Secure web services composition

by

Ousmane Amadou Dia

Bachelor in Applied Mathematics and Computer Science
Ecole Superieure Polytechnique de Dakar (ESP-UCAD)

Master of Science
Ecole Superieure Polytechnique de Dakar (ESP-UCAD)

Submitted in Partial Fulfillment of the

Requirements for the Degree of Doctor of Philosophy in the

Department of Computer Science and Engineering

College of Engineering and Computing

University of South Carolina

2011

Major Professor	Chairman, Examining Committee
Committee Member	Committee Member
Committee Member	Committee Member

Dean of The Graduate School

Acknowledgments

Abstract

Access control systems must adjust to evolving business needs, such as accommodating new and modified business needs. These changes may require new skills or the modification of privileges users have in a given security system. Current access control models are not flexible enough to accommodate such changes. For example, in Role-Based Access Control (RBAC) security systems, assessing employees' competences is a difficult task because RBAC offers limited options to represent users' skills and qualifications. Existing approaches that address this issue rely on outdated user-role assignments or require to constantly engineer new RBAC roles. Role engineering is however costly. Relying on outdated assignments may provide unneeded privileges to the employees or block them from completing their assigned tasks, thus causing more security issues.

In the first part of this proposal, we propose a business-oriented approach to support accurate and dynamic user-role assignments for the RBAC model. Our model, called Business-Driven Role Based Access Control (BD-RBAC), is composed of three layers. The first layer extends the RBAC model with the concepts of *business roles, system roles, credentials*, and *users' capabilities*. The second layer dynamically assigns users to business and system roles, and filters outdated (abnormal) user-role assignments. The third layer supports exception handling and partial authorization. A novel aspect of this work is the agile adaptation of RBAC-based access control policies to changes in organizational needs, while reducing the burden of security administration. We provide algorithms to compute user-role assignments based on

organizational policies, users' requests and their capabilities, and show that their outputs are `permissible` – *a user is authorized to activate the role he is assigned to* –, `complete` – *a user can activate the roles necessary to perform the tasks he requests and for which he possesses the competences* –, and `minimal` – *a user does not receive not-needed privileges.*

Another important challenge we address in this dissertation proposal is the problem of policy composition. In collaborative environments where resources must be shared across multiple sites and between different entities, access control policies of the participants must be composed in order to define a coherent policy. Developing automated policy composition frameworks that are sound and practical has however been a long-standing problem. The difficulty of this problem increases when the policies to compose are stated independently and by autonomous entities. The diversity of the policies to combine may lead to conflicts where two policies yield contradictory decisions. Thus, a crucial step when composing policies is to ensure these policies are compatible, that is they maintain similar levels of security. In this respect, many conflict resolution strategies have been proposed. These strategies address conflicts by establishing a precedence relation among conflicting decisions, therefore enabling to select a unique decision. However, these strategies do not work when the policies are independently stated and there is no global control to dictate the ordering to apply. Other approaches deal with this issue by denying access to resources to which conflicting policies apply. Although such an approach preserves the confidentiality and integrity of the resources, it comes at the cost of restricting their availability. Our work aims thus at developing more advanced techniques to address policy conflicts.

In the second part of this proposal, we propose a technique to compose access policies that are stated in XACML by independent entities. We provide first a precise and non-ambiguous specification of the policies in logic form. We use a logic representation to build the foundations for a solid reasoning about the correctness

and the soundness of the algorithms that we design to detect conflicts and find the causes. Then, we propose a three-level conflicts resolution scheme: 1.) based on some annotations added to the policies, 2.) by means of a defeasible logic reasoning theory, and 3.) by recommendations. The novelty of this work is threefold. First, we provide a mechanism to enable the entities to explicitly specify assertions in their policies they strongly require to be satisfied and those that they consider as only recommended but not mandatory, if any. Such mechanism allows the entities to express compromises they are willing to make in case of conflicts. In real world scenarios, cooporating parties may relinquish some of their requirements in order to reach an agreement. In this work, these compromises, henceforth referred to as annotations, reflect adjustments that the entities are willing to bring in their policies in a way to prevent potential conflicts during the composition. Second, unlike many policy composition proposals which are mainly negotiation-oriented or assign priorities to policies, this work combines access policies without prioritizing any of the participants involved in the composition. Finally, the resulting composite policy appears flexible and easily adjustable to changes in any of the participants' systems with no impacts on its structure.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Today's economic expansion, dynamic markets and technologies bring new challenges. For organizations to remain competitive, they must generally either operate profound transformations in their business contexts such as outsourcing a segment of their production, offering new services, changing their business orientations, or collaborate, merge, or acquire subsidiaries of other organizations. From these business changes arise however many security issues.

First, changes in organizations business needs cause gaps between the specifications of organizational security requirements and their implementation. This, consequently, may lead to severe security breaches in the access control systems of the organizations. As an example, employees that use to perform certain tasks may later hold unneeded privileges after the tasks they were assigned are modified, or they may be no longer able to execute their tasks simply because of the modifications brought in the processes.

Second, collaborations, mergers, or acquisitions of organizations require also profound changes. From a security point of view, the most difficult task is to ensure the participants have similar access specifications or at least that their security policies are compatible. Taking this fact into account is essential as the organizations may enforce different security policies, which may consequently lead to inconsistencies be-

tween their policies. For example, one organization may have a rule over a particular feature while another one may have a contradictory rule over the same feature or expresses only conditional rules.

Dealing with these two issues have been a long-standing problem. In actuality, although many studies have focused on closely related problems in the recent past, most of them are atypical to particular scenarios, and therefore lack extensibility.

1.1 Motivations

Our focus in the first part of this proposal is on organizations that adopt RBAC as their access control model and which business needs evolve over time. RBAC wide adoption in security-sensitive areas such as healthcare, pervasive and mobile computing, and defense shows its potential. By adopting RBAC, organizations want: 1.) to ensure their employees get access to the resources they are permitted to in order to perform their tasks and in accordance with their job profiles, 2.) to reduce the complexity of security administration, and 3.) to maintain adherence to certain compliance standards. However, as the business needs of the organizations evolve over time, these requirements become difficult to observe due to many security issues.

In the second part of this proposal, we address the problem of policy composition in the context where multiple and separate entities want to compose their independently stated XACML access policies. One of the main difficulties in such a scenario is to deal with policy conflicts where one entity expresses a rule on a particular resource while another has a contradictory rule. Although many research proposals have addressed such issues in the past, the subtle difference between the work we propose here and these studies is dealing with policy conflicts in the absence of a hierarchy relation between the entities. Moreover, we also extend our work to address runtime conflicts that arise because of outages or shortage of resources.

1.2 Dissertation Outline

In the remainder of this proposal, we describe first in chapter 2 the security issues that occur as the business contexts of these organizations change and show why RBAC cannot address them. In the same chapter, in section 2.3, we present the limitations of the existing approaches in tackling this issue. We then introduce our approach which consists of extending RBAC with some new concepts such as *credentials, capabilities* and also of decoupling the concept of *role* in RBAC in *business role* and *systme role*. Credentials denote roles membership requirements for users, and capabilities, the competences of the users, their qualifications and skills. In section 2.4, we explain in detail these concepts and formalize them in section 2.5.

In chapter 3, we present a framework called policy composition and conflicts resolution (P2CR) to compose policies and to address conflicts. We give first in section 3.2 some background information to help understand our work. Then, we propose a method to detect policy conflicts in section 3.6.2 and a threefold strategy to address policy conflicts. By means of the annotations that the entities add to their policies, we first attempt in section 3.6.3 to remove the set of values for which the conditions of the conflicting rules hold. We first describe these annotations in section 3.4 and provide means to add them in XACML. In section 3.6.3 also, we present our second approach which uses non-monotonic reasoning to order rules that conflict and for which we cannot remove the conflicts by the annotations. Finally, in the same section, for the rules that conflict and that cannot be ordered, we provide recommendations to the entities owners of the rules as to which requirements in their rules to annotate as recommended so as to resolve the conflicts. In chapter 2 also and in section 3.7, we address another type of conflicts called dynamic conflicts and show how P2CR resulting composite policy can be adjusted to such conflicts.

Finally, in chapter 4, we present our future work which will complete the problems we address in chapters 2 and 3.

Chapter 2

Business Driven User Role Assignment: Nimble Adaptation of RBAC to Organizational Changes

In this chapter, we present our current research results in addressing the adaptation of RBAC-based access control systems to organizational changes. The remaining work is presented in section 4.1.

2.1 Introduction

The Role-Based Access Control (RBAC) model [42, 26, 41] has been widely used in many commercial systems to enforce security. The central idea behind this model is that users in an organization or domain are mapped to roles. A role can be defined as a set of actions and responsibilities associated with a particular working activity. Access rights are associated with the roles, thereby assigning appropriate permissions to the users belonging to these roles. RBAC provides intuitive and powerful access control capabilities. An essential but costly component of RBAC is to engineer the roles. This process, also known as role engineering, consists precisely of defining optimal and persistent sets of roles, permissions, and role-permission assignments

that meet organizational requirements such as adherence to compliance standards while limiting the system administration cost.

However, as organizations evolve, new business tasks arise, existing ones are revised or eliminated. These business changes require to periodically repeat the role engineering process or to depend on outdated user-role assignments. Outdated assignments may however provide unneeded privileges to the users (employees) on the security systems or block them from completing their assigned tasks, thus causing more security risks. Moreover, the capabilities of the employees that is; their aptitude to carry out the assigned tasks without causing harm to the resources or security systems of the organizations, also change. Skills and capabilities of the employees play also a crucial role in influencing organizational decisions. For instance, as new business needs appear, organizations must determine whether their employees have the necessary skills and capabilities to satisfy these needs or to rely on more qualified external users.

In RBAC, users that are unknown within or external to an organization security domain are however generally mapped to default roles such as *guest* or *customer* with limited privileges, or simply discarded. Another non-trivial challenge with RBAC is assessing the employees' competences. This is difficult because RBAC offers limited options to model the employees' skills, experiences and qualifications. Although RBAC provides a hierarchical ordering of roles, such a representation reflects more an organization line of authority and responsibility than the qualifications, skills and experience of its employees. Thus, little of an employee' capabilities may be dynamically inferred from the roles to which the employee is assigned. Organizing RBAC roles according to the employees' competences would require frequent and manual updates as employees acquire new skills and gain experience over time, and manually adding and removing employees in the assigned roles based on their competences is also costly and error prone.

We propose in this chapter to supplement the standard RBAC model with the concepts of capabilities and credentials. The capabilities denote the competences of the users (i.e., their qualifications, skills and experiences). The credentials denote roles membership requirements. Extending RBAC with these concepts enables our model to dynamically adjust RBAC-based security systems according to changes in the business requirements. The work that closely addressed that problem is that of Sandhu et al. [1] in which they supplement RBAC with rules and assign users to roles based on predefined attribute expressions and constraints. However, their approach requires the attribute expressions to be updated when new roles are created or when existing roles are modified. Moreover, they assume the number of roles to be relatively small. As it increases to certain levels (above a hundred), maintaining the rules becomes cumbersome.

We argue that a concise and automated adaptation of security policies to organizational changes requires the understanding of the organizations' business contexts. In this chapter, we explicitly incorporate the business context into RBAC to define persistent user-role and role-permission assignments. We propose to have separated views on the business processes and the permissions over the protection objects they involve. More specifically, we decompose the concrete concept of *role* in RBAC into two different components that are: business roles and system or IT roles. The reason behind this decoupling is that, user-role relationships are the most volatile because of the changes that arise in organizations business needs. In addition, role-permission relationships change less frequently, and role-role relationships rarely. Therefore, as the user-role relationships are particularly sensitive to the security of an organization, only users that have the necessary skills and capabilities must be allowed to perform the organization' business tasks. The decoupling of the system roles and business roles allows us thus to manage changes arising in organizations business contexts without major effects on the role-permission assignments, and to curtail the

considerable amount of needed administration work.

In this chapter, we define a new model, Business-driven Role Based Access Control (BD-RBAC), which consists of three layers. The first layer corresponds to the standard RBAC model enriched with the concepts business role, system role, credentials and capabilities. The second layer or credential-filtering layer provides a flexible way to dynamically assign internal and external users to (new) business tasks and business roles and to filter outdated user-role assignments. Finally, the third layer or exception-handling layer handles partial authorizations i.e., it adjusts authorization decisions according to urgent access needs. The main contribution of our model is thus to adapt organizations access controls that are based on RBAC dynamically to the organizations changing business requirements.

The remainder of the chapter is organized as follows. Section 2.2 discusses the related work. In section 2.3, we give a motivating example to substantiate our work and illustrate our contributions. Section 2.4 describes the different layers of our model and how they interact. Section 2.5 formalizes mathematically the model. We conclude in section 2.6 and give suggestion for future work.

2.2 Related Work

The main focus of this work is to avoid the repetitive and costly role-engineering process while business contexts evolve and yet to provide better security. Over the last few years, role engineering has gained significant attention. Different approaches that can be categorized into *top-down*, *bottom-up*, and *hybrid* have been proposed.

The top-down approach is a business-driven technique where permissions are derived from abstract concepts such as work-patterns or tasks [43, 38, 8, 17]. It defines first the roles from an organization line of business, process descriptions, the organizational structure of the domain, or features of the employees as given by the Human Resources Department of an organization or enterprise, then assigns to these roles

the needed permissions. Each task in the organization is analyzed and decomposed into smaller units to identify the resources and actions needed on the system.

The bottom-up approach starts with the permissions the users have on the system i.e., the existing assignments of users to permissions, and subsequently aggregates them into roles. Unlike the top-down approach, the bottom-up is a discovery process which requires data-mining techniques or tools to extract permissions from the systems' configurations and to group them into roles [32, 48, 44]. Roles, and assignments of users to roles and permissions to roles must be found that approximate the existing user-permission assignments as best as possible while limiting the number of roles that must be maintained.

An alternative to the bottom-up and top-down approaches is the hybrid model which combines both techniques to engineer roles and to derive permissions [22, 21].

While the existing role engineering models aim at optimizing the user-permission assignments as best as possible by finding a minimal set of roles, user-role assignments, and role-permission assignments, they either assume the business processes are static or the users' permissions are known in advance. Hence, very few of these models are fully extensible or applicable in environments where frequent changes occur. In organizations, business needs and users' capabilities change over time. New business processes arise, others are modified or eliminated. Unexpected situations or urgent needs may require to bypass organizational policies. While such situations occur more frequently on the business side, the IT resources are less affected. Therefore, to anticipate the changes, it is important to have separated views over the business processes and the objects they require.

Separating the business roles from the system roles helps to align the business and the IT perspectives by resorbing the gaps between the implementation of an organization security policy and its specification, and also to avoid security breaches that could stem from outdated user-role assignments. Klarl et al. proposed in [28] an

extension to RBAC with a business perspective. In their approach, the business roles are derived from the job profiles and are separated from the system roles. System roles provide an abstract representation of IT systems permissions. However, they have not explain how to map the business roles to system roles, and their approach is not fully applicable in changing business environments.

