AN EFFICIENT AND SCALABLE ACCESS REVIEW EVALUATION MODEL FOR XACML: A SUBJECT-OBJECT GRAPH BASED APPROACH

Malithi Madara Edirisinghe

138211N

Degree of Master of Science

Department of Computer Science and Engineering

University of Moratuwa
Sri Lanka

April 2017

AN EFFICIENT AND SCALABLE ACCESS REVIEW EVALUATION MODEL FOR XACML: A SUBJECT-OBJECT GRAPH-BASED APPROACH

Malithi Madara Edirisinghe

138211N

Degree of Master of Science

Department of Computer Science and Engineering

University of Moratuwa
Sri Lanka

April 2017

DECLARATION

I declare that this is my own work and this dissertation does not incorporate without acknowledgement any material previously submitted for a Degree or Diploma in any other University or institute of higher learning and to the best of my knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgement is made in the text.

In addition, I hereby grant to University of Moratuwa the non-exclusive right to reproduce and distribute my thesis, in whole or in part in print, electronic or other medium. I retain the right to use this content in whole or part in future works.

....

Malithi Madara Edirisinghe Date

I certify that the declaration above by the candidate is true to the best of my knowledge and that this project report is acceptable for evaluation for the Post Graduate Project.

....

Dr. H.M.N Dilum Bandara Date

ABSTRACT

Attribute-Based Access Control (ABAC) never explicitly constructs the access control matrix like traditional access control models such as Role-Based Access Control (RBAC) and Access Control Lists (ACL). Rather, it relies on an access control policy that implicitly defines the access matrix. Thus, in ABAC evaluating access review queries like "*Which objects does this user have access to?*" and "*What actions can this user perform on those objects?*" is computationally expensive. We propose a graph-based model to represent the permission relationships between the subjects and objects, which can be used to solve access review queries more efficiently. The solution is proposed in the context of XACML (eXtensible Access Control Markup Language), the standard that defines a declarative fine-grained, attribute-based access control policy language, an architecture, and a processing model. The proposed solution parses and transforms complex logical expressions in XACML policies into a subject-object relationship graph by extracting conditions starting from the lower levels of a XACML policy tree and constructing condition paths from them. Therefore, the proposed model can isolate matching subjects or objects, for a given access review request, to extract applicable set of conditions efficiently. Using two real-world datasets, we analyzed the computational cost of graph construction, memory usage, and the query evaluation performance of the proposed model with a XACML policy evaluation engine, which implements Multi-Interval Decision Diagrams (MIDD). Compared to MIDD the proposed solution is 100% to 550% and 61.8% to 99.1% efficient in terms of computational and memory requirements, respectively. Further, the solution resolve access review queries 33.9% faster compared to MIDD even for large policy sets.

Keywords: Access Reviews, Attribute-Based Access Control, Authorization Reverse Queries, XACML.

ACKNOWLEDGMENTS

I would like to express profound gratitude to my advisor, Dr. Dilum Bandara, for his invaluable support by providing advice, supervision and useful suggestions throughout this research work. His expertise and continuous guidance enabled me to complete my work successfully.

I express my utmost gratitude to Mr. Prabath Siriwardena, my external project supervisor for providing me the initial project idea and giving immense and invaluable assistance and support throughout the process of finding the optimum solution. Despite his busy schedule, he always encouraged to organize meetings, discuss the progress and clarify issues.

I am grateful for the support and advice given by Dr. Malaka Walpola, by encouraging continuing this research until the end. Especially I would like to thank my dearest friend Ms. Iromi U. Ranaweera, PhD Student at Norwegian University of Science and Technology for helping to get research materials. Further, I would like to thank all my colleagues for their encouragement.

I am as ever, especially indebted to my parents for their love and support throughout my life. Finally, I wish to express my gratitude to all my colleagues at WSO2 for the support given me to manage my MSc research work.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ABAC	Attribute Based Access Control
ACL	Access Control List
ARQ	Axiomatics Reverse Query
DAC	Discretionary Access Control
JSON	JavaScript Object Notation
MAC	Mandatory Access Control
MIDD	Multi Interval Decision Diagram
NIST	National Institute of Standards and Technology
OASIS	Organization for the Advancement of Structured Information Standards
PAP	Policy Administration Point
PDP	Policy Decision Point
PEP	Policy Enforcement Point
PIP	Policy Information Point
PQE	Partial Query Evaluation
RBAC	Role Based Access Control
SOD	Separation of Duty
URI	Universal Resource Identifier
URN	Universal Resource Name
XACL	XML Access Control Language
XACML	Extensible Access Control Markup Language
XML	Extensible Markup Language

1 INTRODUCTION

Providing adequate security for information and information systems is a fundamental management responsibility of any enterprise. Thus, almost all applications include some form of access control, especially when they deal with financial, privacy, administration or defense aspects. The primary goal of access control is to protect system resources against inappropriate or unauthorized user access. Based on the access control requirements of information technology systems a number of access control models with different properties are available to choose from [1].

However, as systems grow, in size and complexity, there is a massive increase in digital data that is stored, which also causes for an equal increase in data that is shared. This leverages the ability to access data anytime and from anywhere. Thus, imposing access control becomes more challenging with traditional access control models such as Discretionary Access Control (DAC), Mandatory Access Control (MAC) and Role-Based Access Control (RBAC), which grants or denies based on the pre-assigned capabilities to users directly or via some predefined attribute types such as roles or groups [1]. They fail to capture the contextual information of the system, users and resources, when an access control request is made, such as access time, location etc., and decide dynamically. Further with the growing number of users and resources participating in the system, managing the association of capabilities become hard. Moreover, roles and groups are often insufficient to express real-word access control requirements. One of the promising alternative is Attribute-Based Access Control (ABAC) [2], which avoids the need for explicit authorizations to be directly assigned to individual users prior to a request to perform an operation on the resource. ABAC grant or deny user requests based on the arbitrary attributes of the user and arbitrary attributes of the resource, and environment conditions, bringing dynamism and context evaluation to access control.

1.1 Attribute Based Access Control

Attribute-Based Access Control (ABAC) allows a higher no of discrete inputs to an access control decision. Therefore, provides a larger set of possible combinations of

those variables to reflect larger set of rules to define access control enterprise policies. In ABAC, access control decisions can change between requests simply by altering request values without changing (*subject, resource*) relationships. This provides dynamic access management capabilities and easy maintenance. Administrators can apply Access control policy without prior knowledge of the specific subject and for an unlimited no of subjects that may require access. They do not need to modify or assign capabilities as new subjects join or leave the system. If the subject is assigned the necessary attributes to access a specific resource, as defined by the access control policy, subject will gain access, which is known as accommodation of external (i.e., unanticipated) user [2].

Due to above properties, ABAC is becoming more popular with the increase of interactions between users, devices, services, data, and external environments. It is more scalable and dynamic over traditional approaches. Thus, ABAC has the potential to dramatically improve access control in modern applications such as e-commerce, and the Internet of Things, becoming more relevant in supporting privacy and security in a Big Data context.

Gartner predicted that 70% of all businesses will use ABAC as a dominant mechanism to protect critical assets by 2020 [3]. Gartner predicts that ABAC will replace RBAC, as the number of possible interactions between users, devices, services, data, and the external environment increases and notes that this trend is evidenced by examples in popular software vendors' product strategies, such as dynamic access control within Microsoft Server 2012, the claims-based architecture in Windows Identity Foundation, and Oracle's Fusion middleware.

1.2 Problem Statement

The ABAC model differs from traditional access control models such as RBAC and ACL. Traditional methods assign capabilities directly to subjects, roles or groups before the access request is made, thus constructs the *access matrix* (i.e., a matrix between subjects and resources, where each row represents a subject and each column represents a resource). However, in ABAC access matrix is never explicitly

constructed. Instead, ABAC relies on an access control policy that implicitly defines the access matrix. Given a subject, a resource and an action, the policy defines whether the subject can perform the action on the resource. Thus, the policy is defining the function to be executed in each cell of the access matrix. While this makes ABAC rich and flexible, it complicates the access review. For example, it is difficult to solve queries like "Which objects does this user have access to?" and "What actions can this user perform on those objects?". Such queries are important in auditing, generating audit reports and to make decisions before executing access requests based on capabilities.

Answering such a query in ABAC involves, evaluating the policy for each cell over a row for subject-centric access review requests such as "Which objects does this user have access to?", or for each cell over a column for resource-centric access review requests such as "Who has access to this resource?". That means policy will be evaluated for each (*subject, resource*) combination over a row or column regardless of the relationship between the subject and resource. Further, based on the complexity of the policy, computing the value for each cell may be computationally expensive. Therefore, ABAC makes it extremely difficult to determine permissions available to a user, because an extremely large number of rules might need to be executed, in the same order in which the system applies them, to successfully determine access.

The biggest concern in supporting access reviews with ABAC model, is the performance, caused due to the cost of determining the value of a cell in the matrix, lack of data representation (how data related and located) which makes it hard to isolate a specific row or a column to go over for a given access review request, and the inability to avoid empty cells in the access matrix where there is no relationship between the specific subject and the resource. Thus, the problem that this research try to address can be stated as follows:

How to perform access reviews in Attribute-Based Access Control (ABAC), while minimizing computational and memory requirements?

1.3 Objectives

The research achieves the following objectives:

- Study existing solutions that support access reviews with XACML, and analyze the access review representation model, computational costs incurred, and limitations.
- Build up a memory efficient data representation model, from an XACML access control policy or policy set, which is scalable and which can be used to evaluate access review requests efficiently.
- Come up with a low latency algorithm to evaluate access review requests that reduces query evaluation cost.

1.4 Research Contribution

ABAC relies on an access control policy, that is evaluated at the time an access control request is made. This makes ABAC rich and flexible in supporting access control requests, but complicates the access review requests. However, support for access reviews is important in auditing, generating audit reports, performing access control administration, when deciding on access control policies, reviewing privileges, and making decisions before executing access requests based on capabilities. This research addresses the problem of supporting access review queries within the ABAC model through the following contributions:

- Development of a subject-object graph representation to XACML policies.
- Propose an algorithm to extract applicable set of conditions from the subject-object graph representation to an access review request.
- The proposed technique is memory and computationally efficient compared to Partial Query Evaluation of XACML Policies [4], and Multi-Interval Decision Diagrams (MIDD) [5].
- Performance analysis using two real-world access policy datasets showed that the proposed solution is 100% to 550% and 61.8% to 99.1% efficient in terms

of computational and memory requirements compared to MIDD and the solution resolves access review queries 33.9% faster compared to MIDD.

1.5 Outline

Rest of the thesis is organized as follows. In Chapter 2, we discuss related work on access control, ABAC and XACML, as well as initiatives taken to address access review requests in ABAC. We further discuss research work on using policy simplification for access review requests and decision diagram approaches taken to pre-compute a data model from XACML policies for performance efficient evaluation.

Research methodology is presented in Chapter 3. We can represent an ABAC policy with a performance efficient data model, from which we can isolate subjects and resources, to easily identify their relationships for a given access review request, how to compute that data model from a given XACML policy set, and finally on how an access review request will be evaluated over the constructed model.

Chapter 4 presents the algorithm that constructs the (*subject, resource*) permission relationship model and will discuss how policy updates can be accommodated. Further, we present the access review request evaluation algorithm, which is based on the data model constructed.

Performance evaluation of the implementation is presented in Chapter 5. Analysis discuss the experiments conducted to evaluate graph construction performance and query evaluation performance.

Chapter 6 discuss the results of performance analysis, capabilities and the limitations of the solution proposed and future improvements and research work.

2 LITERATURE REVIEW

2.1 Access Control Overview

In information System Security, authentication and authorization are two key components of security enforcements. *Authentication* is to bind a subject to an identity that uniquely identifies the respective subject from others. Once subject identity is known, *authorization* decides what that subject may or may not do within the system. Access control is about enforcing appropriate authorization for the system based on the user's identity. Its objective is to protect system resources against inappropriate or undesired user access.

2.2 Concepts

There are several terms and concepts that are commonly used within access control research community. Next, some of the related terms are defined.

- **Subject -** An active entity, generally in the form of a person, process, or device that causes information to flow among resources or other entities in the system or changes the system state [6].
- **Resource/Object -** An entity that contains or receives information. Access to an object implies access to the information it contains. Examples of objects are database records, fields, files, directories, process, processors, video displays, keyboards, clocks, printers, and network resources etc. [6].
- **Action/Operation -** An active process invoked by a subject. For example, in a file system, there will different types of files. A user can perform number of operations on a file. He may read, write, delete etc. [7].
- **Permission/Privilege -** Authorization to perform some action on the system. Mostly permission is defined as the combination of the resource (object) and the action (operation). Thus, an action allowed to be performed on two different resources represents two different permissions. For example, a bank teller machine may have permissions to execute *debit* and *credit* actions on a user's

account through performed transactions, while an accountant in the bank may execute *credit* and *debit* operations on the general ledger based on the transactions of the bank [7].

- **Access Control Matrix (ACM) -** A table in which each row represents a subject and each column represents a resource in the system. Each cell of the table is the set of access rights for that subject to that resource. ACM is a hypothetical model used to describe permission relationships between subjects and resources. It is a sparse matrix with many empty cells, as most subjects do not have access rights to most resources.

- **Separation of Duty (SOD) -** The principle that no user should be given enough privileges to misuse the system. For example, a person authorizing a specific transaction should not be the person who made it. Separation of duties can be enforced either statically by defining conflicting roles (i.e., roles which cannot be executed by the same user) or dynamically by enforcing the control at access time. An example of dynamic separation of duty is the two-person rule. The first user to execute a two-person operation can be any authorized user, whereas the second user can be any authorized user different from the first [1].

2.3 Access Control Models and Mechanisms

When planning an access control system, an enterprise should consider three abstractions [1].

- Access Control Policies
- Access Control Models
- Access Control Mechanisms

Access control policies are the high-level access control requirements of the enterprise. That will specify how access is managed and who may access, what information, under what circumstances. These access control policies are enforced through an access control mechanism that will capture a user's access request over a structure defined by the properties of an access control system. Access control models bridge the gap

between policy and mechanism. Access control mechanism is designed to adhere to the properties of the model.

2.3.1 Discretionary Access Control

DAC [8] is an access control model, in which access to a specific resource is managed by the resource owner or anyone else who is authorized to control the resource's access. For example, in a file system, the owner of the file will control other user's accesses to the file. Only those users specified by the owner may have some combination of read, write, execute and other permissions to the file.

DAC is known to be weak for two reasons.

1. Granting read access is transitive - When Ann grants Bob read access to a file, nothing stops Bob from copying the contents of Ann's file to an object that Bob controls. Bob may now grant any other user access to the copy of Ann's file without Ann's knowledge.
2. DAC policy is vulnerable to Trojan horse attacks - Bob may write a program for Ann such that from top it is like performing some useful function, while at the same time destroys the contents of Ann's files. When investigating the problem, the audit files would indicate that Ann destroyed her own files.

Thus, DAC model includes following drawbacks.

1. Information can be copied from one object to another.
2. No restrictions apply to the information when a specific user has received it.
3. Access privileges for objects are decided by the owner of the object, rather than a global policy or administrator that enforces the organization's security requirements.

Access Control List

ACL [9] is the most common implementation mechanism of DAC model. An ACL associates the permitted operation to a resource and specifies all the subjects that can access that resource, along with their rights to the resource. That is, each entry in the

list is a pair of (*subject, set of rights*). Moreover, an ACL corresponds to a column of the access control matrix. With an ACL, it is easy to answer the question "Who are the users that have access to this resource?", but it is difficult to determine all privileges for a user. In that case, we need to search privileges for that user traversing over each ACL defined for each resource.

Capability List

CL [9] is the inversion of an ACL, where it is attached to the subject and specifies which resources the subject may access. That is, each entry in the list is a pair of (resource, set of rights). Moreover, it corresponds to a row of the access control matrix. The main advantage of a capability list is, that is it easy to review all access that are authorized for a given subject. However, it is hard to derive the subjects that have access to a resource. Thus, with capability lists it is difficult to model DAC policies and therefore, not commercially popular.

2.3.2 Mandatory Access Control

MAC [9] is an access control model, in which access control policy decisions are enforced by a central administrator, not by individual object owners and neither the owners can change access rights. MAC is usually associated with multilevel security models such as Bell-LaPadula Model Confidentiality [10] and Biba Integrity models [11]. MAC is used when the risk of attack is very high and confidentiality is a primary access control concern, or the resources being protected are valuable. Thus, it is commonly used by the military and intelligence agencies to maintain classification policy access restrictions.

The assignment and enforcement of security levels by the system under the MAC model places restrictions on user actions that, while adhering to security policies, prevents dynamic alteration of the underlying policies, and requires large parts of the operating system and associated utilities to be "trusted" and placed outside of the access control framework. MAC systems are difficult and expensive to implement due to the reliance on trusted components and the necessity for applications to be rewritten to adhere to MAC labels and properties. MAC also does not address fine-grained least privilege, dynamic separation of duty or validation of trusted components.

2.3.3 Role Based Access Control

In RBAC [9], access decisions are based on the roles that individual users have been assigned. Access rights are grouped to a role, and the use of resources is restricted to individuals based on the associated role. The operations that a user is permitted to perform are based on the user's role. Assignment of users into roles can be revoked easily and new assignments can be made as the user responsibilities change. Further, permissions for a role can be revised as organizational functions change and evolve. Thus, this model simplifies the administration and management of privileges, because roles can be updated without updating the privileges for every user on an individual basis.

However, in RBAC there is a cost on ensuring least privilege. When a user is associated with a role, the user can be given no more privilege than necessary to perform the job, which is known as least privilege. Because many of the responsibilities overlap between job categories, maximum privilege for each job category could cause unauthorized access. Thus, this requires a certain amount of role engineering to identify the user's job functions, determine the minimum set of privileges required to perform those functions, and to restrict the user to a domain with only those privileges. Even though, RBAC is known to be the best way to address typical access control use cases with compliance and administrative efficiencies, as organizations expand with identity and access management, role management will finally end up with “role explosion”, where more roles exist than actual individuals in the system.

2.3.4 Attribute Based Access Control

ABAC [2] avoids assigning capabilities directly to subjects, roles or groups before the access request is made. Instead when the subject requests access, the ABAC engine can make an access control decision based on the assigned attributes of the requestor, assigned attributes of the object, environment conditions, and a set of policies specified in terms of those attributes and conditions. Thus, in contrast to RBAC, this allows much more fine-grained access control capabilities, by not only combining attributes of the subject or the identity, but also taking context information into account, such as

location, time of the day to access control decisions. Further, ACLs and RBAC are in some ways special cases of ABAC in terms of attributes used. ACLs work on the attribute of 'identity'. RBAC works on the attribute of 'role'. The key difference with ABAC is the concept of policies that express a complex Boolean rule that can evaluate many different attributes.

ABAC, facilitates the establishment of business focused, centralized policy management and a common, consistent access control model for applications across the enterprise. Further, in ABAC, authorization is provided as a service, externalizing the access decision point. This simplifies the software development, as authorization is now not embedded to applications. Moreover, this reduce the cost of maintenance from applications perspective and from policy administration perspective, because decisions are made at runtime based on attributes, thus, changes in access status are immediately recognized when attributes are updated.

Therefore, ABAC is useful when:

- Authorization requests are complex and using RBAC drives for role explosion.
- Authorization rules can be expected to change over time.
- Authorization rules need to take external factors and context information into account when making the authorization decision.

Given these properties, ABAC is commonly to be adopted by industries such as:

- Which have highly distributed and remote operating entities, where access for such remote entities need to be managed by a central authorization system such as Financial/Insurance service sector, Airlines, Telecommunication companies etc.
- Organizations that are mandated by government or regulatory policy to control what information to be provided or what goods may be exported such as Aerospace Manufacturers, Defense Manufacturers, Nuclear Energy Organizations, Oil & Gas companies, Exporters.

- Organizations that have a need for strong protection of their Intellectual Property such as Research & Development Organizations.
- Organizations that have a requirement involving sensitive health data with associated consent management requirements such as Hospitals, Primary, Secondary or Tertiary Healthcare organization and Pharmacies.

In addition to above NIST Guide on ABAC [12], evangelizes the adoption of ABAC in enterprises with a comprehensive set of guidelines on when to migrate to an ABAC model and why it is important. Moreover, ABAC also make it more relevant in supporting privacy and security in a Big Data context, due to the ability of capturing context information for access control decisions, whereas *context* is a key factor in Big Data, just as context is key to understanding privacy [13].

However, there are some complexities incurred with ABAC model. In ABAC, access control rules are defined composing policies, with some policy language where it may be cumbersome for administrators, as they need to expertise on the specific language to express complex, access control rules. However, vendors like Axiomatics have come up with a comprehensive set of tooling to support composing of complex policies and easy administration [14]. Further, it is difficult to perform access reviews specifically being used to audit policies, generate the audit reports and to make decisions before executing access requests based on capabilities, which we are going to address in this thesis.

To enable ABAC implementations, the community has undertaken efforts to develop common terminology and interoperability across access control systems. The Organization for the Advancement of Structured Information Standards' (OASIS) Extensible Access Control Markup Language (XACML) [15] and IBM's XML Access Control Language (XACL) [16] are access control policy specification frameworks that are mainly geared towards supporting ABAC model. The most popular policy language being used to express ABAC policies is XACML, which specifies an industry standard [1].

2.4 XACML

XACML is the OASIS standard for fine grained authorization management based on the concept of ABAC.

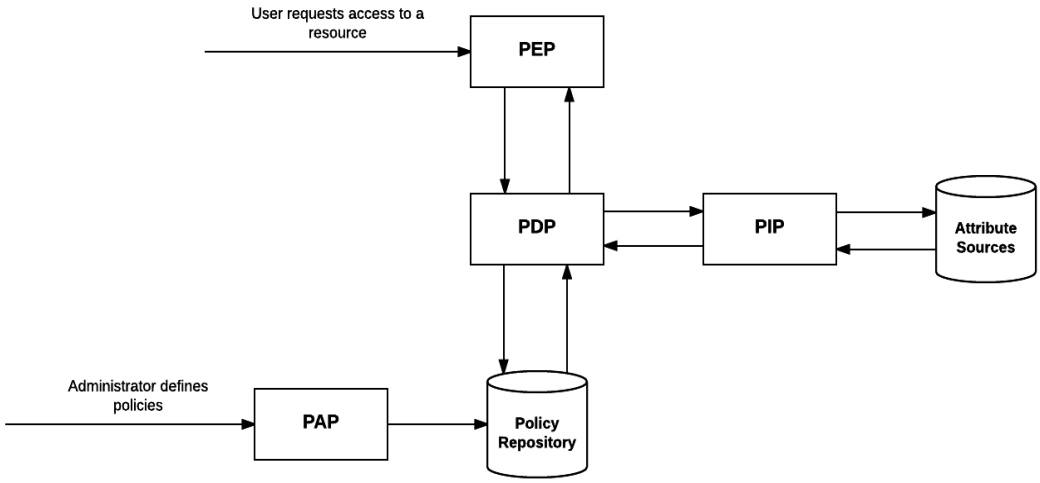

Figure 2.1 – XACML reference architecture.

XACML standardizes three aspects of the authorization process.

- XACML Policy Language - Used to express access control rules and conditions.
- XACML Request/Response Protocol - Used to query a decision engine that evaluates real-world access requests against existing XACML policies. XACML response will include either one of below.
    - Permit - Access request is permitted
    - Deny - Access request is denied
    - Not Applicable - For the access request made no matching rules identified to decide on the access control decision
    - Indeterminate - An error has occurred while evaluating the access request

- XACML Reference Architecture - Provides a standard for the deployment of necessary software modules to achieve efficient enforcement of XACML policies. As represented by Figure 2.1, several modules are defined in the XACML reference architecture that are used to enforce XACML policies. Policy Decision Point (PDP) evaluates policies against access requests provided by Policy Enforcement Points (PEP). PDP or PEP may also need to query Policy Information Point (PIP) to retrieve attributes of the user or the resource to which access is requested. Policies are maintained via a Policy Administration Point (PAP).

2.4.1 XACML Policy Language

XACML Policy Language is XML based. It defines the XACML policy structure, functions and algorithms, attribute categories, data types, and so on. The top-level elements of the policy language model specified in XACML 3.0 Core specification [17] can be represented as in Figure 2.2.

Policies contain rules which can decide from a certain set of attributes whether to permit or deny access for a certain request. For that to happen, any condition of the rule must evaluate to true. Conditions contain declarative logic from a set of predefined functions, which use attributes in the access request as input. These functions can be arithmetic, logical, comparative, set, functional, and so on. Each function is referenced with a unique Universal Resource Name (URN), that is defined in the XACML specification and we can define our own functions too.

