.* **8**(4), 351–387.

[102] Zhang, X., Park, J., Parisi-Presicce, F. and Sandhu, R. [2004], A logical specification for usage control, *in* 'SACMAT '04: Proceedings of the Ninth ACM Symposium on Access Control Models and Technologies', ACM, New York, NY, USA, pp. 1–10.

[103] Zhang, X., Sandhu, R. and Parisi-Presicce, F. [2006], Safety analysis of usage control authorization models, *in* 'Conference on Computer and Communications Security: Proceedings of the 2006 ACM Symposium on Information, Computer and Communications Security', Vol. 21, pp. 243–254.

Appendix A

Source Code

The following listings present source code of the classes of the DENAR prototype implementation. The implementation presents only the DENAR prototype to evaluate the functionalities of the DENAR components work to achieve our approach objectives. However, It is not implemented to a level at which it could be readily commercially utilised.

A.1 DENARs Network Labs

A.1.1 Network Configuration

```
/* --------------------------------------------------------
* @author Ali Alzahrani (STRL, DMU, UK)
* aalzahrani@dmu.ac.uk
----------------------------------------------------------*/

--------------------
maingw.startup file
--------------------
ifconfig eth0 172.16.100.254
route add default gw 10.0.0.254
route add -net 192.168.10.0 netmask 255.255.255.0 gw 172.16.100.1 dev eth0
route add -net 192.168.20.0 netmask 255.255.255.0 gw 172.16.100.2 dev eth0
route add -net 192.168.30.0 netmask 255.255.255.0 gw 172.16.100.3 dev eth0
route add -net 192.168.40.0 netmask 255.255.255.0 gw 172.16.100.4 dev eth0
iptables -t nat -A POSTROUTING -o eth1 -j SNAT --to-source 10.0.0.1
#iptables -t nat -A PREROUTING -s 192.168.10.1 -j DNAT --to-destination 172.16.100.254
#iptables -t nat -A PREROUTING -s 192.168.20.1 -j DNAT --to-destination 172.16.100.254
```

```
#iptables -t nat -A PREROUTING -s 192.168.30.1 -j DNAT --to-destination 172.16.100.254
#iptables -t nat -A PREROUTING -s 192.168.40.1 -j DNAT --to-destination 172.16.100.254
echo "domain project" >> /etc/resolv.conf
echo "search project" >> /etc/resolv.conf
echo "nameserver 172.16.100.254" >> /etc/resolv.conf
/etc/init.d/bind start

--------------------
db.100.16.172 file
--------------------
$TTL   60000
@               IN      SOA     ns.project.    root.project. (
                        2006031201 ; serial
                        28800 ; refresh
                        14400 ; retry
                        3600000 ; expire
                        0 ; negative cache ttl
                        )
@       IN      NS      ns.project.
254     IN      PTR     maingw.project.

--------------------
db.project file
--------------------
$TTL   60000
@               IN      SOA     ns.project.    root.project. (
                        2006031201 ; serial
                        28800 ; refresh
                        14400 ; retry
                        3600000 ; expire
                        0 ; negative cache ttl
                        )
@               IN      NS      ns.project.
ns              IN      A       172.16.100.254
maingw          IN      CNAME   ns
gw              IN      CNAME   maingw
d1              IN      NS      ns.d1.project.
ns.d1   IN      A       192.168.10.1
d2              IN      NS      ns.d2.project.
ns.d2   IN      A       192.168.20.1
d3              IN      NS      ns.d3.project.
ns.d3   IN      A       192.168.30.1
d4              IN      NS      ns.d4.project.
ns.d4   IN      A       192.168.40.1

----------------------
named.conf.local file
----------------------
zone "project" {
                type master;
                file "/etc/bind/db.project";
                allow-update { 127.0.0.1; };
                notify yes;
};
```

```
zone "100.16.172.in-addr.arpa" {
                type master;
                file "/etc/bind/db.100.16.172";
                allow-update { 127.0.0.1; };
                notify yes;
};
```

Listing A.1: maingw Configuration

```
/* -------------------------------------------------------
* @author Ali Alzahrani (STRL, DMU, UK)
* aalzahrani@dmu.ac.uk
-------------------------------------------------------------*/

--------------------
router1.startup file
--------------------
ifconfig eth0 192.168.10.1
ifconfig eth1 172.16.100.1
route add default gw 172.16.100.254
#iptables -t nat -A POSTROUTING -o eth1 -j SNAT --to-source 192.168.10.1
#iptables -t nat -A PREROUTING -s 172.16.100.254 -j DNAT --to-destination 192.168.10.1
#iptables -t nat -A PREROUTING -s 192.168.20.1 -j DNAT --to-destination 192.168.10.1
#iptables -t nat -A PREROUTING -s 192.168.30.1 -j DNAT --to-destination 192.168.10.1
#iptables -t nat -A PREROUTING -s 192.168.40.1 -j DNAT --to-destination 192.168.10.1
echo "domain d1.project" > /etc/resolv.conf
echo "search d1.project" >> /etc/resolv.conf
echo "nameserver 192.168.10.1" >> /etc/resolv.conf
/etc/init.d/bind start
/etc/init.d/dhcp3-server start

--------------------
db.10.168.192 file
--------------------
$TTL 8h
@       IN      SOA     ns.d1.project.   d1-admin.d1.project. (
                2011021801 ; serial
                8h ; refresh
                2h ; retry
                1w ; expire
                0 ; negative cache ttl
)
@       IN      NS      ns.d1.project.

1       IN      PTR     router1.d1.project.

--------------------
db.project.d1 file
--------------------
$TTL 8h
@       IN      SOA     ns.d1.project.   d1-admin.d1.project. (
                2011021801 ; serial
                8h ; refresh
                2h ; retry
                1w ; expire
                0 ; negative cache ttl
)
@       IN      NS      ns.d1.project.
ns      IN      A       192.168.10.1
router1 IN      CNAME   ns
gw      IN      CNAME   router1

----------------------
named.conf.local file
----------------------
zone "d1.project" {
                type master;
                file "/etc/bind/db.project.d1";
                allow-update { key localhost-ddns; };
                notify yes;
};
zone "10.168.192.in-addr.arpa" {
                type master;
                file "/etc/bind/db.10.168.192";
                allow-update { key localhost-ddns; };
                notify yes;
};

------------------------
named.conf.options file
------------------------
key "localhost-ddns" {
                algorithm hmac-md5;
                secret "q0qvEmpG9jSqNNgpa+dJfA==";
        };
options {
                directory "/var/cache/bind";
                forwarders {
                        172.16.100.254;
                };
        forward only;
        recursion yes;
        auth-nxdomain no;
        listen-on-v6 { any; };
};

--------------------
dhcpd.conf file
--------------------
ddns-update-style interim;
ddns-updates on;
ddns-domainname "d1.project";
ddns-rev-domainname "in-addr.arpa";
deny client-updates;

key "localhost-ddns" {
        algorithm hmac-md5;
        secret q0qvEmpG9jSqNNgpa+dJfA==;
        };
option domain-name "d1.project";
option domain-name-servers 192.168.10.1; # local DNS
default-lease-time 300;
max-lease-time 600;
update-static-leases on;
authoritative;
subnet 192.168.10.0 netmask 255.255.255.0
{
option subnet-mask 255.255.255.0;
option routers 192.168.10.1; # Gateway
range 192.168.10.100 192.168.10.120;
allow unknown-clients;
zone d1.project.
        {
                primary 127.0.0.1;
                key localhost-ddns;
        }
zone 10.168.192.in-addr.arpa.
        {
                primary 127.0.0.1;
                key localhost-ddns;
        }

}
```

Listing A.2: router1 Configuration

```
/* -----------------------------------------------------------
* @author Ali Alzahrani (STRL, DMU, UK)
* aalzahrani@dmu.ac.uk
-------------------------------------------------------------*/

--------------------
router2.startup file
--------------------
ifconfig eth0 192.168.20.1
ifconfig eth1 172.16.100.2
route add default gw 172.16.100.254
#iptables -t nat -A POSTROUTING -o eth1 -j SNAT --to-source 192.168.20.1
#iptables -t nat -A PREROUTING -s 172.16.100.254 -j DNAT --to-destination 192.168.20.1
#iptables -t nat -A PREROUTING -s 192.168.10.1 -j DNAT --to-destination 192.168.20.1
#iptables -t nat -A PREROUTING -s 192.168.30.1 -j DNAT --to-destination 192.168.20.1
#iptables -t nat -A PREROUTING -s 192.168.40.1 -j DNAT --to-destination 192.168.20.1
echo "domain d2.project" > /etc/resolv.conf
echo "search d2.project" >> /etc/resolv.conf
echo "nameserver 192.168.20.1" >> /etc/resolv.conf
/etc/init.d/bind start
/etc/init.d/dhcp3-server start

--------------------
db.20.168.192 file
--------------------
```

```
$TTL 8h
@       IN      SOA     ns.d2.project.   d2-admin.d2.project. (
                2011021801 ; serial
                8h ; refresh
                2h ; retry
                1w ; expire
                0 ; negative cache ttl
)
@       IN      NS      ns.d2.project.
1       IN      PTR     router2.d2.project.

--------------------
db.project.d2 file
--------------------
$TTL 8h
@       IN      SOA     ns.d2.project.   d2-admin.d2.project. (
                2011021801 ; serial
                8h ; refresh
                2h ; retry
                1w ; expire
                0 ; negative cache ttl
)
@       IN      NS      ns.d2.project.
ns      IN      A       192.168.20.1
router2 IN      CNAME   ns
gw      IN      CNAME   router2

----------------------
named.conf.local file
----------------------
zone "d2.project" {
                type master;
                file "/etc/bind/db.project.d2";
                allow-update { key localhost-ddns; };
                notify yes;
};
zone "20.168.192.in-addr.arpa" {
                type master;
                file "/etc/bind/db.20.168.192";
                allow-update { key localhost-ddns; };
                notify yes;
};

------------------------
named.conf.options file
------------------------
key "localhost-ddns" {
                algorithm hmac-md5;
                secret "q0qvEmpG9jSqNNgpa+dJfA==";
        };
options {
                directory "/var/cache/bind";
                forwarders {
                        172.16.100.254;
                };
        forward only;
        recursion yes;
        auth-nxdomain no;
        listen-on-v6 { any; };
};

--------------------
dhcpd.conf file
--------------------
ddns-update-style interim;
ddns-updates on;
ddns-domainname "d2.project";
ddns-rev-domainname "in-addr.arpa";
deny client-updates;
key "localhost-ddns" {
        algorithm hmac-md5;
        secret q0qvEmpG9jSqNNgpa+dJfA==;
        };
option domain-name "d2.project";
option domain-name-servers 192.168.20.1; # local DNS
default-lease-time 300;
max-lease-time 600;
update-static-leases on;
authoritative;
subnet 192.168.20.0 netmask 255.255.255.0
{
option subnet-mask 255.255.255.0;
option routers 192.168.20.1; # Gateway
range 192.168.20.100 192.168.20.120;
allow unknown-clients;
zone d2.project.
        {
                primary 127.0.0.1;
                key localhost-ddns;
        }
zone 20.168.192.in-addr.arpa.
        {
                primary 127.0.0.1;
                key localhost-ddns;
        }
}
```

Listing A.3: router2 Configuration

```
/* -------------------------------------------------------
* @author Ali Alzahrani (STRL, DMU, UK)
* aalzahrani@dmu.ac.uk
----------------------------------------------------------*/

--------------------
router3.startup file
--------------------
ifconfig eth0 192.168.30.1
ifconfig eth1 172.16.100.3
route add default gw 172.16.100.254
#iptables -t nat -A POSTROUTING -o eth1 -j SNAT --to-source 192.168.30.1
#iptables -t nat -A PREROUTING -s 172.16.100.254 -j DNAT --to-destination 192.168.30.1
#iptables -t nat -A PREROUTING -s 192.168.40.1 -j DNAT --to-destination 192.168.30.1
#iptables -t nat -A PREROUTING -s 192.168.20.1 -j DNAT --to-destination 192.168.30.1
#iptables -t nat -A PREROUTING -s 192.168.10.1 -j DNAT --to-destination 192.168.30.1
echo "domain d3.project" > /etc/resolv.conf
echo "search d3.project" >> /etc/resolv.conf
echo "nameserver 192.168.30.1" >> /etc/resolv.conf
/etc/init.d/bind start
/etc/init.d/dhcp3-server start

--------------------
db.30.168.192 file
--------------------
$TTL 8h
@       IN      SOA     ns.d3.project.   d3-admin.d3.project. (
                2011021801 ; serial
                8h ; refresh
                2h ; retry
                1w ; expire
                0 ; negative cache ttl
)
@       IN      NS      ns.d3.project.
1       IN      PTR     router3.d3.project.

--------------------
db.project.d3 file
--------------------
$TTL 8h
@       IN      SOA     ns.d3.project.   d3-admin.d3.project. (
                2011021801 ; serial
                8h ; refresh
                2h ; retry
                1w ; expire
                0 ; negative cache ttl
)
@       IN      NS      ns.d3.project.
ns      IN      A       192.168.30.1
router3 IN      CNAME   ns
gw      IN      CNAME   router3

----------------------
named.conf.local file
----------------------
zone "d3.project" {
                type master;
                file "/etc/bind/db.project.d3";
                allow-update { key localhost-ddns; };
                notify yes;
};
zone "30.168.192.in-addr.arpa" {
                type master;
                file "/etc/bind/db.30.168.192";
                allow-update { key localhost-ddns; };
                notify yes;
};

------------------------
named.conf.options file
------------------------
key "localhost-ddns" {
                algorithm hmac-md5;
                secret "q0qvEmpG9jSqNNgpa+dJfA==";
        };
options {
                directory "/var/cache/bind";
                forwarders {
                        172.16.100.254;
                };
        forward only;
        recursion yes;
        auth-nxdomain no;
        listen-on-v6 { any; };
};

--------------------
dhcpd.conf file
--------------------
ddns-update-style interim;
ddns-updates on;
ddns-domainname "d3.project";
ddns-rev-domainname "in-addr.arpa";
deny client-updates;
key "localhost-ddns" {
        algorithm hmac-md5;
        secret q0qvEmpG9jSqNNgpa+dJfA==;
        };
option domain-name "d3.project";
option domain-name-servers 192.168.30.1; # local DNS
default-lease-time 300;
max-lease-time 600;
update-static-leases on;
authoritative;
subnet 192.168.30.0 netmask 255.255.255.0
{
option subnet-mask 255.255.255.0;
option routers 192.168.30.1; # Gateway
range 192.168.30.100 192.168.30.120;
allow unknown-clients;
zone d3.project.
        {
                primary 127.0.0.1;
                key localhost-ddns;
        }
zone 30.168.192.in-addr.arpa.
        {
                primary 127.0.0.1;
                key localhost-ddns;
        }
}
```