Existing role engineering techniques rarely implement exception handling mechanisms or partial authorization. Certain organizations require adapting access decisions to needs; for instance in healthcare, in an emergency situation, it should be allowed to override the access control policy by giving unauthorized users access to certain resources. As specified by HIPAA [13], risk-adaptive access control models (RAdAC) are needed to adapt access decisions according to situations in hand [36].

Access control models in which access decisions are based on changing risks and needs have started to emerge. A new approach called *Brick-The-Glass (BTG)* has been proposed to break or to override access rules in a controlled manner [20, 14, 34]. More intuitively, BTG is a technique that embodies the idea that in certain situations, a user may override a denied-request and access a resource he is not authorized for. In [20], users can override access rules but their actions are recorded and must be justified. Although this approach is more flexible compared to RBAC, it requires knowing the identities of the users and constantly monitoring their actions. Thus, it is more overhead added in the administration work. The approach closer to the method we propose for safely overriding a denied-request is [34]. In this proposal, Marinovic et al. presented a novel BTG policy by establishing why the access was denied [34]. They introduced the notion of *subject' competences and empowerments* to gain more insight into the causes for the access denial, and used Belnap's four-valued logic to represent conflicting and missing or inconsistent knowledge. The model has a declarative query to specify a BTG decision procedure. This allows a policy writer to condition and constrain the BTG access permissions in a fine-grained manner

by embedding integrity constraints into the decision procedure. However, as users' competences change over time, this would require constantly refining the decision procedure. Moreover, accesses that a legitimate user can override are already known by the security system. In many domains, however, it is hard if not impossible to know a priori all the emergency situations and to encode the exceptions. Thus, some emergencies may happen, and the subjects that have the necessary competences to carry out the needed actions be prohibited simply because they are not entitled to perform such actions or because these emergencies were not expected. Finally, some of the must-have competences may not be that relevant in granting the override, thus over-restricting the decision procedure.

2.3 Motivation and Overview

Consider as an example an organization A that implements RBAC to control access to its resources. Assume that **orginally** A's access control specifications allow its employees to perform their tasks. Suppose that **over time** A is extending its business and that this extension requires changes in A's processes; that is some of A' existing tasks are modified to better fit A' business needs, new tasks are added and other tasks are eliminated.

Changes in A' s business context may lead A' s security system to any of the states given in Table 1. Below, we examine three scenarios (changes) that can cause security breaches and adjustments in A security system.

Scenario 1: - ***New tasks are added.***

New tasks may require certain skills and qualifications that A' employees may not have. In this case, to perform these tasks, A must either provide means to its emloyees to acquire the necessary skill set or rely on external users. A' s access control system results then either in *state 1* or *state 3*.

Scenario 2-a: - ***Existing tasks are modified.***

The modified tasks may **require capabilities that were not needed before**. Employees that have the capabilities to perform these tasks may not possess the necessary privileges. Their privileges must then be upgraded in case A relies on them to perform these tasks. Furthermore, certain privileges held by the employees that were executing these tasks before may no longer be needed. To prevent security breaches, these privileges must be dropped. Thus, the user-role assignment must be redefined. A's access control system results in any of the states in Table 2.1.

Scenario 2-b: - ***Existing tasks are modified such that some of the previous privileges are not needed.***

Consider as an example the RBAC role *Payroll admin* to which are assigned A' payroll admins. Assume some of the admins (*Admin1*) work on employees salary payment (*SP*) while the others (*Admin2*) deal with tax payment (*TP*). Also, assume originally that *SP* and *TP* require both a read access r to a database d. Now, suppose that the permission $p = (r, d)$ is no longer needed for performing *SP*. Because A enforces RBAC, to prevent admins in *Admin1* from accessing d, either p has to be dropped, thereby restricting admins in *Admin2* from accessing d, or the role *Payroll admin* has to be splitted into two roles in which one is assigned p. A's access control system results then either in *state 2* or in *state 4*.

Scenario 3: - ***Capabilities of the employees change.***

Employees whose capabilities decrease (e.g., an IT certification that expires) may detain privileges that were assigned based upon the expired capabilities. A's access control system results then in *state 1*, *state 2* or *state 4*.

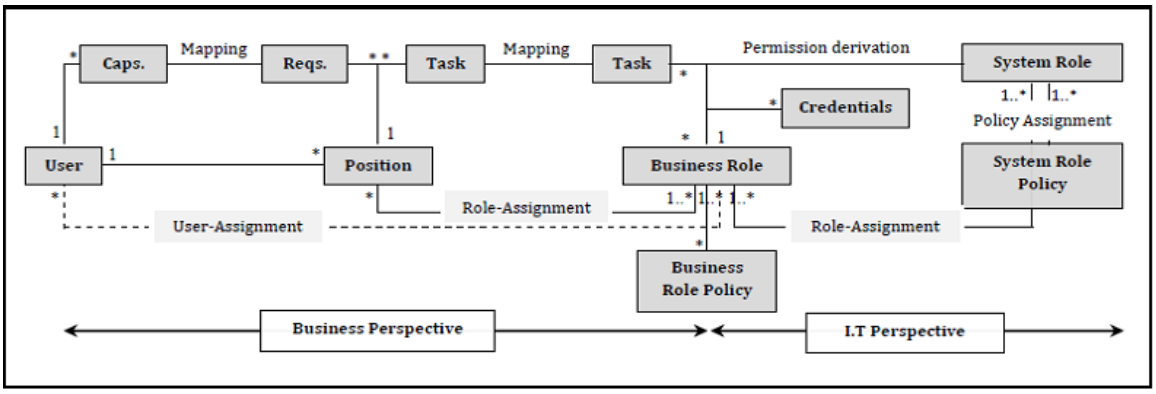

Figure 2.1: Meta-model. *Reqs* are the requirements attached to a position and *Caps* denotes the capabilities of a user.

State 1	➥ Employees are able to perform existing, modified and new business tasks, and
	➥ Employees do not hold unneeded privileges.
	➠ ***No security violation***.
State 2	➥ Employees do not hold unneeded privileges, and
	➥ Employees can perform modified and new tasks if their privileges are upgraded.
	➠ ***Adjust access control specifications***.
State 3	➥ Employees are not able to perform modified or new business tasks, and
	➥ Employees do not hold unneeded privileges.
	➠ ***No security violation.*** Train employees or rely on external users.
State 4	➥ Employees do not hold unneeded privileges but
	➥ Employees' capabilities to perform existing, modified or new tasks decrease.
	➠ ***Security violation***.

Table 2.1: A' s security system after changes in its business context. "➠" indicates the resulting state of A's access system after A operates the changes in its business context.

2.4 The BD-RBAC layout

2.4.1 RBAC-enriched layer

This section describes the different components of the BD-RBAC first layer: the RBAC-enriched layer. The meta-model is depicted in *Figure 2.1*.

A **position** is a job in an organization with a predefined set of business tasks to execute. It has certain requirements that must be fulfilled. For instance, before a position is filled, the background, work experience, technical skills and qualifications

of a group of users are screened to see if they match the position' requirements.

A **user** possesses a set of skills referred to as capabilities. There are two types of users that we consider: users internal to an organization and the external users. Beside the skills required for a given position, a user may possess additional capabilities that may be useful for performing other activities for the organization. A typical example is when new business tasks appear. Instead of hiring employees, an organization may rely on its team to perform these business tasks by dynamically assigning the tasks to its employees possessing the needed capabilities. Thus, knowledge of the employees' skills not currently used helps to fulfill future user-role assignment needs.

A **Business role** is a collection of business tasks performed by a group of users. In the remaining of the chapter, we refer to the *business tasks* as tasks. Examples of tasks are *"analyze financial data"*, *"prepare and record reports"*, *"maintain financial databases and accounting procedures"* that can be grouped into the business role ***accountants***. Approaches to collect or extract the business tasks from organizational requirements and business contexts have already been proposed in the past [38, 8]. Thus, we do not address this issue. Business roles are structured in a hierarchical order. The assignment of a user to a business role is based on a match between a task the user wants to perform and a task associated with the business role. We assume tasks are clearly specified to avoid erroneous task - business role assignments. Authorized users executing the tasks are granted minimal privileges needed to perform the tasks.

Each business role is associated with a set of **credentials**. A credential is a requirement that a user must satisfy in order to be assigned to a business role. A credential can be a qualification, a skill, a responsibility or a level of experience and authority pertaining to an organization's business role that a user must have. Such credentials may be collected without difficulties by the Human Resources Department of an organization during for instance the employees periodic evaluations, or directly

from the employees' resumes. The credentials are useful for dynamically assigning internal users to new business roles and for managing access requests from external users. A match between the capabilities of a user and the credentials required for a task confers to the user the attached privileges to execute the task.

A **Business role policy** expresses a cardinality constraint or a dual constraint. A cardinality constraint limits the number of tasks in business roles, and a dual constraint enforces the separation of duty principle.

A **System role** is a collection of permissions over some IT objects needed to perform a particular task. Interestingly, system roles are useful to enforce least-privileges and need-to-know [11] principles. For instance, in scenario 2-b of *section 2.3*, with the system roles, there is no need to drop the permission p or to split the role *Payroll admins*. *SP* and *TP* are two different tasks though they require both p. Thus, each has its own system role. When p is not needed anymore to perform *SP*, it can be removed seamlessly from *SP* associated system role without affecting the execution of *TP*. The system roles are created by using a scenario-driven role engineering approach [38, 8] i.e., the permissions necessary to perform a task are identified first through a requirement engineering process and then grouped as a system role. The relation system role - task - business role forms an *organization-centric authorization rule* (*o-rule*) that is used to evaluate requests from internal users. Upon internal request, the *o-rules* that apply to the request are first determined. Then, the business roles to consider in order to issue a decision (permit or deny) are extracted from the *o-rules*.

A **System role policy** expresses optimization constraints over objects in order to avoid deadlocks and to prioritize accesses.

2.4.2 Credential-filtering layer

To address the security violations and the adjustment needs expressed in *Table 2.1*, we introduce a second layer in our model: the credential-filtering layer. The role of

the credential-filtering layer is: (1) to purge the user-role assignments based on the capabilities of the users and business role credentials, and (2) to dynamically assign internal and external users to (new) tasks or business roles.

We use an approach similar to subject switching [45]. In [45], access decisions are based on the existence of business roles with certain privileges over requested tasks to which requesting users can be mapped. In our case, if an internal user needs to perform a task, based on the existence of a business role with certain permissions over the requested task, the capabilities of the user are compared with the credentials of the business role. Changes in the business contexts may also require external users' services (e.g., support team, outsourced workforce). The credential-filtering layer can dynamically assign external users to business roles provided there is a match between the credentials attached to these business roles and the external users' capabilities.

To filter the user-role assignments and to dynamically assign internal and external users to business roles, we introduce the *composition-centric authorization rules* (*c-rules*). The *c-rules* extend the *o-rules* with the business role credentials. External users resquesting tasks are assigned to business roles only if they satisfy the mandatory credentials associated with these roles and the disclosure of any information not needed to perform the tasks is evaluated low or unlikely. Upon external request, the *c-rules* that apply to the request are first determined. Then, the business roles to consider in order to evaluate (permit or deny) the request are extracted from the *c-rules*.

2.4.3 Exception handling layer

In the credential-filtering layer, an external user cannot execute any task unless his capabilities fully match the credentials of a business role. In fact, when an external user submits a request for a task t, either the capabilities of the user match all the required credentials for performing t, the user satisfies part of the required constraints but not all of them, or his credentials do not match any of the imposed credentials.

Thus, the external user may possess some of the credentials of a business role but not all of them and still be denied the execution of a task.

In certain circumstances, however, for example in emergency situations, individuals may have to perform activities that they are not authorized for in normal conditions. Thus, access decisions must be flexible enough to accomodate these requirements and be based on changing risks or needs. Research to adapt access decisions according to situations in hand has been progressing recently [36, 20]. In this regard, approaches to render RBAC authorization rules less restrictive have been proposed. However, there are some limitations. For example, Bertino et al. define in [9] a method based on counter proposals. In their approach, acceptable conditions for access grant are sent back to a user supposed to be trustworthy. Although, this approach relaxes the authorization, it presents some limitations because the user's trustworthiness is difficult to assess. Moreover, the user may still be trustworthy but revealing the acceptable conditions for performing a task to the user (assuming the user is external to the organization) is giving him insights on how the organization functions and carries out its business.

Therefore, to respond efficiently to unexpected scenarios and urgent needs, and to render the access decisions less restrictive, we introduce a third layer in our model: the exception handling layer. The exception handling layer implements partial authorization. This means that a user does not necessarily have to satisfy all the credentials of a particular business role to execute a task t but just a subset of them known as mandatory credentials. If the user' capabilities match the mandatory credentials required for t, and based on some prior knowledge, the authorization system evaluates whether not-needed protection objects may be inferred from that operation. If not-needed objects may be inferred, then the request is denied. Otherwise, the request is permitted.

To issue a decision, the idea is to compute conditional probabilities of disclos-

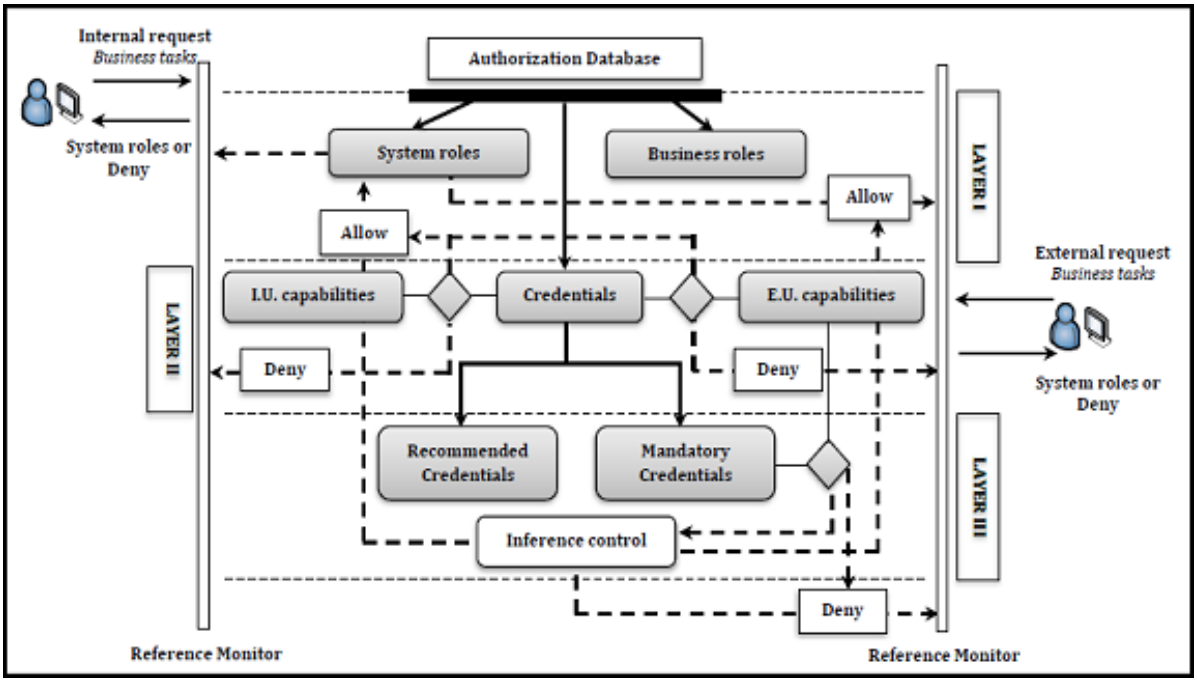

Figure 2.2: Interaction flow diagram. The authorization database is composed of tables that store the identity of internal users, their capabilities and positions, tasks, business roles, system roles, c-rules and o-rules. We have not represented the tables to avoid cluttering the figure. Comparisons between capabilities and credentials are represented by the diamonds.

ing certain objects and compare the results with the objects disclosure thresholds. Assuming all the objects are stored in relational databases, the dependencies between objects can be deduced by means of the `foreign-key constraint` and the `primary-key constraint`. Dependency-probabilities between any two objects can be specified by the organization. If for each needed object, the computed conditional probability is below its disclosure threshold, the requested task t is granted, if above, the request is denied. Due to space restrictions, we however leave the decision inference for future work.