Policy element includes a combining algorithm which decides how to combine results from multiple rules and give the final answer. There is an also set of standard combining algorithms, but custom algorithms can also be used.

The result of each evaluation of each policy is returned to the policy set. Just like with rules, policy sets also have combining algorithms. The final decision for the access request is returned from the policy set based on that.

Policy Set, Policy and Rule elements also includes a Target. Target contains Match elements, which is a set of comparisons on different category of attributes. These

targets are used to determine whether a policy, policy set or individual rule is applicable to the access request. If the policy, policy set or rule is not applicable, the PDP will not use it to determine the outcome of the request.

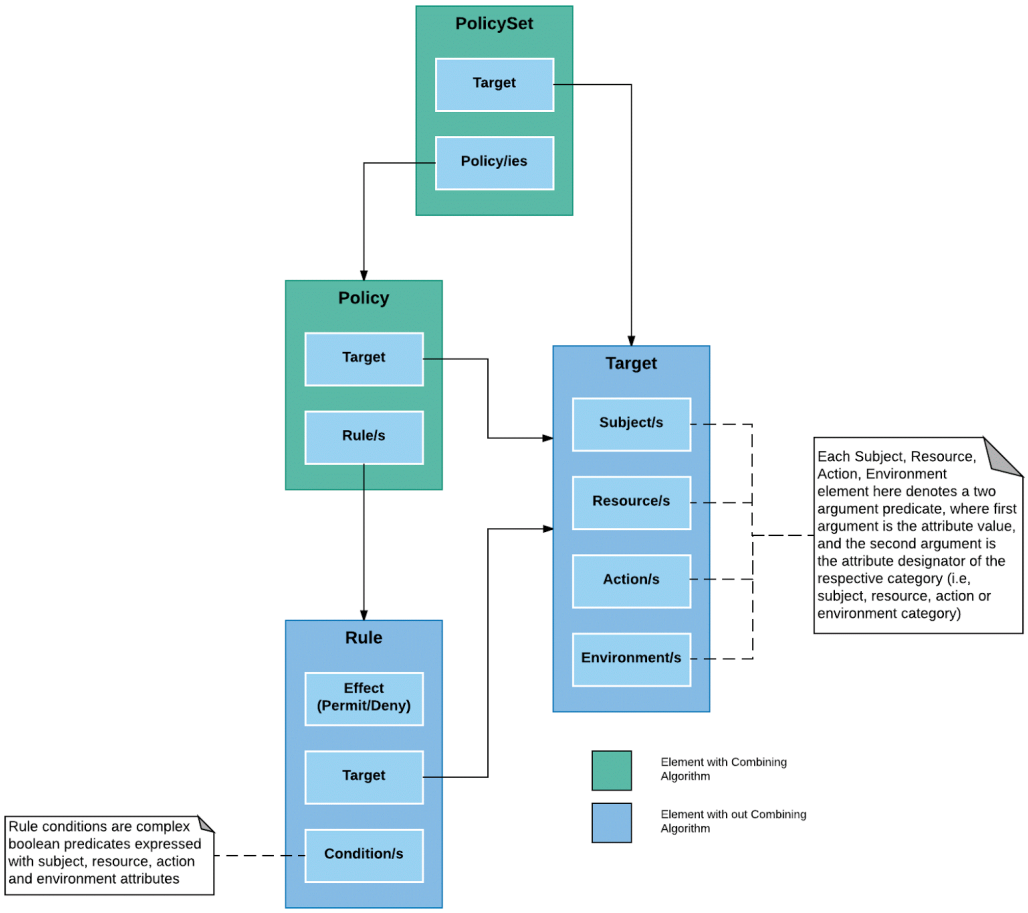

Figure 2.2 – XACML policy language model.

In ABAC, attributes are the variables on which the access control decisions are based. Hence, in XACML they are supplied in the access request as input values for the policy, and they referenced in the policy by their unique Universal Resource Identifier (URI). XACML supports four different attribute categories, namely Subject, Resource, Action and Environment. In these categories, different attributes can be used. For example, subject attributes may include id, username, role, etc. There is a standard set of attributes defined in the XACML specification, but custom ones can also be used. In the access request, they are put in subject, resource, action and environment elements respectively. In the XACML policy, the element, which is used to reference these attributes, is called attribute designator. Attribute designators exist inside

conditions as well as targets. There are four types of attributes designators, subject attribute designators, resource attribute designators, action attribute designators and environment attribute designators matching attribute categories.

XACML is a rule engine. Yet, it defines a rule engine architecture and standard dedicated to access control. XACML rule engine, i.e., PDP, can answer Yes/No questions, with a set of conditions defined based on the attributes of various entities participating. Its extensible standard, provides the capability of using XACML beyond typical access control use cases. For the ease of explanation of XACML access control policies, let us consider a hypothetical use case, which deviates from everyday access control use cases we see. Let us represent that in XACML policy language adhering to the above model.

Suppose at immigration, specific duty free allowance policies being applied by Sri Lankan Customs. Suppose this allowance policy implies below conditions:

- If the immigrant is a foreigner, below conditions will apply:
    - Only 1.5 liters of wine only 1.5 liters of liquor will be permitted.
    - Total value of the items allowed to bring in should be less than $250.
- If the immigrant is a resident, below conditions will apply:
    - Only 2 liters of wine and only 2.5 liters of liquor will be permitted.
    - If the stay abroad is less than 90 days, total value of the items allowed to bring in should be less than $125.
    - If the stay abroad is greater than 90 days and less than 365 days, total value of the items allowed to bring in should be less than $625.
    - If the stay abroad is greater than 365 days, total value of the items allowed to bring in should be less than $1750.

We can express this as an XACML policy similar to that in Figure 2.3. Here the policy is represented in JSON format to improve the readability. The exact XML based XACML policy is included in Appendix A.

```
PolicySet {
  id:DutyFreeAllowances,
  combine-algorithm:first-applicable,
  target:,
  children:[
    Policy {
      id:DutyFreeAllowancesForForeigners,
      combine-algorithm:permit-override,
      target:subject.citizenship = foreigner,
      children:[
        Rule {
          id:WineLiqorAllowanceForForeigners,
          effect:Permit,
          target:resource.id = wine or resource.id = liquor,
          condition:resource.volume <= 1.5,
        },
        Rule {
          id:AllowedItemAllowanceForForeigners,
          effect:Permit,
          target:resource.id = allowedItems,
          condition:resource.value <= 250,
        },
        Rule {
          id:Allowed,
          effect:Deny,
        }
      ]
    },

    Policy {
      id:DutyFreeAllowancesForResidents,
      combine-algorithm:permit-override,
      target:subject.citizenship = local,
      children:[
        Rule {
          id:WineAllowanceForResidents,
          effect:Deny,
          target:resource.id = wine,
          condition:resource.volume <= 2
        },
        Rule {
          id:LiquorAllowanceForResidents,
          effect:Permit,
          target:resource.id = liquor,
          condition:resource.volume <= 2.5
        },
        Rule {
          id:AllowedItemAllowanceForResidents,
          effect:Deny,
          target:resource = allowedItems,
          condition: (subject.stay < 90 and resource.value > 125) or
                     (90 < subject.stay < 365 and resource.value > 625) or
                     (subject.stay > 365 and resource.value > 1750) ,
        },
        Rule {
          id:Allowed,
          effect:Deny,
        }
      ],
    },
  ]
}
```

Figure 2.3 – Hypothetical XACML policy.

In this policy, the top-level element is a *PolicySet*. The *PolicySet* has two *Policies* defined. The *PolicySet* has an empty target that means *the target is always true* (i.e., for any access request this *PolicySet* will be picked up for evaluation). The *combine-algorithm* in the *PolicySet* is defined as *first-applicable*, which means to pick up the first policy from the policies defined, that matches for the access request for

evaluation. Two policies defined, have *permit-override* as the *combining-algorithm*, which means if any of the rules evaluated to *Permit*, policy result also evaluates to *Permit*. Therefore, any rule that evaluated to *Deny* will be overridden if at least one evaluated to a *Permit*. The first policy *DutyFreeAllowancesForForeigners*, will be matched only if the subject is a foreigner, and second policy *DutyFreeAllowancesForResidents* will be matched only if the subject is a local as defined in the *target*. Each policy defined includes a set of *rules*. The top most rules have the *effect* defined as *Permit*, which means if the *conditions* defined in the *rule* evaluates to *true*, the result of the rule will be *Permit*. Moreover, an empty rule defined in each policy at the last, to *Deny*, if none of the rules above matched.

2.4.2 XACML Data Flow Model

The XACML data flow for policy administration and for access requests is represented in Figure 2.4, as specified in XACML 3.0 Core specification [17].

XACML data flow model is as follows:

1. PAPs write policies and policy sets and make them available to the PDP. These policies or policy sets represent the complete policy for a specified target.
2. The access requester sends a request for access to the PEP.
3. The PEP sends the request for access to the context handler in its native request format, including attributes of the subjects, resource, action, environment and other categories that received with the access request.
4. The context handler constructs an XACML request context, adds received attributes, and sends it to the PDP.
5. The PDP requests any additional subject, resource, action, environment attributes from the context handler.
6. The context handler requests the attributes from a PIP.
7. The PIP obtains the requested attributes.
8. The PIP returns the requested attributes to the context handler.

9. Optionally, the context handler includes the resource in the context.

10. The context handler sends the requested attributes and (optionally) the resource to the PDP. The PDP evaluates the policy.

11. The PDP returns the response context (including the authorization decision) to the context handler.

12. The context handler translates the response context to the native response format of the PEP. The context handler returns the response to the PEP.

13. The PEP fulfills the obligations.

14. If access is permitted, then the PEP permits access to the resource; otherwise, it denies access (not shown in Figure 2.4).

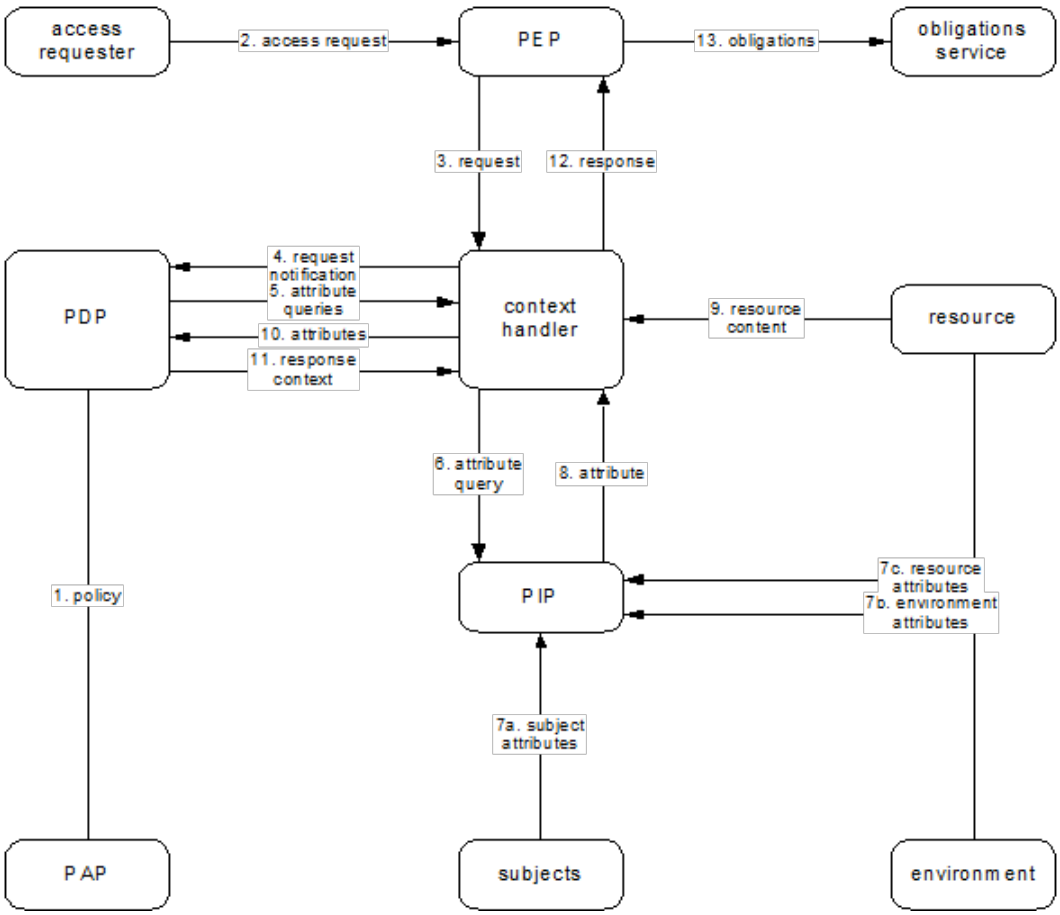

Figure 2.4 – XACML data flow model [17].

Considering the same hypothetical use case, let us see how can a XACML request be made. Suppose, a foreigner named *Alice*, brings one liter of *wine*. Customs now want to know if that amount is permitted for *Alice*. Thus, the access request shown in Figure 2.5 will be made to the PDP. Access Request is represented in JSON format for readability. See Appendix A for the exact XML-based request.

```
Request {
    Category {
      id: subject
      Attribute {
        id: subject.id
        value: Alice
      }
    },
    Category {
      id: resource
      Attribute {
        id: resource.id
        value: wine
      },
      Attribute {
        id: resource.volume
        value: 1
      }
    },
}
```

Figure 2.5 – XACML access control request.

In the access request under each attribute category (subject, resource, action, environment), we define the attributes, so that when the policy is evaluated missing attributes will be picked from the PIP, for the respective category from attributes already received.

In this request, we have asked from the PDP, "*Can subject Alice* bring *1 liter* of *volume* from *wine resource*?" Then, as per the policy defined in Figure 2.3, this request is permitted. The specific evaluation steps are as follows:

1. PDP receives the *subject.id* as *Alice* but it wants the *subject.citizenship* attribute.
2. PIP will solve that to the PDP and say that *subject.citizenship* is *foreigner*.
3. Therefore, the first policy *DutyFreeAllowancesForForeigners* will match for this request as per the condition in the *target*. Thus, that policy will be picked up for evaluation.

4. As the *resource* is *wine* the first rule *WineLiqorAllowanceForForeigners* will be executed as its target will be matched.

5. The condition evaluates to true, thus rule result is *Permit*.

6. Now, the rule-combining algorithm used in the policy is *permit-override*, which gives priority to *Permit* decisions over *Deny* decisions, which means if at least one rule evaluates to *Permit* the net result will be *Permit*. Thus, application of this algorithm in this context will result to a *Permit* decision as the first rule itself evaluates to a *Permit* and there is no need of evaluating the rest.

Therefore, the net result of this access request will be *Permit* and PDP will send the decision response to the PEP (XML based response generated from the PDP is listed in Appendix A). However, asking a query like "*What are the items allowed for Alice*?", rather than specifying the resource, requires evaluating the policy over all resource attribute combinations to decide on the conditions under which *Alice* can bring in that resource.

2.5 Access Reviews with XACML

In this section, we look at related work on improving the performance aspects of XACML policy evaluation where they solved the policy administrative and management functions, and which tried to solve the exact access review problem that we are focusing in this research.

2.5.1 Partial Evaluation of Policies

Sandberg, has conducted a feasibility study at [4], which verifies if reverse queries such as "What resources can this user access", can be initiated as a partial request with some supplied attributes, and get a simplified policy by a partial policy evaluation being performed on top of the attributes received.

Considering the hypothetical example mentioned in Section 2.4.1, suppose we need to know "What items are permitted for Alice". Expressing this in an access request will

include only the subject id, which is known as ‘Alice’. Thus, such a request is termed as a *partial query* in this research.

Partial Query:

subject.id = Alice

The exact policy evaluation for such an access request will return *Not Applicable* or *Indeterminate* indicating an error, as all attributes required to evaluate the policy cannot be extracted with such a partial query. Therefore, what can be conducted is a policy simplification, in which received and extracted attributes from PIPs are applied to simplify policy conditions. This will result in a simplified policy, with a set of conditions that cannot be simplified furthermore, for successful policy evaluation. Thus, results after a partial evaluation will be a set of conditions that need to be satisfied.

Result after partial evaluation:

Permit if resource.id=wine or resource.id = liquor and resource.volume <= 1.5 or

Permit if resource.id=allowedItems and resource.value <= 250

Else Deny

In the suggested approach, when a partial query is received below steps are invoked for partial query evaluation which results in a simplified policy:

1. The partial query is chosen and the supplied attributes are inserted at their referencing designators in the policy.
2. The policy reduced to a simpler form through simplifications, exploiting the information of the inserted attributes.
3. The result of simplification is a piece of logic containing references to attributes that have not been supplied. The context of these attribute references answers the access review query.

Given a XACML policy, policy simplification includes simplifying *Targets* and *Conditions.*

As discussed in [4], *Targets* are simplified by removing *Match* elements that evaluates to true. If the *Target* evaluates to false, what the *Target* designates (Rule, Policy, PolicySet) is removed.

Apply and *Condition* elements are reduced by reducing functions used within these elements applying known attributes. Research further defines the simplification for such functions.

Finally, *Rule*, *Policy* and *PolicySets* are simplified with respect to combining algorithms. For an example the *Deny Overrides* combining algorithm defined in XACML Core specification [17], gives priority to *Deny* decisions over *Permit* decisions. Thus, adhering to that definition of the algorithm, the author defines below simplification rules for the Deny Overrides combining algorithm.

1. If there exists a rule that evaluates to *Deny* whose target is applicable, remove all other rules and return *Deny*.
2. If there exists a rule that evaluates to *Indeterminate* whose target is applicable, remove any rule with a *Permit* effect. (I think we can remove only if the rule effect of the one return *Indeterminate* is a Deny)
3. If there exists a rule that evaluates to *Permit* whose target is applicable, remove all rules that evaluate to *Indeterminate* if their effect is *Permit*.
4. If there exist no rules with a *Deny* effect and at least one rule with a *Permit* effect, remove all other rules and return *Permit*. (Yet if the rule evaluates to Indeterminate it should be Indeterminate)
5. If there exist no rules with a *Deny* effect and all rules evaluate to *Indeterminate,* return *Indeterminate*.

However, there are some inconsistencies with the exact Deny Override algorithm defined in the specification [17]. For example, in rule (2), we can only remove rules with *Permit* effect, if the effect of the rule, which evaluates to *Indeterminate* is a *Deny*. This is because precedence is given for rules with *Deny* effect, thus, if such a rule evaluates to an *Indeterminate*, that result gets precedence over other rules with *Permit*

effect. However, if it is a rule with *Permit* effect that evaluates to Indeterminate, a rule which evaluates to a *Permit* is given precedence over the rule evaluated to error state.

The author explains the simplification approach with an example. A company policy set defines a printer usage policy and an internet access policy for employees. First applicable policy is executed.

Printer access policy:

- An *employee* can print on *PrinterA* or *PrinterB*.
- If *Red* or *Green* ink is used print access denied.
- If *Client-version* is less than 4 or *No Of Pages* to print is greater than 50, print access denied.
- Unless otherwise *employee* is permitted to print.

Internet access policy:

- *send-packets* to *main-router* is permitted.
- If the *MAC-address* is one of (*aa:bb:cc:dd:ee: ff, 00:11:22:33:44:55, 99:88:77:66:55:44, 55 :00:99:11:88:22*) access is denied.

```
<Request>
    <Subject>
        <Attribute AttributeId="role" DataType="string">
            <AttributeValue>employee</AttributeValue>
        </Attribute>
        <Attribute AttributeId="clientVersion" DataType="integer">
            <AttributeValue>5</AttributeValue>
        </Attribute>
    </Subject>
    <Action>
        <Attribute AttributeId="action" DataType="string">
            <AttributeValue>print</AttributeValue>
        </Attribute>
    </Action>
    <Resource>
        <Attribute AttributeId="destination" DataType="string">
            <AttributeValue>Printer A</AttributeValue>
        </Attribute>
    </Resource>
</Request>
```

Figure 2.6 – Partial query request.

The partial query that needs to be executed is "*What affects if an employee may print on Printer A if he uses client version 5?*", which is represented by the XACML 2.0 request protocol [18] as in Figure 2.6.

Steps in the simplification algorithm will be as follows:

1. Replace attribute designators with all known attributes.
2. Keeps references to all the elements, which were inserted, go through each of inserted attributes placed within *Targets* and evaluate each *Target*.
3. Iterate below, as long as policy can be reduced.
    a. Evaluate *Apply* elements.
    b. Evaluate *Condition* elements.
    c. Apply rules of simplification.
    d. Simplify by combining algorithms.

For example, simplification of Condition element, in which attribute designators are replaced with known attribute values is depicted in Figure 2.7. Here, the *Condition* element includes two *Apply* functions, which are combined with *OR* function. First, one is an *integer less than* function and the second one is an *integer greater than* function. The attribute designator value in the first function has been replaced by the *clientVersion* attribute received in the partial request, which is five, whereas the attribute designator of the second function is replaced with an empty element, as its value is not known. Thus, when considering the complete Condition element, first function can be simplified as its value argument values are now known. Applying the function integer less than, to the two arguments evaluates the function to false as,

$$5 < 4 \rightarrow False$$

Thus, the simplified Condition element is now expressed as below.

$$False\ OR\ (x < 50)$$

Here x is the *no of pages* attribute, which is not denoted in Figure 2.7. Thus, the simplified expression now denotes that the *no of pages* should be less than 50 for this condition to be satisfied.

Old:

```
52          <Condition FunctionId="&or;">
53             <Apply FunctionId="&integer-lt;">
54                <Apply FunctionId="&pick-integer;">
55                   <Apply FunctionId="&integer-bag;">
56                      <AttributeValue
                            DataType="&type-integer;"
57                          >5</AttributeValue>
58                   </Apply>
59                </Apply>
60                <AttributeValue DataType="&type-integer;"
61                   >4</AttributeValue>
62             </Apply>
63             <Apply FunctionId="&integer-gt;">
64                <Apply FunctionId="&pick-integer;">
65                   <Apply FunctionId="&integer-bag;">
66                   </Apply>
67                </Apply>
68                <AttributeValue DataType="&type-integer;"
69                   >50</AttributeValue>
70             </Apply>
71          </Condition>
```

New [3]:

```
39       <Condition FunctionId="&or;">
40          <AttributeValue DataType="&type-bool;"
41             >false</AttributeValue>
42          <Apply FunctionId="&integer-gt;">
43             <Apply FunctionId="&pick-integer;">
44                <Apply FunctionId="&integer-bag;">
45                </Apply>
46             </Apply>
47             <AttributeValue DataType="&type-integer;"
48                >50</AttributeValue>
49          </Apply>
50       </Condition>
```

Figure 2.7 – Simplification of condition element [4].

Therefore, finally applying such simplification for the policy will result in a simplified policy, which will denote a set of logical functions to be satisfied, which express the conditions that should be met for the partial query to evaluate. However, the author does not explain on how to interpret the exact answer, for the access review query initiated from the simplified policy. He just highlights that this model is feasible to answer access review requests, which are partial queries in XACML context. Further, compared to exact algorithms defined in XACML, there are some inconsistencies with the derived simplification rules. Moreover, the simplification process drops expressions that evaluates to *Indeterminate* results which simply cannot be neglected when it comes to some logical expressions. For example, when evaluating a Boolean AND function, any argument of the function which evaluates to *True* may be discarded

as the result of the function in that case will depend entirely on the other arguments of the AND function. However, if the first argument of an AND function results into a simplified expression and the second argument results in an Indeterminate, the AND expression cannot be simplified further, and the Indeterminate of the second argument cannot be discarded. The reason is that the result of partial evaluation must be equivalent to the original AND expression with respect to any access control request which is consistent with the partial request. Thus, because the first argument is simplified it may take any value *True*, *False* or *Indeterminate*. If the first argument evaluates to *True* or *Indeterminate*, then the *Indeterminate* of the second argument makes the whole AND expression *Indeterminate*. If the first argument evaluates to *False*, then the whole AND expression is *False*. This is also discussed at [19], as a drawback of [4].

2.5.2 Axiomatics Reverse Query

Axiomatics Reverse Query (ARQ) extend the policy simplification approach defined in [16], to define methods on constructing a simplified policy such that, it reproduces not only *Permit* and *Deny* decisions, but also data relating to errors which lead to *Indeterminate* and *Not Applicable* state. Thus, they intend to address the fact that evaluation of any partial request should be consistent with an evaluation of an access request.