Listing A.4: router3 Configuration

```
/* -------------------------------------------------------
* @author Ali Alzahrani (STRL, DMU, UK)
* aalzahrani@dmu.ac.uk
----------------------------------------------------------*/

--------------------
router4.startup file
--------------------
ifconfig eth0 192.168.40.1
ifconfig eth1 172.16.100.4
route add default gw 172.16.100.254
#iptables -t nat -A POSTROUTING -o eth1 -j SNAT --to-source 192.168.40.1
#iptables -t nat -A PREROUTING -s 172.16.100.254 -j DNAT --to-destination 192.168.40.1
#iptables -t nat -A PREROUTING -s 192.168.30.1 -j DNAT --to-destination 192.168.40.1
#iptables -t nat -A PREROUTING -s 192.168.20.1 -j DNAT --to-destination 192.168.40.1
#iptables -t nat -A PREROUTING -s 192.168.10.1 -j DNAT --to-destination 192.168.40.1
echo "domain d4.project" > /etc/resolv.conf
echo "search d4.project" >> /etc/resolv.conf
echo "nameserver 192.168.40.1" >> /etc/resolv.conf
/etc/init.d/bind start
/etc/init.d/dhcp3-server start

--------------------
db.40.168.192 file
--------------------
$TTL 8h
@       IN      SOA     ns.d4.project.   d4-admin.d4.project. (
                2011021801 ; serial
                8h ; refresh
                2h ; retry
                1w ; expire
                0 ; negative cache ttl
)
@       IN      NS      ns.d4.project.
1       IN      PTR     router4.d4.project.

--------------------
db.project.d4 file
--------------------
$TTL 8h
@       IN      SOA     ns.d4.project.   d4-admin.d4.project. (
                2011021801 ; serial
                8h ; refresh
                2h ; retry
                1w ; expire
                0 ; negative cache ttl
)
@       IN      NS      ns.d4.project.
```

```
ns      IN      A       192.168.40.1
router4 IN      CNAME   ns
gw      IN      CNAME   router4

----------------------
named.conf.local file
--------------------
zone "d4.project" {
                type master;
                file "/etc/bind/db.project.d4";
                allow-update { key localhost-ddns; };
                notify yes;
};
zone "40.168.192.in-addr.arpa" {
                type master;
                file "/etc/bind/db.40.168.192";
                allow-update { key localhost-ddns; };
                notify yes;
};

------------------------
named.conf.options file
------------------------
key "localhost-ddns" {
                algorithm hmac-md5;
                secret "q0qvEmpG9jSqNNgpa+dJfA==";
        };
options {
                directory "/var/cache/bind";
                forwarders {
                        172.16.100.254;
                };
        forward only;
        recursion yes;
        auth-nxdomain no;
        listen-on-v6 { any; };
};

--------------------
dhcpd.conf file
--------------------
ddns-update-style interim;
ddns-updates on;
ddns-domainname "d4.project";
ddns-rev-domainname "in-addr.arpa";
deny client-updates;
key "localhost-ddns" {
        algorithm hmac-md5;
        secret q0qvEmpG9jSqNNgpa+dJfA==;
        };
option domain-name "d4.project";
option domain-name-servers 192.168.40.1; # local DNS
default-lease-time 300;
max-lease-time 600;
update-static-leases on;
```

```
authoritative;
subnet 192.168.40.0 netmask 255.255.255.0
{
option subnet-mask 255.255.255.0;
option routers 192.168.40.1; # Gateway
range 192.168.40.100 192.168.40.120;
allow unknown-clients;
zone d4.project.
        {
                primary 127.0.0.1;
                key localhost-ddns;
        }
zone 40.168.192.in-addr.arpa.
        {
                primary 127.0.0.1;
                key localhost-ddns;
        }
}
```

Listing A.5: router4 Configuration

A.1.2 Centralised PBM

```
/* ---------------------------------------------------------
* @author Ali Alzahrani (STRL, DMU, UK)
* aalzahrani@dmu.ac.uk
-----------------------------------------------------------*/

--------------------
lab.conf file
--------------------
maingw[mem]=256
maingw[0]=xtrnal
maingw[1]=tap,10.0.0.254,10.0.0.1
router1[mem]=256
router1[0]=lan1
router1[1]=xtrnal
router2[mem]=256
router2[0]=lan2
router2[1]=xtrnal
router3[mem]=256
router3[0]=lan3
router3[1]=xtrnal
router4[mem]=256
router4[0]=lan4
router4[1]=xtrnal
pc1-1[mem]=128
pc1-1[0]=lan1
pc2-1[mem]=128
pc2-1[0]=lan1
pc3-1[mem]=128
```

```
pc3-1[0]=lan1
pc4-1[mem]=128
pc4-1[0]=lan1
pc1-2[mem]=128
pc1-2[0]=lan2
pc2-2[mem]=128
pc2-2[0]=lan2
pc3-2[mem]=128
pc3-2[0]=lan2
pc4-2[mem]=128
pc4-2[0]=lan2
CentralPDP[mem]=512
CentralPDP[0]=lan2
pc1-3[mem]=128
pc1-3[0]=lan3
pc2-3[mem]=128
pc2-3[0]=lan3
pc3-3[mem]=128
pc3-3[0]=lan3
pc4-3[mem]=128
pc4-3[0]=lan3
pc1-4[mem]=128
pc1-4[0]=lan4
pc2-4[mem]=128
pc2-4[0]=lan4
pc3-4[mem]=128
pc3-4[0]=lan4
pc4-4[mem]=128
pc4-4[0]=lan4

--------------------
lab.dep file
--------------------
router1 router2: maingw
router3 router4: maingw
pc1-1: router1 router2
pc2-1: router1 router2
pc3-1: router1 router2
pc4-1: router1 router2
pc1-2: router1 router2
pc2-2: router1 router2
pc3-2: router1 router2
pc4-2: router1 router2
CentralPDP: router1 router2
pc1-3: router3 router4
pc2-3: router3 router4
pc3-3: router3 router4
pc4-3: router3 router4
pc1-4: router3 router4
pc2-4: router3 router4
pc3-4: router3 router4
pc4-4: router3 router4
```

Listing A.6: Centralised PBM Configuration

```
/* ------------------------------------------------------
* @author Ali Alzahrani (STRL, DMU, UK)
* aalzahrani@dmu.ac.uk
----------------------------------------------------------*/

-----------------------
CentralPDP.startup file
--------------------
echo 'send host-name "CentralPDP";' >> /etc/dhcp3/dhclient.conf
dhclient eth0

--------------------
pc1-1.startup file
--------------------
echo 'send host-name "pc1-1";' >> /etc/dhcp3/dhclient.conf
dhclient eth

and so on for other PCs
```

Listing A.7: CentralPDP and PCs Configuration in Centralised PBM

A.1.3 Decentralised PBM

```
/* ------------------------------------------------------
* @author Ali Alzahrani (STRL, DMU, UK)
* aalzahrani@dmu.ac.uk
----------------------------------------------------------*/

--------------------
lab.conf file
--------------------
maingw[mem]=256
maingw[0]=xtrnal
maingw[1]=tap,10.0.0.254,10.0.0.1
router1[mem]=256
router1[0]=lan1
router1[1]=xtrnal
router2[mem]=256
router2[0]=lan2
router2[1]=xtrnal
router3[mem]=256
router3[0]=lan3
router3[1]=xtrnal
router4[mem]=256
router4[0]=lan4
router4[1]=xtrnal
pc1-1[mem]=128
pc1-1[0]=lan1
pc2-1[mem]=128
pc2-1[0]=lan1
```

```
pc3-1[mem]=128
pc3-1[0]=lan1
pc4-1[mem]=128
pc4-1[0]=lan1
DENAR1[mem]=128
DENAR1[0]=lan1
pc1-2[mem]=128
pc1-2[0]=lan2
pc2-2[mem]=128
pc2-2[0]=lan2
pc3-2[mem]=128
pc3-2[0]=lan2
pc4-2[mem]=128
pc4-2[0]=lan2
DENAR2[mem]=128
DENAR2[0]=lan2
pc1-3[mem]=128
pc1-3[0]=lan3
pc2-3[mem]=128
pc2-3[0]=lan3
pc3-3[mem]=128
pc3-3[0]=lan3
pc4-3[mem]=128
pc4-3[0]=lan3
DENAR3[mem]=128
DENAR3[0]=lan3
pc1-4[mem]=128
pc1-4[0]=lan4
pc2-4[mem]=128
pc2-4[0]=lan4
pc3-4[mem]=128
pc3-4[0]=lan4
pc4-4[mem]=128
pc4-4[0]=lan4
DENAR4[mem]=128
DENAR4[0]=lan4

--------------------
lab.dep file
--------------------
router1 router2: maingw
router3 router4: maingw
pc1-1: router1 router2
pc2-1: router1 router2
pc3-1: router1 router2
pc4-1: router1 router2
DENAR1: router1 router2
pc1-2: router1 router2
pc2-2: router1 router2
pc3-2: router1 router2
pc4-2: router1 router2
DENAR2: router1 router2
pc1-3: router3 router4
pc2-3: router3 router4
pc3-3: router3 router4
```

```
pc4-3: router3 router4
DENAR3: router3 router4
pc1-4: router3 router4
pc2-4: router3 router4
pc3-4: router3 router4
pc4-4: router3 router4
DENAR4: router3 router4
```

Listing A.8: Decentralised PBM Configuration

```
/* -------------------------------------------------------
* @author Ali Alzahrani (STRL, DMU, UK)
* aalzahrani@dmu.ac.uk
---------------------------------------------------------*/

--------------------
DENAR1.startup file
--------------------
echo 'send host-name "DENAR1";' >> /etc/dhcp3/dhclient.conf
dhclient eth0

--------------------
DENAR2.startup file
--------------------
echo 'send host-name "DENAR2";' >> /etc/dhcp3/dhclient.conf
dhclient eth0

--------------------
DENAR3.startup file
--------------------
echo 'send host-name "DENAR3";' >> /etc/dhcp3/dhclient.conf
dhclient eth0

--------------------
DENAR4.startup file
--------------------
echo 'send host-name "DENAR4";' >> /etc/dhcp3/dhclient.conf
dhclient eth0

--------------------
pc1-1.startup file
--------------------
echo 'send host-name "pc1-1";' >> /etc/dhcp3/dhclient.conf
dhclient eth


and so on for other PCs
```

Listing A.9: DENARs and PCs Configuration in Decentralised PBM

A.2 DENAR Software

A.2.1 XACML Policy and Request

```
/* ------------------------------------------------------
* @author Ali Alzahrani (STRL, DMU, UK)
* aalzahrani@dmu.ac.uk
--------------------------------------------------------*/

<?xml version="1.0" encoding="UTF-8" ?>
<Policy xmlns="urn:oasis:names:tc:xacml:1.0:policy" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance" xsi:schemaLocation="urn:oasis:names:tc:xacml:1.0:policy
cs-xacml-schema-policy-01.xsd" PolicyId="Policy0000" RuleCombiningAlgId=
"urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:first-applicable">
    <Target />
    <Rule RuleId="Rule0001" Effect="Permit">
        <Target>
            <Subjects>
                <Subject>
                    <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:
                    string-equal">
                        <AttributeValue DataType="http://www.w3.org/2001/
                        XMLSchema#string">ali</AttributeValue>
                        <SubjectAttributeDesignator DataType="http://www.w3.org/2001/
                        XMLSchema#string" AttributeId="urn:oasis:names:tc:xacml:1.0:
                        subject:subject-id" />
                    </SubjectMatch>
                </Subject>
            </Subjects>
            <Resources>
                <Resource>
                    <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:
                    function:string-equal">
                        <AttributeValue DataType="http://www.w3.org/2001/
                        XMLSchema#string">/hosthome/XACML-Project/Math1/exam1
                        </AttributeValue>
                        <ResourceAttributeDesignator DataType="http://www.w3.org/2001/
                        XMLSchema#string" AttributeId="urn:oasis:names:tc:xacml:1.0:
                        resource:resource-id" />
                    </ResourceMatch>
                </Resource>
                    <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:
                    function:string-equal">
                        <AttributeValue DataType="http://www.w3.org/2001/
                        XMLSchema#string">/hosthome/XACML-Project/Computer1/exam1
                        </AttributeValue>
                        <ResourceAttributeDesignator DataType="http://www.w3.org/2001/
                        XMLSchema#string" AttributeId="urn:oasis:names:tc:xacml:1.0:
                        resource:resource-id" />
                    </ResourceMatch>
                </Resource>
                    <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:
```

```
                    function:string-equal">
                        <AttributeValue DataType="http://www.w3.org/2001/
                        XMLSchema#string">/hosthome/XACML-Project/Accounting1/exam1
                        </AttributeValue>
                        <ResourceAttributeDesignator DataType="http://www.w3.org/2001/
                        XMLSchema#string" AttributeId="urn:oasis:names:tc:xacml:1.0:
                        resource:resource-id" />
                    </ResourceMatch>
                </Resource>
                    <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:
                    function:string-equal">
                        <AttributeValue DataType="http://www.w3.org/2001/
                        XMLSchema#string">/hosthome/XACML-Project/Research/exam1
                        </AttributeValue>
                        <ResourceAttributeDesignator DataType="http://www.w3.org/2001/
                        XMLSchema#string" AttributeId="urn:oasis:names:tc:xacml:1.0:
                        resource:resource-id" />
                    </ResourceMatch>
                </Resource>
            </Resources>
            <Actions>
                <AnyAction />
            </Actions>
        </Target>
    </Rule>
</Policy>
```