2.4.4 Interactions between layers

The interaction flow between BD-RBAC different layers is depicted in *Figure 2.2*. Layer I refers to the RBAC-enriched layer, layer II corresponds to the credential-filtering and layer III refers to the exception-handling layer.

The interaction flow starts with a user requesting access rights to execute certain

tasks. When the request is submitted, the reference monitor extracts from the Authorization Database the relevant authorization rules. The authorization rules determine the business roles, system roles and the credentials to use in order to evaluate the request. Business role credentials are decomposed into mandatory and recommended credentials. The capabilities of the external users (E.U. in *Figure 2.2*) are not extracted from the authorization database but are rather supplied by the external users at request time. The extracted components together with the external users' capabilities are passed to the credential-filtering layer and the exception-handling layer via the RBAC-enriched layer. The RBAC-enriched layer is also responsible of enforcing the positive decisions (permit) issued by the two other layers.

The credential-filtering layer is in charge of verifying whether the capabilities of the users match with the extracted business role credentials. In the case of a match, the credential-filtering layer notifies the RBAC-enriched layer to grant to the requesting user the necessary system roles to perform the requested tasks. Otherwise, if the requesting user is internal (I.U. in *Figure 2.2*) to the organization, the credential-filtering layer directly denies the request. This step is referred to as the user-role assignment filtering. An external user whose capabilities do not match all the extracted credentials but only a subset of them (i.e., the mandatory credentials) may under certain conditions be allowed to execute the tasks he requests. These conditions are *"whether not-needed objects may be disclosed if the permission to execute the tasks is granted to the user"*.

The role of the exception-handling layer is to verify that such conditions hold during the decision making process. If the conditions hold, the exception-handling layer notifies the RBAC-enriched layer to provide the necessary system roles to the requesting user. In case the conditions are violated, the exception-handling layer directly denies the request. Careful readers will notice that the recommended credentials do not influence much on the decision making process. In fact, they are very specific to

an organization (e.g., seniority of a user within the organization). Therefore, they are more likely to be fulfilled by the internal users than by the external users.

2.5 Formalization

We now formalize the concepts mentioned in the different layers and address the types of problems described in our motivating example. We first start with the concepts introduced in the RBAC-enriched layer.

2.5.1 RBAC-enriched layer

Definition 2.5.1. - ***Preliminary notions***.

- Let $\mathcal{U}$ denote the set of internal users, $\mathcal{BR}$ the set of business roles, T the set of tasks, $\mathcal{C}$ the set of credential expressions and $\mathcal{SR}$ the set of system roles that we will define shortly.
- Let $\mathcal{P}(T), \mathcal{P}(\mathcal{BR})$, and $\mathcal{P}(\mathcal{SR})$ denote the powersets of respectively $T, \mathcal{BR}$, and $\mathcal{SR}$.
- A credential is of the form $\langle$*c-name op value*$\rangle$ where *c-name* denotes the name of the credential, *op* is an arithmetic operator $(=, >, <)$, and *value* is a value in the domain of *c-name*. A credential expression C of a business role is a conjunction of credentials i.e., $C = c_1 \wedge ... \wedge c_k$. Similarly, for a user, a capability expression is a conjunction of his capabilities.
- Let $\mathcal{O}$ denotes the set of protection objects, and $t \in T$ be a task. $\mathcal{O}(t) \subseteq \mathcal{O}$ denotes the set of objects needed to perform the task t. Let $T = \{t_1, ..., t_k\}$ be a set of tasks. Then, $\mathcal{O}(T) = \mathcal{O}(t_1) \cup ... \cup \mathcal{O}(t_k)$.
- $\varphi : \mathcal{U} \rightarrow \mathcal{P}(T)$ maps each $u \in \mathcal{U}$ to a set of tasks.
- $\tau : T \rightarrow \mathcal{SR}$ maps each task $t \in T$ to its corresponding system role.
- $\rho : \mathcal{BR} \rightarrow \mathcal{P}(T)$ maps each business role to a set of tasks.

- $\lambda : \mathcal{BR} \rightarrow \mathcal{C}$ maps each business role to a credential expression. For each $br \in \mathcal{BR}$, $\lambda(br) = \mathcal{MC}_{br} \wedge \mathcal{RC}_{br}$, where $\mathcal{MC}_{br}$ are the required or mandatory credentials, and $\mathcal{RC}_{br}$ are the recommended or non-mandatory credentials.

- $\delta : \mathcal{BR} \rightarrow \mathcal{P}(\mathcal{SR})$ maps each business role to its set of system roles. $\delta(br) = \bigcup_{t \in \rho(br)} \tau(t)$ gives the set of system roles assigned to br.

- $Sl : \mathcal{O} \rightarrow [0, 1]$ maps each object to its sensitivity level. Sensitive objects are assigned high values whereas non-sensitive objects are assigned low values. For any $t \in T$, $Sl(\mathcal{O}(t)) = \sum_{o \in \mathcal{O}(t)} Sl(o)$.

- *isEqualTo*(t', t) is a predicate that returns "true" if t' and t are syntactically equal, and "false" if not.

- *isImpliedBy*(x, y) is a logical implication. It returns "true" if x is implied by y, and "false" if not. □

Next, we give the definition of a system role.

Definition 2.5.2. - ***System role***.

A system role sr is a collection of positive permissions. A permission is of the form $p = (o, a)$, where a defines the type of access (r - *read,* w - *write,* x - *execute*) allowed over an object $o \in \mathcal{O}$. □

The permissions that are necessary to execute a task are identified and grouped as a system role. We adopt the requirement engineering technique proposed by Neumann et al. [38] to identify the needed permissions. We assume that initially each task is associated with a complete and minimal system role; that is:

Definition 2.5.3. - ***System role Properties***.

A system role - task assignment $\tau(t)$ is:

- *Complete* (*weak τ-property*): if the system role $\tau(t)$ associated with t is composed of the necessary permissions to perform t.

- *Minimal* (*strong τ-property*): if the system role $\tau(t)$ associated with t is composed exactly of the permissions necessary to perform t. □

We next examine in *Definition 2.5.4* the hierarchical ordering of the business roles and its properties.

Definition 2.5.4. - ***Business role hierarchy*** - $\mathcal{RH}$.

$\mathcal{RH}$ is the graph G of the partial order relation ($\geq$) between the business roles in $\mathcal{BR}$. If *br, br'* $\in \mathcal{BR}$ and $br \geq br'$ then we say *br dominates br'* or *br is senior to br'*. $\mathcal{RH}$ must satisfy the following properties:

- *ρ-property*: $\forall\, br,\, br' \in \mathcal{BR}\ (br \geq br' \Rightarrow\ \forall\, t' \in \rho(br')\, \exists\, t \in \rho(br) : isEqualTo(t', t))$.
- *λ-property*: $\forall\ br,\ br' \in \mathcal{BR}\ (br \geq br' \Rightarrow isImpliedBy(\lambda(br'),\, \lambda(br)))$.
- *δ-property*: $\forall\ br,\ br' \in \mathcal{BR}\ (br \geq br' \Rightarrow \delta(br) \supseteq \delta(br'))$. □

Intuitively, *ρ-property* means a user assigned to *br* is capable of performing all the tasks in *br'*. *λ-property* means the credentials of *br* dominate the credentials of *br'* and *δ-property* means *br* inherits the system roles of *br'*.

In BD-RBAC, internal users are assigned tasks and based on these tasks, they are assigned to business roles, thereby acquiring the privileges necessary to perform these tasks. The relation between a task, a business role and a system role is defined more formally below.

Definition 2.5.5. - ***Organization-centric Authorization rule***.

An organization-centric authorization rule is of the form *o-rule* $= (t,\, br,\, \tau(t))$ where:

- t is a task,
- *br* is a business role such that $t \in \rho(br)$,
- $\tau(t)$ defines the system role associated with t. □

2.5.2 Credential-filtering layer

The credential-filtering layer provides the security mechanisms to dynamically assign users to business roles and to filter outdated user-role assignments based on a finite set of authorization rules (i.e; *c-rules*) that we define below.

Definition 2.5.6. - ***Composition-centric Authorization rule***.
A composition-centric authorization rule is of the form $c\text{-}rule = (t, br, \lambda(br), \tau(t))$ where:

- t is a task,
- br is a business role such that $t \in \rho(br)$,
- $\lambda(br) = \mathcal{MC}_{br} \wedge \mathcal{RC}_{br}$ are the credentials attached to br,
- $\tau(t)$ defines the system role associated with t. □

In BD-RBAC, a user is assigned to a role and is allowed to execute a task only if he is authorized by the system. We examine in *Definition 2.5.7* the requirements for assigning a user requesting a task to a business role and extend it in *Definition 2.5.10* for a set of tasks.

Definition 2.5.7. - ***Single Task Authorization***.
Let t be a task requested by a user u with the capability expression C'. u is `authorized` to perform t if one the following holds:

- $u \in \mathcal{U}$ and $t \in \varphi(u)$,
- $\exists\ br \in \mathcal{BR} :\ isImpliedBy(\lambda(br), C')$,
- $isImpliedBy(\mathcal{MC}_{br}, C')$ and $(\forall\ o \in \mathcal{O} \setminus \mathcal{O}(t)\ P(o \mid \mathcal{O}(t)) \leq 1\text{-}\ Sl(o))$. □

The third bullet targets specifically external users. It is interpreted as follows: if u requests a task t and u satisfies $\mathcal{MC}_{br}$, then u is granted t only if the execution of t

would not disclose not-needed objects. Otherwise, u's request is denied. $P(o \mid \mathcal{O}(t))$ is the probability of inferring an object o not needed from the execution of t. If $P(o \mid \mathcal{O}(t))$ is below the object's disclosure threshold (i.e., 1- $Sl(o)$), we allow the execution of t. Otherwise, we deny it. We next define some key properties for the generation of the business role assignments.

Definition 2.5.8. - ***Completeness***.

Let T' be a set of tasks requested by a user u and $\mathcal{A} \in \mathcal{P}(\mathcal{BR})$ be a possible assignment of business roles. $\mathcal{A}$ is `complete` if: [$\forall\ t' \in T'$ ($\exists\ br \in \mathcal{A}$, $\exists\ t \in \rho(br)$: $isEqualTo(t, t')$)]. □

Definition 2.5.9. - ***Minimality***.

Let T' be a set of tasks requested by a user u, and $\mathcal{RT}' \in \mathcal{P}(\mathcal{BR})$ be all the possible assignments of business roles for performing T'. Let $\mathcal{A}_i \in \mathcal{RT}'$. $\mathcal{A}_i$ is `minimal` if there does not exist any $\mathcal{A}_j \in \mathcal{RT}'$ such that:

- $F(\mathcal{A}_j) \leq F(\mathcal{A}_i)$ where $F(\mathcal{A}) = \sum_{br \in \mathcal{A}} \sum_{t \in \rho(br) \setminus T'} Sl(\mathcal{O}(t)),\ \mathcal{A} \in \mathcal{RT}'$, and
- $\mathcal{A}_j \subseteq \mathcal{A}_i$. □

In *Figure 2.3* page 31, we define a procedure to generate all the business role assignments for performing a set of tasks T'.

Theorem 2.5.1. - **Generation of Business role Assignment**.

Procedure *ExtractRoles* of page 31 computes the set $\mathcal{RT}'$ of all complete business role assignments $\mathcal{A}$, and $\mathcal{BR}_{T'}$ has the minimal cost $F(\mathcal{A})$. Procedure *ExtractRoles* terminates and $\mathcal{BR}_{T'}$ is complete and minimal. □

Proof. - **Theorem 2.5.1**.

Let us prove that $\mathcal{RT}'$ and $\mathcal{BR}_{T'}$ are complete. `Breadth First Search (BFS)` iteratively visits each level of the connected graph G. At the end, it returns $\mathcal{Q}$ the set of all paths. Because of the way G is constructed, for each $t' \in T'$, for all $\mathcal{A} \in \mathcal{Q}$ there

exists $br \in \mathcal{A}$ such that $t \in \rho(br)$. Thus, $\mathcal{A}$ is complete, so is $\mathcal{Q}$. Then, $\mathcal{RT}'$ is also complete. Since $\mathcal{BR}_{T'} \in \mathcal{RT}'$, $\mathcal{BR}_{T'}$ is also complete.

To prove the assignment $\mathcal{BR}_{T'}$ is `minimal`, let us show that: (1) the sum of the sensitivity level of the objects not needed to perform T' ($F(\mathcal{BR}_{T'})$) is minimal, and (2) no proper subset of $\mathcal{BR}_{T'}$ is a candidate assignment.

Assume by contradiction that there exists a user-business role assignment $\mathcal{BR}'_{T'}$ such that $F(\mathcal{BR}'_{T'}) < F(\mathcal{BR}_{T'})$ and $\mathcal{BR}'_{T'}$ is a candidate assignment. Since $\mathcal{BR}'_{T'}$ is a candidate assignment, it must be in $\mathcal{RT}'$. Then, $\mathcal{BR}'_{T'}$ must also have been selected in $\mathcal{RT}'$. If $F(\mathcal{BR}'_{T'}) < F(\mathcal{BR}_{T'})$, $\mathcal{RT}'$ should not contain $\mathcal{BR}_{T'}$. Therefore, *ExtractRoles* could not have returned $\mathcal{BR}_{T'}$.

Now, let us assume that both $\mathcal{BR}_{T'}$, and $\mathcal{BR}'_{T'}$ are in $\mathcal{RT}'$ i.e., $F(\mathcal{BR}'_{T'}) = F(\mathcal{BR}_{T'})$. Then, $\mathcal{BR}'_{T'} \subset \mathcal{BR}_{T'}$ must hold. However, bullet 4 of *ExtractRoles* would have replaced $\mathcal{BR}_{T'}$ by $\mathcal{BR}'_{T'}$ and $\mathcal{BR}_{T'}$ could have not be returned.