The authors provide a method for partially evaluating an ABAC policy, creating a representation of the simplified policy [19]. This representation includes a dedicated data field associated with an expression, for storing intermediate results of evaluation of the expression itself or an expression subordinate. Following rules are imposed with simplification:

- An expression evaluable only to False is formed in the simplified ABAC policy for each expression in the full ABAC policy which is not completely evaluable under the partial request and which is connected by a Boolean AND function to at least one expression that evaluates under the partial request to False.
- An expression evaluable only to Indeterminate is formed in the simplified ABAC policy for each expression in the full ABAC policy which is not

completely evaluable under the partial request and which is connected by a combining algorithm to at least one expression that evaluates under the partial request to Indeterminate.

Adhering to above simplification rules, authors define a way to perform partial evaluation. The fundamental goal in this approach is to construct a simplified ABAC policy equivalent to a full ABAC policy.

1. Input a full ABAC policy that includes attribute dependent expressions, wherein each expression is evaluable to one of *Not applicable*, *Indeterminate* and either *Permit* or *Deny.*

2. Input a partial request comprising at least one attribute value and at least one attribute identified as variable.

3. Partially evaluate the full ABAC policy by substituting at least one attribute value for a corresponding attribute appearing in the policy. Then form a simplified ABAC policy equivalent to the full ABAC policy based on the evaluation result thus obtained, and predetermined simplification rules.

4. Simplified ABAC policy includes an expression having a result data field for storing evaluation result.

Going forward, they propose a system for evaluating access review queries at [20] based on the above policy simplification model. The system is depicted in Figure 2.8. Following steps are invoked by the system with respect to the evaluation of an access review request (reverse query):

1. Receive a reverse query indicating a given decision (d) (Permit or Deny), and a set (R) of admissible access requests, each of which comprises one or more attributes appearing in the ABAC policy and explicit values assigned to them.

2. Extract attributes to which all access requests in the set (R) assign identical values.

3. Reduce the ABAC policy at least by substituting values for the extracted attributes.

4. Cache the policy after said reducing, as a simplified policy (P′).
5. Translate the cached simplified policy (P′) and the given decision (d) into a satisfiable logic proposition in Boolean variables.
6. Derive all variable assignments satisfying the logic proposition.
7. Extract, based on the variable assignments thus derived, all access requests from the set (R) for which the ABAC policy (P) yields the given decision (d).

Thus, the extracted access requests denote the respective conditions that should be met to yield the given decision by the reverse query.

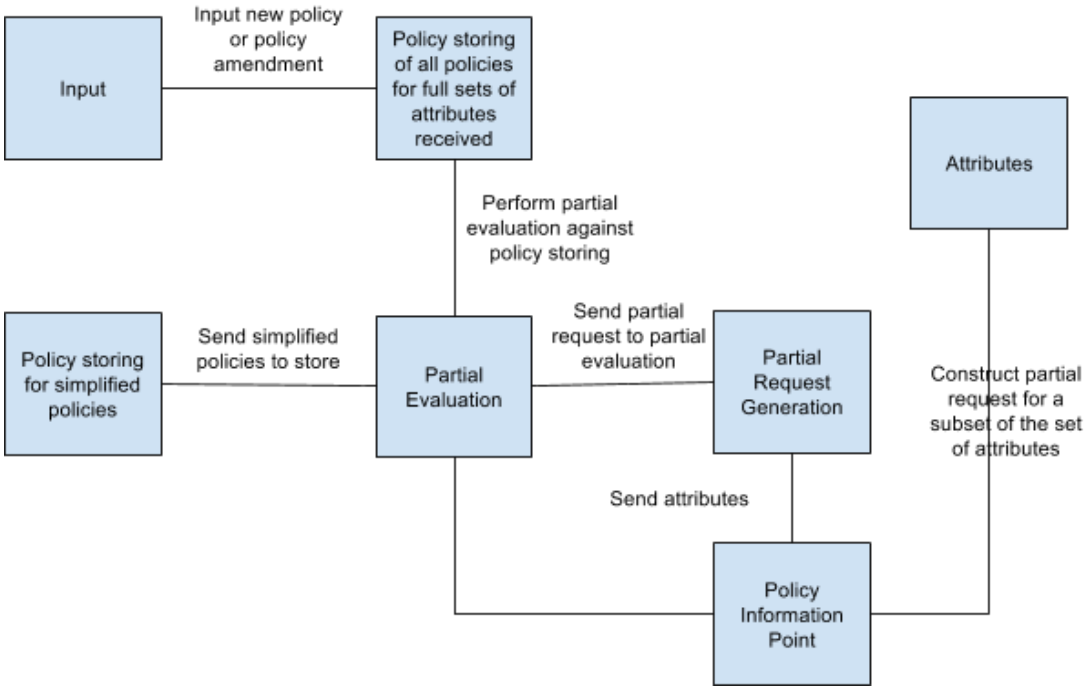

Figure 2.8 – Reverse query system.

Based on above inventions, Axiomatics have come up with a comprehensive solution on supporting access review requests with XACML, which is known as Axiomatics Reverse Queries (ARQ) [21]. This solution is capable of extracting conditions for a given access review query. Further, it supports producing the support in various formats. For example, ARQ can generate SQL queries based on conditions extracted for an access review request [22].

Figure 2.9 illustrates a simple such use case in a health care industry, where a doctor views a page with medical records, where he or she is only allowed list them based on the location from which they try to access.

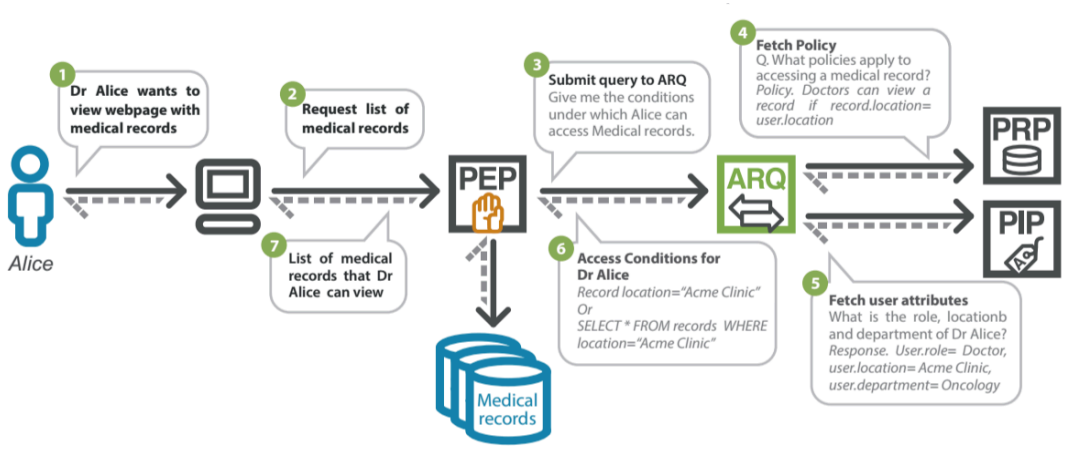

Figure 2.9 – Axiomatics reverse query in action [22].

2.5.3 Multi Interval Decision Diagrams

Ngo et al. [5] propose a decision diagram approach using the data interval partition aggregation, to improve policy evaluation performance. Proposed solution can parse and transform complex logical expressions in policies into decision tree structures to improve policy evaluation. The model proposed can be used to solve access review queries as well.

As defined in XACML, specification [17], *Match*, *AllOf*, *AnyOf*, *Target* and *Condition* elements, will always evaluate to one of the results in Table 2.1, on evaluation of the logical function expressed from each element, via subject, resource, action or environment attributes and attribute values. The authors denote these evaluation results by V_M where,

$$V_M = \{T, F, IN\}$$

Table 2.1 – Match, AllOf, AnyOf, Target, Condition result.

Element Result	Notation
Match	T
No Match	F
Indeterminate	IN

Similarly, per [17], Rule, Policy and PolicySet segments will evaluate to one of the results in Table 2.2.

Table 2.2 – Rule, Policy, PolicySet result.

Element Result	Notation
Permit	P
Deny	D
Not Applicable	N
Indeterminate	IN

These evaluation results are denoted by V_R where,

$$V_R = \{P, D, N, IN\}$$

Then, to represent functions having signature over each domain above, the authors define two types of decision diagrams, which are derived by considering variable interval partitions.

Let us look at the sample policy listed in Figure 2.10 to understand how decision diagrams are structured. This sample policy includes two rules, one with *Permit* effect, and the other with *Deny* effect. Each rule includes a target condition, which is a logical function with three variables, *vol* (volume), *t* (time), *p* (price). In addition, each rule includes an obligation [17] as well. Rules are combined in the policy via permit-override combining algorithm. If we list the interval partitions that the variable *vol* can take in above policy, it will be similar to the following:

```
In Rule R1: [100,150), (300,500)

In Rule R2: [100], [100,300], [500]

In Policy P0: [100, 500]
```

```
Policy {
   id: P0;
   combine_algo: permit-override;
   target: {(vol ≥ 100) ∧ (vol ≤ 500)};
   childrens: {R1, R2}
}

Rule {
   id: R1;
   effect: Permit;
   target: {[(100 ≤ vol ≤ 150) ∧ (12 ≤ t ≤ 17) ∧ (3 ≤ p ≤ 4)] ∨
           [(300 ≤ vol ≤ 500) ∧ (1 ≤ p ≤ 2)] ∨
           [(100 ≤ vol ≤ 150)∧ (6 ≤ t ≤ 9)∧ (1 ≤ p ≤ 2)]};
   obligations: {O1, Permit}
}

Rule {
   id: R2;
   effect: Deny;
   target: {[vol = 100)∧ (t = 17)] ∨
           [(100 ≤ vol ≤ 300)∧ (t = 9)] ∨
           [(vol = 500) (t ≥ 12)]};
   obligations: {O2, Deny}
}
```

Figure 2.10 – Sample XACML policy [5].

Now, if we extract the first sub expression of the complete logical expression defined in Rule R1's target, we can denote that as a logical function as below.

$$f(vol, t, p) = (100 \leq vol \leq 150) \wedge (12 \leq t \leq 17) \wedge (3 \leq p \leq 4) \tag{2.1}$$

For the interval partition *P* of variable *vol*, when *P*{[100, 150]}, we can define a partial function of the function above, like below.

$$f_{vol^{[100,150]}} = (12 \leq t \leq 17) \wedge (3 \leq p \leq 4) \tag{2.2}$$

By defining a boolean function for the interval partition of variable *vol* where,

$$h_{vol}^{[100,150]} = 1 \;\; if \; vol \; \in [100,150],$$

$$h_{vol}^{[100,150]} = 0 \;\; otherwise,$$

Equation 2.1 can be represented as,

$$f(vol, t, p) = h_{vol}^{[100,150]} \wedge f_{vol^{[100,150]}} \tag{2.3}$$

Thus, for interval partition [100,150] of variable vol, above function can be represented in a decision diagram, as in Figure 2.11.

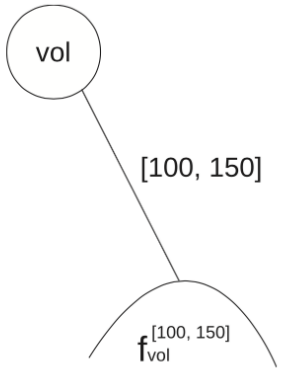

Figure 2.11 – Example of function decomposition [5].

Similarly, a logical function *f*, can be represented as a rooted, acyclic graph *G(V,E)*, with the node set *V,* having internal nodes containing variables and leaf nodes containing Boolean values. Each outgoing edge represents the boolean function over the partition variable. Therefore, each subgraph of a variable is a partial function. This structure is illustrated in Figure 2.12.

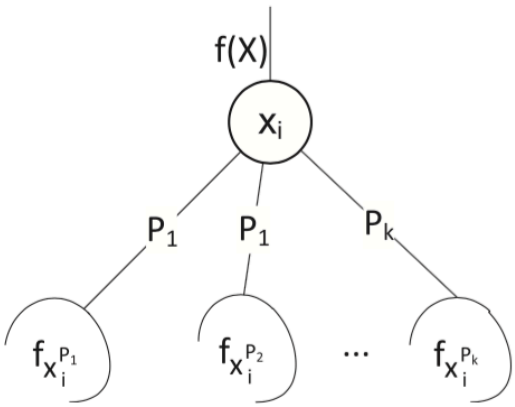

Figure 2.12 – Decision diagram illustration for function decomposition [5].

Authors then extends this representation to Multi Interval Decision Diagrams (MIDD), where two types of decision diagrams are defined as MIDD and X-MIDD to represent V_M and V_R domains respectively.

Definition: MIDD

MIDD is the *G(V, E)* representing a function having the signature over V_M domain.

- Each internal node *m* in MIDD is the tuple of (Node variable, State value, Set of tuples that represent an outgoing edge which contains a reduced interval partition).
- State value can be *(F, IN)*. If node variable is marked as required state value is *IN*, else it is *F*.
- Descendent node can be another internal node or a leaf node containing *T* value, which is called as *T-leaf-node*.

Adhering to above definition, MIDD of the *R1*'s target can be represented as in Figure 2.13.

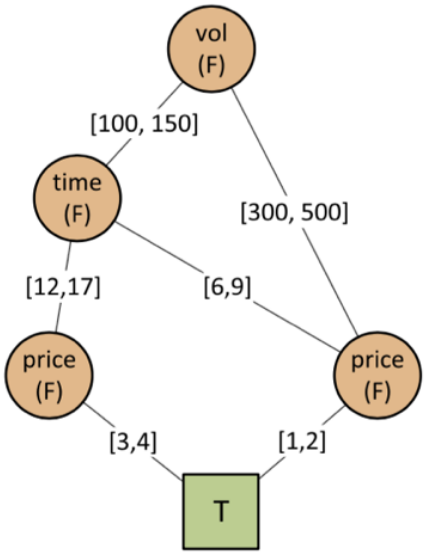

Figure 2.13 – MIDD of *R1*'s Target [5].

Definition: X-MIDD

X-MIDD is the *G(V, E)* representing a function having the signature over V_R domain.

- An internal node *m* is a tuple of (Node variable, State (state value in V_R domain), Obligations and Advices if state is *Permit* or *Deny* else empty, Set of tuples that represent outgoing edges).

- Leaf node contains the policy evaluation result, which is a tuple of (State (state value in V_R domain), Obligations and Advices if state is *Permit* or *Deny*, else empty).

Adhering to above definition, X-MIDD of *R1* can be represented as in Figure 2.14.

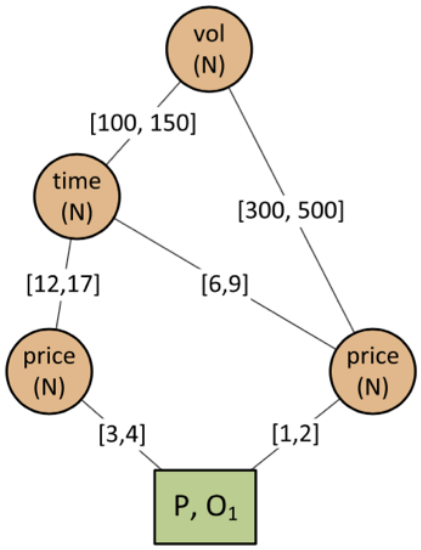

Figure 2.14 – X-MIDD of R1 [5].

Finally, given a XACML policy tree, an equivalent X-MIDD is constructed as below.

1. Extract intervals and build MIDDs representing Target and Condition elements.
2. Create the X-MIDD for each rule from its MIDDs following the mechanism.
3. For a Policy or PolicySet, join X-MIDDs of its children by equivalent combining operators to construct the final X- MIDD instance representing the root policy.

Therefore, the final decision diagram constructed for the policy *P0* can be represented as in Figure 2.15.

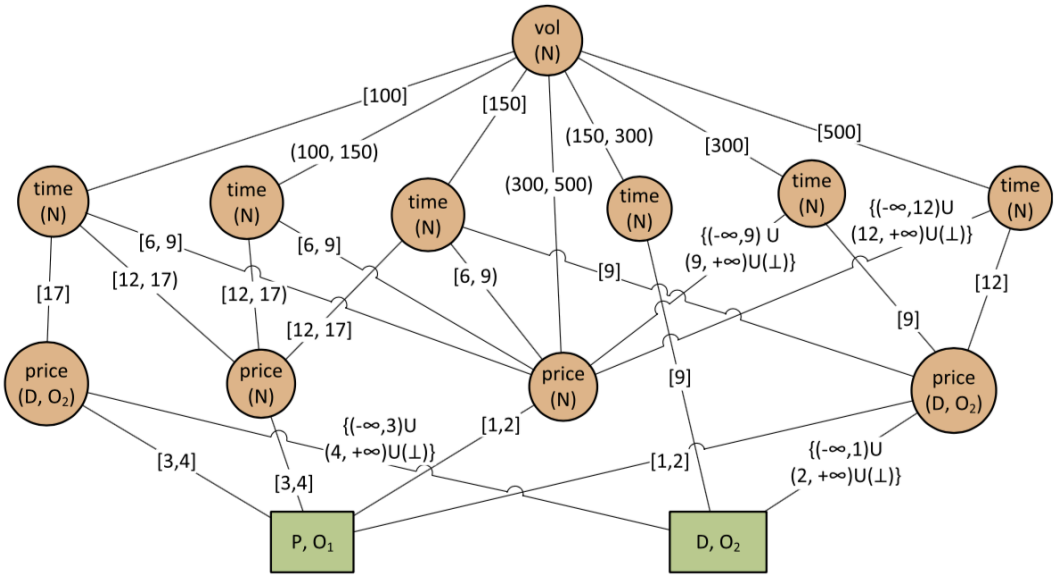

Figure 2.15 – X-MIDD of sample policy *P0* [5].

The advantage of this model is that the decision paths are precomputed, thus answering an access request or an access review request is just a matter of traversing over the graph. For example, suppose an access review request as below is initiated.

"Which resources can the subject Alice access during 9am-6pm?".

Given an X-MIDD representing the policies, the problem now becomes how to enumerate all possible paths reaching the permit decision for the partial request initiated.

{subject.id = 'Alice', action = 'read', time ∈ [9,18]}.

However, there are several other drawbacks of this solution.

- The depth of the graph will grow with respect to the number of variables in the policy.
- No of nodes in the graph will grow with respect to the number of variables and the number of value intervals that each variable could take.
- Ordering of variables may affect the complexity of the graph (the order in which variables are picked to construct the graph).

- Adding a new policy will involve constructing the graph for that policy and merging with the existing graph. However, when updating a policy, it is not clear how to accommodate newly added variable conditions or modified conditions to the graph. Therefore, updating a policy may need to build the graph for whole policy set from beginning.

3 PROPOSED SOLUTION

Two approaches have been proposed to solve the problem of performing access reviews in XACML space ([4], [5]). The first approach performs partial evaluation of policies using the usual policy evaluation approach [4]. This approach takes more time, as each policy needs to be evaluated and simplified, against a subset of attributes that needed to evaluate a policy, to extract the applicable set of conditions. The second approach, construct a decision graph that represent XACML policy sets. Due to pre-constructed representation, this approach reduces evaluation complexity, but incurs scalability problems as the graph grows with respect to the number of variables in the policy set and the distinct value intervals that each variable could take.

Thus, in this research, we look to construct a data model for XACML policy representation, that could represent the permission relationships between subjects and objects and could isolate matching subjects or objects, for a given access review request to extract applicable set of conditions efficiently. Reconsidering the problem statement, we can conclude about access control in ABAC with following facts:

1. Goal of access control is to define permissions between subjects and objects and restrict access, based on those permissions.
2. In access control models, permissions define the connections between the subjects and objects.
3. Similarly, in ABAC, a policy defines the permissions between subjects and objects.
4. In a policy, both the subjects and objects are defined with their attributes. Then there are actions and context information, which define permissions between subjects and objects.
5. Therefore, ultimately, an ABAC policy will define a specific set of subjects with a specific set of attributes, similarly it will define a specific set of objects with a specific set of object attributes. Thus, from a policy it should be possible to identify subject sets and resource sets and model the permissions among them as connections.

6. In such a model, asking a query like *'What objects can Alice read?'* means identifying the subject set that Alice belongs to, from the attributes of Alice, and identifying resource sets that such a subject will have permission to. That means, the query will return the conditions that define the respective resource sets that *'Alice'* will be permitted, and the conditions that should be met to reach each resource set.

Therefore, going in this direction we should be able to model the relationship between subjects and objects via the set of conditions defined in a policy, from which we could easily extract the applicable set of conditions to answer an access review request.

3.1 Subject – Object Relationship Model

In XACML, there are four attribute categories namely, subject, resource, action and environment. Here, subject and resource attributes represent subject sets and resource sets. Action and environment attributes define permissions among such subject and resource sets. Thus, given a XACML policy tree, we can identify resource sets and subject sets and model permissions among them as a directed graph. For example, assume a company policy that defines access to employee appraisal forms as follows:

- During the annual performance appraisal time, appraisal forms will be opened for employees to fill in and submit within a specified period. During that period employees, can view and edit their own appraisal forms and submit. After the deadline for submission, they will no longer be able to edit it, but they can view their own appraisal form.
- Managers can view and comment on the submitted appraisal forms, of his/her subordinates before the deadline defined for manager approval. Soon after they will no longer be able to comment, but they will be able to view them.
- HR personnel can view appraisal forms of all the employees anytime.

Suppose in this context,

- Each user has an attribute as *id*, which is the user identifier.

- Employees, managers, and HR personnel are identified by the roles assigned to them, which is denoted by the *role* attribute.
- Managers have an additional attribute as *subordinateIds*, which holds the set of identifiers of his or her subordinates.
- An appraisal form is a resource. It is identified as an appraisal form by the *id* attribute. Resource will have an attribute *sid*, which defines the subject identifier that the resource belongs to.
- Users can perform *read*, *write* or *comment* actions on appraisal forms as denoted by *id* attribute of actions.
- Submission period and the approval period are environmental conditions, that are defined by *submission-period* and *approval-period* environmental attributes respectively.

If we model this access control policy, categorizing subjects and resources as vertices and permissions among them as edges, we can generate the following graph as shown in Figure 3.1. Thus, vertices in the graph represent subject category or resource category entities that will include a collection of predicates, which describe the subject or the resource. Edges will represent the access control decision and the conditions that should be satisfied for that decision, which include predicates constructed with action and environment category attributes.

Vertices in the graph will be either subjects or resources. Therefore, a vertex in the graph will hold following properties.

- Category: *Subject* or *Resource*
- List of (*Attribute Id*, *Attribute Value*, *function*) tuples, i.e. the conditions that should be satisfied

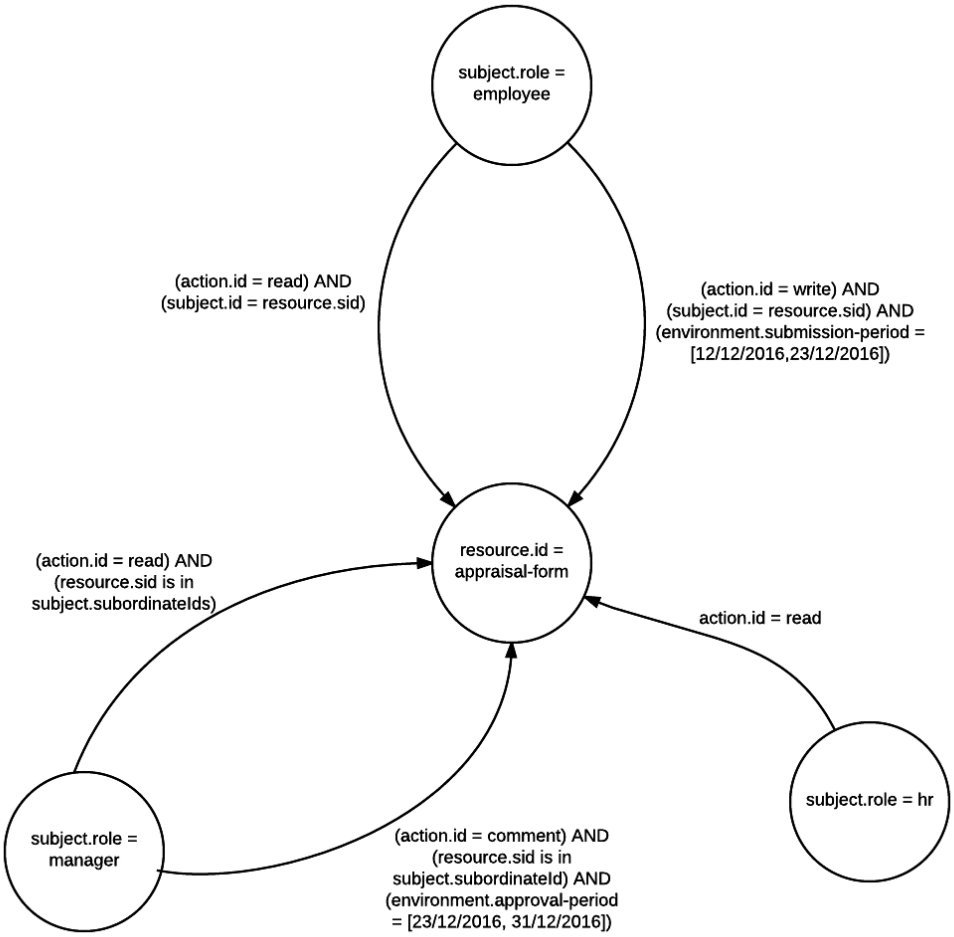

Figure 3.1 – Subject–Object relationship model.