Listing A.10: XACML Policy (Policy.xml)

```
/* -------------------------------------------------------
* @author Ali Alzahrani (STRL, DMU, UK)
* aalzahrani@dmu.ac.uk
------------------------------------------------------------*/

<?xml version="1.0" encoding="UTF-8" ?>
<Policy xmlns="urn:oasis:names:tc:xacml:1.0:policy" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance" xsi:schemaLocation="urn:oasis:names:tc:xacml:1.0:policy
cs-xacml-schema-policy-01.xsd" PolicyId="Policy0000" RuleCombiningAlgId=
"urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:first-applicable">
    <Target />
    <Rule RuleId="Rule0001" Effect="Permit">
        <Target>
            <Subjects>
                <Subject>
                    <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:
                    string-equal">
                        <AttributeValue DataType="http://www.w3.org/2001/
                        XMLSchema#string">ali</AttributeValue>
                        <SubjectAttributeDesignator DataType="http://www.w3.org/2001/
                        XMLSchema#string" AttributeId="urn:oasis:names:tc:xacml:1.0:
                        subject:subject-id" />
                    </SubjectMatch>
                </Subject>
            </Subjects>
            <Resources>
                <Resource>
                    <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:
                    function:string-equal">
                        <AttributeValue DataType="http://www.w3.org/2001/
                        XMLSchema#string">/hosthome/XACML-Project/Math1/exam1
                        </AttributeValue>
                        <ResourceAttributeDesignator DataType="http://www.w3.org/2001/
                        XMLSchema#string" AttributeId="urn:oasis:names:tc:xacml:1.0:
                        resource:resource-id" />
                    </ResourceMatch>
                </Resource>
            </Resources>
            <Actions>
                <AnyAction />
            </Actions>
        </Target>
    </Rule>
</Policy>
```

Listing A.11: XACML Policy (sub-policy1.xml)

```
/* ----------------------------------------------------------
* @author Ali Alzahrani (STRL, DMU, UK)
* aalzahrani@dmu.ac.uk
-----------------------------------------------------------*/

<?xml version="1.0" encoding="UTF-8" ?>
<Policy xmlns="urn:oasis:names:tc:xacml:1.0:policy" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance" xsi:schemaLocation="urn:oasis:names:tc:xacml:1.0:policy
cs-xacml-schema-policy-01.xsd" PolicyId="Policy0000" RuleCombiningAlgId=
"urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:first-applicable">
    <Target />
    <Rule RuleId="Rule0001" Effect="Permit">
        <Target>
            <Subjects>
                <Subject>
                    <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:
                    string-equal">
                        <AttributeValue DataType="http://www.w3.org/2001/
                        XMLSchema#string">ali</AttributeValue>
                        <SubjectAttributeDesignator DataType="http://www.w3.org/2001/
                        XMLSchema#string" AttributeId="urn:oasis:names:tc:xacml:1.0:
                        subject:subject-id" />
                    </SubjectMatch>
                </Subject>
            </Subjects>
            <Resources>
                    <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:
                    function:string-equal">
                        <AttributeValue DataType="http://www.w3.org/2001/
                        XMLSchema#string">/hosthome/XACML-Project/Computer1/exam1
                        </AttributeValue>
                        <ResourceAttributeDesignator DataType="http://www.w3.org/2001/
                        XMLSchema#string" AttributeId="urn:oasis:names:tc:xacml:1.0:
                        resource:resource-id" />
                    </ResourceMatch>
                </Resource>
            </Resources>
            <Actions>
                <AnyAction />
            </Actions>
        </Target>
      </Rule>
    </Policy>
```

Listing A.12: XACML Policy (sub-policy2.xml)

```
/* -------------------------------------------------------
* @author Ali Alzahrani (STRL, DMU, UK)
* aalzahrani@dmu.ac.uk
---------------------------------------------------------*/

<?xml version="1.0" encoding="UTF-8" ?>
<Policy xmlns="urn:oasis:names:tc:xacml:1.0:policy" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance" xsi:schemaLocation="urn:oasis:names:tc:xacml:1.0:policy
cs-xacml-schema-policy-01.xsd" PolicyId="Policy0000" RuleCombiningAlgId=
"urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:first-applicable">
    <Target />
    <Rule RuleId="Rule0001" Effect="Permit">
        <Target>
            <Subjects>
                <Subject>
                    <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:
                    string-equal">
                        <AttributeValue DataType="http://www.w3.org/2001/
                        XMLSchema#string">ali</AttributeValue>
                        <SubjectAttributeDesignator DataType="http://www.w3.org/2001/
                        XMLSchema#string" AttributeId="urn:oasis:names:tc:xacml:1.0:
                        subject:subject-id" />
                    </SubjectMatch>
                </Subject>
            </Subjects>
            <Resources>
                    <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:
                    function:string-equal">
                        <AttributeValue DataType="http://www.w3.org/2001/
                        XMLSchema#string">/hosthome/XACML-Project/Accounting1/exam1
                        </AttributeValue>
                        <ResourceAttributeDesignator DataType="http://www.w3.org/2001/
                        XMLSchema#string" AttributeId="urn:oasis:names:tc:xacml:1.0:
                        resource:resource-id" />
                    </ResourceMatch>
                </Resource>
            </Resources>
            <Actions>
                <AnyAction />
            </Actions>
        </Target>
    </Rule>
</Policy>
```

Listing A.13: XACML Policy (sub-policy3.xml)

```
/* --------------------------------------------------------
* @author Ali Alzahrani (STRL, DMU, UK)
* aalzahrani@dmu.ac.uk
-----------------------------------------------------------*/

<?xml version="1.0" encoding="UTF-8" ?>
<Policy xmlns="urn:oasis:names:tc:xacml:1.0:policy" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance" xsi:schemaLocation="urn:oasis:names:tc:xacml:1.0:policy
cs-xacml-schema-policy-01.xsd" PolicyId="Policy0000" RuleCombiningAlgId=
"urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:first-applicable">
    <Target />
    <Rule RuleId="Rule0001" Effect="Permit">
        <Target>
            <Subjects>
                <Subject>
                    <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:
                    string-equal">
                        <AttributeValue DataType="http://www.w3.org/2001/
                        XMLSchema#string">ali</AttributeValue>
                        <SubjectAttributeDesignator DataType="http://www.w3.org/2001/
                        XMLSchema#string" AttributeId="urn:oasis:names:tc:xacml:1.0:
                        subject:subject-id" />
                    </SubjectMatch>
                </Subject>
            </Subjects>
            <Resources>
                    <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:
                    function:string-equal">
                        <AttributeValue DataType="http://www.w3.org/2001/
                        XMLSchema#string">/hosthome/XACML-Project/Research/exam1
                        </AttributeValue>
                        <ResourceAttributeDesignator DataType="http://www.w3.org/2001/
                        XMLSchema#string" AttributeId="urn:oasis:names:tc:xacml:1.0:
                        resource:resource-id" />
                    </ResourceMatch>
                </Resource>
            </Resources>
            <Actions>
                <AnyAction />
            </Actions>
        </Target>
    </Rule>
</Policy>
```

Listing A.14: XACML Policy (sub-policy4.xml)

```
/* -------------------------------------------------------
* @author Ali Alzahrani (STRL, DMU, UK)
* aalzahrani@dmu.ac.uk
----------------------------------------------------------*/

<?xml version="1.0" encoding="UTF-8" ?>
<Policy xmlns="urn:oasis:names:tc:xacml:1.0:policy" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance" xsi:schemaLocation="urn:oasis:names:tc:xacml:1.0:policy
cs-xacml-schema-policy-01.xsd" PolicyId="Policy0000" RuleCombiningAlgId=
"urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:first-applicable">
    <Target />
    <Rule RuleId="Rule0001" Effect="Permit">
        <Target>
            <Subjects>
                <Subject>
                    <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:
                    string-equal">
                        <AttributeValue DataType="http://www.w3.org/2001/
                        XMLSchema#string">ali</AttributeValue>
                        <SubjectAttributeDesignator DataType="http://www.w3.org/2001/
                        XMLSchema#string" AttributeId="urn:oasis:names:tc:xacml:1.0:
                        subject:subject-id" />
                    </SubjectMatch>
                </Subject>
            </Subjects>
            <Resources>
                    <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:
                    function:string-equal">
                        <AttributeValue DataType="http://www.w3.org/2001/
                        XMLSchema#string">/hosthome/XACML-Project/Math1/exam1
                        </AttributeValue>
                        <ResourceAttributeDesignator DataType="http://www.w3.org/2001/
                        XMLSchema#string" AttributeId="urn:oasis:names:tc:xacml:1.0:
                        resource:resource-id" />
                    </ResourceMatch>
                </Resource>
            </Resources>
            <Actions>
                <AnyAction />
            </Actions>
        </Target>
    </Rule>
</Policy>
```

Listing A.15: The Case Study 1 Policy (PolicyCS1.xml)

```
/* -------------------------------------------------------
```

```
* @author Ali Alzahrani (STRL, DMU, UK)
* aalzahrani@dmu.ac.uk
-----------------------------------------------------------*/

<?xml version="1.0" encoding="UTF-8" ?>
<Policy xmlns="urn:oasis:names:tc:xacml:1.0:policy" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance" xsi:schemaLocation="urn:oasis:names:tc:xacml:1.0:policy
cs-xacml-schema-policy-01.xsd" PolicyId="Policy0000" RuleCombiningAlgId=
"urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:first-applicable">
    <Target />
    <Rule RuleId="Rule0001" Effect="Permit">
        <Target>
            <Subjects>
                <Subject>
                    <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:
                    string-equal">
                        <AttributeValue DataType="http://www.w3.org/2001/
                        XMLSchema#string">ali</AttributeValue>
                        <SubjectAttributeDesignator DataType="http://www.w3.org/2001/
                        XMLSchema#string" AttributeId="urn:oasis:names:tc:xacml:1.0:
                        subject:subject-id" />
                    </SubjectMatch>
                </Subject>
            </Subjects>
            <Resources>
                    <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:
                    function:string-equal">
                        <AttributeValue DataType="http://www.w3.org/2001/
                        XMLSchema#string">/hosthome/XACML-Project/Computer1/exam1
                        </AttributeValue>
                        <ResourceAttributeDesignator DataType="http://www.w3.org/2001/
                        XMLSchema#string" AttributeId="urn:oasis:names:tc:xacml:1.0:
                        resource:resource-id" />
                    </ResourceMatch>
                </Resource>
            </Resources>
            <Actions>
                <AnyAction />
            </Actions>
            <Condition FunctionId="PDPOverlay_Constraint_Checker">
             <SubjectCondition DataType="http://www.w3.org/2001/XMLSchema#string">
             Foundation</SubjectCondition>
             <ObjectCondition DataType="http://www.w3.org/2001/XMLSchema#string">
             (/Computer1/exam1)=OpenBook</ObjectCondition>
             <EnvironmentCondition DataType="http://www.w3.org/2001/XMLSchema#string">
             01/Jan/2013</EnvironmentCondition>
             <EventCondition DataType="http://www.w3.org/2001/XMLSchema#string">
             NotDone(ali,(/Computer1/exam1), Download)</EventCondition>
            </Condition>
        </Target>
    </Rule>
</Policy>
```