Clearly, *ExtractRoles* terminates since `BFS` complexity is linear ($\Theta(|V| + |E|)$) in V the set of nodes of G and E the set of edges. □

Definition 2.5.10. - ***Multiple Tasks Authorization***.
Let T' be a set of tasks requested by a user u with the capability expression C', and $\mathcal{RT}'$ be the business role assignments generated by the procedure *ExtractRoles* in *Figure 2.3*. u is `authorized` to perform T' if one the following holds:

- $u \in \mathcal{U}$ and $T' \subseteq \varphi(u)$,
- $\exists\ \mathcal{A} \in \mathcal{RT}' : \forall\ br \in \mathcal{A}\ isImpliedBy(\lambda(br), C')$,
- $\exists\ \mathcal{A} \in \mathcal{RT}' : (\forall\ br \in \mathcal{A}\ isImpliedBy(\mathcal{MC}_{br}, C'))$ and $(\forall\ t' \in T'\ \forall\ o \in \mathcal{O} \setminus \mathcal{O}(t') : P(o \mid \mathcal{O}(t)) \leq 1\text{-}\ Sl(o))$. □

Next, we formalize in *Definition 2.5.11* how to select the business role(s) to assign to an internal user upon request.

Definition 2.5.11. - ***Business role assignment***.

Let $u \in \mathcal{U}$ be an internal user with the capability expression C', and $T' \subseteq T$ a set of tasks requested by u. Let $\mathcal{RT}'$ be the candidate business roles assignments generated by *ExtractRoles*. $\mathcal{A} \in \mathcal{RT}'$ is a candidate assignment for u to perform T' if $\mathcal{A}$ is `complete` (see *Definition 2.5.8*).

Then, for each task $t' \in T'$, there exists an *o-rule* = $(t', br, \tau(t'))$ where $br \in \mathcal{A}$ that gives to u the system role $\tau(t')$ necessary to perform t'. The set of system roles necessary to perform all the tasks in T' is: $\bigcup_{t' \in T'} \tau(t')$.

In section 2.3, we presented changes and potential security violations in organization A security system. We mentioned before that RBAC cannot address these changes and we described schemes to supplement RBAC with in order to guarantee protection of the resources and control efficiently users' access requests. *Definitions 2.5.12, 2.5.13* and *2.5.14* below formalize each of these schemes.

Definition 2.5.12. - ***Internal user - Business role Dynamic Assignment***.

Let T' be a set of tasks requested by a user $u \in \mathcal{U}$ with the capability expression C' such that $T' \not\subseteq \varphi(u)$. Let $\mathcal{RT}'$ be the candidate business roles assignments generated by *ExtractRoles* for performing T'. We say that u is assigned dynamically to a candidate assignment $\mathcal{A} \in \mathcal{RT}'$ to perform T' if :

- $\forall\ br \in \mathcal{A}\ isImpliedBy(\lambda(br), C')$.
- $\forall\ br \in \mathcal{A} : \exists\ t \in \rho(br), t \notin \varphi(u), t \notin T'\ (\delta(br) = \delta(br) \cup \{\overline{\tau(t)}\})$.

To prevent any task in a business role in $\mathcal{A}$ not assigned to u and not requested by u to be executed, the second bullet requires the associated system role to be blocked.

Definition 2.5.13. - ***Outdated user-business role assignment Filtering***.

Let T' be a set of tasks originally executed by a user $u \in \mathcal{U}$. Let C' be u current capability expression. Let $\mathcal{BR}_{T'}$ be the candidate business roles assignment generated

by *ExtractRoles* and u' s original assignment for performing T'. $\mathcal{BR}_{T'}$ is outdated if : $\exists\ br \in \mathcal{BR}_{T'} :\ \neg isImpliedBy(\lambda(br), C')$. □

Definition 2.5.14. - ***External user - Business role Dynamic Assignment***. newline Let T' be a set of tasks requested by an external user u with the capability expression C'. Let $\mathcal{RT}'$ be the candidate business roles assignments generated by *ExtractRoles* for performing T'. We say u is assigned dynamically to $\mathcal{A} \in \mathcal{RT}'$ to perform T' if:

- $\forall\ br \in \mathcal{A}\ isImpliedBy(\lambda(br), C')$.
- $\forall\ br \in \mathcal{A} : \exists\ t \in \rho(br), t \notin T'\ (\delta(br) = \delta(br) \cup \{\overline{\tau(t)}\})$. □

To prevent any task in a business role in $\mathcal{A}$ not requested by u to be executed, the second bullet requires the associated system role to be blocked.

2.5.3 Exception-handling layer

The role of the exception-handling layer is to provide authorization mechanisms for external users satisfying only the mandatory credentials.

Definition 2.5.15. - ***Partial match***.

Let T' be a set of tasks requested by an external user u with the capability expression C'. Let $\mathcal{RT}'$ be the candidate business roles assignments generated by *ExtractRoles* for performing T'. u partially satisfies the requirements for executing T' if:

- $\exists\ \mathcal{A}_i\ \in \mathcal{RT}'\ \forall\ br \in \mathcal{A}_i :\ isImpliedBy(\mathcal{MC}_{br}, C')$,
- $\forall\ \mathcal{A}\ \in \mathcal{RT}'\ \forall\ br \in \mathcal{A} : \neg isImpliedBy(\mathcal{RC}_{br}, C')$. □

Definition 2.5.15 entails: if the capabilities of u match only the mandatory credentials and not all the recommended ones, then u partially satisfies the credentials associated with br. Then, as in *Definition 2.5.10* if:
[$\forall\ t' \in T'\ \forall\ o \in \mathcal{O} \setminus \mathcal{O}(t') : P(o \mid \mathcal{O}(t)) \leq$ 1- $Sl(o)$] holds, we dynamically assign u to $\mathcal{A}_i$ (*Definition 2.5.12*) and allow it to execute T'. Otherwise, we deny the request.

Definition 2.5.16. - ***Denial***.

Let T' be a set of tasks requested by an external user u with the capability expression C'. Let $\mathcal{RT}'$ be the candidate business role assignments generated by *ExtractRoles*. We deny the request for the execution of T' if one of the following holds:

- $\forall\ \mathcal{A} \in \mathcal{RT}'\ \forall\ br \in \mathcal{A}\ (\neg isImpliedBy(\mathcal{MC}_{br}, C'))$.
- $\forall\ t' \in T'\ \forall\ o \in \mathcal{O} \setminus \mathcal{O}(t') : P(o \mid \mathcal{O}(t)) >$ 1- $Sl(o)$. □

We next introduce the algorithms for assigning users to roles and for filtering abnormal (outdated) user-role assignments. These algorithms are in *page 10*.

2.5.4 Algorithms

Algorithm 1 dynamically assigns roles to internal users requesting tasks and ensures that they have the necessary capabilities. Algorithm 1 operates as follows: in *line 1*, we verify if the requester u is authorized to execute the tasks in T', and we extract the optimal business roles assignment necessary to perform T'. Then, we verify whether the capabilities u holds satisfy the credentials of each business role in the assignment. If yes, we grant the necessary system roles, otherwise we deny the request.

Algorithm 2 dynamically assigns external users to business roles. *ExtractRoles* extracts the set of business roles necessary to perform *T'*. We compare C' with the mandatory and the recommended credentials of the extracted business roles (*lines 7-11*). If we find a match, the user is granted the necessary system roles to perform *T'*. If *C'* matches only the mandatory credentials and not all the recommended credentials, we evaluate in *line 12* whether not-needed objects may be disclosed.

Theorem 2.5.2. - **Internal User - Role assignment**.

Algorithm I computes the sets of business roles $\mathcal{BR}_{T'} \in \mathcal{RT}'$ and system roles $\mathcal{SR}_{T'}$ necessary to perform T'. It terminates and $\mathcal{BR}_{T'}$ is `complete`, `minimal` and `authorized`. □

Proof. - **Theorem 2.5.2**.

The proof of `completeness` and `minimality` follows from Theorem 2.5.1. Line 4 of Algorithm 1 ensures the `authorization` property. □

Theorem 2.5.3. - **External User - Role assignment**.

Algorithm 2 computes the sets of business roles $\mathcal{BR}_{T'} \in \mathcal{RT}'$ and system roles $\mathcal{SR}_{T'}$ necessary to perform T'. It terminates and $\mathcal{BR}_{T'}$ is `complete`, `minimal` and `authorized`. □

Proof. - **Theorem 2.5.3**.

The proof of`completeness` and `minimality` follows from Theorem 2.5.1. As expressed in *lines 6-13*, external users are `authorized` to access tasks only if they satisfy the mandatory credentials and the disclosure of not needed information is considered low. □

Algorithm 1: **Internal user - role assignment filtering** & **Dynamic assignment.**

Input : T': set of tasks requested by $u \in \mathcal{U}.$, C' capabilities of the requester u.

Output: $\{\mathcal{BR}_{T'}, \mathcal{SR}_{T'}\}$: minimal and complete set of business roles and system roles.

`/* user-role assignment filtering */`

1 **if** $T' \subseteq \varphi(u)$ **then**

2 $\quad \{\mathcal{BR}_{T'}, \mathcal{SR}_{T'}, \mathcal{RT}'\} \leftarrow ExtractRoles(T')$;

3 $\quad$ **foreach** $br \in \mathcal{BR}_{T'}$ **do**

4 $\quad\quad$ **if** $\neg isImpliedBy(\lambda(br), C')$ **then**

$\quad\quad\quad$ `/* capabilities do not satisfy requirements */`

5 $\quad\quad\quad$ **return** $\emptyset$;

6 $\quad$ **return** $\{\mathcal{BR}_{T'}, \mathcal{SR}_{T'}\}$;

`/* dynamic assignment of business roles and system roles */`

7 **else**

8 $\quad$ *repeat steps 2-6*

Algorithm 2: **External user - Business role Assignment.**

Input : T', set of tasks requested by u, C', capabilities of the requester u.

Output: $\{\mathcal{BR}_{T'}, \mathcal{SR}_{T'}\}$, minimal and complete set of business roles and system roles.

1 $\{\mathcal{BR}_{T'}, \mathcal{SR}_{T'}, \mathcal{RT}'\} \leftarrow ExtractRoles(T')$;

2 **repeat**

3 $\quad \mathcal{BR}_{T'} \leftarrow Dequeue(\mathcal{RT}')$;

4 $\quad bool \leftarrow false$; `/* match not yet found.` */

5 $\quad$ **foreach** $t' \in T'$ **do**

$\quad\quad$ `/* verify credentials satisfaction` */

6 $\quad\quad$ **foreach** $br' \in \mathcal{BR}_{T'}$ **and** $t' \in \rho(br')$ **do**

7 $\quad\quad\quad$ **if** $\neg isImpliedBy(\mathcal{MC}_{br'}, C')$ **then**

8 $\quad\quad\quad\quad$ *go to step 3*

9 $\quad\quad\quad$ **if** $\neg isImpliedBy(\mathcal{RC}_{br'}, C')$ **then**

10 $\quad\quad\quad\quad$ $\mathcal{O} \leftarrow \{\mathcal{O}(t) : t \in \rho(br), t \neq t'\}$;

$\quad\quad\quad\quad$ `/* inference control` */

11 $\quad\quad\quad\quad$ for each $o \in \mathcal{O}$, if $P(o \mid \mathcal{O}(t')) > 1 - Sl(o)$ *go to step 3*

12 $\quad bool \leftarrow true$; `/* match found.` */

13 $\quad$ `/* generate` $\mathcal{SR}_{T'}$ `as in step 5 in` **Extract roles** */

14 $\quad$ **return** $\{\mathcal{BR}_{T'}, \mathcal{SR}_{T'}\}$;

15 **until** $\mathcal{RT}' = \emptyset$ **or** $bool = true$;

16 **return** $\emptyset$;

2.6 Concluding Remarks

Business Driven Role Based Access Control (BD-RBAC) is a flexible model that incorporates business context into RBAC. BD-RBAC explicitly defines tasks and roles and relates them to system level roles. BD-RBAC also supports partial authorization through the exception handling module. A significant contribution of our model is the ability to easily adapt information access control policies to organizational changes.

In addition, BD-RBAC dynamically assigns users to roles in an intuitive manner based on the users' capabilities. We have presented tractable algorithms to generate minimal and complete business roles and system roles. One of the key findings of this work was to realize the benefits of separating the monolithic concept of role in RBAC into business role and system role in order to resorb the gaps between the implementation of an organization security policy and its specification.

Our results will be submitted to the ACM Symposium of Access Control Models and Technologies (SACMAT, Newark), in January 13^{th}, 2012.

Procedure:	**Extract roles** – *Minimal and complete business role assignments.*
INPUT	$T' = \{t'_1, .., t'_k\}$
OUTPUT	$\{\mathcal{BR}_{T'}, \mathcal{SR}_{T'}, \mathcal{RT}'\}$.

METHOD

1. For each $t'_i \in T'$, let $H[t'_i] = \{br \mid (t'_i, br, \tau(t'_i))$ is an *o-rule*$\}$.
2. Maintain a hash table K and a directed acyclic graph G. Let s_0 and f (dummy nodes) be the source and end nodes of G defined as follows.
   - For each $br \in H[t'_1]$ create the edge (s_0, br), and let $K[br] \leftarrow \rho(br) \setminus T'$. Let s_1 be a dummy node. For each $br \in H[t'_1]$ create the edge (br, s_1).
   - For each $br' \in H[t'_2]$, create the edge (s_1, br'), and let $K[br'] \leftarrow \rho(br') \setminus T'$. Let s_2 be a dummy node. For each $br' \in H[t'_2]$ create the edge (br', s_2).
   - Apply the previous bullet for all $t'_j \in T'$, $3 \leq j \leq k$.
   - Finally, for each $br_l \in H[t'_k]$, create the edge (br_l, f), and let $K[br_l] \leftarrow \rho(br_l) \setminus T'$.
3. For each node $br \in G$, maintain a set of its predecessors π. Set initially $\pi \leftarrow NIL$. Modify `BFS` to return all complete paths of G (from s_0 to f) as follows.
   - For each visited node in G, enqueue all its predecessors.
   - If the node br being visited is f, then $\{\pi(br), br\}$ is a set of nodes that form a complete path from s_0 to f.
   - Define $\mathcal{Q} = \{\{\pi(br), br\} \setminus \{s_0, ..., s_{k-1}, f\}\}$ the set of sets of intermediate nodes $\mathcal{A}$ of all the complete paths from s_0 to f generated by the modified `BFS`.
4. Let $\mathcal{RT}' = \{\mathcal{A}' \in \mathcal{Q} \mid F(\mathcal{A}') = \min_{\mathcal{A} \in \mathcal{Q}} F(\mathcal{A})\}$ where $F(\mathcal{A}) = \sum_{br \in \mathcal{A}} \sum_{t \in K[br]} Sl(\mathcal{O}(t))$. Select a business role assignment $\mathcal{BR}_{T'} \in \mathcal{RT}'$. Iteratively, if there exists $\mathcal{A}_i \in \mathcal{RT}'$ such that $\mathcal{A}_i \subset \mathcal{BR}_{T'}$, change $\mathcal{BR}_{T'}$ to $\mathcal{A}_i$.
5. Extract $\mathcal{SR}_{T'} = \{\tau(t) \mid t \in \rho(br) \setminus K[br],\ br \in \mathcal{BR}_{T'}\}$ the set of system roles.
6. Return $\{\mathcal{BR}_{T'}, \mathcal{SR}_{T'}, \mathcal{RT}'\}$.