For example, this could be formulated as follows:

```
node: {
  id: employee_node
  category: subject
  conditions: [
    condition
    {
      attribute_id: subject.role
      attribute_value: employee
      function: string-equal
    }
  ]
}
```

```
node {
  id:appraisal_form_node
  category: resource
  conditions: [
    condition {
      attribute_id:resource.type
      attribute_value:appraisal_form
      function:string-equal
    }
  ]
}
```

To match with a node, the category of the node should be matched and each attribute should be matched to the attribute value when the defined function is applied.

Edge in the graph will store,

- Decision: *Permit* or *Deny*
- List of (*Attribute Id*, *Attribute Value*, *function*) tuples, i.e. the conditions that should be satisfied
- Subject node: subject category node that this edge links to
- Resource node: resource category node that this edge links to

For example,

```
edge {
  id:edge_employee_to_appraisal_form
  decision: Permit
  subject_node:employee_node
  resource_node:appraisal_form_node
  conditions: [
    match_attribute {
      attribute_id:action.id
      attribute_value:read
      function:string-equal
    }
  ]
}
```

For an edge to be evaluated to the decision specified, each attribute should be matched to the attribute value when the defined function is applied.

3.2 Methodology

Constructing the above XACML policy representation model, requires traversing over each policy set and policy in the top most XACML policy set, to extract the predicates

defined in each policy, and represent them in a data structure. In XACML, a predicate evolves from the bottom of a XACML policy tree, with *Match* and *Apply* elements within *Targets* and *Conditions* respectively. A predicate defined with *Match* or *Apply* is a single function with few variables, whereas each variable in the function is either a subject, resource, action or environment attribute. *Targets* and *Conditions* are then derived by combining such simple functions with *AND* or *OR* operators, to express complex predicates that should be evaluated to *true* for the respective *Target* or *Condition* element to be matched. Finally, by combining the result of *Target* and *Condition* elements, a policy evaluates to either *Permit* or *Deny*. Therefore, starting from bottom most *Match* and *Apply* elements, we can build a data structure to express each predicate that lead to a *Permit* or *Deny* decision. That means, such a predicate combines conditions defined with subject, resource, action and environment attributes with *AND* operator. Representing that in the data structure will also include the decision that the successful evaluation of that predicate will lead to, i.e., either *Permit* or *Deny*. In a policy, there will be a set of such predicates that lead to either *Permit* or *Deny* decision, based on the conditions defined in that policy, as such conditions merge with OR operator. Therefore, the data structure, which represents a predicate that lead to a Permit or Deny decision, will have,

- a subject node, which includes conditions that define the subject.
- a resource node, which includes conditions that define the resource.
- a link node, which includes conditions that define the subject and resource relationship.
- a decision node, which will say if this leads to a *Permit* or *Deny.*

However, if the predicate does not include subject conditions, the data structure will not have a subject node. Similarly, it may not have a resource or link node as well. However, at least the data structure will include just a decision node (in case of empty rules, which defines only the rule effect). At most, there will be only four nodes in this data structure. This is because, conditions will always be categorized into either subject, resource, or link, and with the decision node, the data structure will have only four nodes at most. Further, because this data structure is organized hierarchically,

with each node having a directed edge exactly from a one other node, it could be considered as a tree. Figure 3.2 represent the model of a predicate tree. Each internal node of this tree will include either a set of conditions derived from subject attributes, a set of conditions derived from resource attributes or a set of conditions derived from action or environment attributes. The external node or the leaf node will include the decision, the predicate will lead to, i.e., either *Permit* or *Deny*. Therefore, here onwards, this tree will be termed as a *predicate tree*.

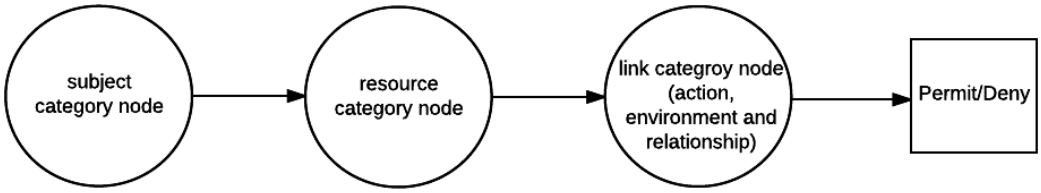

Figure 3.2 – Predicate tree.

Once all predicates in the XACML policy are extracted to a set of predicate trees that group subject-category conditions, resource-category conditions and link-category conditions, the subject-resource relationship graph can be built by combining them together. For each predicate tree perform following,

- Create a vertex for subject, equivalent to the subject category node.
- Create a vertex for resource, equivalent to the resource category node.
- Create an edge between the subject and the resource node, in which the decision is the value of the decision node and add all the conditions of the link node to the edge created, to denote the subject-resource relationship conditions that should be satisfied to meet the decision.

This defines the relationship between a specific subject category and a resource category.

Note that if either subject or resource node is missing, the predicate tree does not represent the subject and resource relationship. Such trees are ignored when building the relationship. Ideally, empty trees could result only with empty rules, which means

there are no conditions within such a rule. Therefore, ignoring them could not have an impact on evaluation, which ultimately extracts the condition set.

For each such relationship, add the subject node to the graph labelling as a subject node, if it is not already present. If such a subject node already exists, merge the edges of node with the edges of the existing node. That means, if the same edge exists in the existing subject node pointing to the same resource node ignore that edge. Else, link the edge with the existing subject node. Similarly, add resource nodes to the graph and merge edges of the node to the existing node if such resource node exists. This will construct a graph as in Figure 3.3, where subjects and resources can be easily identified and the permissions among them can be easily derived.

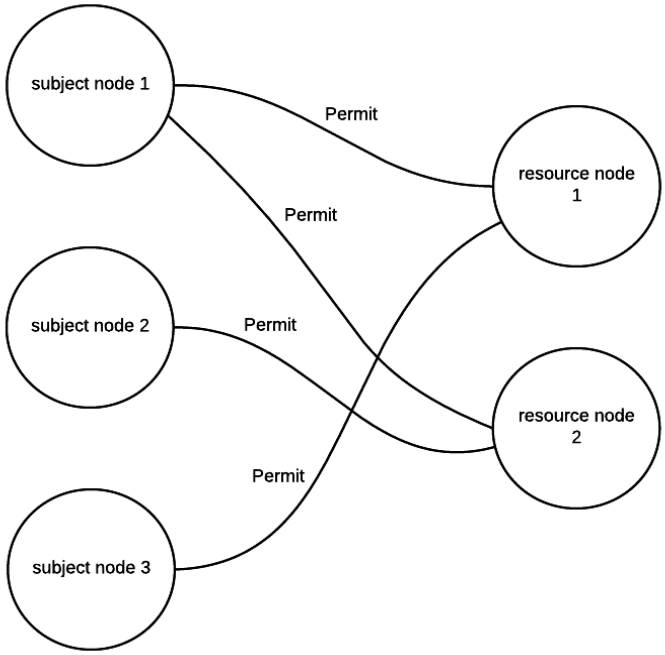

Figure 3.3 – Subject–Object relationship graph.

Now with the above graph constructed, answering an access review request includes,

1. Identifying the set of subject nodes or identifying the set of resource nodes, from the attributes received or can be derived from the access review request.

2. Traversing over the edges that the identified set of subject nodes or resource nodes linked to, and identify the set of resource nodes or subject nodes that relates.

Thus, answering a query like *'What objects can Alice read?'* means identifying the subject node set that Alice belongs to, from the attributes of Alice. Then from the edges of those nodes, identifying the set of resource nodes that such a subject will have permission to. Therefore, the query will return the full set of conditions defined in the path from each subject node to each resource node isolated. This greatly simplifies the query resolution leading the fast and accurate access reviews.

3.3 Theoretical Comparison

In this section, we compare the time complexity and space complexity of the proposed solution, with the Partial Query Evaluation approach [4] and MIDD approach [5] in literature.

3.3.1 Space Complexity

Space Complexity of Subject-Object Relationship Model

In this model, conditions that should be matched for subjects and resources are categorized based on the attributes and attribute values appearing in the policy. There are a set of nodes representing types of subjects via a set of conditions derived from subject attributes. Similarly, there are a set of nodes representing types of resources via a set of conditions derived from resource attributes.

Number of nodes representing subjects and the number of nodes representing resources will always be less than the total number of subjects and resources in the system. Because, the model will not uniquely define each subject or resource, rather it represents subject types and resource types via a set of conditions derived from subject or resource attributes. Therefore, each subject or resource in the system, may match one or more subject nodes or resource nodes based on its attributes.

Suppose, there are m no of subject nodes and n no of resource nodes. Therefore,

$$\text{Total no of nodes in the subject} - \text{object relationship graph} = (m + n) \quad (3.1)$$

If number of edges linking subject and resource nodes is *l*, in the worst case, there will be a link from every subject to every object. Therefore, in the worst case,

$$\text{Total no of edges in the subject} - \text{object relationship graph} = (m.n) \quad (3.2)$$

Thus, in this model, space complexity will depend on the number of subject and resource conditions defined and how uniquely do they categorize subject sets and resource sets. Moreover, in a typical access control policy, there will not be a link from every subject to every object, because the access control matrix that the policy defines will include many empty cells, where permission relationship between the subject and resource not defined. Having a link from every subject to every object means, each cell in the access control matrix is occupied, which is not the practical case.

Space Complexity of Partial Query Evaluation

Partial Query Evaluation model does not change the default XACML policy tree structure which is represented in Figure 3.4, considering only the top-level elements defined by XACML Policy Language Model. Thus, in the XACML policy tree a node will be represented by a policy set, policy or a rule and each edge represents the parent – child relationship among policy sets, policy sets and policies, or policies and rules.

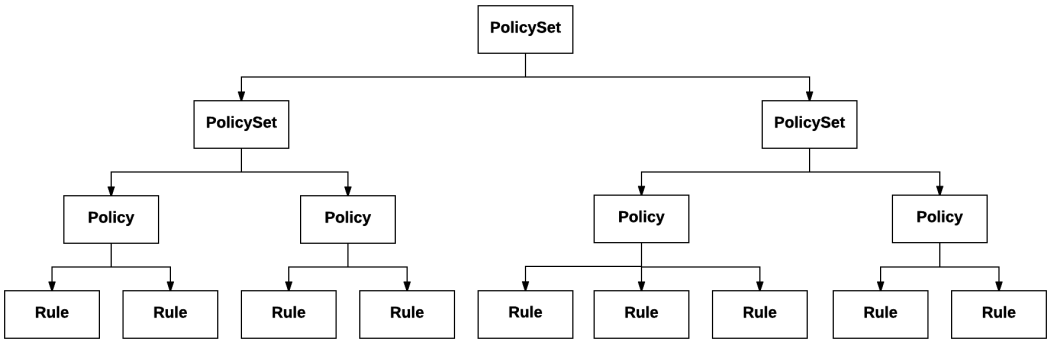

Figure 3.4 – XACML policy tree structure.

Suppose, in such a policy set defined, there are *n* policy sets, *m* policies and *r* rules. In that case,

$$\text{Total no of nodes in the XACML tree structure} = (m + n + r) \tag{3.3}$$

Thus, in this model, the space complexity depends on the number of policy sets, policies and rules defined. The tree structure grows as more policies, rules and rule conditions come in to the system.

Space Complexity of MIDD Approach

Suppose, there are n attributes defined in the policy set, and the attribute set A is denoted as follows:

$$A = \{a_1, a_2, a_3, \dots, a_i, \dots, a_n\} \tag{3.4}$$

If a_i attribute has only one value a' appearing in the policy set, distinct value intervals for attribute a_i is derived as follows:

$\{(-\infty, a'), [a'], (a', +\infty)\}$

Therefore, there are three distinct value intervals for only one value appearing in the policy set. If there are two values a' and a'' appearing in the policy set, where $a' < a''$, distinct value intervals for attribute a_i is derived as follows:

$\{(-\infty, a'), [a'], (a', a''), [a''] (a'', +\infty)\}$

Thus, there are five distinct value intervals. Therefore, if a_i attribute has k_i different values appearing in the policy set, distinct value intervals for attribute a_i can be separated into $2k_i + 1$ intervals. Therefore, the MIDD representing the policy tree has at most $2k_i + 1$ outgoing edges from any node at level l_i (for n no of attributes in the policy there will be n+1 levels). Therefore,

$$\text{Maximum no of nodes at level } l_i = \prod_{j=1}^{i}(2k_j + 1) \tag{3.5}$$

$$\text{Total no of nodes in the MIDD in worst case} = \sum_{i=1}^{n}\prod_{j=1}^{i}(2k_j + 1) \tag{3.6}$$

$$\text{Total no of edges in the MIDD in worst case} = \sum_{i=1}^{n}\prod_{j=1}^{i}(2k_j + 1) \tag{3.7}$$

Thus, in this model the space complexity in the worst case does not depend on the number of policies, or the complexity of logical formulas in target or condition expressions. It depends on the number of attributes and number of distinct attribute values in the full policy set. Moreover, the size of the MIDD is affected heavily by the attribute ordering.

3.3.2 Time Complexity

Time Complexity of Subject – Object Relationship Model

In the proposed model, subject sets and resource sets defined by the access control policy, are isolated. Therefore, for a subject centric access review, such as '*What are the resources can this user access*?', it is a matter of finding the set of subject nodes that the request match. Each subject node in the model, groups a set of conditions defined with certain set of subject attributes. Therefore, the subject in the access review request may match with one or more subject nodes in the model, based on the attribute values of that subject.

Suppose there are *m* such subject nodes in the model, if subject nodes are denoted by *S*,

$$S = \{s_1, s_2, s_3, \dots, s_i, \dots, s_m\} \tag{3.8}$$

If s_i has k_i conditions, time complexity to evaluate each node and find the matching subject set is as follows:

$$\text{Time Complexity of Subject} - \text{Object Relationship Model} = k_1 + k_2 + \cdots + k_m = \sum_{i=1}^{m} k_i$$

$$\text{Therefore, worst case complexity is } = O\left(\sum_{i=1}^{m} k_i\right) \tag{3.9}$$

Same evaluation time complexity applies for resource centric access reviews as well.

Time Complexity of Partial Query Evaluation

In XACML, access control logic resides in *Target* elements, of policy sets, policies, rules and *Condition* elements of rules. Thus, evaluation includes traversing over the

XACML policy tree, and evaluating *Targets* of policy sets, policies, rules and *Conditions* of rules as represented in Figure 3.5.

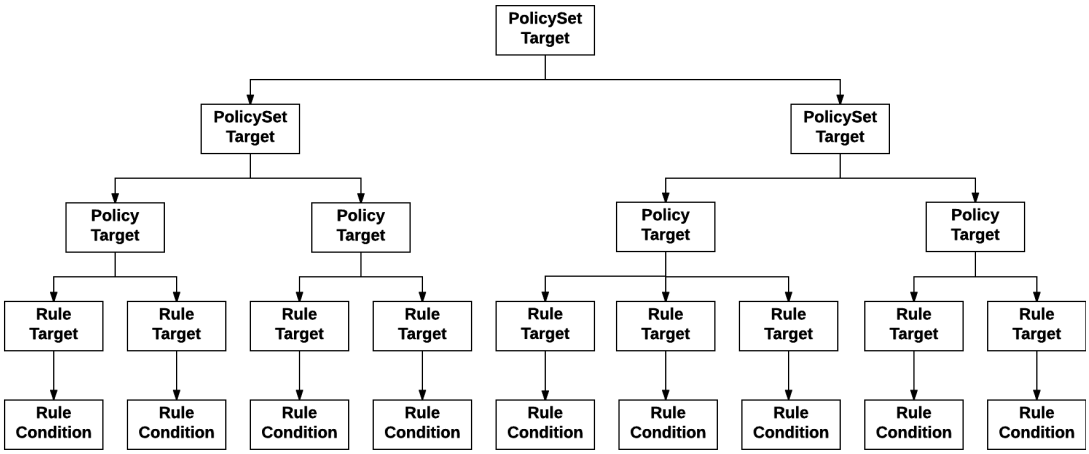

Figure 3.5 – XACML policy evaluation structure.

Partial Query Evaluation (PQE) approach follows the same model. In a typical access control request, policy evaluation expects all attributes needed to evaluate the policy successfully. However, in an access review request, only a subset of the attributes will be available. Thus, policy simplification includes traversing over the tree and simplifying each *Target* and *Condition* for available attributes in the worst case.

Suppose, in such a policy set defined, there are *n* policy set targets, *m* policy targets, *r* rule targets and *k* rule conditions. In that case,

$$\text{Time Complexity of PQE Model} = O(m + n + r + k) \tag{3.10}$$

Time Complexity of MIDD Approach

In this model, for a given access review request, which is a partial access request with few attributes, we need to traverse the graph over the paths that will lead to a *Permit* decision node, for given attribute values. Each path traversed, denotes the conditions that should be met. Thus, evaluation of an access review request is a depth first graph search. Therefore, in the worst case, we will be visiting all nodes and edges, where the time complexity can be given as follows:

$$\text{Time Complexity of MIDD approach} = O(V + E) \tag{3.11}$$

Where V is the number of vertices and as E is the number of edges, which is given by Equation 3.7.

3.3.3 Discussion

Table 3.1 summarizes the space complexities computed for the proposed model and the prevalent solutions. As derived in Equation 3.3, the space complexity of the PQE approach solely depends on the number of policy sets m, policies n, and rules r introduced to the access control system. Such policies or rules may refer the same set of subjects or resources, but irrespectively the structure will grow.

Table 3.1 – Summary of space complexities.

Solution	Number of Nodes	Number of Edges
Subject – Object Relationship Model	$(m+n)$	$(m.n)$
PQE	$(m+n+r)$	$(m+n+r)-1$
MIDD	$\sum_{i=1}^{n}\prod_{j=1}^{i}(2k_j+1)$	$\sum_{i=1}^{n}\prod_{j=1}^{i}(2k_j+1)$

In MIDD approach, space complexity depends on the number of attributes defined in the XACML policy set and the distinct value intervals they could take. Both, the number of nodes and the number of edges in the model incurs the same complexity, with respect to the number of attributes and the distinct value intervals they take. Thus, the graph grows when more attributes come into the access control policy. Moreover, if we look at the space complexity derived for MIDD approach, the size of the graph is mainly determined by the distinct attribute value intervals. Further, the graph complexity will vary with the attribute ordering considered at graph construction. Thus, this affects the scalability of this model.

As derived in Equation 3.1 and Equation 3.2, in Subject – Object relationship model the graph depends on the number of subject sets m and resource sets n identified from the XACML policy set. Therefore, the graph will grow only with the addition of a new subject set or a resource set. The model clearly isolates the subject groups and resource groups and links among them. Thus, this model is more scalable over the PQE and MIDD approaches.

Table 3.2 summarizes the time complexities computed for the proposed model and the prevalent solutions. The time complexity of PQE approach depends on the number of policy set *m*, policy targets *n*, rule targets *r*, and rule conditions *k* in the access control policy. During the access review query, each condition needs to be evaluated against the specified functions over simplification rules. This adds a considerable cost to the query evaluation. Moreover, in a typical access review request, the attributes received will be a collection of subject category, action category and environment category, or it will be a collection of resource category, action category and environment category, based on whether it is a subject-centric access review request or resource-centric access review request. However, in PQE model, irrespective of the nature of the access review request, all targets and conditions needs to be evaluated for simplification.

Table 3.2 – Summary of time complexities.

Solution	**Time Complexity**
Subject – Object Relationship Model	$O(\sum_{i=1}^{m} k_i)$
PQE	$O(m+n+r+k)$
MIDD	$O(V+E)$

In MIDD approach, for the worst case, all nodes and edges in the tree will be visited. Thus, as the number of nodes and number of edges of the tree depends on the number of attributes and the distinct attribute intervals they can take as derived in Equation 3.6 and Equation 3.7, time complexity depends on the same variables. The advantage of this model is that some complex function evaluations are skipped from query evaluation time, because the representation derives paths based on the value intervals. However, the evaluation complexity is still affected by the attribute ordering.

In Subject – Object relationship model, attribute categories received are taken into consideration, from which subjects and resources can easily be identified. Therefore, each condition defined in the policy is not evaluated. Thus, this reduces the cost of evaluation considerably when compared to PQE method. However, to find the matching set of subjects or the matching set of resources, all identified subject sets or resource sets will be evaluated and condition evaluation will happen at the time of

processing the query. Therefore, when compared with the MIDD approach, for a smaller number of attributes and distinct attribute intervals in the policy, MIDD approach will perform well. Yet, for higher number of attributes, attribute value intervals and with respect to attribute ordering, the evaluation time of the MIDD approach will be effected heavily, whereas the Subject – Object relationship model will not be effected at the same rate, as it grows at a relatively slower rate with respect to those variables.

Therefore, looking at the space complexities derived, we can conclude that the proposed model is more scalable over PQE and MIDD approach. Then with respect to time complexities, proposed model is not much affected by the number of conditions or the number of attributes introduced to the access control system and fluctuate in query evaluation times with respect to those variables. Thus, the proposed model will perform efficiently, irrespective of the complexity of the XACML policy tree.

4 IMPLEMENTATION

This chapter illustrates the algorithm that constructs the subject-object relationship model proposed in Chapter 3. The hypothetical example presented in Section 2.4.1 is used to explain the algorithm.

4.1 Graph Construction

In a XACML policy, predicates are defined with *Target* elements of a policy set, policy, or rule and *Conditions* element of a rule. When an access request is made, those are the elements that are evaluated against the received set of attributes. Therefore, extracting the predicates from a XACML policy and constructing *predicate trees*, include parsing *Target* elements of policy set, policy, and rules, *Conditions* element of rules, and finally combining them together.

This research focus on parsing *Target* elements only. But the same model can be followed to parse rule *Conditions* to build *predicate trees,* to include them in the subject – object relationship graph.

Let us consider the hypothetical example presented in Section 2.4.1. In the sample policy presented, we can express access control predicates from policy and rule *Target* elements and build up the evaluation structure of the XACML policy tree as in Figure 4.1. Now, let us see how we can parse above policy structure to construct the subject-object relationship model proposed in Chapter 3. An overview of the algorithm, which builds the subject-object relationship graph, is represented in Figure 4.2. Next each of the key steps are discussed in detail.