Listing A.16: The Case Study 2 Policy (PolicyCS2.xml)

```
/* -------------------------------------------------------
* @author Ali Alzahrani (STRL, DMU, UK)
* aalzahrani@dmu.ac.uk
----------------------------------------------------------*/

<?xml version="1.0" encoding="UTF-8" ?>
<Policy xmlns="urn:oasis:names:tc:xacml:1.0:policy" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance" xsi:schemaLocation="urn:oasis:names:tc:xacml:1.0:policy
cs-xacml-schema-policy-01.xsd" PolicyId="Policy0000" RuleCombiningAlgId=
"urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:first-applicable">
    <Target />
    <Rule RuleId="Rule0001" Effect="Permit">
        <Target>
            <Subjects>
                <Subject>
                    <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:
                    string-equal">
                        <AttributeValue DataType="http://www.w3.org/2001/
                        XMLSchema#string">ali</AttributeValue>
                        <SubjectAttributeDesignator DataType="http://www.w3.org/2001/
                        XMLSchema#string" AttributeId="urn:oasis:names:tc:xacml:1.0:
                        subject:subject-id" />
                    </SubjectMatch>
                </Subject>
            </Subjects>
            <Resources>
                    <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:
                    function:string-equal">
                        <AttributeValue DataType="http://www.w3.org/2001/
                        XMLSchema#string">/hosthome/XACML-Project/Accounting1/exam1
                        </AttributeValue>
                        <ResourceAttributeDesignator DataType="http://www.w3.org/2001/
                        XMLSchema#string" AttributeId="urn:oasis:names:tc:xacml:1.0:
                        resource:resource-id" />
                    </ResourceMatch>
                </Resource>
            </Resources>
            <Actions>
                <AnyAction />
            </Actions>
            <Condition FunctionId="PDPOverlay_Constraint_Checker">
             <SubjectCondition DataType="http://www.w3.org/2001/XMLSchema#string">
             Foundation</SubjectCondition>
             <ObjectCondition DataType="http://www.w3.org/2001/XMLSchema#string">
             (Research1/exam1)=MutipleChoice</ObjectCondition>
             <EnvironmentCondition DataType="http://www.w3.org/2001/XMLSchema#string">
             01/Jan/2013</EnvironmentCondition>
             <EventCondition DataType="http://www.w3.org/2001/XMLSchema#string">
             Done(ali,(Research1/exam1), Download)</EventCondition>
            </Condition>
        </Target>
    </Rule>
</Policy>
```

Listing A.17: The Case Study 3 Policy (PolicyCS3.xml)

```
/* -------------------------------------------------------
* @author Ali Alzahrani (STRL, DMU, UK)
* aalzahrani@dmu.ac.uk
-----------------------------------------------------------*/

<?xml version="1.0" encoding="UTF-8" ?>
<Policy xmlns="urn:oasis:names:tc:xacml:1.0:policy" xmlns:xsi="http://www.w3.org/
2001/XMLSchema-instance" PolicyId="LimitedAccess" RuleCombiningAlgId="urn:oasis:
names:tc:xacml:1.0:rule-combining-algorithm:ordered-permit-overrides">
    <Target />
    <Rule RuleId="Rule0001" Effect="Permit">
        <Target>
            <Subjects>
                <Subject>
                    <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:
                    string-equal">
                        <AttributeValue DataType="http://www.w3.org/2001/
                        XMLSchema#string">ali</AttributeValue>
                        <SubjectAttributeDesignator DataType="http://www.w3.org/
                        2001/XMLSchema#string" AttributeId="urn:oasis:names:tc:
                        xacml:1.0:subject:subject-id" />
                    </SubjectMatch>
                </Subject>
            </Subjects>
            <Resources>
                <Resource>
                    <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:
                    function:string-equal">
                        <AttributeValue DataType="http://www.w3.org/2001/
                        XMLSchema#string">/hosthome/XACML-Project/users/ali
                        </AttributeValue>
                        <ResourceAttributeDesignator DataType=
                        "http://www.w3.org/2001/XMLSchema#string" AttributeId=
                        "urn:oasis:names:tc:xacml:1.0:resource:resource-id" />
                    </ResourceMatch>
                </Resource>
            </Resources>
            <Actions>
                <Action>
                    <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:
                    function:string-equal">
                        <AttributeValue DataType="http://www.w3.org/2001/
                        XMLSchema#string">Download</AttributeValue>
                        <ActionAttributeDesignator DataType="http://
                        www.w3.org/2001/XMLSchema#string" AttributeId="urn:
                        oasis:names:tc:xacml:1.0:action:action-id" />
                    </ActionMatch>
                </Action>
            </Actions>
            <Condition FunctionId="PDPOverlay_Constraint_Checker">
                <SubjectCondition DataType="http://www.w3.org/2001/
                XMLSchema#string">Research</SubjectCondition>
                <ObjectCondition DataType="http://www.w3.org/2001/
                XMLSchema#string">Secret</ObjectCondition>
                <EnvironmentCondition DataType="http://www.w3.org/
                2001/XMLSchema#string">01 Jan 2012
                </EnvironmentCondition>
                <EventCondition DataType="http://www.w3.org/2001/
                XMLSchema#string">has allowed</EventCondition>
            </Condition>
        </Target>
    </Rule>
</Policy>
```

Listing A.18: XACML Policy (LimitedAccessPolicy.xml)

```
/* ------------------------------------------------------
* @author Ali Alzahrani (STRL, DMU, UK)
* aalzahrani@dmu.ac.uk
----------------------------------------------------------*/

<?xml version="1.0" encoding="UTF-8" ?>
<Request xmlns="urn:oasis:names:tc:xacml:1.0:context" xmlns:xsi="http://www.w3.org/
2001/XMLSchema-instance">
    <Subject>
        <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
        DataType="urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name">
            <AttributeValue>ali</AttributeValue>
        </Attribute>
    </Subject>
    <Resource>
        <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
        DataType="http://www.w3.org/2001/XMLSchema#string">
            <AttributeValue>exam</AttributeValue>
        </Attribute>
    </Resource>
    <Action>
        <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
        DataType="http://www.w3.org/2001/XMLSchema#string">
            <AttributeValue>read</AttributeValue>
        </Attribute>
    </Action>
</Request>
```

Listing A.19: XACML Request

A.2.2 Client Class

```
/* ----------------------------------------------------
* @author Ali Alzahrani (STRL, DMU, UK)
* aalzahrani@dmu.ac.uk
------------------------------------------------------*/


import java.io.*;
import java.net.*;
import java.util.*;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
public class client {
    Socket clientsocket = null;
    PrintWriter ToServer = null;
    BufferedReader FromServer = null;
    InputStream input = null;
    OutputStream output = null;
    String serverip = null;
    String username = null;
    String password = null;
    String challenge = null;
    String hashedpasswd = null;
    String Line = null;
    String delims = "[ ]+";
    String delims1 = "/";
    String[] tokens = null;
    File F1 = null;
    String fname = null;
    String[] tokens1 = null;
    String outFileName = null;
    int fileNameadd = 1;
    public client() throws Exception {
        String UserInput = null;
        System.out.println("Please type the server address ");
        BufferedReader FromUser = new BufferedReader(new InputStreamReader(System.in));
        Console console = System.console();
        serverip = FromUser.readLine();
        try {
            clientsocket = new Socket(serverip, 5555);
            ToServer = new PrintWriter(clientsocket.getOutputStream(), true);
            FromServer = new BufferedReader(new InputStreamReader(clientsocket.
            getInputStream()));
            input = clientsocket.getInputStream();
            output = clientsocket.getOutputStream();
            System.out.println("Connected to the server " + serverip);
        } catch (Exception e) {
            System.out.println("Unable to connect to " + serverip);
            System.exit(1);
        }
//Authentication
        challenge = FromServer.readLine();
        System.out.println("Enter Username:");
        username = FromUser.readLine();
        password = FromUser.readLine(); // new String (console.readPassword
        ("Enter password: "));
        hashedpasswd = SHA1(challenge + password);
        ToServer.println(username + ":" + hashedpasswd);
        if ((FromServer.readLine()).equals("1")) {
            System.out.println("--------------- SUCCESSFUL AUTHENTICATION
            [ " + username + " ]---------------");
            System.out.println("Usage: \n To download a file---> Download {file}
            \n To upload a file-----> Upload {file} {Destination Directory}
            \n To delete a file-----> Delete {file} \n To list a directory-->
            List {directroy} \n To quit, Quit \n");
            UserInput = FromUser.readLine();
            while (!UserInput.equals("Quit")) { // compare by reference
                toTokenise(UserInput);
                UserInput = FromUser.readLine();
            }//end of while
            System.out.println("Bye.........!");
            clientsocket.close();
        } //end of if
        else {
            System.out.println("Authentication failure. Exiting ..........");
            clientsocket.close();
        }
    } //end of client constructor
    public void toTokenise(String s1) {
        tokens = s1.split(delims);
        if (tokens[0].equals("Download")) {
            try {
                F1 = new File(tokens[1]);
                if (!(F1.isDirectory())) {
                    System.out.println("Trying to download " + tokens[1]
                    + ".........");
                    ToServer.println(tokens[0] + " " + tokens[1]);
                    try {
                        Line = FromServer.readLine();
                        if (Line.equals("NotApplicable")) {
                            System.out.println("        " + Line);
                        } else {
                            fname = tokens[1];
                            tokens1 = fname.split(delims1);
                            outFileName = tokens1[(tokens1.length - 1)];
                            int bytesRead = 0;
                            byte[] mybytearray = new byte[Integer.parseInt(Line)];
                            System.out.println("File size " + mybytearray.length);
                            int current = 0;
                            //Receive file
                            bytesRead = input.read(mybytearray, 0, mybytearray.length);
                            if (bytesRead == 0) {
                                System.out.println("Failed.[0 Bytes] Downloaded, the
                                file is either empty or does not exist");
                            } else {
                                FileOutputStream fos = new FileOutputStream(fileNameadd
                                + outFileName);
                                current = bytesRead;
                                // thanks to A. C diz for the bug fix
                                do {
                                    bytesRead = input.read(mybytearray, current,
                                    (mybytearray.length - current));
                                    if (bytesRead >= 0) {
                                        current += bytesRead;
                                    }
                                } while (bytesRead > 0);
                                fos.write(mybytearray, 0, current);
                                System.out.println("File Downloaded");
                                fos.flush();
                                fileNameadd++;
                            }
                        }
                    } catch (IOException e) {
                        System.out.println(e);
                    }
                    ToServer.flush();
                } else {
                    System.out.println("ERROR: " + tokens[1] + " is a directory.
                    You can download files only");
                }
            } catch (ArrayIndexOutOfBoundsException e) {
                System.err.println("ERROR: Make sure that the Download command is
                formated properly \n    e.g. Download [File_location]");
            }
        }//end of downloading file
        else {
            System.out.println("ERROR: Unknow command");
        }
    } //end of toTokenise
////////////// hashing the password /////////////
// This code was taken from
//http://www.anyexample.com/programming/java/java_simple_class_to_compute
//_sha_1_hash.xml
    private static String convertToHex(byte[] data) {
        StringBuffer buf = new StringBuffer();
        for (int i = 0; i < data.length; i++) {
            int halfbyte = (data[i] >>> 4) & 0x0F;
            int two_halfs = 0;
            do {
                if ((0 <= halfbyte) && (halfbyte <= 9)) {
                    buf.append((char) ('0' + halfbyte));
                } else {
                    buf.append((char) ('a' + (halfbyte - 10)));
                }
                halfbyte = data[i] & 0x0F;
            } while (two_halfs++ < 1);
        }
        return buf.toString();
    }
    public static String SHA1(String text)
            throws NoSuchAlgorithmException, UnsupportedEncodingException {
        MessageDigest md;
        md = MessageDigest.getInstance("SHA-1");
        byte[] sha1hash = new byte[40];
        md.update(text.getBytes("iso-8859-1"), 0, text.length());
        sha1hash = md.digest();
        return convertToHex(sha1hash);
    }
///////////////// end of hashing ////////////////////////////////////////////
    public static void main(String[] args) throws Exception {
        client c1 = new client();
    } // end of main
} //end of class
```