Figure 2.3: H in line 1 is obtained by querying the relational table *o-rule*. $\mathcal{SR}_{T'}$ in line 6 is obtained by querying the table *s-rule*. `BFS` for Breadth-first Search is an algorithm for searching a graph. The dummy nodes $s_0, ..., s_{k-1}, f$ are introduced just to optimize the number of edges of the connected graph G. They are at even positions in $\{\pi(br), br\}$, thus can be removed easily. *ExtractRoles* and `BFS` have the same complexity $\Theta(|V| + |E|)$.

Chapter 3

P2CR: A Pratical Framework for XACML Policies Composition and Conflicts Resolution in the Clouds

In this chapter, we present our current research results addressing the problem of policy composition in a community cloud environment where tenants specify independently their access control and want to retain control over their own resources.

3.1 Introduction

Recent years have witnessed a growing number of special-purpose communities in which different organizations (or tenants) with common interests and needs interact and share pools of configurable resources governed by a cloud service provider. Community clouds have many advantages. They enable organizations that are technologically different and geographically separated to collaborate in a seamless manner. However, they can be difficult to manage especially when the tenants have differing access policies. The diversity of the policies of the entities may lead to serious obstacles in establishing a safe collaboration within the cloud. An important requirement for precisely achieving this goal is that each entity, tenant as well as cloud service

provider, abides by the security, compliance and risk management requirements of the others. Thus, to allow the entities to interact safely, their access policies must necessarily be compared and composed.

Leveraging the community clouds as an illustrative example, we address in this chapter the policy composition problem in a broader scenario in which different entities (e.g., owners of computing resources – *memory, computing power, bandwidth, etc.* – and services) are interested in composing their independently stated access policies while retaining their autonomy i.e., maintaining the control over their own resources. A non-trivial challenge generally faced in this context is the occurrence of conflicts. Two access policies may apply to same objects and yield upon request of the objects contradictory evaluation results. Access control systems governed by such policies cannot deterministically decide whether to grant access to the requested objects or to deny the access. Consequently, they may even allow certain users to access resources they are not authorized for or deny the access to the legitimate ones. Thus, to enable access policies in individual systems to unambiguously evaluate users requests, many conflict resolution strategies have been proposed [18, 7, 19] among which:

- *Most specific* (*general*) *takes precedence*: A rule that applies to a more specific (general) entity (subject or resource) takes precedence over a rule that applies to a more general (specific) entity e.g., a rule that applies to a user takes precedence over a rule that applies to a group, or a rule that applies to a subfolder or file takes precedence over a rule that applies to the subfolder's or file's parent.

- *Deny takes precedence*: Deny rules takes precedence over Permit rules.

- *Order-based precedence*: Rules are totally ordered, usually based on time (most recent in time takes precedence) by the policy author, so the author can explicitly state which rules take precedence over others.

However, these strategies do not address conflicts that arise in situations where

several entities want to integrate their independent access policies. Conflicts that occur in this scenario are difficult to eliminate because of the diversity of the policies of the entities, and more importantly because of the conflict resolution strategies that they use.

Currently, no effective technique exists for resolving these conflicts while the policies are being integrated. An intuitive approach could however be to pick the conflict resolution strategy of a random entity and adopt it as the conflict resolution technique of all the policies, or to deny access to resources to which conflicting policies apply, thereby preserving the confidentiality and integrity of the resources though at the cost of restricting availability. Unfortunately, because each entity enforces the strategy it finds more suitable to its needs, such an approach would result in many cases inconclusive. A typical example is two entities, A that applies the *Deny-overrides* [19] scheme to restrict access to its resources, and B that uses the *Permit-overrides* [19] method to ensure the availability of its data. In this case, if the strategy that B uses is applied, then resources of A may be accessed by unauthorized users. Conversely, if we opt for the strategy of A, then access to resources of B may be severely restricted.

Over the past years, considerable work in composing independently stated access policies has been carried out [31, 35, 12, 10, 40]. The approach common to many of the proposed studies is to combine these policies based on the priorities they are assigned with. Assigning priorities to policies is however difficult especially if the number of the policies is high, and understanding them even more because the priorities are generally represented as numbers and no semantic is attached to them to reflect their meaning [2, 30]. In addition, in many studies, the composite policy resulting from the integration of the policies is enforced in only one point. What this entails is either the party that administers the single point of enforcement is heading all the entities, or it is mandated by them to combine their individual access policies and to manage the resulting policy. However, entities that are interested in combining their policies may

be under the authority of different parties or reluctant to part with the administration of their resources. Lastly, in many proposals, conflicts are detected manually (e.g., [2]) and their causes usually overlooked (e.g., [35]). However, without a precise knowledge of what causes a conflict, it is difficult to guarantee the effectiveness of the solution that one would adopt to resolve the conflict. Furthermore, in a context where the number of policies to compose is high and the policy composition task is automated, considering the causes is essential to a successful resolution of the conflicts.

In this chapter, we propose a policy composition and conflict resolution (P2CR) framework to integrate XACML access policies and a novel technique to merge their decision results. We provide first a precise and non-ambiguous specification of the policies in logic form, then we detect conflicting policies and identify the causes. The advantage of using a logic representation of the policies is that it allows us to reason about the correctness and the soundness of the algorithms that we put in place to detect conflicts and identify the causes. Finally, by means of the annotations that the entities attach to their policies, we remove if possible the conflicts. Otherwise, we rely on a defeasible logic theory to resolve them. The annotations carry important information for allowing a conflict-free policy integration. Essentially, they reflect adjustments that the entities are willing to bring in their policies in a way to prevent potential conflicts during the composition. Intuitively, the entities, aware that maintaining their policies as they are may render their composition impossible, anticipate then the conflicts and describe their means to obviate the conflicts via the annotations. As an example, an entity may annotate a condition in its policy as recommended to indicate that the satisfaction of the condition is not compulsory or, restrict the scope of the policy. In comparison to many policy composition approaches which are mainly negotiation-oriented or assign priorities to policies, our approach combines access policies without prioritizing any of the participants involved in the composition, and more importantly, the resulting composite policy appears flexible

and easily adjustable to changes in any of the participants' systems with no impacts on its structure.

The remaining of the chapter is organized as follows. Section 3.2 surveys XACML and Defeasible Logic. We provide in section 3 a motivating example to give an overview of the problems that we address. In section 4, we present the annotations the entities add to their policies. We introduce P2CR in section 5 and describe its core components in section 6. In section 7, we show how to enforce the policy created by P2CR with respect to each entity systems and resources status. Section 7 discusses the related work. We conclude in section 8 and give suggestions for future work.

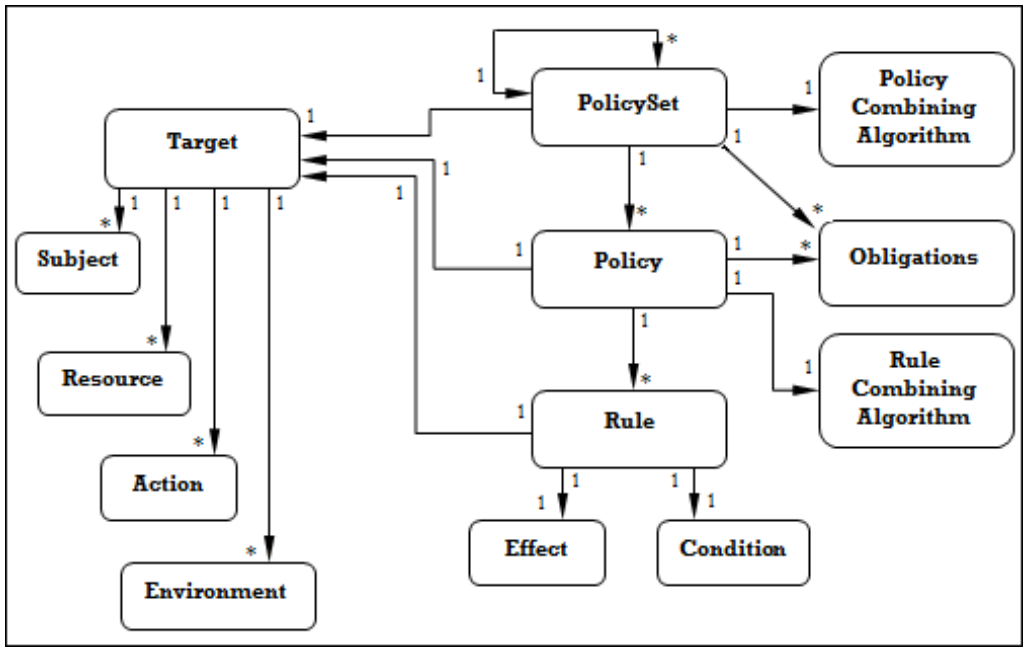

Figure 3.1: XACML context [19].

3.2 Background Information

Before proceeding further, we first present some necessary background for understanding the remainder of the chapter.

3.2.1 Overview of XACML policies

XACML is an access control language standardized by OASIS (Organization for the Advancement of Structured Information Standards) for the specification of access control policies. XACML is widely adopted in many domains especially in service oriented environments as the language of choice for expressing access control policies. The core components of an XACML document are (see *Figure 3.1*):

PolicySet. A policy set is a container that comprises policies or policy sets along with a `policy combining algorithm`, and eventually references to other policies found in remote places.

Policy. A policy consists of a `Target`, a set of `Rules`, a `rule combining algorithm`, and `Obligations`. The target contains three components or target attributes (*Subject, Action, and Resource*) that identify requests to which the policy is applicable.

Rule. A rule is the most basic unit of an XACML policy that evaluates access requests. It consists of a `Target` that has a structure similar to the target of an XACML policy, an `Effect` which is either permit or deny, and `Conditions`. The conditions refine further the applicability of the rule established by the target and denote the constraints that must be satisfied for a request to be permitted or denied by the rule as specified in its Effect.

Obligations. Obligations are functions that must be executed in conjunction with the enforcement of an authorization decision (a permit or a deny decision). XACML defines two types of obligations: positive obligations and negative obligations.

Combining algorithm. A rule (policy) combining algorithm is used as a procedure for resolving conflicts among rules (policies) that yield upon a request different evaluation results. It combines individual evaluation results of rules (policies) into one final decision. Rule and policy combining algorithms are semantically similar. XACML defines four rule (policy) combining algorithms:

- `Permit-overrides(PO)` evaluates to `Permit` if there is at least a rule (policy) that

evaluates to `Permit`, and to `Deny` if no rule (policy) evaluates to `Permit` and there is at least one rule (policy) that evaluates to `Deny`.

- `Deny-overrides(DO)` evaluates to `Deny` if there is at least a rule (policy) that evaluates to `Deny`, and to `Permit` if no rule (policy) evaluates to `Deny` and there is at least one rule (policy) that evaluates to `Permit`.

- `First-applicable(FA)` evaluates to the decision of the first applicable rule (policy) in a policy (policy set).

- `Only-one-applicable(OA)` evaluates to the decision of the unique rule (policy) in the policy (policy set) that applies. If only one policy is applicable, the decision is that of the policy. If no policy or more than one policy is applicable, then the decision is `NotApplicable`.

Because `FA` and `OA` evaluate each rule (policy) in the order that it appears in an XACML policy (policy set), for simplicity, we consider in the remaining of this chapter only policies which combining algorithms are either `PO` or `DO`.

In addition to the rule (policy) combining algorithms, XACML defines five actors to handle access decisions (see *Figure 3.2*): Policy Enforcement Point (PEP), Policy Administration Point (PAP), Policy Decision Point (PDP), Policy Information Point (PIP), and a context handler. The PEP is responsible for making access control decision requests to the PDP and the enforcement of the given decisions. The PDP is the decision point for the access requests. It approves or denies an access request by matching it against applicable policies. The PAP is the repository for the policies. Upon a request, it provides to the PDP the policies to evaluate the request. The PIP is the point where the attributes necessary for the policy evaluation are retrieved. The attributes can be retrieved from the resource to be accessed, the environment (system e.g., time), the subjects, etc. The context handler translates requests made by users to XACML canonical format.

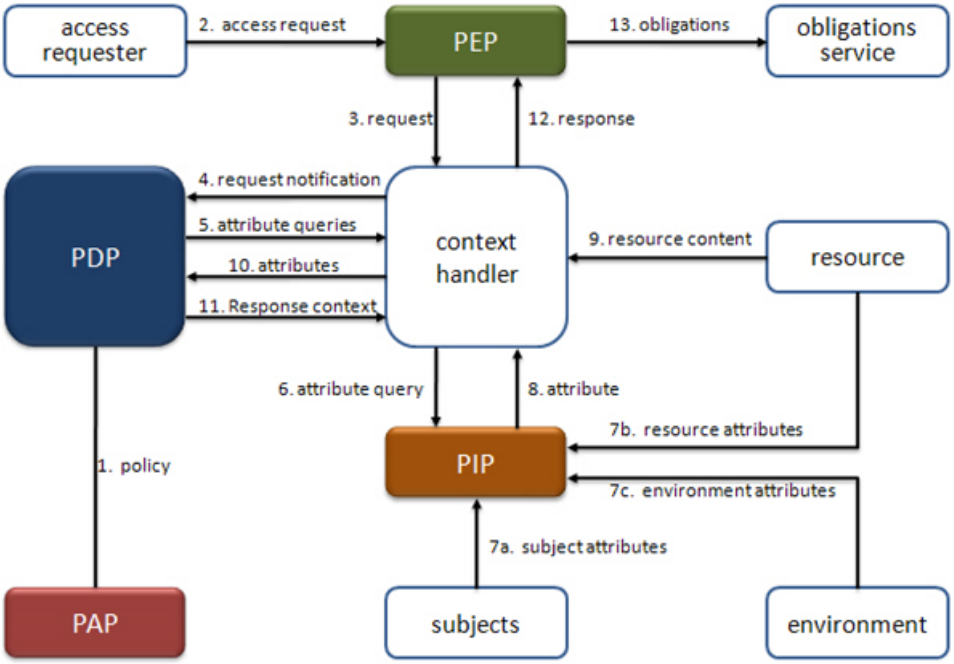

Figure 3.2: XACML Actors and Data flow [19].