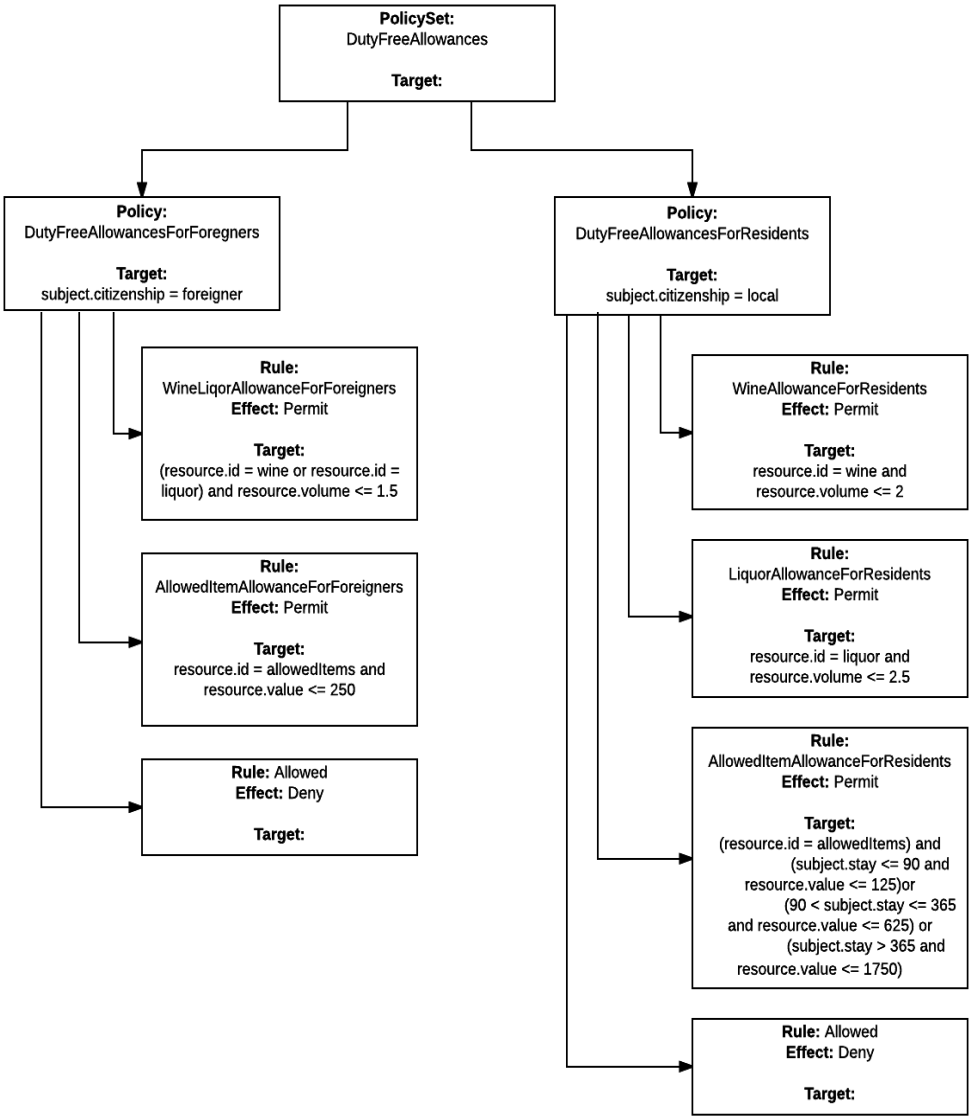

Figure 4.1 – XACML policy tree of hypothetical example.

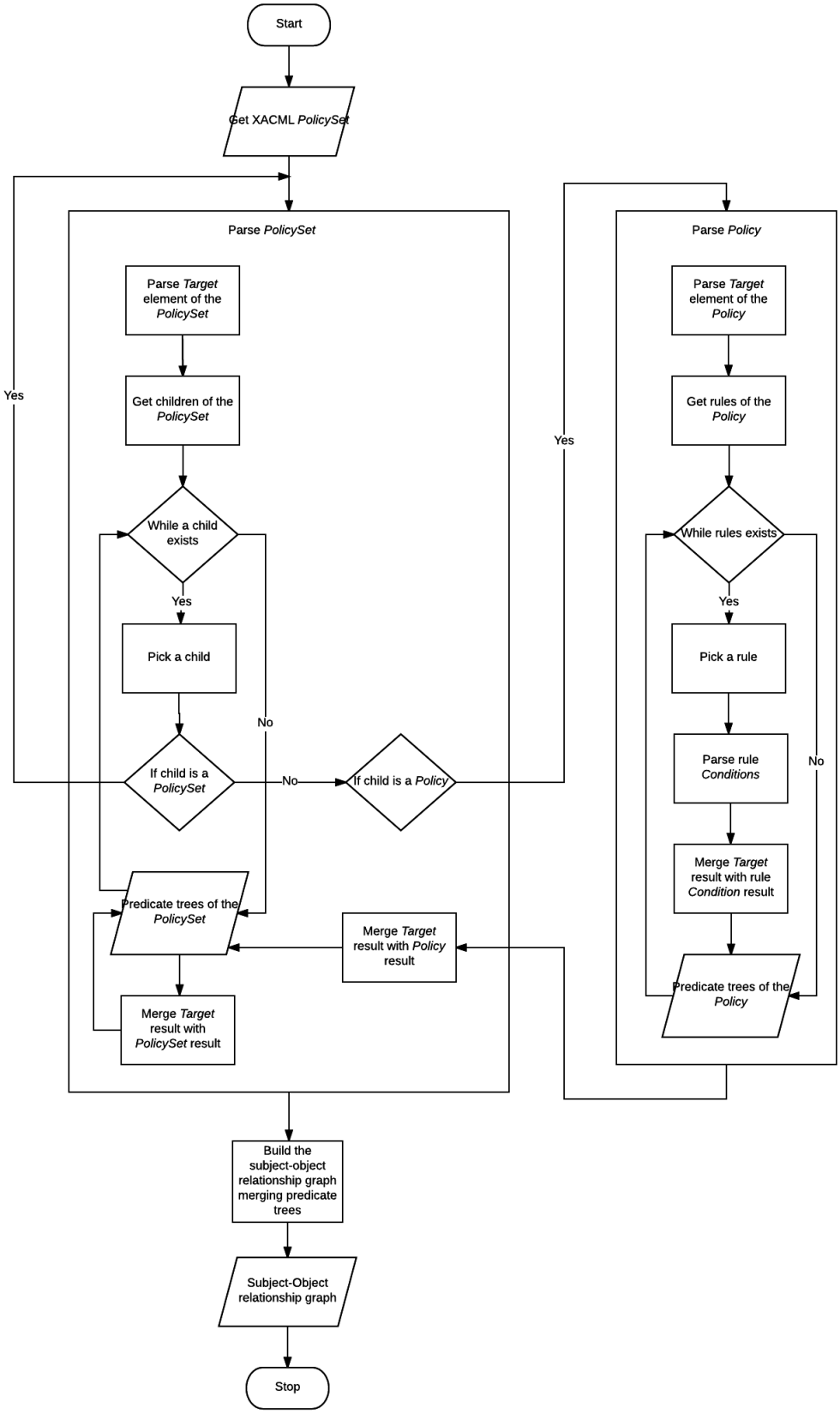

Figure 4.2 – Overview of the subject-object relationship graph construction.

Step 1: Start with rule *Target* elements and construct the set of *predicate trees* that defines the *Target* expression.

Let us consider the *WineLiqorAllowanceForForeigners* rule target. In XML-based XACML policy, this rule target will be represented as shown in Figure 4.3. To improve readability namespaces of functions, attributes, attribute categories and data types are removed. Refer Appendix A to find the complete XACML policy.

```
<Target>
    <AnyOf>
        <AllOf>
            <Match MatchId="string-equal">
                <AttributeValue DataType="string">wine</AttributeValue>
                <AttributeDesignator AttributeId="resource-id"
Category="resource" DataType="string">
                </AttributeDesignator>
            </Match>
        </AllOf>
        <AllOf>
            <Match MatchId="string-equal">
                <AttributeValue
DataType="string">liquor</AttributeValue>
                <AttributeDesignator AttributeId="resource-id"
Category="resource" DataType="string">
                </AttributeDesignator>
            </Match>
        </AllOf>
    </AnyOf>
    <AnyOf>
        <AllOf>
            <Match MatchId="double-greater-than-or-equal">
                <AttributeValue DataType="double">1.5</AttributeValue>
                <AttributeDesignator AttributeId="volume"
Category="resource" DataType="double">
                </AttributeDesignator>
            </Match>
        </AllOf>
    </AnyOf>
</Target>
```

Figure 4.3 – Target expression of WineLiqorAllowanceForForeigners rule.

In a XACML policy, a *Target* expression includes several other constructs, that combines predicates over *OR* and *AND* operators, to build up the complete logical expression. Thus, a *Target* can be represented as in Figure 4.4.

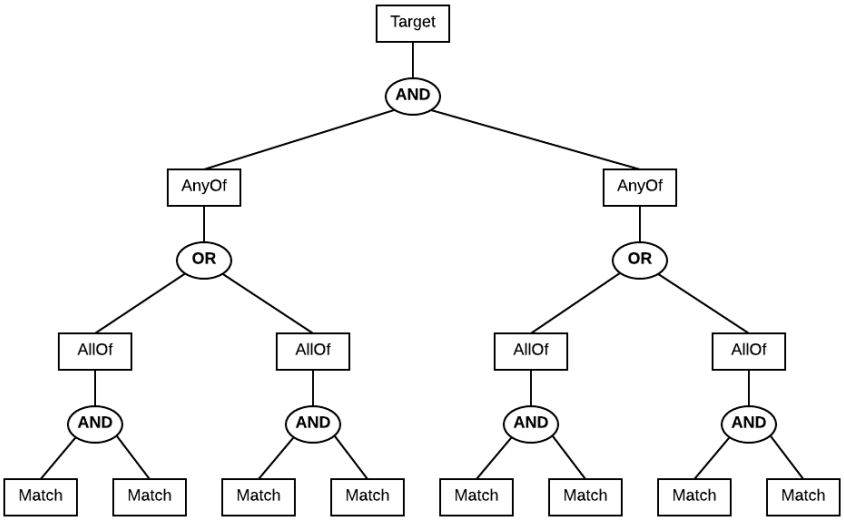

Figure 4.4 – Target expression tree structure.

In XACML, a *Target* expression constructs as follows:

- *Target* includes a set of *AnyOf* elements. Predicates expressed from these *AnyOf* elements are combined with the *AND* operator, which represents the complete *Target* expression.
- *AnyOf* elements include a set of *AllOf* elements. Predicates expressed from these *AllOf* elements are combined with the *OR* operator, which represents the complete *AnyOf* expression.
- *AllOf* elements include a set of *Match* elements. Predicates expressed from these *Match* elements are combined with the *AND* operator, which represents the complete *AllOf* expression.
- *Match* element is composed from a tuple of (*match-id*, *attribute value*, *attribute id*), where *match-id* is a two-operand predicate Boolean function. Thus, each *Match* element represents a Boolean function, which defines a subject, resource or an action or environmental condition to be satisfied.

Thus, parsing the *Target* element includes,

1. Traverse up to *Match* elements.

2. Extract the Boolean predicate from the *Match* element as a condition.

   For example, let us consider the *Match* element depicted in Figure 4.5 of the *Target* depicted in

   Figure 4.3. Here predicate is,

   resource-id = wine

   Extracting this predicate as a condition which is a tuple of (*attribute*, *attribute value*, *function*), we can construct the condition structure as below.

```
condition {
  attribute: resource-id,
  attributeValue: wine,
  function: string-equal
}
```

```
<Match MatchId="string-equal">
    <AttributeValue DataType="string">wine</AttributeValue>
    <AttributeDesignator AttributeId="resource-id"
Category="resource" DataType="string">
    </AttributeDesignator>
</Match>
```

Figure 4.5 – Match expression example.

3. For each such *Match* element within a *AllOf* element, identify the condition and the category that condition belongs, from the *attribute*. Then, add the condition to the respective node that group conditions of that category as in Figure 4.6.

Figure 4.6 – Resource node example.

4. Link each node created, such that it forms a tree, where the leaf node will be a node with value *True*.

This denotes that all conditions of all nodes should be satisfied to lead to a *True* value. Thus, as the XACML core specification defines, each *Match* condition in *AllOf* element is merged with AND operator and represented in a tree.

```
<AllOf>
    <Match MatchId="string-equal">
        <AttributeValue
DataType="string">wine</AttributeValue>
        <AttributeDesignator AttributeId="resource-id"
Category="resource" DataType="string">
        </AttributeDesignator>
    </Match>
</AllOf>
```

Figure 4.7 – AllOf expression example 1.

For example, let us consider the *AllOf* expression depicted in Figure 4.7. Parsing this *AllOf* expression results the tree shown in Figure 4.8. Similarly, parsing AllOf expression in Figure 4.9 will result the tree in Figure 4.10.

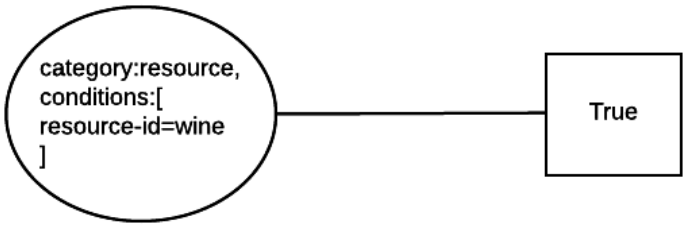

Figure 4.8 – Tree structure resulted from AllOf expression example 1.

```
<AllOf>
    <Match MatchId="string-equal">
        <AttributeValue
DataType="string">liquor</AttributeValue>
        <AttributeDesignator AttributeId="resource-id"
Category="resource" DataType="string">
        </AttributeDesignator>
    </Match>
</AllOf>
```

Figure 4.9 – AllOf expression example 2.

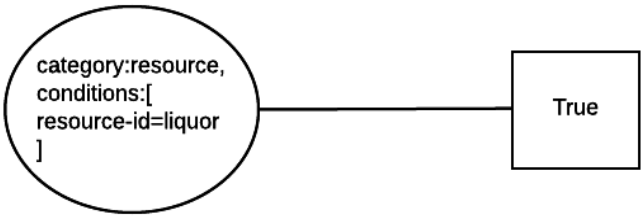

Figure 4.10 – Tree structure resulted from AllOf expression example 2.

Suppose cost of extracting the condition from *Match* is *C1* and adding the condition to the node is *C2*. Therefore, time, T_{sum} taken to parse *Match*

expression is *C1+ C2*. For k_i such conditions (*Match* expressions) within $AllOf_i$ expression the number of iterations is k_i. Thus, time complexity of parsing all *Match* expressions is $k_i(C1+ C2)$. If, cost of constructing the tree and linking nodes is *C3,* time consumption, T_{sum} of parsing $AllOf_i$ expression is k_i *(C1+ C2) + C3*. Therefore, complexity of parsing $AllOf_i$ is $O(k_i)$.

5. Next, parse *AnyOf* expressions, parsing each *AllOf* expression within *AnyOf* as explained in Step 4 above. As per the XACML core specification *AllOf* expressions are merged with *OR* operator within a *AnyOf* expression.

   For example, *AnyOf* expression defined in Figure 4.11 includes two *AllOf* expressions represented in Figure 4.7 and Figure 4.9 respectively. Thus, parsing this *AnyOf* expression parse each *AllOf* expression, which constructs a predicate tree as, discussed in Step 3. Each such tree constructed will be added to a List, because with *AnyOf*, any such predicate result in true means, the expression evaluates to true.

```
<AnyOf
    <AllOf>
        <Match MatchId="string-equal">
            <AttributeValue
DataType="string">wine</AttributeValue>
            <AttributeDesignator AttributeId="resource-id"
Category="resource" DataType="string">
            </AttributeDesignator>
        </Match>
    </AllOf>
    <AllOf>
        <Match MatchId="string-equal">
            <AttributeValue
DataType="string">liquor</AttributeValue>
            <AttributeDesignator AttributeId="resource-id"
Category="resource" DataType="string">
            </AttributeDesignator>
        </Match>
    </AllOf>
</AnyOf>
```

Figure 4.11 – AnyOf expression example.

Having k_j no of *AllOf* elements means there are *kj* possible expressions, and if at least one evaluates to *True, AnyOf* expression evaluates to *True*. Thus, parsing *AnyOf* will return the set of predicate trees. For, k_j number of *AllOf*

elements, time required (T_{sum}) to parse $AnyOf_j = \sum_{i=1}^{k_j} AllOf_i$ can be given as follows:

$$T_{sum} = \sum_{i=1}^{k_j} Ck_i + C'$$

Therefore, complexity of parsing $AnyOf_j$ is $O(k_j k_i)$.

6. Parse *Target* expression, parsing each *AnyOf* expression within *Target* as described in Step 5, and merge the lists of predicate trees resulted from *AnyOf* expressions using *AND* operator.

   We can express *AllOf*, *AnyOf* and Target expressions as Boolean functions as in Equation 4.1, 4.2 and 4.3 respectively.

$$AnyOf_1 = AllOf_a + AllOf_b \quad (4.1)$$

$$AnyOf_2 = AllOf_c + AllOf_d \quad (4.2)$$

$$Target = AnyOf_1 . AnyOf_2 \quad (4.3)$$

   Thus, *Target* can be expressed as follows:

$$Target = (AllOf_a + AllOf_b).(AllOf_c + AllOf_d) \quad (4.4)$$

   Applying distributive law, we can express *Target* expression as follows:

$$Target = AllOf_a.AllOf_c + AllOf_a.AllOf_d + AllOf_b.AllOf_c + AllOf_b.AllOf_d \quad (4.5)$$

   Therefore, as each predicate tree in the list of trees returned by parsing *AnyOf* expression, represent the Boolean predicate of each *AllOf* expression, merging two *AnyOf* expressions means iterating over each list and merging each predicate tree of list 1 with each predicate tree of list 2 and building new list of predicate trees.

   As described in Step 3, parsing any *AllOf* expression will result in a predicate tree with a maximum of four nodes, if the *AllOf* expression includes predicates defined with each attribute category, i.e. subject, resource, action or environment. Such a tree is represented in Figure 4.12. Therefore, considering

Equation 4.5, merging $AllOf_a$ and $AllOf_b$ with *AND* operator means merging the predicate tree resulted from each expression as in Figure 4.13.

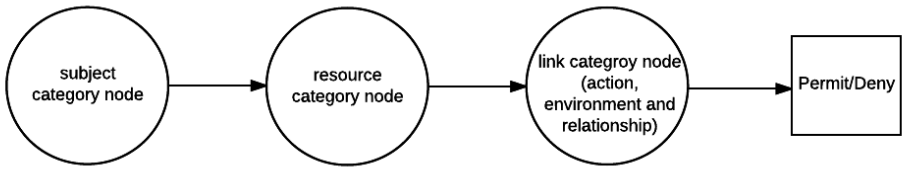

Figure 4.12 – Predicate tree resulted from AllOf expression.

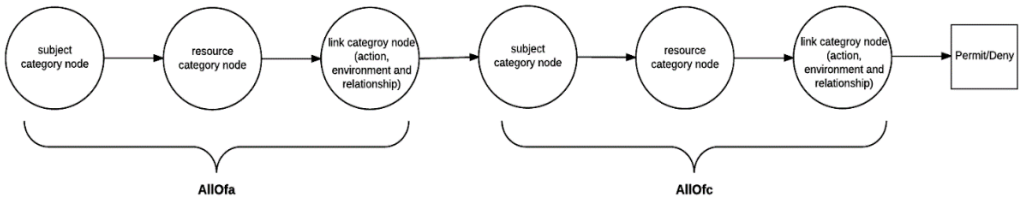

Figure 4.13 – $AllOf_a.AllOf_b$.

Then reduce the tree merging nodes of same category as follows:

i. Start from first node and mark it as current node.

ii. Iterate over the nodes.

iii. If a node of same category of the current node found merge with the current node.

iv. Move to the next node and start from Step ii

This will return an ideal predicate tree defined, with maximum of four nodes (i.e., subject, resource, link, and decision node). For example, let us consider the *WineLiqorAllowanceForForeigners* rule target represented in

Figure 4.3. If we denote the first *AnyOf* expression in the *Target* expression as $AnyOf_1$, the predicate tree list of $AnyOf_1$ can be illustrated as in Figure 4.14. Similarly, if we denote the second *AnyOf* expression in the *Target* expression as $AnyOf_2$, the predicate tree list of $AnyOf_2$ can be given as in Figure 4.15. Therefore, $AnyOf_1.AnyOf_2$ will result in the predicate tree list in Figure 4.16. Reducing each predicate will result the predicate tree list in Figure 4.17.

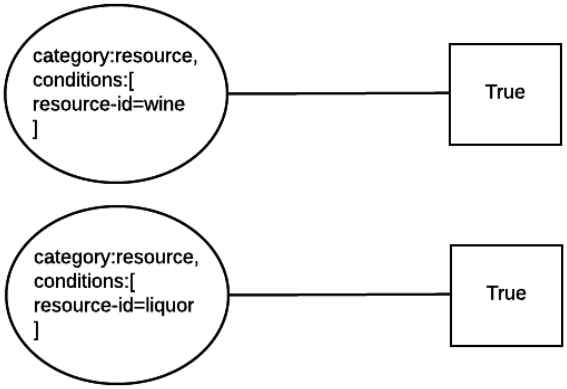

Figure 4.14 – Predicate tree list of *AnyOf*$_{1.}$

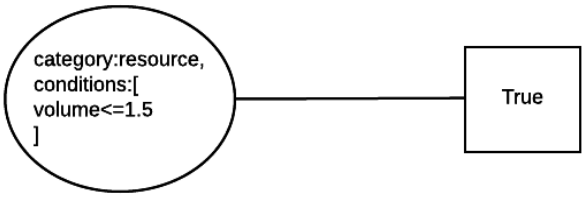

Figure 4.15 – Predicate Tree List of *AnyOf*$_{2.}$

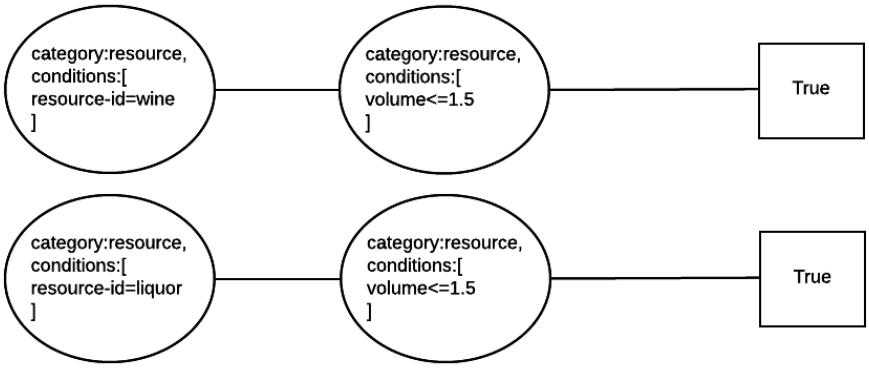

Figure 4.16 – Predicate tree list of *AnyOf*$_{1}$.*AnyOf*$_{2.}$

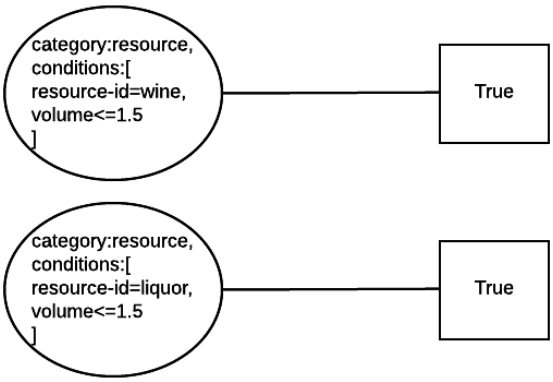

Figure 4.17 – Predicate tree list after reduction.

Let us analyze the cost of reducing the predicate tree. For n number of internal nodes (without the decision node) in the tree, cost of time T_{sum} of reduction can be stated as:

$$T_{sum} \leq n + (n-1) + (n-2) + \cdots + 1 \tag{4.6}$$

$$T_{sum} \leq \sum_{i=1}^{n} i$$

$$T_{sum} \leq n(n+1)/2$$

As the maximum number of internal nodes in the merged predicate can be only six, cost of time, T_{sum} of reduction will always be ≤ 21. Now, suppose $AnyOf_j$ expression includes k_j number of predicate trees. Thus, merging predicate trees of $AnyOf_j$ and $AnyOf_{j+1}$ includes $k_j\, k_{j+1}$ iterations. Therefore, cost of time, T_{sum} of merging $AnyOf_j$ and $AnyOf_{j+1} \leq k_j\, k_{j+1}(21)$.

For m *AnyOf* expressions, cost of time T_{sum} of merge operation can be stated as:

$$T_{sum} \leq \big(k_1 k_2(21)\big) k_3(21) \ldots k_m(21) \tag{4.7}$$

$$T_{sum} \leq (21)^{m-1}(k_1 k_2 \ldots k_m)$$

$$T_{sum} \leq (21)^{m-1} \prod_{i=1}^{m} k_i$$

Therefore, the complexity of parsing the *Target* expression is,

$O((21)^{m-1} \prod_{i=1}^{m} k_i)$

Step 2: Similarly construct the set of predicate trees that defines the *Conditions* expression.

Parsing rule *Conditions* is skipped in this research now, because the present implementation supports only two argument functions. However, *Conditions* can be parsed similarly and a set of predicate trees can be constructed. This requires

supporting different types of functions and merging them together to construct Boolean expressions.