Listing A.20: Client Class

A.2.3 DENAR Class

```
/* -------------------------------------------------------
* @author Ali Alzahrani (STRL, DMU, UK)
* aalzahrani@dmu.ac.uk
-----------------------------------------------------------*/

import java.io.*;
import java.net.*;
import java.util.*;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

class ClientInstance implements Runnable {
    public boolean foundString = false;
    private Socket csokt;
    public String AB = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ";
    ClientInstance(Socket clientsokt) {
        this.csokt = clientsokt;
        System.out.println("New connection from " + csokt.getInetAddress() +
        " on port " + csokt.getPort());
    } // end of ClientInstance constructor
    public synchronized void run() { // assign thread for each client
        String line;
        String delims = "[ ]+";
        String delims1 = "/";
        String delims2 = ":";
        String baseDir = null;
        String[] tokens = null;
        String[] baseDirTokens = null;
        String username = null;
        String password = null;
        String challenge = null;
        String hashedpasswd = null;
        BufferedReader FromClient = null;
        PrintWriter ToClient = null;
        InputStream input = null;
        OutputStream output = null;
        File toBeDeleted = null;
        File toBeListed = null;
        SimplePDP xacmlFram = new SimplePDP();
        String authorisationResult = null;
        try {
            FromClient = new BufferedReader(new InputStreamReader(csokt.
            getInputStream()));
            ToClient = new PrintWriter(csokt.getOutputStream(), true);
            input = csokt.getInputStream();
            output = csokt.getOutputStream();
        } catch (IOException e) {
            System.out.println("I/O Error");
            System.exit(-1);
        }
//Authentication
        try {
            challenge = randomString(12);
            ToClient.println(challenge);
            line = FromClient.readLine();
            String[] tokens2 = line.split(delims2);
            String userRecord = search(tokens2[0]);
            if (!userRecord.equals("0")) {
                String[] frompassfile = userRecord.split(delims2);
                try {
                    hashedpasswd = SHA1(challenge + frompassfile[1]);
                } catch (NoSuchAlgorithmException nsae) {
                }
                if (tokens2[0].equals(frompassfile[0]) && tokens2[1].equals
                (hashedpasswd)) {
                    username = frompassfile[0];
                    System.out.println("[ " + username + " ] AUTHENTICATED
                    SUCCESSFULLY");
                    ToClient.println("1");
/////////////End of Authentication/////////////
                    while (true) {
                        try {
                            line = FromClient.readLine();
                            System.out.println(line);
                            tokens = line.split(delims);
                            baseDirTokens = tokens[1].split(delims1);
                            baseDir = baseDirTokens[0];
                            for (int j = 1; j < baseDirTokens.length - 1; j++) {
                                baseDir += delims1 + baseDirTokens[j];
                            }
                            //reconstruct the base directory of user request
                            if (tokens[0].equals("Download")) {
                                System.out.println("Authorising " + username +
                                "'s requests to download " + tokens[1] +
                                " ............");
                                //baseDir = baseDirTokens[0] + "/" + baseDirTokens[1]
                                + "/" + baseDirTokens[2] + "/" + baseDirTokens[3];
                                try {
                                    authorisationResult = xacmlFram.getXACMLDecision
                                    (username, baseDir, tokens[0]);
                                    System.out.println(authorisationResult);
                                    if (authorisationResult.equals("Permit")) {
```

```
                                File myFile = new File(tokens[1]);
                                ToClient.println(myFile.length());
                                byte[] mybytearray = new byte[(int)
                                myFile.length()];
                                FileInputStream fis = new FileInputStream
                                (myFile);
                                fis.read(mybytearray, 0, mybytearray.length);
                                output.write(mybytearray, 0, mybytearray.
                                length);
                                output.flush();
                            } else {
                                ToClient.println(authorisationResult);
                            }
                        } catch (Exception e) {
                            System.out.println("Unable to donwload " + "("
                            + e.getMessage() + ")");
                        }
                    }
                    else {
                        ToClient.println("Invalid request");
                    }
                } catch (IOException e) {
                    System.out.println("Read failed");
                }
            } // end of while
        }
        ToClient.println("0");
        csokt.close();
        System.out.println("Disconnected");
    } //end of if
    else {
        ToClient.println("0");
        csokt.close();
        System.out.println("Disconnected");
    }
} catch (IOException e) {
} //end of try
} // end of run
/////////////////////// search passwords file //////////////////////////////////
    public String search(String s1) {
        String filename = "passwords.txt";
        BufferedReader inpass = null;

// I initialize the buffered reader to start at the begining every time
        try {
            inpass = new BufferedReader(new FileReader(new File(filename)));
        } catch (IOException e) {
        }
        String searchFor = s1;
        String lineContent = null;
        int currentLine = 0;
        // this will be set to true if the string was found
        while (true) {
            currentLine++;
            // get a line of text from the file
            try {
                lineContent = inpass.readLine();
            } catch (IOException e) {
                break;
            }
            // checks to see if the file ended (in.readLine() returns null if
            the end is reached)
            if (lineContent == null) {
                break;
            }
            if (lineContent.indexOf(searchFor) == -1) {
                continue;
            } else {
                return lineContent;
            }
        } //end of while
        if (!foundString) {
            System.out.println("Authentication failure, Disconnecting .........");
            return "0";
        }
        try {
            inpass.close();
        } catch (IOException ioe) {
        }
        return "0";
    } //end of search function

//////////////// random string generating //////////////////////////////////////////
    // This method was developed by maxp at http://stackoverflow.com/
    //http://stackoverflow.com/questions/41107/how-to-generate-a-random-alpha-numeric-
    string-in-java
    String randomString(int len) {
        Random rnd = new Random();
        StringBuilder sb = new StringBuilder(len);
        for (int i = 0; i < len; i++) {
            sb.append(AB.charAt(rnd.nextInt(AB.length())));
        }
        return sb.toString();
    }
/////////////////////////// hashing password ////////////////////////////////////
// This code was taken from http://www.anyexample.com/programming/java/java_simple
_class_to_compute_sha_1_hash.xml
    private static String convertToHex(byte[] data) {
        StringBuffer buf = new StringBuffer();
        for (int i = 0; i < data.length; i++) {
            int halfbyte = (data[i] >>> 4) & 0x0F;
            int two_halfs = 0;
            do {
                if ((0 <= halfbyte) && (halfbyte <= 9)) {
                    buf.append((char) ('0' + halfbyte));
                } else {
                    buf.append((char) ('a' + (halfbyte - 10)));
                }
                halfbyte = data[i] & 0x0F;
            } while (two_halfs++ < 1);
        }
        return buf.toString();
    }
    public static String SHA1(String text)
            throws NoSuchAlgorithmException, UnsupportedEncodingException {
        MessageDigest md;
        md = MessageDigest.getInstance("SHA-1");
        byte[] sha1hash = new byte[40];
        md.update(text.getBytes("iso-8859-1"), 0, text.length());
        sha1hash = md.digest();
        return convertToHex(sha1hash);
    }
//////////////// end of hashing ///////////
} // end of ClientInstance
public class server {
    ServerSocket Serversocket = null;
    server() {
    } // end of server constructor
    public synchronized void listenSocket() {
        try { //try opening server socket
            Serversocket = new ServerSocket(5555);
            System.out.println("The server is listening on port " + Serversocket
            .getLocalPort());
        } catch (IOException e) {
            System.out.println("Server is unable to listen on port: 5555");
            System.exit(-1);
        }
        while (true) { //try opening clinet socket
            ClientInstance insClient;
            try {
                insClient = new ClientInstance(Serversocket.accept());
                Thread thread = new Thread(insClient);
                thread.start();
            } catch (IOException e) {
                System.out.println("Unble to accept connections on port: 5555");
                System.exit(-1);
            }
        } //end of while
    } // end of listenSocket
    protected void finalize() { //clean up and release the resources
        try {
            Serversocket.close();
        } catch (IOException e) {
            System.out.println("Could not close socket");
            System.exit(-1);
        }
    } //end of finalize
    public static void main(String[] args) {
        server server1 = new server();
        server1.listenSocket();
    }
}
```

Listing A.21: DENAR Class

A.2.4 tester Class

```
/* --------------------------------------------------------
* @author Ali Alzahrani (STRL, DMU, UK)
* aalzahrani@dmu.ac.uk
----------------------------------------------------------*/


import java.io.*;
import java.util.Random;
public class tester {
    public tester() {
    }
    public static void wait1sec() {
        long t0, t1;
        t0 = System.currentTimeMillis();
        do {
            t1 = System.currentTimeMillis();
        } while (t1 - t0 < 100);
    }
    public static void main(String[] args) throws Exception {
        System.out.println(args[0]);
        System.out.println(args[1]);
        System.out.println(args[2]);
        Random xRandom = new Random(Integer.parseInt(args[3]));
        int randomNumber;
        int randomUser;
        int randomFile;
        File toBeListed;
        String uploadDestination;
        String Basedir = "/hosthome/XACML-Project/users/";
        String userNames[] = {"adam", "ali"};
        for (int i = 0; i < 50; ++i) {
            randomNumber = xRandom.nextInt(1);
            randomUser = xRandom.nextInt(1);
            //System.out.println("Generated: " + randomNumber);
            switch (randomNumber) {
                case 0:
                    wait1sec();
                    toBeListed = new File(Basedir + "ali");
                    File[] dFiles = toBeListed.listFiles();
                    randomFile = xRandom.nextInt(dFiles.length);
                    System.out.println("Download" + dFiles[randomFile].toString());
                    break;
                case 1: //exit
                    System.exit(0);
                default:
                    System.out.println("\n This case is from 0 to 3\n");
            } //end of case
        }
        System.out.println("Quit");
    }
}
```

Listing A.22: tester Class

A.2.5 SimplePDP Class

```
/* -------------------------------------------------------
* @author Ali Alzahrani (STRL, DMU, UK)
* aalzahrani@dmu.ac.uk
----------------------------------------------------------*/

import java.io.*;
import java.util.*;
import com.sun.xacml.*;
import com.sun.xacml.ctx.*;
import com.sun.xacml.finder.*;
import com.sun.xacml.finder.impl.*;
import org.w3c.dom.*;
import org.xml.sax.*;
import javax.xml.parsers.*;
import javax.xml.transform.*;
import javax.xml.transform.dom.*;
import javax.xml.transform.stream.*;
import java.io.FileInputStream;
import java.util.*;
import com.sun.xacml.PDPConfig;
import com.sun.xacml.ParsingException;
import com.sun.xacml.cond.FunctionFactory;
import com.sun.xacml.cond.FunctionFactoryProxy;
import com.sun.xacml.cond.StandardFunctionFactory;
import com.sun.xacml.ctx.RequestCtx;
import com.sun.xacml.ctx.ResponseCtx;
import com.sun.xacml.finder.AttributeFinder;
import com.sun.xacml.finder.PolicyFinder;
import com.sun.xacml.finder.impl.CurrentEnvModule;
import com.sun.xacml.finder.impl.FilePolicyModule;

public class SimplePDP {
    ConstraintCheckerFunction ConstraintCheckerFunction;
    public void SimplePDP() {
    }   ;
 public String getXACMLDecision(String subjectString, String resourceString,
 String actionString) throws Exception {
        ConstraintCheckerFunction = new ConstraintCheckerFunction();
        // load the policies
        FilePolicyModule policyModule = new FilePolicyModule();
        policyModule.addPolicy("LimitedAccessPolicy.xml");
        // setup the policy finder
        PolicyFinder policyFinder = new PolicyFinder();
        Set policyModules = new HashSet();
        policyModules.add(policyModule);
        policyFinder.setModules(policyModules);
        // module to provide the current date & time
        CurrentEnvModule envModule = new CurrentEnvModule();
        // setup the attribute finder
        AttributeFinder attrFinder = new AttributeFinder();
        List attrModules = new ArrayList();
        attrModules.add(envModule);
        attrFinder.setModules(attrModules);
        // load the constraint checker function designed to access the
        //coordination object of the overlay network
        FunctionFactoryProxy proxy = StandardFunctionFactory.getNewFactoryProxy();
        FunctionFactory factory = proxy.getConditionFactory();
        factory.addFunction(new ConstraintCheckerFunction());
        FunctionFactory.setDefaultFactory(proxy);
        // create the PDP
        PDP pdp = new PDP(new PDPConfig(attrFinder, policyFinder, null));
        // now work on the request
        String xacmlRequest = "<?xml version=\"1.0\" encoding=\"UTF-8\"?>"
                + "<Request     xmlns=\"urn:oasis:names:tc:xacml:1.0:context\"
                xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instanc\"
                xsi:schemaLocation=\"urn:oasis:names:tc:xacml:1.0:context
                cs-xacml-schema-context-01.xsd\">"
                + "<Subject>"
                + "<Attribute AttributeId=\"urn:oasis:names:tc:xacml:1.0:
                subject:subject-id\"   DataType=\"http://www.w3.org/2001/
                XMLSchema#string\">"
                + "<AttributeValue>" + subjectString + "</AttributeValue>"
                + "</Attribute>"
                + "</Subject>"
                + "<Resource>"
                + "<Attribute AttributeId=\"urn:oasis:names:tc:xacml:1.0:
                resource:resource-id\" DataType=\"http://www.w3.org/2001/
                XMLSchema#string\">" + "<AttributeValue>" + resourceString +
                "</AttributeValue>"
                + "</Attribute>"
                + "</Resource>"
                + "<Action>"
                + "<Attribute AttributeId=\"urn:oasis:names:tc:xacml:1.0:
                action:action-id\" DataType=\"http://www.w3.org/2001/
                XMLSchema#string\">" +
                "<AttributeValue>" + actionString + "</AttributeValue>"
                + "</Attribute>"
                + "</Action>"
                + "</Request>";
//convert the string to xml
        DocumentBuilderFactory docFactory0 = DocumentBuilderFactory.newInstance();
        DocumentBuilder requestDocument = docFactory0.newDocumentBuilder();
        InputSource inputSourceReq = new InputSource();
        inputSourceReq.setCharacterStream(new StringReader(xacmlRequest));
        Document xacmlRequestDoc = requestDocument.parse(inputSourceReq);
        Node requestNode = xacmlRequestDoc.getDocumentElement();
//xacml request and response
        RequestCtx request = RequestCtx.getInstance(requestNode);
        ResponseCtx response = pdp.evaluate(request);
// re-create xacml response to extract only the decision values
        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        response.encode(baos);
        String responseString = baos.toString();
        DocumentBuilderFactory docFactory1 = DocumentBuilderFactory.newInstance();
        DocumentBuilder responseDocument = docFactory1.newDocumentBuilder();
        InputSource inputSourceRes = new InputSource();
        inputSourceRes.setCharacterStream(new StringReader(responseString));
        Document xacmlResponseDoc = responseDocument.parse(inputSourceRes);
        xacmlResponseDoc.getDocumentElement().normalize();
        NodeList nodeLst = xacmlResponseDoc.getElementsByTagName("Result");
        Node resultNode = nodeLst.item(0);
        String decisionValue = null;
        if (resultNode.getNodeType() == Node.ELEMENT_NODE) {
            Element resultElmnt = (Element) resultNode;
            NodeList decisionNodeLst = resultElmnt.getElementsByTagName("Decision");
            Element decisionNodeElmnt = (Element) decisionNodeLst.item(0);
            NodeList decisionNode = decisionNodeElmnt.getChildNodes();
            decisionValue = ((Node) decisionNode.item(0)).getNodeValue();
            getNodeValue());
        }
        decisionValue = ConstraintCheckerFunction.PolicyChecker(subjectString,
        resourceString, actionString);
        return decisionValue;
    }
}
```