3.2.2 Defeasible Logic

When composing access policies stated independently by different entities, conflicts may occur any time during the composition process. Thus, there is clearly some uncertainty on whether the policies are composable or not. In this section, we present a type of logic, defeasible logic, which particular aspect is to allow reasoning over partial, incomplete and inconsistent knowledge. With respect to policy composition, this inconsistent knowledge can be interpreted as the presence of conflicting policy components or rules within the set of policies to compose.

Defeasible logic is of the family of non-monotonic logics that aim to model human common-sense reasoning. In real life, humans tend to make decisions based upon partial evidence and revise their conclusions as new information is acquired [30]. For example, if we know that the policy P_1 is applicable to a resource o, and that upon request of o, P_1 can evaluate the request, then the decision rendered by the P_1 is the effective one. However, if we find another policy P_2 applicable to o that yields

an opposite decision, then we cannot consider any of the two decisions as these two policies are conflicting. Defeasible logic is simple and computationally efficient [23], and it has been used in many areas such as automated negotiation, contracts [23], and for formalizing XACML policies [29].

Reasoning with defeasible logic requires defining a defeasible theory that consists of *facts, strict rules, defeasible rules, defeaters*, and *the inference mechanism* also referred to as *superiority relation*.

`Facts` are indisputable statements, such as $Bird(dove)$ to state that *"a dove is a bird"*.

`Strict rules` are rules in the classical sense, that is whenever their premises are indisputable then so are their conclusions. An example of a strict rule is *"professors are faculty members"* which is formally represented as

$$Professor(x) \rightarrow FacultyMember(x)$$

`Defeasible rules` are rules that can be defeated by contrary evidence. In the context of policy composition, for example, two rules that conflict defeat each other depending on which rule is evaluated first. A more illustrative example is *"people giving lectures are faculty members"* [23], formally represented as

$$GivesLectures(x) \Rightarrow FacultyMember(x)$$

`Defeaters` are used to prevent some conclusions. An example is *"tutors might not be faculty members"* [23] which is formally represented as

$$Tutors(x) \rightsquigarrow \neg FacultyMember(x)$$

Defeasible and defeater rules together define the knowledge base of the defeasible reasoning theory.

The `superiority relation` ($\succ$) defines a partial ordering of the non-strict rules. In case of conflicts, Defeasible logic usually relies on a hierarchy (priority information) defined over the defeasible rules and expressed by means of the superiority relation to resolve the conflicts. For example [23],

r: $GivesLectures(x) \Rightarrow FacultyMember(x)$

r': $GuestLecturer(x) \Rightarrow \neg FacultyMember(x)$

$r' \succ r$

means that the conclusion of r' overrides that of r. Without the priority information "$r' \succ r$", we would not be able to conclude whether "*a guest lecturer is a faculty member*".

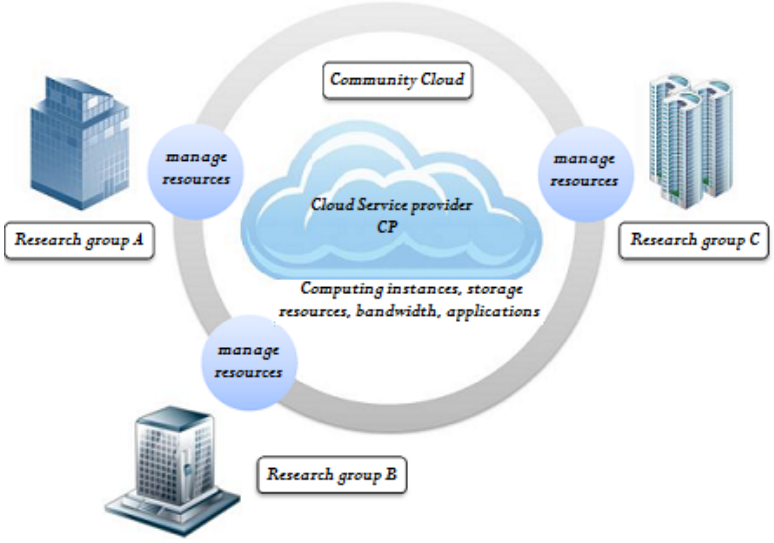

Figure 3.3: Community cloud.

3.3 Motivation and Overview

Consider initially a cloud in which three research groups A, B and C with common research interests deploy their resources on leased server instances owned and maintained by CP the cloud service provider. Assume A, B, and C manage access to their respective resources – *technical reports, sources, documentations, research applications* – by means of their XACML policies, and that the bandwidth usage of C is monitored by CP. Assume that A uses a `DO` combining algorithm, B and C a `PO` combining algorithm. Suppose now that A, B, and C want to transform the cloud to a community cloud (*Figure 3.3*) where their staffs can share research results and collaborate in projects, each group retaining its control over its own resources.

Policy A. *Senior researchers, researchers and interns can read, write or download technical reports of any size except from 21:00 to 23:00 when the systems are backuped. Researchers in general with less than 10 publications and 5 years of experience cannot modify validated reports.*

Policy B. *Only senior researchers can change reports already validated, and this only between 22:00 and 00:00. Interns are denied access to the reports after 20:00. A senior researcher is a researcher with minimum 7 publications and 4 years of research experience at least.*

Policy C. *Interns cannot download reports of size greater than 3GB. CP ensures that the cumulative downloads do not exceed 10GB per day and that files of size greater than 3GB cannot be downloaded between 10:00 and 11:00.*

1.	*An intern requesting access to validated reports of size less than 3GB from 20:00 to 21:00.*
	Evaluations. *A, C: Permit, B: Deny.*
2.	*An intern downloading technical reports of size less than 1GB from 11:00 to 12:00.*
	Evaluations. *A, B, C: Permit, CP*[1]*: Deny*
3.	*A senior researcher requesting access for making changes to a validated report from 23:00 to 00:00.*
	Evaluations. *A: Deny, B: Permit.*

Table 3.1: Occurrences of conflicts.

As we notice, the requirements that A defines for accessing the *technical reports* are different from those of B, while C imposes no restrictions except on the size of the reports being accessed. Moreover, unlike A that denies researchers with *less than 10 publications* and *5 years of experience* to modify *validated reports*, B allows instead senior researchers with *at least 7 publications* and *4 years of experience* to change already validated reports. These differences between the policies may lead

[1]The downloads of the day when cumulated exceed *10GB*.

to the scenarios in *Table 3.1* in which conflicts occur. To resolve these conflicts, we may be tempted to apply the combining algorithm of either *A, B* or *C*. However, this would undermine the safety or availability requirements of the entities. As an example, applying the algorithm of A to resolve the conflict in *row 3* of *Table 3.1* results in denying senior researchers of B from carrying changes on validated reports. If however we use the algorithm of B, then even non senior researchers in A will be able to modify validated reports of B.

The question that we then address here is whether this yet-to-create community cloud would preserve the security requirements of each entity. The answer to this question lies in how we construct the policy that combines the access specifications of the entities. In some cases, such a policy may be impossible to create. However, with the annotations each entity adds to its policy, we argue that it may be possible to define this composite policy while satisfying the entities requirements. Our aim is thus to create this policy based exclusively on the access specifications and the annotations each entity provides. The policy shall be flexible to render it adjustable to the status of the systems of the entities it is built to protect and to minimize potential changes on its structure. This is particularly important because in some situations (e.g., system outages or shortage of resources) an entity may be compelled to modify its policy, thereby impacting the composite one.

Thus, we restate this objective as follows. Given k complete and consistent access policies $P_1 = \{r_{11}, ..., r_{n1}\}$, $P_2 = \{r_{12}, ..., r_{n2}\}, ..., P_k = \{r_{1k}, ..., r_{nk}\}$ defined independently by k autonomous entities $A_1, A_2, ..., A_k$ along with their annotations, how to possibly define a globally enforceable meta-policy $\mathcal{Q}$ such that:

- *Completeness.* Rules in $\mathcal{P}_1, ..., \mathcal{P}_k$ over same resources are enforceable with $\mathcal{Q}$.
- *Consistency.* $\mathcal{Q}$ is conflict-free.
- *Minimality.* There exists no authorization rule in $\mathcal{Q}$ such that if removed the

behavior of $\mathcal{Q}$ remains unchanged.

The problem that motivates our work may be perceived as similar to trust negotiation. Winslett et al. [47] defined this paradigm as "an approach to establishing trust between parties so that interaction can take place, through the use of access control policies that specify what combination of digital credentials a stranger must disclosed to gain access to a local resource". However, our work differs from trust negotiation in four aspects. *i.*) Our goal is to determine from the access policies that the entities provide a composite policy free of conflicts that satisfies the entity security requirements. *ii.*) We consider policies of the form *subject-action-target* (*SAT*) written in XACML rather than digital credentials. *iii.*) No negotiation takes place in our scenario, and each entity retains its authority over its resources. *iv.*) The annotations that we introduce are used only in case the policies to compose are conflicting.

3.4 P2CR Input Policies

We now explain the policies we attempt to compose. We refer to them as meta-policies. These meta-policies are stated independently by different entities and consist of XACML policies enriched with annotations. Likewise, the annotations are also specified by the entities to indicate assertions to modify in their meta-policies so as to remove conflicts that occur during the policy composition. Each entity annotates its policy separately i.e., with no prior knowledge over the policies of the other entities, and this based on the requirements in its policy that it considers as mandatory to satisfy and those that are under certain circumstances only recommended. Note that although in some cases the annotations may help to eliminate conflicts, in others, however, they may be irrelevant. Below, we describe the annotations the entities attach to their policies.

3.4.1 Adjustment of Conditions

Generally, two `XACML` rules conflict if they have the same target, different effects, and their conditions are mutually satisfiable. As an example, for an object o, two entities A and B may possess each a rule, r_1 for A in which it authorizes o to be accessed only by users who satisfy the constraints *"email = .edu"* and *"Citizenship = US"* i.e., users who have a *".edu"* email and are of citizenship *"US"*, and r_2 for B in which it denies access to object o to all users who satisfy the constraint *"email = .edu"*. Here, *email* and *Citizenship* are condition attributes, *".edu"* and *"US"* their domain of permitted values. Cleary, r_1 and r_2 conflict since they will evaluate requests from the users with a *".edu"* email to different decision results.

The constraints over the condition attributes of an `XACML` rule define the `Conditions` of the rule. Upon a request, their truth must be evaluated for the rule to be applicable to the request and yield a decision. Sometimes, however, the decision rendered may simply depend on the satisfiability of some of the constraints but not all of them i.e., those the entity owner of the rule deems more relevant; hence the need for more flexibility in their specification. As in the example before, entity A could have specified "*email = .edu*" as only recommended but not strictly required; in which case the conflict would not occur. Similarly, *section 3.3*, A could have indicated the "*5 years experience*" as simply recommended; allowing thus *senior researchers* with "*4 years experience*" to modify *validated reports*.

The `Conditions` of an `XACML` rule may however be empty, in which case the rule is applicable to any request that matches its target. Such a rule conflicts with any rule that has the same target and a different effect regardless of whether the `Conditions` of the latter rule is empty or not. Thus, for an entity that has a rule with no conditions in its policy to possibly obviate conflicts between its policy and other policies, an appropriate approach would be that it restricts the scope of the rule. As an example, entity A in *section 3.3* may have a rule r that allows *interns* to access any *report*. For

the purpose of composing its policy with that of B, A may realize that maintaining r as it is may cause their policies to conflict, especially if B also hires *interns*. Then, A may decide that r applies only to *US citizen interns*. The constraint *"Citizenship = US"* becomes therefore mandatory (i.e., required) for any intern to access reports of A as part of its collaboration with B. In the remaining, we refer to attributes that restrict the scope of a rule such as *Citizenship* as *scope restriction attributes*, and we consider that rules with empty conditions are supplied with such attributes.

3.4.2 Activation of Requirements

Conflicts may also occur when the amount of target objects available is limited because of outages that an entity may face or as a result of high demands it cannot handle in particular times. To prevent additional overheads that could stem from users requests, and to guarantee at the same time a minimum level of availability of its resources to its users, the entity may consider to adjust temporarily some of its authorization rules. As an example, an entity that is confronted with bandwidth bottlenecks may change all *download* actions to *read only*, it may *block* access to resources that are of *size* greater than a certain threshold. After the outage, to compensate activity losses, the entity may extend the *access time* it has set before in order to enable more users to gain access to its resources. For such changes to take effect in a system though, they must be dictated by the underlying environment e.g., *status of a system, resources availability, usage peaks*. With XACML, this contextual information can precisely be encoded as attributes and stored in the XACML policy information point (`PIP`) for later use during the policy evaluation process by the XACML policy decision point (`PDP`).

With the annotations, we provide the means to express the activation of these requirements. Each entity may use the annotations to specify the actions that it allows to be taken with respect to the status of its system or resources. As one can notice, conflicts that occur because of outages or shortage of resources, also referred

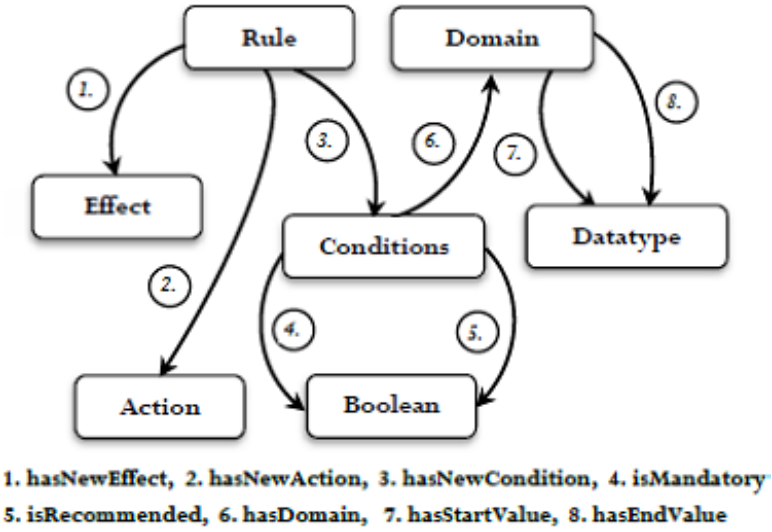

Figure 3.4: OWL schema for XACML policy annotations.

to as dynamic conflicts, can be checked only at runtime. Because our framework detects only static conflicts, also known as modality conflicts, it cannot therefore eliminate these dynamic conflicts even with the annotations that the entities supply. However, what we aim for here is to render the composite policy flexible enough and dynamically adjustable to the status of the system where such conflicts arise.

3.4.3 Adding Annotations into XACML

We now explain how the annotations can be formalized and added into `XACML` policies. As a basis for this formalization, we use the `OWL` ontology schema depicted in *Figure 3.4*. The boxes in the schema represent `OWL` classes while the labeled arrows are `OWL` properties. In this section, we focus mainly on the components `Conditions`, *hasDomain*, *isRecommended isMandatory*, and *hasNewCondition*. These are the components that the entities leverage in order to annotate their `XACML` policies. The remaining components, which we illustrate in *Figure 3.4* only for ease of understanding, will be covered in section 7.