Step 3: Merge rule *Target* with rule *Conditions* with *AND* operator and derive the set of predicate trees

Predicate trees derived by parsing *Target* expression can be merged with predicate trees derived by parsing *Conditions* expression, using the same algorithm used to merge *AnyOf* expressions in *Target*, which is explained under Step 1, item 6.

Step 4: For each rule based on the rule effect change the decision of each tree to *Permit* or *Deny*

For example, considering the predicate tree list derived for *WineLiquorAllowanceForForeigners* rule defined in the hypothetical policy in Figure 4.1, they can be converted to predicate trees in Figure 4.18 considering the rules effect. For n predicate trees, complexity of this operation is $O(n)$.

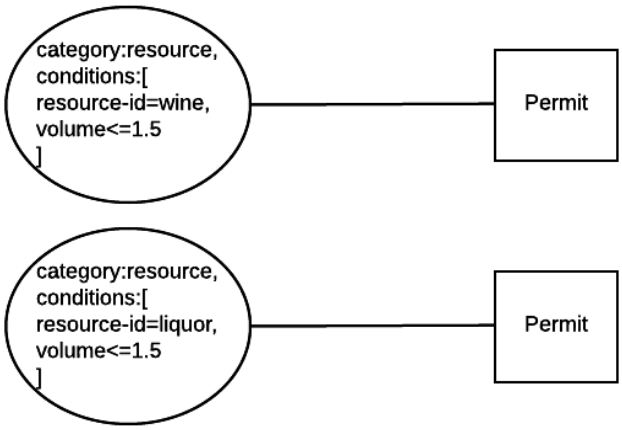

Figure 4.18 – Predicate tree list of WineLiquorAllowanceForForeigners rule.

Step 5: For each policy, parse policy *Target* and derive the list of predicate trees. Then merge with rule predicates with *AND* operator.

This operation is performed using the same algorithm used to merge *AnyOf* expressions in *Target*, which is explained under Step 1, item 6. For example, the predicate tree list derived for *DutyFreeAllowancesForForeigners* policy can be given as in Figure 4.19.

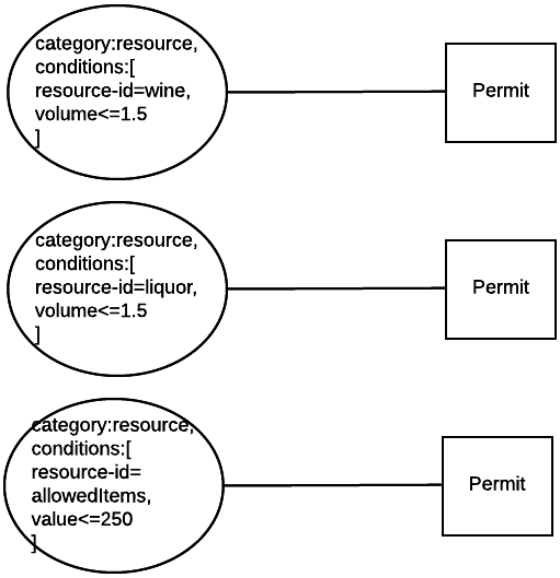

Figure 4.19 – Predicate tree list of DutyFreeAllowancesForForeigners policy.

Step 6: Perform Step 5 for each Policy Set.

Step7: Build the Subject-Object Relationship graph merging all predicate trees.

Following steps are used to build the subject-object relationship graph.

1. For each predicate tree,
    a. Create a vertex for subject, equivalent to the subject category node.
    b. Create a vertex for resource, equivalent to the resource category node.
    c. Create an edge between the subject and the resource node, in which the decision is the value of the decision node, and add all the conditions of the link node to the edge created, to denote the subject-resource relationship conditions that should be satisfied to meet the decision.
2. For each such relation,
    a. Add the subject node to the graph if no matching node found. Matching the node includes,
        - Checking if both are of same category

- Checking if both includes same set of conditions.

  If i^{th} node has k_i conditions and j^{th} node has k_j conditions, this includes $k_i + k_j$ iterations.

b. If a matching subject node is found, add the edges of the node to the matching node.

c. Add the resource node to the graph if no matching node found.

d. If a matching resource node is found, add the edges of the node to the matching node.

For example, Subject-Object Relationship graph for *DutyFreeAllowancesForForeigners* policy, in the hypothetical policy listed in Figure 4.1, will be as in Figure 4.20. Thus, considering the full hypothetical XACML policy listed in Figure 4.1, Subject-Object Relationship graph will be constructed as in Figure 4.21.

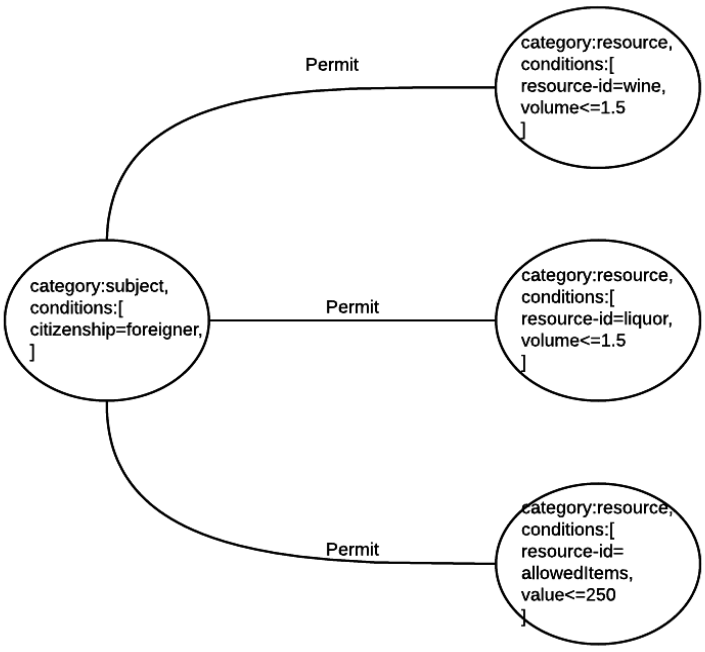

Figure 4.20 – Subject-Object relationship graph of DutyFreeAallowancesForForeigners policy.

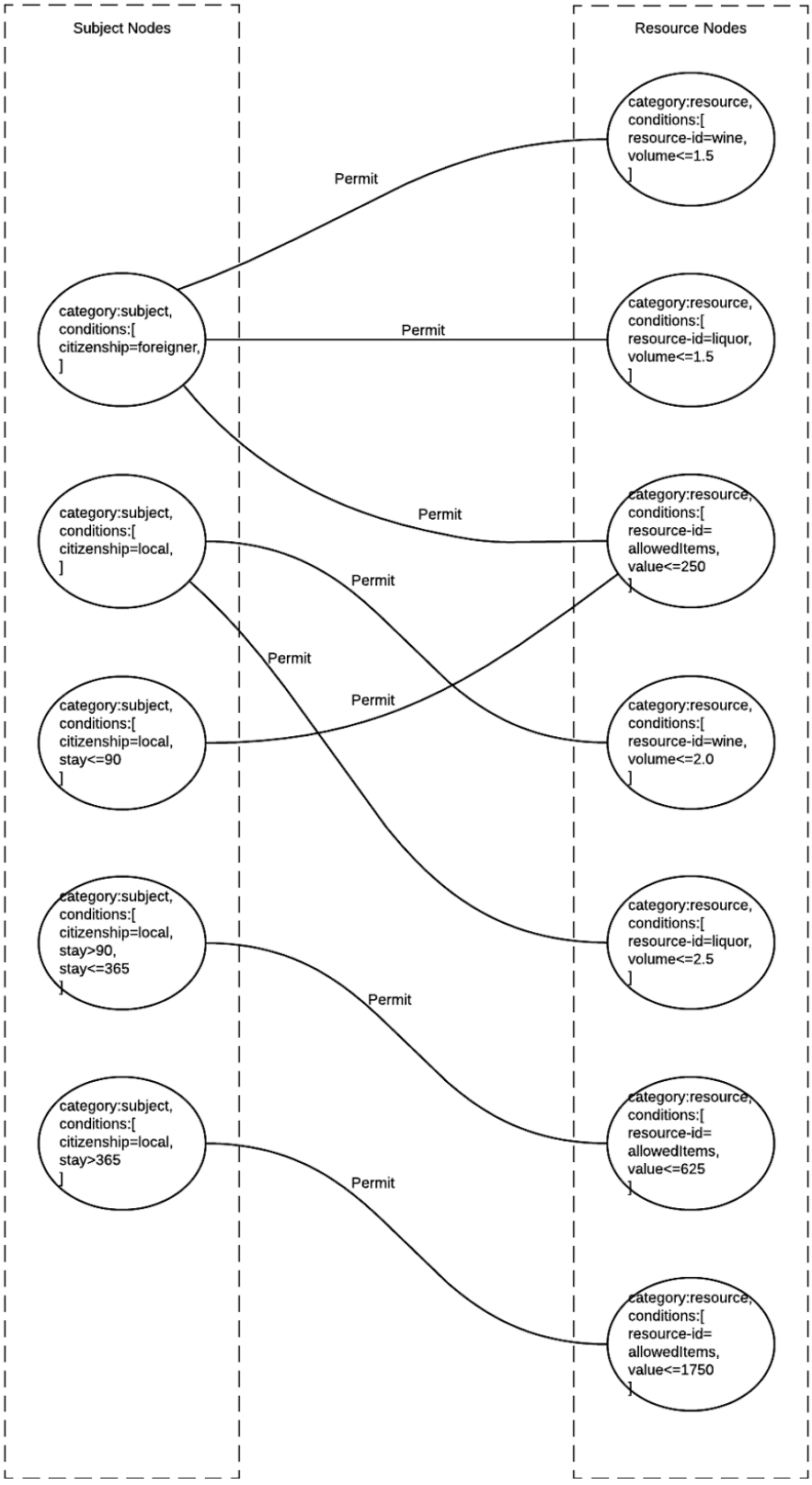

Figure 4.21 – Subject-Object relationship graph of the hypothetical policy example.

4.2 Graph Update

The Subject-Object Relationship graph constructed from the XACML policy tree in Section 4.1, will need to reflect certain updates to the XACML policy tree as well. We can categorize such updates as follows:

1. Adding a new *PolicySet.*
2. Adding a new *Policy.*
3. Adding a new *Rule* to a *Policy.*
4. Updating an existing *PolicySet Target*, *Policy Target*, *Rule Target* or a *Rule Condition*.

To accommodate updates related to the first three cases above, the corresponding predicate trees need to be constructed for the newly added *PolicySet*, *Policy* or *Rule*. Later those needs to be merged with the existing graph. For a newly introduced *PolicySet*, predicate trees can be derived by performing Step 1 to Step 6 (see Section 4.1). Then merge the derived set of predicate trees with the existing graph by performing Step 7. Similarly, adding a new *Policy* includes deriving the set of predicate trees, by performing Step 1 to Step 5 (see Section 4.1), and merging them with the existing graph by performing Step 7. Adding a new *Rule* includes deriving the set of predicate trees of that *Rule*, by performing Step 1 to Step 4 (Section 4.1), then merging them with the parent *Policy Target* of that *Rule*, by performing Step 5, and finally merging them with the existing graph by performing Step 7.

Yet, accommodating updates to the graph for the fourth case above is challenging. This is because, Boolean predicates defined from *Targets* and *Conditions* in the XACML Policy Tree are already combined with a specific set of nodes and edges in the graph. Updates to *Targets* and *Conditions* will change the way such predicates are combined. Therefore, it is not possible to simply isolate the predicates that changed, and update them, because that does not reflect the changes to predicate combination. One option is the reconstruction of the graph. However, effective accommodation of such updates can be researched more, which we leave as future work.

4.3 Access Review Request Evaluation

With the Subject-Object Relationship graph constructed, an access review request will be performed as follows. An overview of this algorithm is represented in Figure 4.24.

1. Extract subject category attributes from the set of received attributes to a List.
2. Extract resource category attributes from the set of received attributes to a List.
3. Add all attributes received for a separate List.
4. If subject category attributes are available, find the matching set of subject nodes from the subject node list. In this case evaluating a node includes,
    a. Requesting the missing subject attribute values that are being used in the conditions of the node, from PIP, providing the received set of subject attributes.
    b. For all available subject attribute values now available, evaluate each condition in the node.

       In evaluation, there can be conditions that cannot be evaluated due to missing attribute value(s). That means the respective condition should satisfy to pick up the respective node as a match. From access review perspective, such conditions need to be communicated in the response, which means such conditions are evaluated as true.

       If any other condition evaluates to false, that means the respective node is not a match for the received set of attributes. Else if, all other conditions evaluated to true, the node is picked as a matching node.
5. Extract the set of matching edges from the matching set of subject nodes, against all attributes received. Same algorithm used to evaluate subject nodes in Step 4, is used to find matching edges as well.
6. If resource category attributes are also available, for each edge filtered out, check if the resource node that the edge points is a match against the set of resource category attributes. Same algorithm used to evaluate subject nodes in Step 4, is

used to find matching resource nodes as well. If the resource node that the edge points is a match, filter out that edge as a successful match to the access review request received. If no resource category attributes are available, the set of edges filtered out in will be the paths matching the access review request.

7. For each edge extracted, the subject and resource node that the edge points to denote the subject and resource conditions that should be satisfied and the conditions in the edges denotes the action, environment and subject to resource linking conditions that should be satisfied. Thus, the result of the access review request is the extracted set of edges. Therefore, build the response from the set of edges extracted and return.

8. If no subject category attributes are available, check if resource category attributes are available in the access review request. If so, find the matching set of resource nodes from the resource node list, performing the algorithm in Step 4, against the resource category attributes.

9. Extract the set of matching edges from the matching set of resource nodes, against all attributes received. Same algorithm used to evaluate subject nodes in Step 4, is used to find matching edges as well. The extracted set of edges is the result of the access review request. Therefore, build the response from the set of edges extracted and return.

Let us consider the same hypothetical access control policy defined in Section 2.4.1. Suppose we want to know '*What are the items that Malithi can bring in?*', given the fact, that *Malithi* is a resident, and she has travelled for United States for two weeks to attend to a business conference. Such as access review request initiated with XACML policy language, will be like in Figure 4.22. To improve readability namespaces of functions, attributes, attribute categories and data types are removed.

```
<Request>
    <Attributes Category="subject">
        <Attribute AttributeId="subject-id">
            <AttributeValue DataType="string">
                Malithi
            </AttributeValue>
        </Attribute>
        <Attribute AttributeId="stay">
            <AttributeValue DataType="integer">
                14
            </AttributeValue>
        </Attribute>
    </Attributes>
</Request>
```

Figure 4.22 – Access review request.

Now, let us apply the algorithm defined above to this access review request. Then query evaluation is as follows:

1. Performing Step 1, we can identify that this request includes the two subject attributes where,

   subject.subject-id = Malithi

   subject.stay=14

2. Performing Step 2, we can identify that there are no resource category attributes.

3. Performing Step 3, will extract the two subject attributes that we already identified.

4. Performing Step 4, will request the *citizenship* attribute from the PIP, and the PIP will return the value as *local.* Now, the available set of attributes of this subject is as below.

   subject.subject-id = Malithi

   subject.stay=14

   subject.citizenship=local

5. Now, matching above attributes against each subject node, in the subject node list of the graph in Figure 4.21, will pick subject to resource links in Figure 4.23 after performing Step 4 and Step 5 respectively.

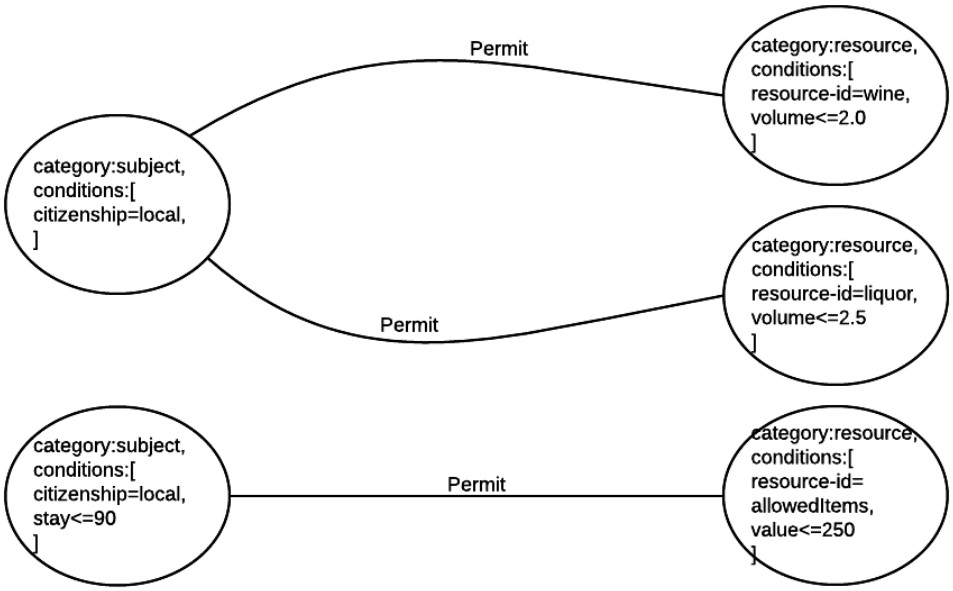

Figure 4.23 – Links extracted for the access review request.

6. Step 6 will be skipped as no resource category attributes are received. Thus, the result of the access review request is the subject to resource links extracted as in Figure 4.23, which will be returned in Step 7.

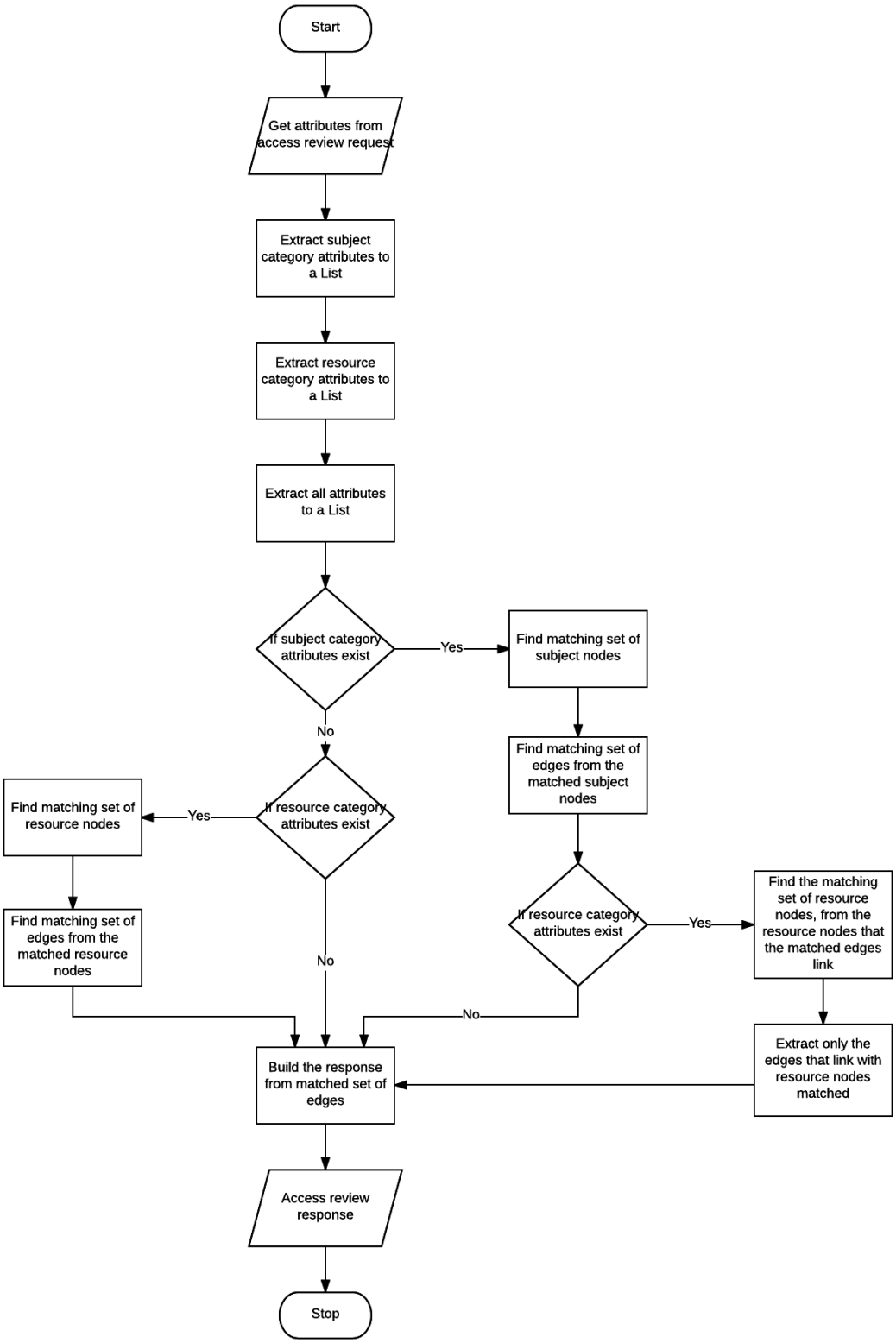

Figure 4.24 – Overview of access review request evaluation.

5 PERFORMANCE ANALYSIS

In this chapter, we discuss the evaluation experiments conducted on the implementation of the solution proposed. In our experiments, we compare our implementation with the SNE-XACML engine [23], which implements XACML policy evaluation with MIDDs [5]. Originally, SNE-XACML engine supports only XACML access control requests. Therefore, SNE-XACML engine implementation was improved to support access review requests using depth first graph traversal technique, to use it with access review query evaluation experiments.

5.1 Environment and datasets

We use two of the three datasets used in [5] and those are summarized in Table 5.1. The first dataset is a real-world policy taken from GEYSERS project [24]. The policy set from this project applies access control to a logical network infrastructure based on network roles. There are three network roles defined in the policy, each allowed to perform a defined set of operations on a defined set of logical network infrastructure resources. The second dataset applies access control to a web-based application that supports submission, review, discussion, and notification phases of conferences based on roles [25]. This dataset is converted to from XACML 2.0 to a XACML 3.0 policy with appropriate attribute categories in place.

We implement the proposed solution in JRE 1.8. Experiments are carried on an OS X 10.11.6 system with Intel Core i7 (I7-4750HQ) quad-core 2GHz processor and 8 GB DDR3 RAM.

Table 5.1 – XACML 3.0 sample policy datasets.

ID	Datasets	No of Policy Levels	No of Policy Sets	No of Policies	No of Rules	No of Attributes
1	GEYSERS	3	6	7	33	3
2	Continue-a	6	111	266	298	14

5.2 Graph Construction Evaluation

Figure 5.1 shows the average time taken to construct the subject-object relationship graph by parsing the policy set. Average construction time was computed by processing the graph 10 times for each dataset. In Figure 5.1 the proposed solution is denoted as *Subject-Object Relationship Model* and the MIDD approach is denoted from *MIDD*. Here, we observe that our solution is 100% faster in graph construction for the *GEYSERS* dataset and 550% faster for the *Continue-a* dataset than the MIDD approach, which is highly dependent on the number of attributes n and number of attribute values k.

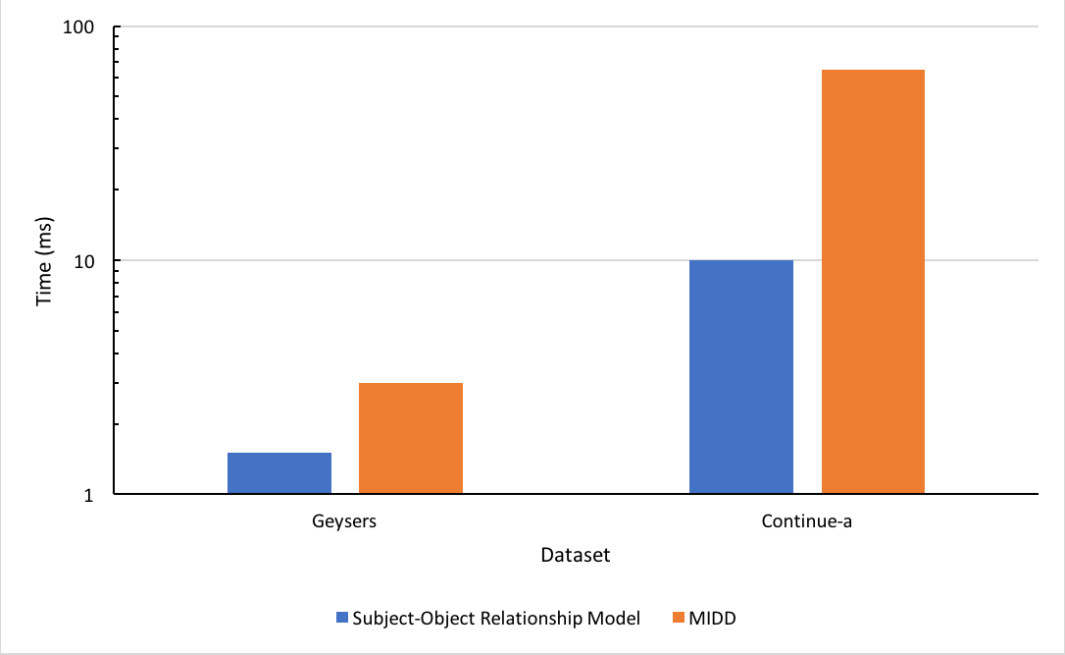

Figure 5.1 – Average graph construction time.