Listing A.23: SimplePDP Class

A.2.6 ConstraintCheckerFunction Class

```
/* -------------------------------------------------------
* @author Ali Alzahrani (STRL, DMU, UK)
* aalzahrani@dmu.ac.uk
---------------------------------------------------------*/

import com.sun.xacml.EvaluationCtx;
import com.sun.xacml.attr.AttributeValue;
import com.sun.xacml.attr.BooleanAttribute;
import com.sun.xacml.attr.IntegerAttribute;
import com.sun.xacml.attr.StringAttribute;
import com.sun.xacml.cond.EvaluationResult;
import com.sun.xacml.cond.FunctionBase;
import java.text.SimpleDateFormat;
import java.util.*;
import java.io.*;
import java.net.*;
import org.w3c.dom.Document;
import org.w3c.dom.*;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.DocumentBuilder;
import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;
import java.util.List;

public class ConstraintCheckerFunction extends FunctionBase {
    /**
     * The identifier for this function
     */
    public static final String NAME =
            "PDPOverlay_Constraint_Checker";
    PIPcoordinator PIPcoordinator;
    boolean success;
    int currentValue;
    boolean isPermitted;
    /**
     * The attributes that the policy must pass onto this function are:
     *String subject name
     *String resource name
     *String action name
     *String environment name
     * Constructor that accepts mixed data type inputs as specified for Sun's
     * XACML PDP implementation
     */
    public ConstraintCheckerFunction() {
        super(NAME, 0, attributes, areBags, BooleanAttribute.identifier, false);
        System.out.println("ConstraintCheckerFunction Function Starting");
    }
    public String PolicyChecker(String subjectRequest, String objectRequest,
    String actionRequest) {
        String ruleID = "";
        String subjectPolicy = "";
        String objectPolicy = "";
        String actionPolicy = "";
        boolean conditionEvaluationResult = false;
        String conditionPolicy = "";
        String subjectCondition = "";
        String objectCondition = "";
        String environmentCondition = "";
        String eventCondition = "";
        String decisionResult = "";
        try {
            DocumentBuilderFactory docBuilderFactory = DocumentBuilderFactory.
            newInstance();
            DocumentBuilder docBuilder = docBuilderFactory.newDocumentBuilder();
            Document doc = docBuilder.parse(new File("LimitedAccessPolicy.xml"));
            // normalize text representation
            doc.getDocumentElement().normalize();
            NodeList listOfPersons = doc.getElementsByTagName("Subject");
            int totalPersons = listOfPersons.getLength();
            for (int s = 0; s < listOfPersons.getLength(); s++) {
                Node firstPersonNode = listOfPersons.item(s);
                if (firstPersonNode.getNodeType() == Node.ELEMENT_NODE) {
                    Element firstPersonElement = (Element) firstPersonNode;
                    //-------
                    NodeList firstNameList = firstPersonElement.
                    getElementsByTagName("AttributeValue");
                    Element firstNameElement = (Element) firstNameList.item(0);
                    NodeList textFNList = firstNameElement.getChildNodes();
                    subjectPolicy = ((Node) textFNList.item(0)).getNodeValue().trim();
                }//end of if clause
            }//end of for loop with s var
            NodeList listOfResources = doc.getElementsByTagName("Resource");
            int totalResources = listOfResources.getLength();
            for (int s = 0; s < listOfResources.getLength(); s++) {
                Node firstResourceNode = listOfResources.item(s);
                if (firstResourceNode.getNodeType() == Node.ELEMENT_NODE) {
                    Element firstResourceElement = (Element) firstResourceNode;
                    //-------
                    NodeList firstObjectList = firstResourceElement.
                    getElementsByTagName("AttributeValue");
                    Element firstObjectElement = (Element) firstObjectList.item(0);
                    NodeList textFOList = firstObjectElement.getChildNodes();
                    objectPolicy = ((Node) textFOList.item(0)).getNodeValue().trim();
                    objectPolicy);
                }//end of if clause
            }//end of for loop with s var
///////////////------------------------------
            NodeList listOfActions = doc.getElementsByTagName("Action");
            int totalActions = listOfActions.getLength();
            for (int s = 0; s < listOfActions.getLength(); s++) {
                Node firstActionNode = listOfActions.item(s);
                if (firstActionNode.getNodeType() == Node.ELEMENT_NODE) {
                    Element firstActionElement = (Element) firstActionNode;
                    //-------
                    NodeList firstActList = firstActionElement.
                    getElementsByTagName("AttributeValue");
                    Element firstActElement = (Element) firstActList.item(0);
                    NodeList textFAList = firstActElement.getChildNodes();
                    actionPolicy = ((Node) textFAList.item(0)).getNodeValue().trim();
                }//end of if clause
            }//end of for loop with s var
///////////////------------------------------
            NodeList listOfConditions = doc.getElementsByTagName("Condition");
            int totalConditions = listOfConditions.getLength();
            //System.out.println("Total no of Conditions : " + totalConditions);
            for (int c = 0; c < listOfConditions.getLength(); c++) {
                Node firstConditionNode = listOfConditions.item(c);
                if (firstConditionNode.getNodeType() == Node.ELEMENT_NODE) {
                    Element firstConditionElement = (Element) firstConditionNode;
                    subjectCondition = getTagValue("SubjectCondition",
                    firstConditionElement);
                    objectCondition = getTagValue("ObjectCondition",
                    firstConditionElement);
                    environmentCondition = getTagValue("EnvironmentCondition",
                    firstConditionElement);
                    eventCondition = getTagValue("EventCondition",
                    firstConditionElement);
                    System.out.println("Subject Condition  : " + subjectCondition);
                    System.out.println("Object Condition  : " + objectCondition);
                    System.out.println("Environment Condition  : " +
                    environmentCondition);
                    System.out.println("Event Condition  : " + eventCondition);
                }//end of if clause
            }//end of for loop with s var
///////////////------------------------------
            System.out.println("subject Request = " + subjectRequest);
            System.out.println("subject Policy = " + subjectPolicy);
            System.out.println("object Request = " + objectRequest);
            System.out.println("object Policy = " + objectPolicy);
            System.out.println("action Request = " + actionRequest);
            System.out.println("action Policy = " + actionPolicy);
            //System.out.println("Calling PIP");
            String y = "";
            String g = "Alui";
        } catch (SAXParseException err) {
            System.out.println("** Parsing error" + ", line "
                    + err.getLineNumber() + ", uri " + err.getSystemId());
            System.out.println(" " + err.getMessage());
        } catch (SAXException e) {
            Exception x = e.getException();
            ((x == null) ? e : x).printStackTrace();
        } catch (Throwable t) {
            t.printStackTrace();
        }
////////------------
        try {
            PIPcoordinator = new PIPcoordinator();
            System.out.println("Calling PIPcoordanitor");
            String PIPfind = "PIPdata.xml";
            // PIPdata.xml is supposed value here but the value can be found in
            obligation context
            String job = "findAttribute";
            // can be sent form PEP, it maybe ("findAttribute" for authrisation)
            or ("addAttribute" or "updateAttribute" for obligation).
            String nodeAttribute = "";
            // this value is assigned in update attribute the node updated can
            be found in obligation context to be updated in PIP
            String oldAttribute = "";
            // this value is the old value in PIP for the above node
            String newAttribute = "";
            // this value is the new value in PIP for the above node  which can
            be found in obligation context
            conditionEvaluationResult = PIPcoordinator.PIPcoordinatorTask(PIPfind,
            job, subjectPolicy, objectPolicy, actionPolicy, subjectCondition,
            objectCondition, environmentCondition, eventCondition,
            nodeAttribute, oldAttribute, newAttribute);
            System.out.println("Policy decision Result = " + decisionResult);
            System.out.println("condition Evaluation Result = " +
            conditionEvaluationResult);
            if (conditionEvaluationResult == true) {
                decisionResult = "Permit";
                System.out.println("decisionResult = " + decisionResult);
            } else {
                decisionResult = "NotApplicable";
            }
        } catch (Exception e) {
            System.out.print("Connect to another PDP in network? (y or n): ");
        }
        return (decisionResult);
    }
//////////
    private static String getTagValue(String sTag, Element eElement) {
        NodeList nlList = eElement.getElementsByTagName(sTag).item(0).getChildNodes();
        Node nValue = (Node) nlList.item(0);
        return nValue.getNodeValue();
    }
//////////////
    public EvaluationResult evaluate(List inputs, EvaluationCtx context) {
        System.out.println("result=");
        System.out.println(context);
        System.out.println("ssssssssssssssss");
        //Get the current value from the PDP Overlay coordination object
        int[] x = new int[2];
        try {
            PIPcoordinator = new PIPcoordinator();
            System.out.println("Calling PIPcoordanitor");
            String Attr = "";
            System.out.println("success = " + success);
            if (x[0] == 1) {
                isPermitted = false;
            } else {
                isPermitted = true;
            }
            if (success == true) {
                isPermitted = true;
            } else {
                isPermitted = false;
            }
            isPermitted = true;
            System.out.println("isPermitted = " + isPermitted);
        } catch (Exception e) {
            Constraint Checker");
            System.out.print("Connect to another PDP in network? (y or n): ");
        }
        //Return the evaluation result using the getInstance class designed
        by Sun microsystems
        return EvaluationResult.getInstance(isPermitted);
    }
}
```