The class `Conditions` models the adjustment of conditions of section 4.1. As illustrated in *Figure 3.4*, it models only one atomic constraint. To annotate more than a constraint in a rule, we instantiate the class `Conditions` commensurately to

the number of constraints in the `Conditions` element of the rule. The `OWL` class `Conditions` is associated with the `OWL` property *hasDomain*. It specifies the domain of values for which a constraint, instance of the class `Conditions`, is *isRecommended* or *isMandatory*. A constraint cannot however be annotated as mandatory and recommended at the same time. Thus, we specify in the ontology that *isRecommended* and *isMandatory* are disjoint properties by adding the `owl:disjointWith` construct. To annotate a constraint of a rule as mandatory or recommended, it suffices to declare as specified *Table 3.2* the *AttributeId* of the attribute on which the constraint is defined as a `URI` pointing to an object in an ontology. Properties of that object can be extracted by means of the `RDQL` query language. An example of a query that extracts the constraints annotated as recommended in rule r_1 is given below[2]. To restrict the scope of a rule, one can simply add constraints to it by instantiating the `OWL` property *hasNewCondition*. *hasNewAction* of r_1.

Select ?c From ⟨http://www.onto_orga.rdf⟩ Where (?x, ⟨ciae:hasNewCondition⟩, ?c), (?c, ⟨ciae:isRecommended⟩, True) AND ?x = "r_1"

Element	***Annotations***
Condition	*⟨AttributeSelector Datatype = "http://www.w3.org/2001/XMLSchema#int" AttributeId="uri:http:// www.orga.com/onto_orga.rdf#age"⟩*

Table 3.2: An example of annotations.

3.5 P2CR Framework

Our framework, `P2CR`, the architecture of which is sketched in *Figure 3.5*, consists of two levels: the policy translation level and the policy composition level.

The *policy translation level* is composed by the `policy translation module` which receives a set of meta-policies $\mathcal{P} = \{P_1, ..., P_k\}$ to compose, each meta-policy

[2] `ciae denotes the namespace of the ontology.`

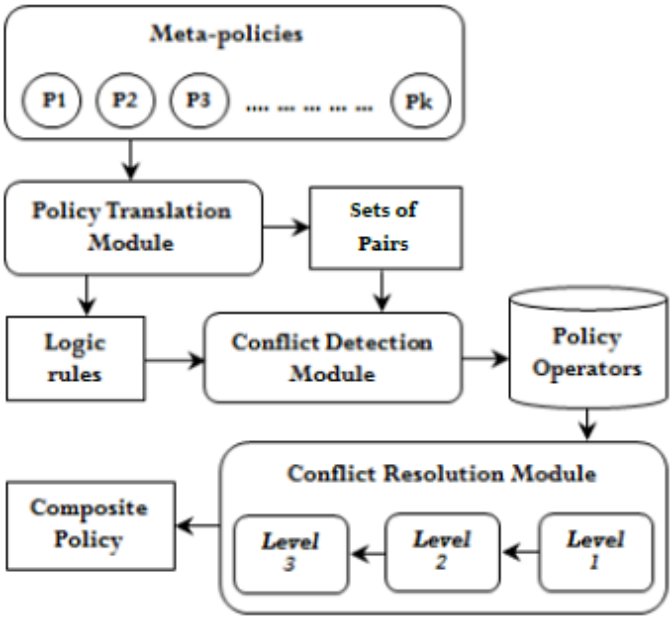

Figure 3.5: P2CR Architecture.

consisting of `XACML` rules and annotations. The policy translation module converts each `XACML` rule of a meta-policy to a set of logic expressions, and the annotation attached to it to a set of attribute-domain pairs, where in each pair the domain is either recommended or mandatory for the attribute. The logic expressions define the set of defeasible rules that are used in the policy composition level for reasoning purposes.

The core components of the *policy composition level* are the `conflict detection module` and the `conflict resolution module`. The conflict detection module verifies whether the sets of rules that it receives from the policy translation module present conflicts; in which case the conflict detection module identifies the causes. Finding conflicts in the collection $\mathcal{P}$ entails deriving a contradiction i.e., finding any two rules in $\mathcal{P}$ with mutually satisfiable conditions, that have the same target, and different effects. In case the conflict detection module finds conflicts, it selects the policy operators to apply for removing the conflicting space —*the collection of values for which the conditions of the two rules are mutually satisfiable*—, fetches the annotations attached to the conflicting rules, and sends the rules, the operators and the

annotations altogether to the `policy integration module`. The policy operators that `P2CR` supports are:

Intersection ($\cap$) is useful for finding mutually satisfiable `XACML` rule `Conditions` elements.

Scoping restriction ($\hat{}$) adjusts a rule to a set of subjects, objects, actions and conditions based on the status of a system or the availability of some resources. Scoping is particularly useful for managing runtime conflicts.

Set difference ($\setminus$) restricts a rule by removing the values for which the condition attributes of the rule are annotated as recommended.

The use of each of the policy operators will be clear shortly. The `conflict resolution module` is in charge of removing or resolving the conflicts. It is composed of three levels. The process for removing in *level-1* a conflict between any two rules consists of examining the annotations attached to the condition attributes of the conflicting rules, and determining whether the collection of permitted values in the conflicting space of one of the attributes can be removed i.e., if the condition attribute is annotated as recommended for that specific range of values. In case the annotations attached to the conflicting rules cannot remove the conflict, the conflict resolution module sends the rules to its second level, *level-2*. The purpose of *level-2* is to establish a precedence relation over the rules it receives from the conflict detection module. For this, we construct a defeasible logic theory, the sets of defeasible rules and defeaters of which are composed by the translated rules and the conflicting rules. When, no ordering can be of these rules established, the conflict resolution module, via its third level, *level-3*, determines the constraints in the conditions of any of the rules such that if annotated as recommended can then help remove definitely the conflicts.

Informally, the interaction between the two levels can be summarized as follows. The policy translation module converts the meta-rules to logic expressions such that conflicting meta-rules become inconsistent logic expressions. The conflict detection

module verifies if the sets of the logic expressions presents conflicts. If any, it finds the causes of the conflicts, attempts to remove them, or simply sends that information to the reasoning module. The mapping function that translates the meta-policies in $\mathcal{P}$ to logic expressions $\mathcal{L}$ is defined by

$$\begin{array}{rccc} \zeta: & \mathcal{P} & \rightarrow & \mathcal{L} \\ & x & \mapsto & \zeta(x) \end{array}$$

3.6 P2CR Core Components

In this section, we describe the core components of P2CR, namely the policy translation module, the conflict detection module and the conflict resolution modules.

3.6.1 Policy Translation Module

In this section, we start first with showing how to translate a standard XACML rule to a logic expression. We then extend this translation to provide a logical formalization of meta-rules and meta-policies.

Given an XACML rule in xml format, we can parse it to identify its subject, action, resource, and condition attributes. Once, these attributes are identified, we can then transform any XACML rule into logic expressions.

Example 3.6.1. - *A Parsed XACML Rule x.*

Target	***Subject***: *email = mycloud.com, email = .edu*
	Action: *read, download.*
	Resource: *reports.*
Effect	*Permit.*
Condition	*Citizenship = "US"* $\wedge$ $t \geq$ 8am $\wedge$ $t \leq$ 10am.

The domain of permitted values for *Citizenship* in the rule x ($dom(x, Citizenship)$) is *"US"*, and that of t ($dom(x, t)$) is any time between 8am and 10am.

Definition 3.6.1. - *Generic Rule*.

Let x be an XACML rule, $S(x)$, $A(x)$, and $R(x)$ be respectively its finite set of subject, action, and resource attributes.

We formalize the `Target` of x as $target(s, a, o)$, a ternary predicate that returns 1 (true) if the rule x explicitly authorizes or denies a user with the subject attribute s to perform an action a on an object o, and 0 (false) if not.

$$target(s, a, o) = \begin{cases} 1 & if\ (s, a, o) \in S(x) \times A(x) \times R(x) \\ 0 & Otherwise \end{cases}$$

We define the `Conditions` element of x as $cond(x) = c_1 \triangleright v_1 \wedge ... \wedge c_n \triangleright v_n$, where $c_i, 1 \leq i \leq n$ is a condition attribute, $v_i, 1 \leq i \leq n$ a constant, and $\triangleright \in \mathcal{O} = \{<, \leq, =, \neq, \geq, >\}$ an operator. We denote by $C(x) = \{c_1, ..., c_n\}$ the set of condition attributes of x.

An XACML rule may present alternative `Conditions` elements. In such a case, we split the rule into multiple rules, each having one `Conditions` element.

We formalize the `Effect` element of x as the placeholder $effect(s, a, o)$. If it is a `Permit`, then $effect(s, a, o)$ becomes $permit(s, a, o)$ i.e., permit a user with the attribute s to perform action a on o. If the effect is `Deny`, it becomes $deny(s, a, o)$ i.e., deny the user to perform action a on o.

The rule x is then translated to the generic expression

$$\zeta(x):\ \forall\ (s, a, o) \in S(x) \times A(x) \times R(x)\ (target(s, a, o) \rightarrow (cond(x) \rightarrow effect(s, a, o))).$$

The predicate $cond(x)$ is omitted if $C(x)$ is empty, in which case x is translated to the following expression

$$\zeta(x):\ \forall\ (s, a, o) \in S(x) \times A(x) \times R(x)\ (target(s, a, o) \rightarrow effect(s, a, o)).$$

Definition 3.6.2. - *XACML Rule*.

We define formally an XACML rule x as the non-empty set $\mathcal{X}$ of all the instances

Subject (s)	***Action*** (a)	***Resource*** (o)
$email = mycloud.com$	$read$	$reports$
$email = mycloud.com$	$download$	$reports$
$email = .edu$	$read$	$reports$
$email = .edu$	$download$	$reports$

Table 3.3: Subject, action, and resource attributes.

of $\zeta(x)$ i.e., $\mathcal{X} = \{target(s,a,o) \rightarrow (cond(x) \rightarrow effect(s,a,o)) \colon \ (s,a,o) \in S(x) \times A(x) \times R(x)\}$. Each element of $\mathcal{X}$ is referred to as an atomic rule.

Example 3.6.2. - ***Atomic Rule***.

The rule x in *Example 5.1* is formalized as the set $\mathcal{X}$ of four atomic rules with a `Permit` effect having different targets but same conditions ($Citizenship = US \wedge t \geq$ 8am $\wedge\ t \leq$ 10am). The target attributes of the atomic rules of $\mathcal{X}$ are provided in *Table 3.3*.

Example 3.6.3. - ***Annotations***.

To the XACML rule x given in *Example 5.1* can be attached the following annotations.

1. *"Citizenship = "US"" is mandatory for downloading the reports.*
2. *"Citizenship = "US"" is recommended for the users with "email = .edu" to read the reports.*

Annotation *1* applies to the principals with the subject attribute *"email = mycloud.com"* or *"email = .edu"*, whereas annotation *2* applies only to those with *"email = .edu"*. Consequently, annotation *1* is attached to each atomic rule in $\mathcal{X}$, whereas annotation *2* applies only to two specific atomic rules in $\mathcal{X}$ of target attributes the last two rows of *Table 3.3*. Non-annotated constraints (e.g., $t \geq$ 8am $\wedge\ t \leq$ 10am) are assumed by default to be mandatory.

In prelude to the formalization of the meta-rules and meta-policies, note that the target of an XACML rule may be empty. In such a case, as specified by the XACML

standard [19], we assign the target value of the policy containing the rule to the rule i.e., the constraints in the policy target are propagated down to the target of the rule. Likewise, for any XACML policy, we also propagate the constraints in the target of the policy down to its rules. We thus ignore the targets of the XACML policies.

As mentioned before, recall also that a meta-policy consists of XACML rules, each of which defined before as the set $\mathcal{X}$ is enriched with annotations. In the following, we use the term meta-rule to refer to an annotated atomic rule. For a given meta-rule, we can parse it to identify the subject, action and resource of its atomic rule and also its condition attributes and annotations. The extracted annotations are organized as sets of pairs, each pair consisting of a condition attribute or a rule restriction attribute and a range of values for which the attribute is annotated as mandatory or recommended.

Definition 3.6.3. - ***Meta-rule***.

We define a meta-rule r as the triple $r = \langle x, \delta, \mathcal{A} \rangle$, where x is an atomic XACML rule, $\mathcal{A} = \{(a_1, V_1), ..., (a_n, V_n)\}$ the annotations attached to x, and $\delta \colon C(r) \rightarrow \mathcal{V}$ a mapping function defined as follows.

$$\delta(a_i) = \begin{cases} V_i & if\ (a_i \in C(x) \wedge isRecommended(a_i, V_i)) \\ & \vee (a_i \notin C(x) \wedge isMandatory(a_i, V_i)) \\ \emptyset & if\ (isMandatory(a_i, dom(x, a_i)) \wedge \\ & a_i \in C(x)) \end{cases}$$

where $C(r) = \{a_1, ..., a_n\}$, each $a_i, 1 \leq i \leq n$ being either a **condition attribute** ($a_i \in C(x)$) **annotated as recommended or mandatory** for a range of values $V_i \in \mathcal{V} = \{V_1, ..., V_n\}$, or a **scope restriction attribute annotated as mandatory** for a range of values $V_i \in \mathcal{V}$ in $\mathcal{A}$. If no scope restriction attributes are added to r, then $C(r) = C(x)$. If however $C(x) = \emptyset$, then $C(r)$ is composed only of scope

restriction attributes. The condition $(a_i \in C(x) \wedge isRecommended(a_i, V_i))$ if true, entails $V_i \subseteq dom(x, a_i), V_i \neq \emptyset$, where $dom(x, a_i)$ denotes the domain of permitted values of a_i in x.

Definition 3.6.4. - *Meta-policy*.
A meta-policy P is of the form $P = \langle f, \mathcal{R}_p, \mathcal{R}_d \rangle$, where f is a rule combining algorithm (either `PO` or `DO`), and $\mathcal{R}_p = \{r_1, ..., r_n\}$ and $\mathcal{R}_d = \{r'_1, ..., r'_m\}$ the sets of meta-rules of P with respectively a `Permit` and a `Deny` effect.

For a better presentation, we use in the following the dot notation (**"."**) to refer to a specific component of a meta-policy or a meta-rule. For instance, $r.x$ will denote the atomic rule of r, and $r.\mathcal{A}$ the attached set of annotations. To differentiate meta-rules of different meta-policies, we also use when necessary subscript notations; for example r_{ij} will denote the meta-rule j of the meta-policy P_i.

3.6.2 P2CR Conflicts Detection Module

We now explore an important component of our policy composition framework, the conflict detection module. The main purpose of this module is to ensure that the meta-policies translated by the policy translation module are composable. For this, it first searches for potential conflicts between the meta-rules of the meta-policies, determines the causes of the conflicts, and selects the relevant policy operators to use to remove the conflicts. Two meta-rules r and r' conflict if their atomic rules $r.x$ and $r'.x'$ satisfy one of the following conditions.