Figure 5.2 shows the growth of the subject-object relationship graph by computing the number of nodes that will reside in the graph. Here, we observe that the proposed solution does not include a large number of nodes in the graph with respect to the complexity of the policy. The graph has grown only by 38.1% though the policy set of *Continue-a* dataset has a higher number of conditions, attributes, and attribute values compared to the *GEYSERS* dataset. This is because the proposed solution maintains a separate node only for each unique subject type and resource type defined in the policy. In Equation 3.1, we derived that the number of nodes of the subject-object relationship

graph, is the summation of uniquely identified subject sets and resource sets. In *GEYSERS* dataset, only a one subject attribute and a resource attribute is used to identify subjects and resources. The subject attribute has three distinct values, which used to define three subject sets and the resource attribute has 18 distinct values, which used to define 18 resource sets. Therefore, the total number of nodes in the subject-object relationship graph is 21, which is the same we received by constructing the graph for the *GEYSERS* dataset. Similarly, *Continue-a* dataset also has one subject attribute and resource attribute used to identify subjects and resources. The subject attribute has four distinct values, which defined four subject sets and the resource attribute has 25 distinct values, which defined 25 resource sets. This results 29 nodes in the graph for *Continue-a* dataset, which is the same resulted from graph construction. However, in Figure 5.2, graph of *Continue-a* dataset for MIDD approach, is 50 times larger compared to graph of *GEYSERS* dataset. This is because the graph size is proportional to the number of attributes n and number of attribute values k in the policy as derived in Equation 3.6. GEYSERS dataset has three string type attributes defined, where each has 3, 18 and 30 distinct attribute values defined respectively. Because, the attribute is of string type, which is not a continuous data type, and string equal function is used to define conditions, we can conclude that each attribute has 3, 18 and 30 distinct attribute value intervals, which is less than the worst-case attribute value intervals derived in Equation 3.6. If we calculate the total number of nodes for these numbers using Equation 3.6, starting from the same order, we get that the graph has 1677 number of nodes in the worst case. But, constructing the graph from the implementation resulted only 55 number of nodes. This is because, each target condition defined in the XACML policy does not use all attributes as its variables. Therefore, the MIDD graph, will not have edges from one level to the next level for each distinct attribute value interval that attribute can take. Also, if we traverse over a path from the root node to a decision node, that path may not visit nodes defined for each attribute in the policy. Yet, the graph has grown with respect to the number of attributes and number of attribute values in the policy. Comparing the graph size of the proposed solution and the MIDD approach for *GYSERS* dataset we can observe that proposed solution is 61.8% memory efficient than the MIDD approach. With

respect to the *Continue-a* dataset proposed solution is 99.1% memory efficient than the MIDD approach.

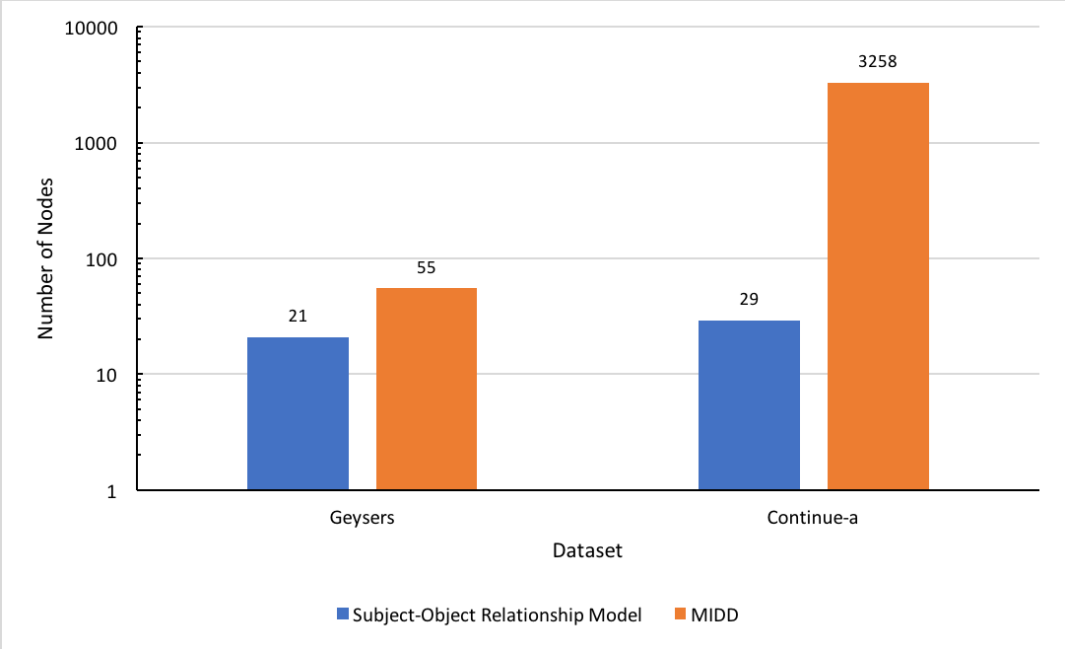

Figure 5.2 – Number of nodes in the graph.

Next, we analyze the average graph construction time while varying the number of policy sets in the second level. For this we choose *Continue-a* dataset as it spans up to six levels, includes 25 policy sets in the second level, and has a high policy complexity than the *GEYSERS* dataset. As seen in Figure 5.3 number of policy sets is varied from five to 25 in increments of five policy sets in the second level. Average graph construction time was calculated by processing the graph 10 times by randomly selecting the expected number of policies in the second level during each iteration. Then, using the same policy set used for computation in Figure 5.3, average number of nodes is analyzed while varying the number of policy sets in the second level of *Continue-a* dataset as shown in Figure 5.4. While the graph construction time of both the solutions increase with increasing number of second-level policies, rate of increase for MIDD is about 20 times greater than the graph construction time taken for the proposed solution (see Figure 5.3). Similarly, as seen in Figure 5.4 graph size grows with the number of second-level policies.

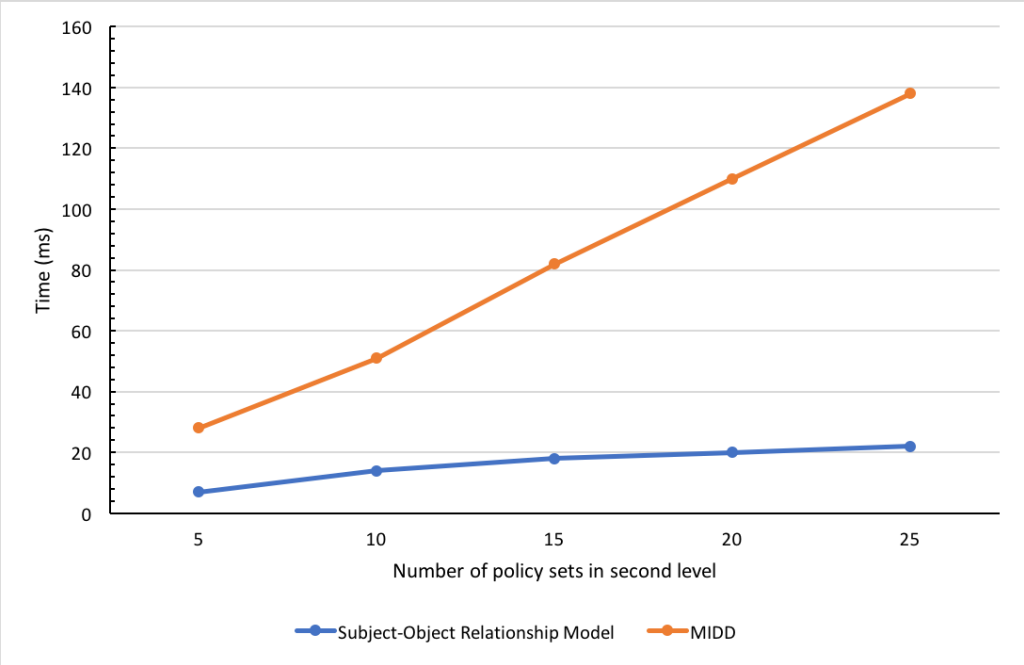

Figure 5.3 – Average graph construction time for varied number of policy sets.

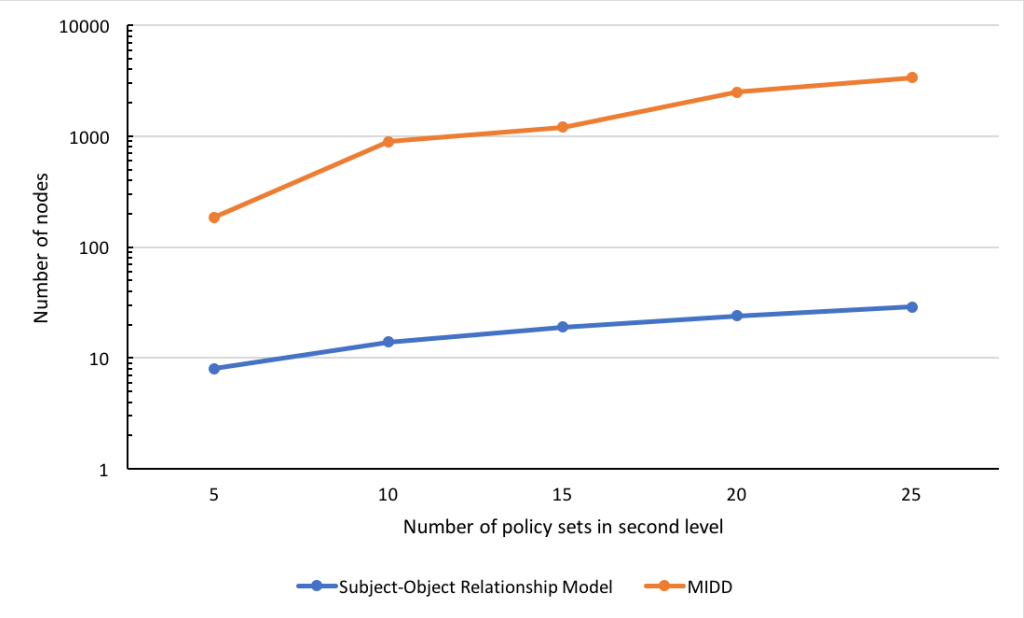

Figure 5.4 – Average number of nodes for varied number of policy sets.

However, graph size of the MIDD approach grows at a rate that is about 150 times greater than the proposed solution. This pattern is observant in both graph construction time and the graph size because MIDD approach depends on the number of attributes and attribute values in the policy. Whereas the proposed approach is dependent only

on parsing the newly introduced policy constructs and extracting the conditions. Thus, the graph of the proposed solution will grow only with the addition of a new subject set or a resource set. Therefore, we can conclude that the proposed solution is faster in graph construction and uses less memory than the MIDD approach. Thus, the proposed solution is more scalable over the MIDD approach.

5.3 Query Resolution

Access review requests were performed for a set of query types as listed in Table 5.2. For each query type, 100 random queries were generated and evaluated 10 times against the proposed solution (*Subject-Object Relationship Model*) and the MIDD (*MIDD*) approach. XACML policy engines supports access requests only. They result only *Permit* or *Deny* for an access request. Therefore, to perform an access review request with XACML for each subject, resource, action, or environment condition available in the system, access requests should be generated. This requires to access attribute sources of the system, such that for each subject or resource their attributes can be retrieved. The evaluation performance depends how many subjects and resources are available in the system. However, both solutions considered for this analysis does not require to plug attribute sources as they return the possible set of conditions. Given the respective attribute values that exists in the policy, both these solutions can successfully return the set of conditions. Thus, as the outputs and inputs deviates, performing an access review request against an XACML policy engine that supports only XACML access control requests by evaluating the XACML policy tree, is not considered for this analysis.

Table 5.2 – Access review request query types.

Query Type	Description
subject-only	Access review requests with subject attributes only
resource-only	Access review requests with resource attributes only
subject-resource	Access review requests with subject and resource attributes
subject-resource-link	Access review requests with subject, resource, action or environment attributes

Figure 5.5 shows the average query resolution time for the four query types on Table 5.2 with *GEYSERS* dataset. Figure 5.6 shows the standard deviation of query resolution time for the four query types with *GEYSERS* dataset. In Figure 5.5 we can observe that MIDD approach has less average query resolution times than the proposed approach for subject-only, resource-only, and subject-resource-link query types, which is 29.7%, 7.7% and 41.7% lesser than the solution proposed. *GEYSERS* is a small policy set with less number of attributes. Therefore, graph size of the MIDD approach is small. Moreover, the query resolution time of the MIDD approach depends on the number of nodes and edges visited from the depth first graph traversal for an access review request as we derived in Equation 3.11. In the proposed solution, the received set of attributes will be evaluated against the conditions in each subject node or resource node in the graph, based on the attribute category received as derived in Equation 3.9. Moreover, the function evaluation will also happen in the query evaluation time while evaluating conditions. Therefore, for a small graph size, MIDD approach will perform well in query evaluation when compared with the proposed model, as we observe in Figure 5.5. Further, we can observe that both models have high query resolution times for *subject-only* query type where only subject attributes are received. *GEYSERS* policy set has only one subject attribute defined with three distinct values. Also, there is only one resource attribute and action attribute defined but there are number of conditions defined with respect to resource attribute and action attribute. Thus, there are number of distinct attribute values for resource and action attributes as well. In that case, in MIDD approach having only the subject attribute will cause to traverse a larger part of the graph. Further, if the subject attribute is used only in a lower level of the graph while traversing from the root of the graph many unnecessary nodes and edges will be visited. This is visible from the average query resolution time incurred for the *resource-only* query type for MIDD approach. Resolution time for *subject-only* query type is very low when compared with the resolution time for *resource-only* query type, which means from the root level respective path is selected and the graph is visited optimally. In the proposed model average query resolution time for subject-only query type is high because after selecting the respective subject node by evaluating each subject node in the graph for the received attribute values, each edge and resource node should be traversed to

extract the conditions to be satisfied. Now as the conditions should be evaluated this requires more time than the MIDD approach. For other query types, we cannot observe large differences in resolution times between the two approaches. However, we can observe that for *subject-resource* query type the proposed model has performed 80% faster than the MIDD approach. This could be because MIDD approach is highly dependent on the attribute ordering.

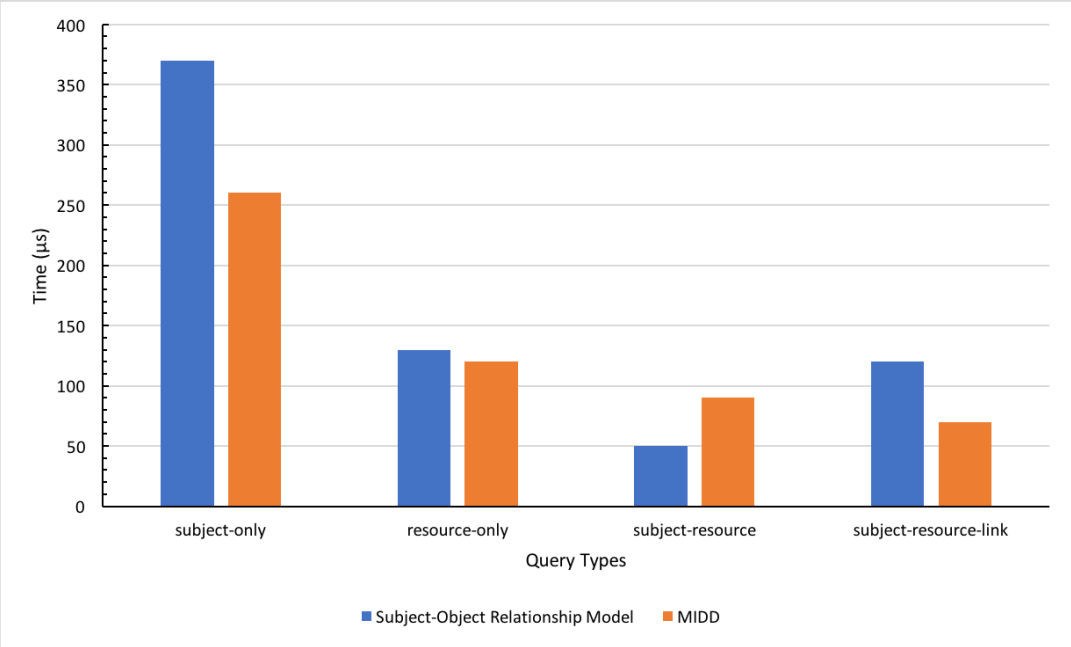

Figure 5.5 – Average query resolution times for *GEYSERS* dataset.

From Figure 5.6 we can observe that the proposed solution has larger standard deviation for *subject-only* query type. The minimum query resolution time that it took was 0 µs and maximum query resolution time was 8000 µs. The MIDD approach took only 6000 µs in maximum while minimum was 0 µs. That means a specific subject node should be linked with a smaller set of resource nodes while others link with a larger set of resource nodes or vice versa. This is clear because the MIDD approach also has a larger standard deviation for *subject-only* query type comparatively to other query types.

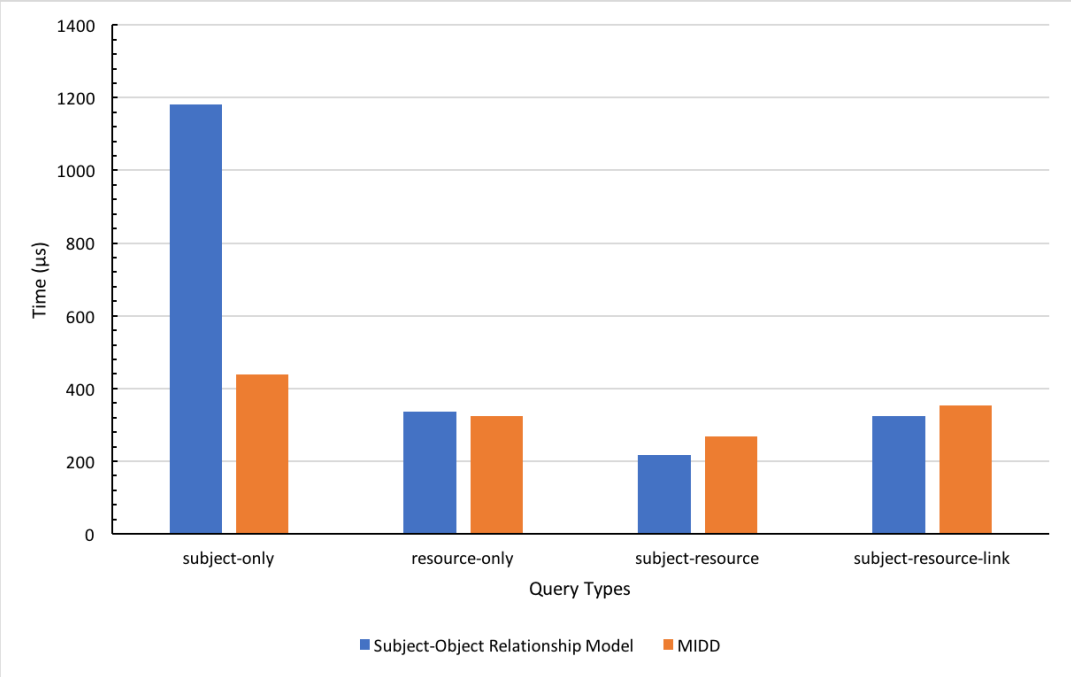

Figure 5.6 – Standard deviation of query resolution time for *GEYSERS* dataset.

Figure 5.7 and 5.8 show the average query resolution time and standard deviation for the four query types with the *Continue-a* dataset. *Continue-a* policy set is complex than the *GEYSERS* and includes a larger condition set with more attributes present in the policy. Therefore, the graph constructed in MIDD approach for this dataset is large, because it is dependent on the number of attributes and distinct attribute values. However, the graph size of the proposed solution grows only with the subject sets and resource sets that can be uniquely identified in the policy as we derived in Equation 3.1. Similar to the *GEYSERS* policy set *Continue-a* policy set also includes one subject attribute with four distinct values defined. However, there are many action and environment attributes defined in the policy. As such conditions define the relationship among the subject and the resource, the graph of the proposed solution will not grow similar to the graph of the MIDD approach. As we derived in Equation 3.11, time complexity of MIDD approach depends upon the number of nodes and the number of edges in the graph, which depends on the number of attributes and distinct attribute value intervals each attribute will take, as derived in Equation 3.6 and 3.7. Thus, for *Continue-a* policy set graph size of the MIDD approach grows with respect to all attribute types, subject, resource, action and environment, and their distinct attribute values. This impacts on its query resolution time. From Figure 5.7 we can clearly

observe for a complex and large policy set the proposed solution outperforms the MIDD approach in query evaluation. Moreover, from Figure 5.7 we can see that the MIDD approach has a higher query resolution time when compared to the proposed approach for *subject-only* query type. Here, MIDD approach is 90.1% slower than the proposed model. For the four query types considered the proposed solution performs 33.9% faster in query evaluation than the MIDD approach in average.

From Figure 5.8 we can see that the MIDD approach has a large standard deviation for the *subject-only* query type. This is different from the results obtained for subject-only query type for *GEYSERS* dataset for both the solutions. In this case, we can observe this deviation only for the MIDD approach. Thus, we can conclude that this is caused not due to the different complexities among subject and resource relationships, but due to the attribute ordering, which highly impacts the performance of the MIDD approach.

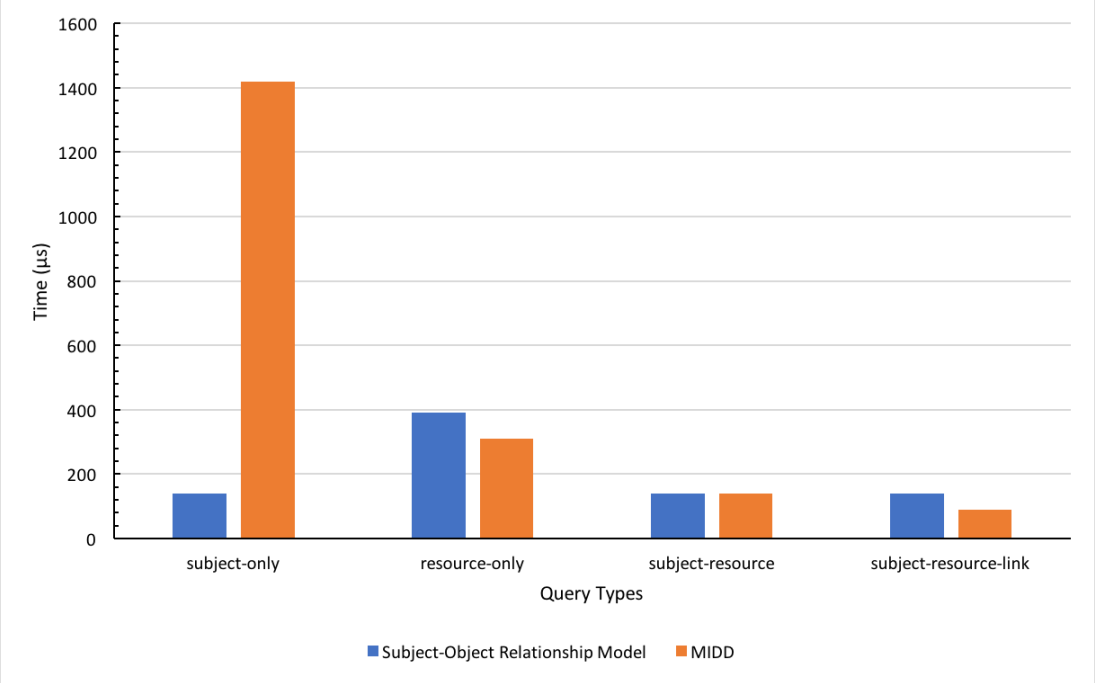

Figure 5.7 – Average query resolution times for *Continue-a* dataset.