Listing A.24: ConstraintCheckerFunction Class

A.2.7 PIPcoordinator Class

```
/* ------------------------------------------------------
* @author Ali Alzahrani (STRL, DMU, UK)
* aalzahrani@dmu.ac.uk
--------------------------------------------------------*/
```

```
import java.util.*;
import java.io.*;
import java.net.*;
import org.w3c.dom.Document;
import org.w3c.dom.*;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.DocumentBuilder;
import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;

public class PIPcoordinator {
    public static PIP pip;
    public static RemotePIPcoordinator remotePIPcoordinator;
//////////////////////
// Finding is PIPs local or remote
/////////////////////
    public boolean PIPcoordinatorTask(String PIPfind, String job, String subjectPolicy,
    String objectPolicy, String actionPolicy, String subjectCondition, String
    objectCondition, String environmentCondition, String eventCondition, String
    nodeAttribute, String oldAttribute, String newAttribute) throws Exception {
        String RemotePIPindicator = "@";
        boolean RemoteAttributeSubject = false;
        boolean RemoteAttributeObject = false;
        boolean RemoteAttributeEnvironment = false;
        boolean RemoteAttributeEvent = false;
        boolean isPermitted = false;
        if (subjectCondition.indexOf(RemotePIPindicator.charAt(0)) >= 0) {
            RemoteAttributeSubject = true;
        }
        if (objectCondition.indexOf(RemotePIPindicator.charAt(0)) >= 0) {
            RemoteAttributeObject = true;
        }
        if (environmentCondition.indexOf(RemotePIPindicator.charAt(0)) >= 0) {
            RemoteAttributeEnvironment = true;
        }
        if (eventCondition.indexOf(RemotePIPindicator.charAt(0)) >= 0) {
            RemoteAttributeEvent = true;
        }
        if (RemoteAttributeSubject == false || RemoteAttributeObject == false ||
        RemoteAttributeEnvironment == false || RemoteAttributeEvent == false) {
            if (job.equals("addAttribute")) {
                addToLocalPIP(PIPfind, subjectPolicy, objectPolicy, actionPolicy,
                        subjectCondition, objectCondition, environmentCondition,
                        eventCondition);
                isPermitted = true;
            } else if (job.equals("updateAttribute")) {
                updateLocalPIP(PIPfind, subjectPolicy, objectPolicy, actionPolicy,
                        nodeAttribute, oldAttribute, newAttribute);
                isPermitted = true;
            } else if (job.equals("findAttribute")) {
                isPermitted = getAttributeLocalPIP(subjectPolicy, objectPolicy,
                actionPolicy, subjectCondition, objectCondition, environmentCondition,
                eventCondition);
                isPermitted = true;
            }
        } else {
            if (job.equals("addAttribute")) {
                addToRemotePIP(PIPfind, job, subjectPolicy, objectPolicy, actionPolicy,
                subjectCondition, objectCondition, environmentCondition, eventCondition);
                isPermitted = true;
            } else if (job.equals("updateAttribute")) {
                updateRemotePIP(PIPfind, job, subjectPolicy, objectPolicy,
                actionPolicy, nodeAttribute, oldAttribute, newAttribute);
                isPermitted = true;
            } else if (job.equals("findAttribute")) {
                isPermitted = getAttributeRemotePIP(PIPfind, job, subjectPolicy,
                objectPolicy, actionPolicy, subjectCondition, objectCondition,
                environmentCondition, eventCondition);
            }
        }
        return (isPermitted);
    }

///////////////////
// Local PIPs
///////////////////
    public static void addToLocalPIP(String PIPadd, String subject, String resource,
    String action, String subjectAttribute, String objectAttribute,
    String environmentAttribute, String eventAttribute) throws Exception {
        pip.addAttributesToDB(PIPadd, subject, resource, action, subjectAttribute,
        objectAttribute, environmentAttribute, eventAttribute);
    }
////////////////////
    public static void updateLocalPIP(String PIPupdate, String subject, String
    resource, String action, String nodeAttribute, String oldAttribute, String
    newAttribute) throws Exception { pip.updateAttributesDB(PIPupdate, subject,
    resource, action, nodeAttribute, oldAttribute, newAttribute);
    }
/////////////////////
    public boolean getAttributeLocalPIP(String subjectPolicy, String objectPolicy,
    String actionPolicy, String subjectCondition, String objectCondition, String
    environmentCondition, String eventCondition) {
        boolean isPermitted = false;
        String subjectPIP = subjectPolicy;
        String objectPIP = objectPolicy;
        String actionPIP = actionPolicy;
        String subjectconditionPIP = "";
        String objectconditionPIP = "";
        String environmentconditionPIP = "";
        String eventConditionPIP = "";
        try {
            System.out.println("Calling PIP");
            subjectconditionPIP = pip.getSubjectCondtionAttribute(subjectPIP,
            objectPIP, actionPIP);
            objectconditionPIP = pip.getObjectCondtionAttribute(subjectPIP,
            objectPIP, actionPIP);
            environmentconditionPIP = pip.getEnvironmentCondtionAttribute
            (subjectPIP, objectPIP, actionPIP);
            eventConditionPIP = pip.getEventCondtionAttribute(subjectPIP,
            objectPIP, actionPIP);
            if (subjectconditionPIP.equals(subjectCondition) &&
            objectconditionPIP.equals(objectCondition) &&
            environmentconditionPIP.equals(environmentCondition) &&
            eventConditionPIP.equals(eventCondition)) {
                isPermitted = true;
            }
            //System.out.println("PIP called");
        } catch (Throwable t) {
            t.printStackTrace();
        }
        return (isPermitted);
    }

///////////////////////
// Remote PIPs through remote PIPcoordinator
////////////////////////
    public static void connectionToRemotePIP() throws Exception {
        ServerSocket PIPcoordinatorSocket = null;
        try {
            PIPcoordinatorSocket = new ServerSocket(7777);
        } catch (IOException e) {
            System.err.println("Could not listen on port: 7777.");
            System.exit(1);
        }
        Socket PIPsocket = null;
        try {
            PIPsocket = PIPcoordinatorSocket.accept();
        } catch (IOException e) {
            System.err.println("Accept failed.");
            System.exit(1);
        }
        PrintWriter out = new PrintWriter(PIPsocket.getOutputStream(), true);
        BufferedReader in = new BufferedReader(
                new InputStreamReader(
                PIPsocket.getInputStream()));
        String inputLine, outputLine;
        RemotePIPProtocol kkp = new RemotePIPProtocol();
        outputLine = kkp.processInput(null);
        out.println(outputLine);
        while ((inputLine = in.readLine()) != null) {
            outputLine = kkp.processInput(inputLine);
            out.println(outputLine);
            if (outputLine.equals("Bye.")) {
                break;
            }
        }
        out.close();
        in.close();
        PIPsocket.close();
        PIPcoordinatorSocket.close();
    }
///////////////
    public static void addToRemotePIP(String PIPadd, String job, String subject,
    String resource, String action, String subjectAttribute, String objectAttribute,
    String environmentAttribute, String eventAttribute) throws Exception {
        connectionToRemotePIP();
        boolean addingNewAttribute = false;
        String nodeAttribute = "";
        String oldAttribute = "";
        String newAttribute = "";
        addingNewAttribute = remotePIPcoordinator.connectionToRemotePIP(PIPadd,
        job, subject, resource, action, subjectAttribute, objectAttribute,
        environmentAttribute, eventAttribute, nodeAttribute, oldAttribute,
        newAttribute);
    }

//////////////////
    public static void updateRemotePIP(String PIPupdate, String job, String subject,
    String resource, String action, String nodeAttribute, String oldAttribute,
    String newAttribute) throws Exception {
        connectionToRemotePIP();
        boolean updatingAttribute = false;
        String subjectAttribute = "";
        String objectAttribute = "";
        String environmentAttribute = "";
        String eventAttribute = "";
        updatingAttribute = remotePIPcoordinator.connectionToRemotePIP(PIPupdate,
        job, subject, resource, action, subjectAttribute, objectAttribute,
        environmentAttribute, eventAttribute, nodeAttribute, oldAttribute,
        newAttribute);
    }
///////////////////////
    public boolean getAttributeRemotePIP(String PIPfind, String job, String
    subjectPolicy, String objectPolicy, String actionPolicy, String
    subjectCondition, String objectCondition, String environmentCondition,
    String eventCondition) throws Exception {
        connectionToRemotePIP();
        boolean evaluatedAttribute = false;
        String nodeAttribute = "";
        String oldAttribute = "";
        String newAttribute = "";
        try {
            evaluatedAttribute = remotePIPcoordinator.connectionToRemotePIP(
            PIPfind, job, subjectPolicy, objectPolicy, actionPolicy,
                    subjectCondition, objectCondition, environmentCondition,
                    eventCondition, nodeAttribute, oldAttribute,
                    newAttribute);
        } catch (Throwable t) {
            t.printStackTrace();
        }
        return (evaluatedAttribute);
    }
}
```

Listing A.25: PIPcoordinator Class

A.2.8 RemotePIPcoordinator Class

```
/* -----------------------------------------------------
* @author Ali Alzahrani (STRL, DMU, UK)
* aalzahrani@dmu.ac.uk
-------------------------------------------------------*/


import java.util.*;
import java.io.*;
import java.net.*;
import org.w3c.dom.Document;
import org.w3c.dom.*;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.DocumentBuilder;
import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;

public class RemotePIPcoordinator {
    public static PIP pip;
////////////////
    public boolean connectionToRemotePIP(String PIPtoConnect, String job, String
    subject, String resource, String action, String subjectAttribute, String
    objectAttribute, String environmentAttribute, String eventAttribute, String
    nodeAttribute, String oldAttribute, String newAttribute) throws Exception {
        boolean isPermitted = false;
        Socket pipsocket = null;
        PrintWriter out = null;
        BufferedReader in = null;
        try {
            pipsocket = new Socket("146.227.66.178", 8888);
            out = new PrintWriter(pipsocket.getOutputStream(), true);
            in = new BufferedReader(new InputStreamReader(pipsocket.getInputStream()));
        } catch (UnknownHostException e) {
            System.err.println("Don't know about host: PIPcoordinator.");
            System.exit(1);
        } catch (IOException e) {
            System.err.println("Couldn't get I/O for the connection to: PIPcoordinator");
            System.exit(1);
        }
        BufferedReader stdIn = new BufferedReader(new InputStreamReader(System.in));
        String fromPIPcoordinator;
        String fromPIP;
        while ((fromPIPcoordinator = in.readLine()) != null) {
            System.out.println("PIPcoordinator: " + fromPIPcoordinator);
            if (fromPIPcoordinator.equals("Bye.")) {
                break;
            }
            fromPIP = stdIn.readLine();
            if (fromPIP != null) {
                System.out.println("PIP: " + fromPIP);
                out.println(fromPIP);
            }
        }
        if (job.equals("addAttribute")) {
            addToLocalPIP(PIPtoConnect, subject, resource, action, subjectAttribute,
            objectAttribute, environmentAttribute, eventAttribute);
        }
        if (job.equals("updateAttribute")) {
            updateLocalPIP(PIPtoConnect, subject, resource, action, nodeAttribute,
            oldAttribute, newAttribute);
        }
        if (job.equals("findAttribute")) {
            isPermitted = getAttributeLocalPIP(subject, resource, action,
            subjectAttribute, objectAttribute, environmentAttribute, eventAttribute);
        }
        out.close();
        in.close();
        stdIn.close();
        pipsocket.close();
        return (isPermitted);
    }

//////////////
// Local PIPs
///////////////
    public static void addToLocalPIP(String PIPadd, String subject, String resource,
    String action, String subjectAttribute, String objectAttribute, String
    environmentAttribute, String eventAttribute) throws Exception {
        pip.addAttributesToDB(PIPadd, subject, resource, action, subjectAttribute,
        objectAttribute, environmentAttribute, eventAttribute);
    }
/////////////////////
    public static void updateLocalPIP(String PIPupdate, String subject,
    String resource, String action, String nodeAttribute, String oldAttribute,
    String newAttribute) throws Exception { pip.updateAttributesDB(PIPupdate,
    subject, resource, action, nodeAttribute, oldAttribute, newAttribute);
    }
////////////////////////
    public boolean getAttributeLocalPIP(String subjectPolicy, String objectPolicy,
    String actionPolicy, String subjectCondition, String objectCondition, String
    environmentCondition, String eventCondition) {
        boolean isPermitted = false;
        String subjectPIP = subjectPolicy;
        String objectPIP = objectPolicy;
        String actionPIP = actionPolicy;
        String subjectconditionPIP = "";
        String objectconditionPIP = "";
        String environmentconditionPIP = "";
        String eventConditionPIP = "";
        try {
            System.out.println("Calling PIP");
            subjectconditionPIP = pip.getSubjectCondtionAttribute(subjectPIP,
            objectPIP, actionPIP);
            objectconditionPIP = pip.getObjectCondtionAttribute(subjectPIP,
            objectPIP, actionPIP);
            environmentconditionPIP = pip.getEnvironmentCondtionAttribute(subjectPIP,
            objectPIP, actionPIP);
            eventConditionPIP = pip.getEventCondtionAttribute(subjectPIP, objectPIP,
            actionPIP);
            if (subjectconditionPIP.equals(subjectCondition) && objectconditionPIP.
            equals(objectCondition) && environmentconditionPIP.equals(environmentCondition)
            && eventConditionPIP.equals(eventCondition)) {
                isPermitted = true;
            }
        } catch (Throwable t) {
            t.printStackTrace();
        }
        return (isPermitted);
    }
}
```

Listing A.26: RemotePIPcoordinator Class

A.2.9 PIP Class

```
/* --------------------------------------------------------
* @author Ali Alzahrani (STRL, DMU, UK)
* aalzahrani@dmu.ac.uk
----------------------------------------------------------*/


import java.util.Vector;
import java.util.ListIterator;
import java.io.*;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerException;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;
import org.w3c.dom.Document;
import org.w3c.dom.NodeList;
import org.w3c.dom.Node;
import org.w3c.dom.Element;
import org.w3c.dom.NamedNodeMap;
import org.xml.sax.SAXException;

public class PIP {
    String x;
    public static String getSubjectCondtionAttribute(String subjectValue, String
    objectValue, String actionValue) {
        String conditionValue = "";
        String SubjectPIP = "";
        String ObjectPIP = "";
        String ActionPIP = "";
        try {
            DocumentBuilderFactory docBuilderFactory = DocumentBuilderFactory.
            newInstance();
            DocumentBuilder docBuilder = docBuilderFactory.newDocumentBuilder();
            Document doc = docBuilder.parse(new File("PIPdata.xml"));
            doc.getDocumentElement().normalize();
            NodeList listOfRules = doc.getElementsByTagName("rule");
            int totalRules = listOfRules.getLength();
            for (int s = 0; s < listOfRules.getLength(); s++) {
                Node firstRuleNode = listOfRules.item(s);
                if (firstRuleNode.getNodeType() == Node.ELEMENT_NODE) {
                    Element firstRuleElement = (Element) firstRuleNode;
//-------
                    NodeList subjectConditionList = firstRuleElement.
                    getElementsByTagName("Subject");
                    Element subjectConditionElement = (Element) subjectConditionList.
                    item(0);
                    NodeList textFSList = subjectConditionElement.getChildNodes();
                    SubjectPIP = ((Node) textFSList.item(0)).getNodeValue().trim();
//-------
                    NodeList objectConditionList = firstRuleElement.
                    getElementsByTagName("Object");
                    Element objectConditionElement = (Element) objectConditionList.
                    item(0);
                    NodeList textFOList = objectConditionElement.getChildNodes();
                    ObjectPIP = ((Node) textFOList.item(0)).getNodeValue().trim();
//-------
                    NodeList actionConditionList = firstRuleElement.
                    getElementsByTagName("Action");
                    Element actionConditionElement = (Element) actionConditionList.
                    item(0);
                    NodeList textFAList = actionConditionElement.getChildNodes();
                    ActionPIP = ((Node) textFAList.item(0)).getNodeValue().trim();
//-------
                    if (subjectValue.equals(SubjectPIP) && objectValue.equals(ObjectPIP)
                    && actionValue.equals(ActionPIP)) { NodeList firstConditionList =
                    firstRuleElement.getElementsByTagName("SubjectAttribute");
                        Element firstConditionElement = (Element) firstConditionList.
                        item(0);
                        NodeList textFNList = firstConditionElement.getChildNodes();
                        conditionValue = ((Node) textFNList.item(0)).getNodeValue().
                        trim();
                        break;
                    }//end of if clause
                }//end of if clause
            }//end of for loop with s var
        } catch (Throwable t) {
            t.printStackTrace();
        }
        return (conditionValue);
    }//end of getSubjectCondtionAttribute

/////////////
    public static String getObjectCondtionAttribute(String subjectValue, String
    objectValue, String actionValue) {
        String conditionValue = "";
        String SubjectPIP = "";
        String ObjectPIP = "";
        String ActionPIP = "";
        try {
            //System.out.println("Reading the PIP Database");
            DocumentBuilderFactory docBuilderFactory = DocumentBuilderFactory.
            newInstance();
            DocumentBuilder docBuilder = docBuilderFactory.newDocumentBuilder();
            Document doc = docBuilder.parse(new File("PIPdata.xml"));
            // normalise text representation
            doc.getDocumentElement().normalize();
            NodeList listOfRules = doc.getElementsByTagName("rule");
            int totalRules = listOfRules.getLength();
            for (int s = 0; s < listOfRules.getLength(); s++) {
                Node firstRuleNode = listOfRules.item(s);
                if (firstRuleNode.getNodeType() == Node.ELEMENT_NODE) {
                    Element firstRuleElement = (Element) firstRuleNode;
//-------
                    NodeList subjectConditionList = firstRuleElement.
                    getElementsByTagName("Subject");
                    Element subjectConditionElement = (Element) subjectConditionList.
                    item(0);
                    NodeList textFSList = subjectConditionElement.getChildNodes();
                    SubjectPIP = ((Node) textFSList.item(0)).getNodeValue().trim();
//-------
                    NodeList objectConditionList = firstRuleElement.
                    getElementsByTagName("Object");
                    Element objectConditionElement = (Element) objectConditionList.
                    item(0);
                    NodeList textFOList = objectConditionElement.getChildNodes();
                    ObjectPIP = ((Node) textFOList.item(0)).getNodeValue().trim();
//-------
                    NodeList actionConditionList = firstRuleElement.
                    getElementsByTagName("Action");
                    Element actionConditionElement = (Element) actionConditionList.
                    item(0);
                    NodeList textFAList = actionConditionElement.getChildNodes();
                    ActionPIP = ((Node) textFAList.item(0)).getNodeValue().trim();
//-------
                    if (subjectValue.equals(SubjectPIP) && objectValue.equals(ObjectPIP)
                    && actionValue.equals(ActionPIP)) { NodeList firstConditionList =
                    firstRuleElement.getElementsByTagName("ObjectAttribute");
                        Element firstConditionElement = (Element) firstConditionList.
                        item(0);
                        NodeList textFNList = firstConditionElement.getChildNodes();
                        conditionValue = ((Node) textFNList.item(0)).getNodeValue().
                        trim();
                        conditionValue);
                        break;
                    }//end of if clause
                }//end of if clause
            }//end of for loop with s var
        } catch (Throwable t) {
            t.printStackTrace();
        }
        return (conditionValue);
```

```
    }//end of getObjectCondtionAttribute
////////////////////////////
    public static String getEnvironmentCondtionAttribute(String subjectValue,
    String objectValue, String actionValue) {
        String conditionValue = "";
        String SubjectPIP = "";
        String ObjectPIP = "";
        String ActionPIP = "";
        try {
            //System.out.println("Reading the PIP Database");
            DocumentBuilderFactory docBuilderFactory = DocumentBuilderFactory.
            newInstance();
            DocumentBuilder docBuilder = docBuilderFactory.newDocumentBuilder();
            Document doc = docBuilder.parse(new File("PIPdata.xml"));
            // normalize text representation
            doc.getDocumentElement().normalize();
            NodeList listOfRules = doc.getElementsByTagName("rule");
            int totalRules = listOfRules.getLength();
            for (int s = 0; s < listOfRules.getLength(); s++) {
                Node firstRuleNode = listOfRules.item(s);
                if (firstRuleNode.getNodeType() == Node.ELEMENT_NODE) {
                    Element firstRuleElement = (Element) firstRuleNode;
//-------
                    NodeList subjectConditionList = firstRuleElement.
                    getElementsByTagName("Subject");
                    Element subjectConditionElement = (Element)
                    subjectConditionList.item(0);
                    NodeList textFSList = subjectConditionElement.
                    getChildNodes();
                    SubjectPIP = ((Node) textFSList.item(0)).getNodeValue().
                    trim();
//-------
                    NodeList objectConditionList = firstRuleElement.
                    getElementsByTagName("Object");
                    Element objectConditionElement = (Element)
                    objectConditionList.item(0);
                    NodeList textFOList = objectConditionElement.
                    getChildNodes();
                    ObjectPIP = ((Node) textFOList.item(0)).getNodeValue().
                    trim();
//-------
                    NodeList actionConditionList = firstRuleElement.
                    getElementsByTagName("Action");
                    Element actionConditionElement = (Element) actionConditionList.
                    item(0);
                    NodeList textFAList = actionConditionElement.getChildNodes();
                    ActionPIP = ((Node) textFAList.item(0)).getNodeValue().trim();
//-------
                    if (subjectValue.equals(SubjectPIP) && objectValue.equals(ObjectPIP)
                    && actionValue.equals(ActionPIP)) {
                        NodeList firstConditionList = firstRuleElement.
                        getElementsByTagName("EnvironmentAttribute");
                        Element firstConditionElement = (Element) firstConditionList.
                        item(0);
                        NodeList textFNList = firstConditionElement.getChildNodes();
                        conditionValue = ((Node) textFNList.item(0)).getNodeValue().
                        trim();
                        break;
                    }//end of if clause
                }//end of if clause
            }//end of for loop with s var
        } catch (Throwable t) {
            t.printStackTrace();
        }
        return (conditionValue);
    }//end of getEnvironmentCondtionAttribute
////////////////
    public static String getEventCondtionAttribute(String subjectValue, String
    objectValue, String actionValue) {
        String conditionValue = "";
        String SubjectPIP = "";
        String ObjectPIP = "";
        String ActionPIP = "";
        try {
            //System.out.println("Reading the PIP Database");
            DocumentBuilderFactory docBuilderFactory = DocumentBuilderFactory.
            newInstance();
            DocumentBuilder docBuilder = docBuilderFactory.newDocumentBuilder();
            Document doc = docBuilder.parse(new File("PIPdata.xml"));
            // normalize text representation
            doc.getDocumentElement().normalize();
            NodeList listOfRules = doc.getElementsByTagName("rule");
            int totalRules = listOfRules.getLength();
            for (int s = 0; s < listOfRules.getLength(); s++) {
                Node firstRuleNode = listOfRules.item(s);
                if (firstRuleNode.getNodeType() == Node.ELEMENT_NODE) {
                    Element firstRuleElement = (Element) firstRuleNode;
//-------
                    NodeList subjectConditionList = firstRuleElement.
                    getElementsByTagName("Subject");
                    Element subjectConditionElement = (Element) subjectConditionList.
                    item(0);
                    NodeList textFSList = subjectConditionElement.getChildNodes();
                    SubjectPIP = ((Node) textFSList.item(0)).getNodeValue().trim();
//-------
                    NodeList objectConditionList = firstRuleElement.
                    getElementsByTagName("Object");
                    Element objectConditionElement = (Element) objectConditionList.
                    item(0);
                    NodeList textFOList = objectConditionElement.getChildNodes();
                    ObjectPIP = ((Node) textFOList.item(0)).getNodeValue().trim();
//-------
                    NodeList actionConditionList = firstRuleElement.
                    getElementsByTagName("Action");
                    Element actionConditionElement = (Element) actionConditionList.
                    item(0);
                    NodeList textFAList = actionConditionElement.getChildNodes();
                    ActionPIP = ((Node) textFAList.item(0)).getNodeValue().trim();
//-------
                    if (subjectValue.equals(SubjectPIP) && objectValue.equals(ObjectPIP)
                    && actionValue.equals(ActionPIP)) { NodeList firstConditionList =
                    firstRuleElement.getElementsByTagName("EventAttribute");
                        Element firstConditionElement = (Element) firstConditionList.
                        item(0);
                        NodeList textFNList = firstConditionElement.getChildNodes();
                        conditionValue = ((Node) textFNList.item(0)).getNodeValue().
                        trim();
                        break;
                    }//end of if clause
                }//end of if clause
            }//end of for loop with s var
        } catch (Throwable t) {
            t.printStackTrace();
        }
        return (conditionValue);
    }//end of getEventCondtionAttribute

////////////////////
    public static void addAttributesToDB(String PIPadd, String subject, String
    resource, String action, String subjectAttribute, String objectAttribute,
    String environmentAttribute, String eventAttribute) {
// PIPadd can be any PIP database in the system instead of (PIPdata.xml) below
        File docFile = new File("PIPdata.xml");
        Document doc = null;
        try {
            DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
            DocumentBuilder db = dbf.newDocumentBuilder();
            doc = db.parse(docFile);
        } catch (java.io.IOException e) {
            System.out.println("Can't find the file");
        } catch (Exception e) {
            System.out.print("Problem parsing the file.");
        }
        Element root = doc.getDocumentElement();
        NodeList children = root.getChildNodes();
        Element ruleElement = doc.createElement("rule");
        Node updateText = doc.createTextNode("");
        ruleElement.appendChild(updateText);
        Element subjectAdd = doc.createElement("Subject");
        String str_subject = subject;
        Node subjectAddNode = doc.createTextNode(str_subject);
        subjectAdd.appendChild(subjectAddNode);
        ruleElement.appendChild(subjectAdd);
//-------
        Element objectAdd = doc.createElement("Object");
        String str_object = resource;
        Node objectAddNode = doc.createTextNode(str_object);
        objectAdd.appendChild(objectAddNode);
        ruleElement.appendChild(objectAdd);
//-------
        Element actionAdd = doc.createElement("Action");
        String str_action = action;
        Node actionAddNode = doc.createTextNode(str_action);
        actionAdd.appendChild(actionAddNode);
        ruleElement.appendChild(actionAdd);
```

```
//-------
        Element SubjectAttributeAdd = doc.createElement("SubjectAttribute");
        String str_SubjectAttribute = subjectAttribute;
        Node SubjectAttributeNode = doc.createTextNode(str_SubjectAttribute);
        SubjectAttributeAdd.appendChild(SubjectAttributeNode);
        ruleElement.appendChild(SubjectAttributeAdd);
//-------
        Element ObjectAttributeAdd = doc.createElement("ObjectAttribute");
        String str_ObjectAttribute = objectAttribute;
        Node ObjectAttributeNode = doc.createTextNode(str_ObjectAttribute);
        ObjectAttributeAdd.appendChild(ObjectAttributeNode);
        ruleElement.appendChild(ObjectAttributeAdd);
//-------
        Element EnvironmentAttributeAdd = doc.createElement("EnvironmentAttribute");
        String str_EnvironmentAttribute = environmentAttribute;
        Node EnvironmentAttributeNode = doc.createTextNode(str_EnvironmentAttribute);
        EnvironmentAttributeAdd.appendChild(EnvironmentAttributeNode);
        ruleElement.appendChild(EnvironmentAttributeAdd);
//-------
        Element EventAttributeAdd = doc.createElement("EventAttribute");
        String str_EventAttribute = eventAttribute;
        Node EventAttributeNode = doc.createTextNode(str_EventAttribute);
        EventAttributeAdd.appendChild(EventAttributeNode);
        ruleElement.appendChild(EventAttributeAdd);
//-------
        root.appendChild(ruleElement);
    }

/////////////
    public static void updateAttributesDB(String PIPupdate, String subjectValue,
    String objectValue, String actionValue, String nodeAttribute, String
    oldAttribute, String newAttribute) {
        String conditionValue = "";
        String SubjectPIP = "";
        String ObjectPIP = "";
        String ActionPIP = "";
        try {
            // PIPupdate can be any PIP database in the system instead of
            (PIPdata.xml) below
            String filepath = "c:\\file.xml";
            DocumentBuilderFactory docFactory = DocumentBuilderFactory.newInstance();
            DocumentBuilder docBuilder = docFactory.newDocumentBuilder();
            Document doc = docBuilder.parse(filepath);
            // normalize text representation
            doc.getDocumentElement().normalize();
            NodeList listOfRules = doc.getElementsByTagName("rule");
            int totalRules = listOfRules.getLength();
            for (int s = 0; s < listOfRules.getLength(); s++) {
                Node firstRuleNode = listOfRules.item(s);
                if (firstRuleNode.getNodeType() == Node.ELEMENT_NODE) {
                    Element firstRuleElement = (Element) firstRuleNode;
//-------
                    NodeList subjectConditionList = firstRuleElement.
                    getElementsByTagName("Subject");
                    Element subjectConditionElement = (Element) subjectConditionList.
                    item(0);
                    NodeList textFSList = subjectConditionElement.getChildNodes();
                    SubjectPIP = ((Node) textFSList.item(0)).getNodeValue().trim();
//-------
                    NodeList objectConditionList = firstRuleElement.
                    getElementsByTagName("Object");
                    Element objectConditionElement = (Element) objectConditionList.
                    item(0);
                    NodeList textFOList = objectConditionElement.getChildNodes();
                    ObjectPIP = ((Node) textFOList.item(0)).getNodeValue().trim();
//-------
                    NodeList actionConditionList = firstRuleElement.
                    getElementsByTagName("Action");
                    Element actionConditionElement = (Element) actionConditionList.
                    item(0);
                    NodeList textFAList = actionConditionElement.getChildNodes();
                    ActionPIP = ((Node) textFAList.item(0)).getNodeValue().trim();
//-------
                    if (subjectValue.equals(SubjectPIP) && objectValue.equals(ObjectPIP)
                    && actionValue.equals(ActionPIP)) { NodeList firstConditionList =
                    firstRuleElement.getElementsByTagName(nodeAttribute);
                        Element firstConditionElement = (Element) firstConditionList.
                        item(0);
                        NodeList textFAttList = firstConditionElement.getChildNodes();
                        conditionValue = ((Node) textFAttList.item(0)).getNodeValue().
                        trim();
                        if (nodeAttribute.equals(firstConditionElement.getNodeName())
                        && conditionValue.equals(oldAttribute)) {
                         firstConditionElement.setTextContent(newAttribute);
                        }//end of if clause
                        break;
                    }//end of if clause
                }//end of if clause
            }//end of for loop with s var
            // write the content into xml file
            TransformerFactory transformerFactory = TransformerFactory.newInstance();
            Transformer transformer = transformerFactory.newTransformer();
            DOMSource source = new DOMSource(doc);
            StreamResult result = new StreamResult(new File(filepath));
            transformer.transform(source, result);
        } catch (Throwable t) {
            t.printStackTrace();
        }
    }
}
```

Listing A.27: PIP Class