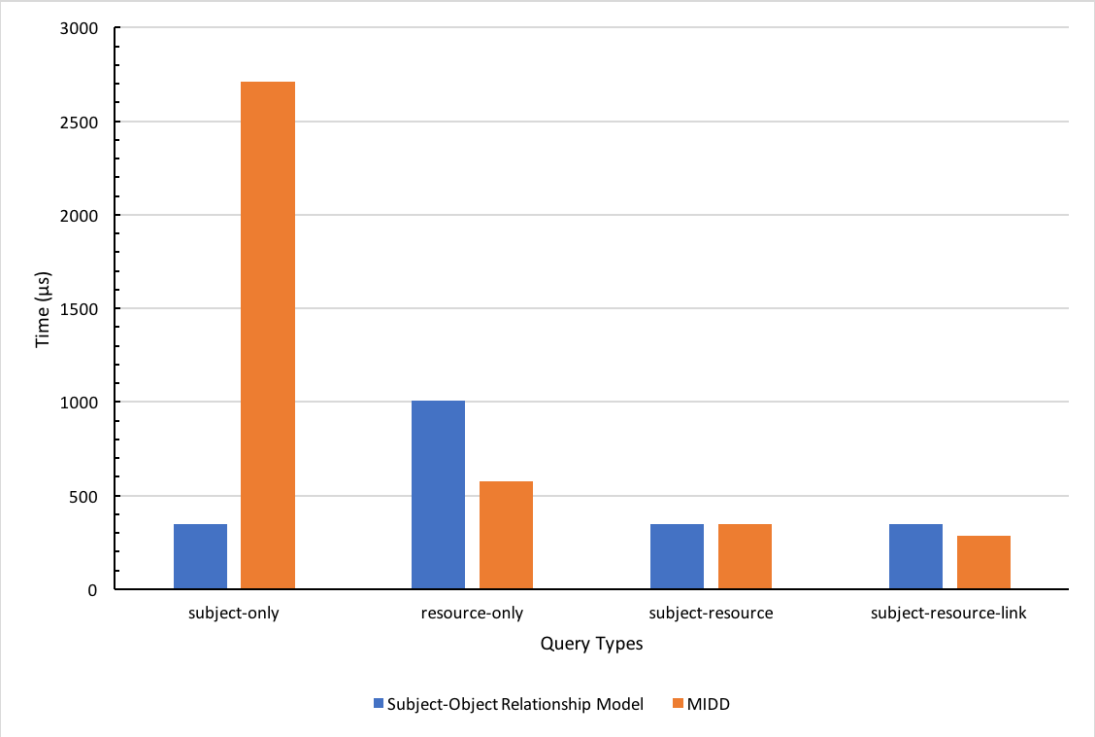

Figure 5.8 – Standard deviation of query resolution time for *Continue-a* dataset.

In Figure 5.9, we analyze the average query resolution time while varying the number of policy sets in the second level. For this we choose *Continue-a* dataset as it spans up to six levels, includes 25 policy sets in the second level, and has a high policy complexity than the *GEYSERS* dataset. As seen in Figure 5.9 number of policy sets is varied from five to 25 in increments of five policy sets in the second level. Average query resolution time was calculated by generating 100 random queries for each query type in Table 5.2, against a subject-object relationship graph and MIDD graph constructed 10 times by randomly selecting the expected number of policies in the second level in each iteration. *Continue-a* policy set includes one subject attribute with four distinct values defined in each second level policy set. Also, each second level policy set is defined for a distinct resource which is uniquely identified from a resource attribute. However, there are many action and environment attributes defined, that varies with the number of second level policy sets used. Therefore, even though the number of second level policy sets is varied, subject attributes and subject attribute values defined in the policy set is not varied. Rather, only resource, action and environment conditions are varied. Thus, with varied number of policy sets in second

level graph size of the MIDD will grow as it is dependent on the number of attributes and distinct attribute values.

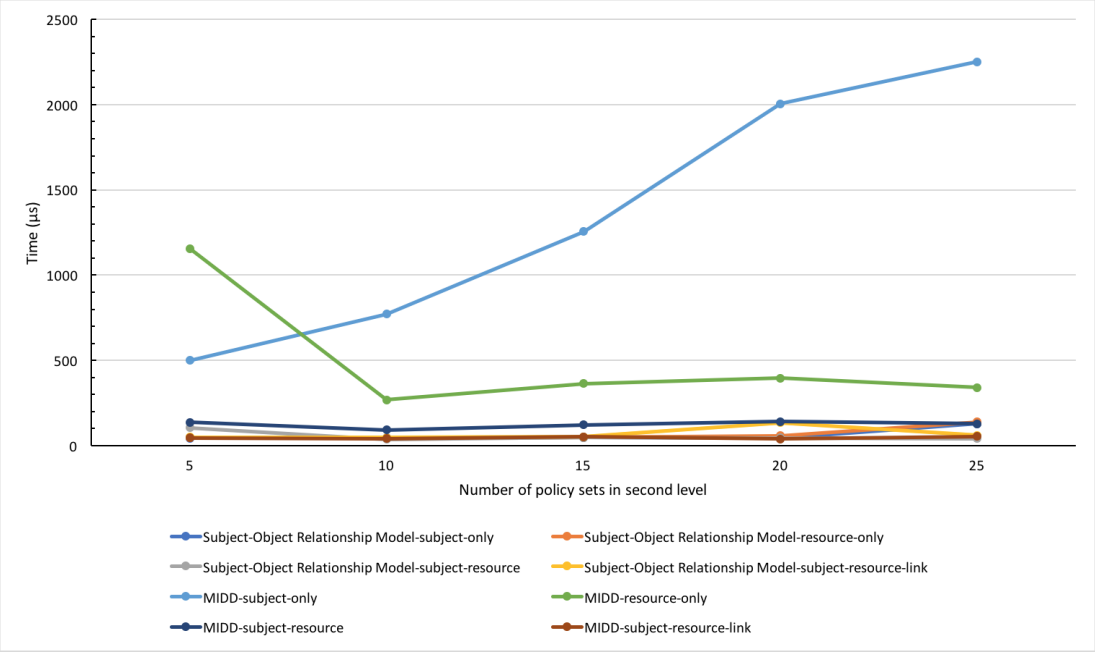

Figure 5.9 – Average query resolution time for varied number of policy sets.

However, the graph of the proposed solution will not grow similar to the graph of the MIDD approach, as the graph size of the proposed solution grows only with the subject sets and resource sets that can be uniquely identified in the policy, and action and environment conditions define the relationship among the subjects and the resources. Due to this reason, we cannot observe variations of average query resolution time of the proposed solution for each query type with the varied number of policy sets in Figure 5.9. Thus, we can conclude that this is caused not due to different complexities among subject and resource relationships, but due to attribute ordering, which highly impacts the performance of the MIDD approach. For MIDD approach, query resolution time of subject-only queries is higher and grows heavily with the varied number of policy sets. This is caused as the graph size of MIDD grow with respect to the attributes and attribute values introduced with the increasing number of policies, and attribute ordering. Also, for resource-only queries MIDD approach has higher query resolution times, but when more attributes are included in the access review request with subject-resource and subject-resource-link query types, MIDD approach

has lower query resolution times. Therefore, with these observations, we can conclude that the query resolution of MIDD approach is heavily dependent on attribute ordering.

6 SUMMARY AND FUTURE WORK

6.1 Summary

Traditional access control models such as RBAC and ACL have a precomputed access matrix with capabilities directly assigned to subjects, roles, and groups before an access request is made. However, ABAC never explicitly constructs this access control matrix. Rather it relies on an access control policy that implicitly defines the access matrix. Therefore, it is difficult to solve access review queries like "*Which objects does this user have access to?*" and "*What actions can this user perform on those objects?*" with ABAC.

This research contributes towards supporting access review queries with ABAC model, which is the biggest drawback to use ABAC with access control administration, auditing and reviewing. In this research, we analyzed two approaches that have been proposed to solve the problem of performing access reviews in XACML, which is a popular ABAC standard. The first approach performs partial evaluation of policies using the usual policy evaluation [4]. This approach takes more time, as each policy needs to be evaluated and simplified, against a subset of attributes that needed to evaluate a policy, to extract the applicable set of conditions. The second approach [5], construct a decision graph that represent XACML policy sets which is named as Multi Interval Decision Diagram (MIDD). Due to pre-constructed representation, this approach reduces evaluation complexity, but incurs scalability problems as the graph grows with respect to the number of variables in the policy set and the distinct value intervals that each variable could take. Analyzing the computational costs, limitations in each approach and the XACML policy representation, we proposed a graph-based model that can represent the permission relationships between subjects and objects. The proposed solution parses and transforms complex logical expressions in XACML policies into a subject-object relationship graph by extracting conditions starting from the lower levels of a XACML policy tree and constructing condition paths from them. Therefore, that can isolate matching subjects or objects, for a given access review request to extract applicable set of conditions efficiently.

Theoretical comparison in Section 3.3 proved that the proposed solution is more scalable over partial query evaluation and MIDD approaches and has less space complexity than those approaches. With respect to time complexity, the proposed solution is not much affected by the number of attributes or number of conditions introduced to the access control system, while the other two are highly dependent on policy complexity, number of attributes and attribute values in the policy. Therefore, irrespective of the complexity of the XACML policy tree the proposed model can perform well. In the performance analysis conducted in Chapter 5 against the MIDD approach, it was seen that the proposed solution is 100% faster in graph construction than the MIDD approach for a small policy set and around 550% faster for a large policy set. The proposed solution was 61.8% to 99.1% memory efficient than the MIDD approach. This clearly proves the fact that the proposed solution is highly scalable, and computational and memory efficient in graph construction compared to the MIDD approach. Experiments conducted further illustrates that the solution efficiently performs query evaluation for larger and complex policy sets when compared with the MIDD approach which is dependent on the number of attributes, distinct attribute values in the policy set, and attribute ordering selected when constructing the decision graph in 33.9%. Therefore, we can conclude that the proposed solution has efficient evaluation performance in time complexity, highly scalable as it incurs less space complexity and less evaluation time complexity with respect to the complexity of the XACML policy set.

6.2 Limitations

There are few limitations in the proposed solution and its implementation. The present implementation parses only *Target* elements in the policy sets, policies, and rules to extract the conditions defined in the XACML policy set. However, in an XACML policy, rule *Conditions* also define access control conditions that should be met for the successful evaluation. This is because the present implementation only supports two argument functions such as equal, greater than, and less than. However, this can be improved to support complex functions with multiple arguments on different data types.

This research does not discuss on how to accommodate updates to the subject-object relationship graph when an existing *PolicySet Target*, *Policy Target*, *Rule Target*, or *Rule Condition* get updated. Accommodating such updates to the graph is challenging, because updates to *Targets* and *Conditions* will change the way Boolean predicates defined in the XACML policy tree are combined with the nodes and edges in the graph now. Therefore, the graph will need to be computed again to accommodate such changes. While this warrants future work, we believe such reconstruction is acceptable as polies change infrequently.

XACML allows multi-valued attribute requests, where an attribute can have a list of values. For example, a person can have several roles such as employee and manager. The present implementation of the proposed solution supports access review requests only with a single-valued attribute.

6.3 Future Work

This research can be improved to support complex functions with multiple arguments on different data types. Thus, support parsing rule *Conditions* to accommodate them in the subject-object relationship graph model. This is an improvement in the implementation level of the solution proposed.

Further, effective accommodation of updates to the subject-object relationship graph when updating an existing *PolicySet Target*, *Policy Target*, *Rule Target* or *Rule Condition* can be researched more in future, rather than processing the graph again for the complete policy set. In this case, subject-object relationship graph should be designed in a way such updates can be easily applied.

Proposed solution acts similar to the PDP defined in the XACML reference architecture. Rather than answering access control requests with a *Permit* or *Deny* decision, it answers access review requests with the set of conditions that should be met to satisfy the query. Yet, from the requestor perspective, this is not a complete access review request answer. For example, if the requestor asks "*What are the resources Alice can read?*" the answer, that the requestor expects is the set of resources that *Alice* will have access to, not the condition set that *Alice* should meet to access a

specific type of resource. However, given a XACML policy set, the request evaluation point can only extract the conditions from the policy set. Therefore, there should be some other module that can access parse the extracted set of conditions, access attribute sources in the system and answer with the specific set of subjects or objects. Thus, this solution can be improved in future by designing another such module, that can convert the condition set retrieved from the proposed solution to a set of data access queries, which will retrieve data from attribute data sources to respond with an exact answer for an access review query.

Moreover, the solution proposed in this research focus only on solving access review queries. However, as it constructs a precomputed data model, which represents the XACML policy tree, it can be improved to support access control requests as well. This will improve the access control query evaluation efficiently. Therefore, this research can be extended to represent XACML policy sets to support access control requests in future.

REFERENCES

[1] V. C. Hu, D. F. Ferraiolo and D. R. Kuhn, "Assessment of Access Control Systems," National Institute of Standards and Technology , Gaithersburg, 2006.

[2] V. C. Hu, D. R. Kuhn and D. F. Ferraiolo, "Attribute-Based Access Control," *Computer,* vol. 48, pp. 85-88, February 2015.

[3] R. Wagner, "Identity and Access Management 2020," *Information Systems Security Association Journal,* pp. 26-30, 2014.

[4] J. Sandberg, "Administrative Queries in XACML-feasibility of partial-query evaluation," 2006.

[5] C. Ngo, Y. Demchenko and C. d. Laat, "Decision Diagrams for XACML Policy Evaluation and Management," *Computers and Security,* vol. 49, pp. 1-16, 21 November 2014.

[6] National Computer Security Center, "Glossary of Computer Security Terms," 1988.

[7] D. Ferraiolo, D. Kuhn and R.Chandramouli, "Role-Based Access Control," *Computer Security Series,* 2003.

[8] National Computer Security Center (NCSC), "A Guide to Understanding Discretionary Access Control in Trusted System," 1987.

[9] C. P. Pfleeger, Security In Computing, 2nd Edition ed., Prentice-Hall, 1997.

[10] D. Bell and L. LaPadula, "Secure computer system: Unified exposition and multics interpretation," March 1976.

[11] K. Biba, "Integrity considerations for secure computer systems," 1977.

[12] V. C. Hu, D. Ferraiolo and R. Kuhn, "Guide to Attribute Based Access Control (ABAC) Definition and Considerations," National Institute of Standards and Technology , Gaithersburg, 2014.

[13] A. Cavoukian, M. Chibba, G. Williamson and A. Ferguson, "The Importance of ABAC: Attribute-Based Access Control to Big Data: Privacy and Context," 2015.

[14] Axiomatics AB, "Axiomatics Policy Server 6.0 White Paper," 2015.

[15] OASIS, "OASIS eXtensible Access Control Markup Language (XACML) TC | OASIS," 2016. [Online]. Available: https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml.

[16] "XML Access Control Language (XACL)," 2001. [Online]. Available: http://xml.coverpages.org/xacl.html.

[17] OASIS, "eXtensible Access Control Markup Language (XACML) Version 3.0," 2013. [Online]. Available: http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html.

[18] OASIS, "eXtensible Access Control Markup Language (XACML) Version 2.0," 2005. [Online]. Available: http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf.

[19] Axiomatics, "System and method for performing partial evaluation in order to construct a simplified policy". US Patent US20140317685 A1, 23 Oct 2014.

[20] Axiomatics, "System and method for evaluating a reverse query". Patent US9223992 B2, 29 December 2015.

[21] S. Kumar, "Blimey! What's Axiomatics Reverse Query?," 2015. [Online]. Available: http://www.axiomatics.com/blog/entry/blimey-what-s-axiomatics-reverse-query.html.

[22] "XACML Meets SQL - ARQ 2.0 Just the Cure for Secure Data Sharing," Axiomatics, 2016. [Online]. Available: https://www.axiomatics.com/resources/96-webinars/250-webinar-xacml-meets-sql-arq-2-0-just-the-cure-for-secure-data-sharing.html.

[23] C. Ngo, "SNE-XACML," [Online]. Available: https://github.com/canhnt/sne-xacml.

[24] GEYSERS, "GEYSERS - generalised architecture for dynamic infrastructure services," 2010. [Online]. Available: http://www.geyser.eu/.

[25] F. K, K. S, M. LA and T. MC, "Verification and change-impact analysis of access-control policies," in *Proceedings of 27th International Conference on Software Engineering ICSE'05*, New York, 2005.

[26] "Docs.oasis-open.org," 2016. [Online]. Available: http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en_files/image002.gif.

Appendix A

Hypothetical XACML Policy defined in Figure 2.3

```
<PolicySet xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
PolicyCombiningAlgId="urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:first-
applicable"
PolicySetId="DutyFreeAllowances"
Version="1.0">
<Target></Target>
<Policy PolicyId="DutyFreeAllowancesForForeigner"
RuleCombiningAlgId="urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:permit-
overrides" Version="1.0">
<Target>
<AnyOf>
<AllOf>
<Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
<AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">foreigner
</AttributeValue>
<AttributeDesignator AttributeId="citizenship"
Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="true">
</AttributeDesignator>
</Match>
</AllOf>
</AnyOf>
</Target>
<Rule Effect="Permit" RuleId="WineLiquorAllowanceForForeigners">
<Target>
<AnyOf>
<AllOf>
<Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
<AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">wine
</AttributeValue>
<AttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="true">
</AttributeDesignator>
</Match>
</AllOf>
<AllOf>
<Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
<AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">liquor
</AttributeValue>
<AttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="true">
</AttributeDesignator>
</Match>
</AllOf>
</AnyOf>
<AnyOf>
<AllOf>
<Match MatchId="urn:oasis:names:tc:xacml:1.0:function:double-greater-than-or-equal">
<AttributeValue DataType="http://www.w3.org/2001/XMLSchema#double">1.5
</AttributeValue>
<AttributeDesignator AttributeId="volume"
Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
DataType="http://www.w3.org/2001/XMLSchema#double" MustBePresent="true">
</AttributeDesignator>
</Match>
</AllOf>
</AnyOf>
</Target>
</Rule>
<Rule Effect="Permit" RuleId="AllowedItemAllowanceForForeigners">
<Target>
<AnyOf>
<AllOf>
```

```
<Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
<AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">allowedItems
</AttributeValue>
<AttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="true">
</AttributeDesignator>
</Match>
</AllOf>
</AnyOf>
<AnyOf>
<AllOf>
<Match MatchId="urn:oasis:names:tc:xacml:1.0:function:integer-greater-than-or-equal">
<AttributeValue DataType="http://www.w3.org/2001/XMLSchema#integer">250
</AttributeValue>
<AttributeDesignator AttributeId="value"
Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
DataType="http://www.w3.org/2001/XMLSchema#integer" MustBePresent="true">
</AttributeDesignator>
</Match>
</AllOf>
</AnyOf>
</Target>
</Rule>
<Rule Effect="Deny" RuleId="AccessDenyForForeigner"></Rule>
</Policy>
<Policy PolicyId="DutyFreeAllowancesForResidents"
RuleCombiningAlgId="urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:permit-
overrides" Version="1.0">
<Target>
<AnyOf>
<AllOf>
<Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
<AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">local
</AttributeValue>
<AttributeDesignator AttributeId="citizenship"
Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="true">
</AttributeDesignator>
</Match>
</AllOf>
</AnyOf>
</Target>
<Rule Effect="Permit" RuleId="WineAllowanceForResidents">
<Target>
<AnyOf>
<AllOf>
<Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
<AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">wine
</AttributeValue>
<AttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="true">
</AttributeDesignator>
</Match>
<Match MatchId="urn:oasis:names:tc:xacml:1.0:function:double-greater-than-or-equal">
<AttributeValue DataType="http://www.w3.org/2001/XMLSchema#double">2
</AttributeValue>
<AttributeDesignator AttributeId="volume"
Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
DataType="http://www.w3.org/2001/XMLSchema#double" MustBePresent="true">
</AttributeDesignator>
</Match>
</AllOf>
</AnyOf>
</Target>
</Rule>
<Rule Effect="Permit" RuleId="LiquorAllowanceForResidents">
<Target>
<AnyOf>
<AllOf>
<Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
```

```
<AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">liquor
</AttributeValue>
<AttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="true">
</AttributeDesignator>
</Match>
<Match MatchId="urn:oasis:names:tc:xacml:1.0:function:double-greater-than-or-equal">
<AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#double">2.5</AttributeValue>
<AttributeDesignator AttributeId="volume"
Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
DataType="http://www.w3.org/2001/XMLSchema#double" MustBePresent="true">
</AttributeDesignator>
</Match>
</AllOf>
</AnyOf>
</Target>
</Rule>
<Rule Effect="Permit" RuleId="AllowedItemAllowance">
<Target>
<AnyOf>
<AllOf>
<Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
<AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">allowedItems
</AttributeValue>
<AttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="true">
</AttributeDesignator>
</Match>
</AllOf>
</AnyOf>
<AnyOf>
<AllOf>
<Match <MatchId="urn:oasis:names:tc:xacml:1.0:function:integer-greater-than-or-
equal">
<AttributeValue DataType="http://www.w3.org/2001/XMLSchema#integer">90
</AttributeValue>
<AttributeDesignator AttributeId="stay"
Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
DataType="http://www.w3.org/2001/XMLSchema#integer" MustBePresent="true">
</AttributeDesignator>
</Match>
<Match MatchId="urn:oasis:names:tc:xacml:1.0:function:integer-greater-than-or-equal">
<AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#integer">250</AttributeValue>
<AttributeDesignator AttributeId="value"
Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
DataType="http://www.w3.org/2001/XMLSchema#integer" MustBePresent="true">
</AttributeDesignator>
</Match>
</AllOf>
<AllOf>
<Match MatchId="urn:oasis:names:tc:xacml:1.0:function:integer-less-than">
<AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#integer">90</AttributeValue>
<AttributeDesignator AttributeId="stay"
Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
DataType="http://www.w3.org/2001/XMLSchema#integer" MustBePresent="true">
</AttributeDesignator>
</Match>
<Match MatchId="urn:oasis:names:tc:xacml:1.0:function:integer-greater-than-or-equal">
<AttributeValue DataType="http://www.w3.org/2001/XMLSchema#integer">365
</AttributeValue>
<AttributeDesignator AttributeId="stay"
Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
DataType="http://www.w3.org/2001/XMLSchema#integer" MustBePresent="true">
</AttributeDesignator>
</Match>
<Match MatchId="urn:oasis:names:tc:xacml:1.0:function:integer-greater-than-or-equal">
```

```
<AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#integer">625</AttributeValue>
<AttributeDesignator AttributeId="value"
Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
DataType="http://www.w3.org/2001/XMLSchema#integer" MustBePresent="true">
</AttributeDesignator>
</Match>
</AllOf>
<AllOf>
<Match MatchId="urn:oasis:names:tc:xacml:1.0:function:integer-less-than">
<AttributeValue DataType="http://www.w3.org/2001/XMLSchema#integer">365
</AttributeValue>
<AttributeDesignator AttributeId="stay"
Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
DataType="http://www.w3.org/2001/XMLSchema#integer" MustBePresent="true">
</AttributeDesignator>
</Match>
<Match MatchId="urn:oasis:names:tc:xacml:1.0:function:integer-greater-than-or-equal">
<AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#integer">1750</AttributeValue>
<AttributeDesignator AttributeId="value"
Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
DataType="http://www.w3.org/2001/XMLSchema#integer" MustBePresent="true">
</AttributeDesignator>
</Match>
</AllOf>
</AnyOf>
</Target>
</Rule>
<Rule Effect="Deny" RuleId="AccessDenyForLocal"></Rule>
</Policy>
</PolicySet>
```

XACML Request to the above policy.

```
<Request xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17”
                CombinedDecision="false"
                ReturnPolicyIdList="false">
<Attributes Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject">
<Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
                        IncludeInResult="false">
<AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">Alice</AttributeValue>
</Attribute>
</Attributes>
<Attributes Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"/>
<Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
                        IncludeInResult="false">
<AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">wine
</AttributeValue>
</Attribute>
<Attribute AttributeId="volume" IncludeInResult="false">
<AttributeValue DataType="http://www.w3.org/2001/XMLSchema#double">1.0
</AttributeValue>
</Attribute>
</Attributes>
</Request>
```

XACML Response to the above Request.

```
<Response xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
                 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                 xsi:schemaLocation="urn:oasis:names:tc:xacml:3.0:core:schema:wd-
                 17 http://docs.oasis-open.org/xacml/3.0/xacml-core-v3-schema-wd-
                 17.xsd">
    <Result>
    <Decision>Permit</Decision>
    </Result>
</Response>
```