1. $r.x$ and $r'.x'$ must have the same target, mutually satisfiable conditions, and be of different effects.

2. The positive obligations of $r.x(r'.x')$ must be in contradiction with the access specifications of $r'.x'(r.x)$.

3. The negative obligations of $r.x(r'.x')$ must be in contradiction with the access specifications of $r'.x'(r.x)$.

In this chapter, we focus only on condition 1 that we explain further in *Definition 3.6.5.*

Definition 3.6.5. - ***Conflict*** (*Semantics*).
Two XACML rules x and x' conflict if x and x' have the same target, $cond(x)$ and $cond(x')$ are mutually satisfiable, and upon a request, x yields `Permit` (resp. `Deny`) and x' yields `Deny` (resp. `Permit`) depending on their effects.

Definition 3.6.6. - ***Notations***.

- Let $\mathcal{P} = \{P_1, ..., P_k\}$ be a set of translated meta-policies, $\mathcal{PR} = \bigcup_{P_i \in \mathcal{P}} P_i.\mathcal{R}_p$, $\mathcal{DR} = \bigcup_{P_i \in \mathcal{P}} P_i.\mathcal{R}_d$ the sets of meta-rules of $\mathcal{P}$ with respectively a permit and a deny effect, and $\mathcal{L} = \mathcal{PR} \cup \mathcal{DR}$ the set of all the meta-rules of $\mathcal{P}$.

- Let $\mu \colon \mathcal{L} \to \mathcal{P}$ be the function that maps each meta-rule $r_{ij} \in \mathcal{L}$ to its containing meta-policy. $\mu(r_{ij})$ returns always a unique meta-policy $\mu(r_{ij}) = P_i$.

- Let φ be a function that maps each $P_i \in \mathcal{P}$ to its set of rules $\varphi(P_i) = P_i.\mathcal{R}_p \cup P_i.\mathcal{R}_d$, and let $T(P_i) = \bigcup_{r_{ij} \in \varphi(P_i)} S(r_{ij}.x_{ij}) \times A(r_{ij}.x_{ij}) \times R(r_{ij}.x_{ij})$ be its set of triples of the form (s, a, o).

- Let $\mathcal{T} = \bigcup_{P_i \in \mathcal{P}} T(P_i)$ be the set of triples of $\mathcal{P}$.

- Let $\gamma_1 \colon \mathcal{T} \to \mathcal{PR}$ be the function that maps each triple $(s, a, o) \in \mathcal{T}$ to the set of rules in $\mathcal{PR}$ which targets evaluate to 1 on (s, a, o) $(target(s, a, o) = 1)$.

- Let $\gamma_2 \colon \mathcal{T} \to \mathcal{DR}$ be the function that maps each triple $(s, a, o) \in \mathcal{T}$ to the set of rules in $\mathcal{DR}$ which targets evaluate to 1 on (s, a, o) $(target(s, a, o) = 1)$.

- Let $\lambda \colon \mathcal{L} \times \mathcal{L} \to \mathcal{L}$ be the function that returns for any pair of rules (r, r') the dominant rule $\lambda(r, r')$ i.e., $\lambda(r, r') = r$ if $r \succ r'$, and $\lambda(r, r') = r'$ if $r' \succ r$.

- Let ψ be the function that returns for any attribute-domain pair (a, V), the set of literals on a such that $V = \bigcap_{l \in \psi(a,V)} dom(l, a)$. We denote by $\neg\psi(a, V)$ the set of negative literals $\{\neg l : \ l \in \psi(a, V)\}$.

Methodology – We first group all the rules in $\mathcal{P}$ based on their premises ($target(s, a, o)$). We then split each group into two sub-groups, one that contains only rules with a permit conclusion ($permit(s, a, o)$), and the other, the rules with a deny conclusion ($deny(s, a, o)$). We denote by $\Gamma_{sao} = \gamma_1(s, a, o)$ the sub-group of permit rules, and by $\overline{\Gamma}_{sao} = \gamma_2(s, a, o)$ the sub-group of deny rules for a triple $(s, a, o) \in \mathcal{T}$. *Definition 3.6.7* below formalizes the notion of conflict between two rules as defined in *Definition 3.6.5*.

Definition 3.6.7. - ***Conflict*** (*Syntax*).
Two meta-rules $r \in \Gamma_{sao}$ and $r' \in \overline{\Gamma}_{sao}$ conflict if their atomic rules $r.x$ and $r'.x'$ satisfy one of the following two conditions holds.

1. $C(r.x) = C(r'.x') = \emptyset$.

2. $C(r.x) = \emptyset \wedge C(r'.x') \neq \emptyset$ or $C(r.x) \neq \emptyset \wedge C(r'.x') = \emptyset$.

3. $(I_c = C(r'.x) \cap C(r'.x')) \neq \emptyset$, and $\exists\ c_i \in I_c$ such that $(I_d = dom(r.x, c_i) \cap dom(r'.x', c_i)) \neq \emptyset$.

In condition 2, a constraint over any condition attribute $c \in C(r.x)$ ($c' \in C(r'.x')$) that is satisfiable in r (r') is also satisfiable in r' (r) because $C(r'.x') = \emptyset$ ($C(r.x) = \emptyset$).

To find conflicts between rules in Γ_{sao} and rules in $\overline{\Gamma}_{sao}$ and identify the causes, we first transform each atomic rule of a meta-rule to a disjunction of predicates. An example of a permit atomic rule x transformed to a boolean formula is: $\neg target(s, a, o) \vee \neg cond(x) \vee permit(s, a, o)$. $\neg cond(x)$ is in `CNF` and can be expressed as the set $L(x) = \{l_1, ..., l_n\}$ of literals where $l_i = \neg(c_i \triangleright v_i)$. Because the set of operators $\mathcal{O}$ is closed under negation ($\neg$), each l_i can then be transformed to a positive literal $l_i = c_i \triangleleft v_i$ where $\triangleleft \in \mathcal{O}$. In the following, we use these literals, and we denote by $dom(l_i, c_j)$ the domain of c_j for l_i.

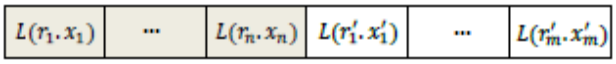

Figure 3.6: Data structure.

We then build the array T depicted in *Figure 3.6*. The shaded part of T is composed of literals of the permit rules while the other part comprises deny rules. In the following, $r_{[i]}(r'_{[j]})$ denotes a permit (deny) rule at position i (j) in the first (second) part of T.

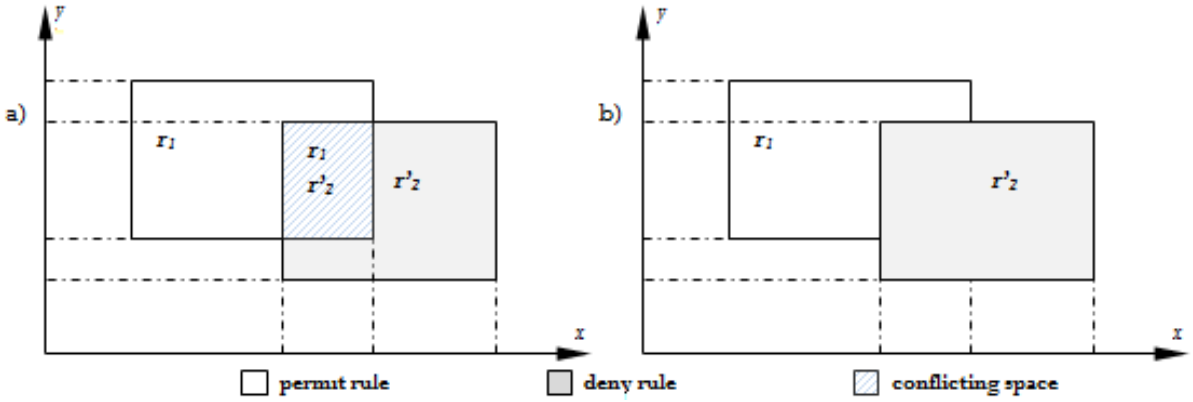

Figure 3.7: Conflicting space. a.) Two rules, r_1 and r'_2, that conflict in a 2-dimensional space. b. The conflict is removed by means of the annotations attached to r_1.

3.6.3 P2CR Conflict Resolution Module

. We propose three levels of conflict resolution capabilities:

- ***Level-1***: by using the annotations supplied by the collaborating entities.
- ***Level-2***: by means of a defeasible logic reasoning theory.
- ***Level-3***: by providing recommendations based on the policy specifications of the entities.

Level-1 – For each rule r in the first part of T, we verify if there exists a rule r' in the other part of T such that $cond(r.x)$ and $cond(r'.x')$ are mutually satisfiable. If we find such a rule r', then r and r' w.r.t *Definition 3.6.5* conflict. To remove the conflict, we use the annotations of the rules and update $cond(r.x)$ or $cond(r'.x')$. That is, if c_i is

a condition attribute such that $I_d = dom(r, c_i) \cap dom(r', c_i)$ is not empty, to remove the conflicting space I_d, we check which of the annotations of r or r' enables eliminating the conflicts i.e., if $r.\delta(c_i) \supseteq I_d$ or $r'.\delta'(c_i) \supseteq I_d$. Assuming it is $r.\delta(c_i)$, we redefine the domain of c_i in the literal l_k where it appears as $dom(l_k, c_i) = dom(l_k, c_i) \setminus I_d$ (see Appendix *Figure 3.7*). If $dom(l_k, c_i)$ becomes empty, we simply ignore l_k. If $r.\delta(c_i) \not\supseteq I_d$, and $r'.\delta'(c_i) \supseteq I_d$, we use instead the annotations attached to r' and proceed as before. If both annotations can help correct the conflicts, then we randomly select one of the literal where c_i appears and redefine its domain in that literal as $dom(l, c_i) = dom(l, c_i) \setminus I_d$ with $l \in L(r)$ or $l \in L(r')$. If the annotations cannot help remove the conflicts, then we apply our reasoning method (see ***level-2***) over $\{(r, r')\}$ to resolve the conflict.

To detect rules that conflict for a triple $(s, a, o) \in \mathcal{T}$ and find the causes, we use Algorithm 3. Two rules with a conflicting space are in conflict except when they belong to the same policy. This, because the combining algorithm of their policy already resolves that conflict (*line 6*). The attributes the permitted values of which in $r_{[i]}$ and $r'_{[j]}$ overlap are grouped in I_c then stored in H, a matrix indexed by i and j, the positions of $r_{[i]}$ and $r'_{[j]}$ respectively in the first and second part of T. In each I_c, Algorithm 4 selects the attribute to which is attached the annotation that helps to remove the conflict between $r_{[i]}$ and $r'_{[j]}$ and possibly between $r_{[i]}$ and other rules $r'_{[k],k\neq j}$ (*lines 3-5*, and *line 7* with $\arg\max_{a\in I} K[a]$). It updates then the domain of c in the rule where it selected c by removing the values for which c is recommended (*lines 16-19, 20-23*). If none of the annotations attached to the attributes in I_c can help remove the conflicts, then the conflicting rules are stored as pairs in $\mathcal{F}$, an array indexed by the attributes in I_c (*lines 12-14*). At the end, Algorithm 4 partitions the collection $\{\Gamma_{sao}, \overline{\Gamma}_{sao}\}$ into a set of permit rules $\mathcal{P}$, a set of deny rules $\mathcal{D}$, and a set of conflicting rules $\mathcal{F}$. Note that $\mathcal{P}$ and $\mathcal{D}$ do not conflict.

Definition 3.6.8. - ***Completeness & Soundness***.

Completeness. A conflict detection method over the collection $\{\Gamma_{sao}, \overline{\Gamma}_{sao}\}$ is complete if there exist no rules $r_i \in \Gamma_{sao}$ and $r'_j \in \overline{\Gamma}_{sao}$ that conflict but the conflict detection method does not identify them as conflicting rules.

Soundness. A conflict detection method over the collection $\{\Gamma_{sao}, \overline{\Gamma}_{sao}\}$ is sound if any two rules $r_i \in \Gamma_{sao}$ and $r'_j \in \overline{\Gamma}_{sao}$ that the conflict resolution method identifies as conflicting are indeed conflicting.

Algorithm 3: **Conflicts Detection and Causes Identification.**

Input : T of size n+m, with n, m sizes of Γ_{sao}, $\overline{\Gamma}_{sao}$.

Output: matrix H of attributes. Initially, $H = \emptyset$.

1 $i \leftarrow 1$;
2 **while** $i \leq n$ **do**
3 $L_i \leftarrow \mathrm{T}[i]$; $j \leftarrow n+1$;
4 **while** $j \leq n+m$ **do**
5 $L'_j \leftarrow \mathrm{T}[j]$; $C'_j \leftarrow C(r'_{[j]})$; $C_i \leftarrow C(r_{[i]})$;
/* no conflicts between $r_{[i]}$ and $r'_{[j]}$. */
6 **if** $\mu(r_{[i]}) = \mu(r'_{[j]})$ **then**
7 *go to line 21;*
8 **if** $L'_j = \emptyset$ **and** $L_i = \emptyset$ **then**
9 **foreach** $(a, V) \in r_{[i]}.\mathcal{A}_{[i]}$ **do**
10 $L_i \leftarrow L_i \cup \neg\psi(a, V)$;
11 **foreach** $(a', V') \in r'_{[j]}.\mathcal{A}'_{[j]}$ **do**
12 $L'_j \leftarrow L'_j \cup \neg\psi(a', V')$;
13 **if** $L_i = \emptyset$ **and** $L'_j \neq \emptyset$ **then**
14 $I_c \leftarrow C'_j$;
15 **if** $L'_j = \emptyset$ **and** $L_i \neq \emptyset$ **then**
16 $I_c \leftarrow C_i$;
17 **if** $L'_j \neq \emptyset$ **and** $L_i \neq \emptyset$ **then**
18 $I_c \leftarrow \{c \in C_i \cap C'_j : \exists\ l = c \triangleleft v \in L_i,\ l' = c \triangleleft v' \in L'_j\ s.t\ dom(l, c) \cap dom(l', c) \neq \emptyset\}$;
/* $r_{[i]}$ and $r'_{[j]}$ are conflicting. */
19 **if** $I_c \neq \emptyset$ **then**
20 $H[i, j] \leftarrow I_c$;
21 $j \leftarrow j + 1$;
22 $i \leftarrow i + 1$;
23 **return** H;

Theorem 3.6.1. Algorithm 3 *is* `complete` and `sound`. □

Proof. - ***Theorem 5.1***.

Assume to the contrary that Algorithm 3 is not complete i.e., for a triple $(s, a, o) \in \mathcal{T}$ there exist two conflicting rules r_i and r'_j which targets evaluate to 1 on (s, a, o) that Algorithm 3 does not detect. With respect to *Definition 3.6.5*, $r_i.x_i$ and $r'_j.x'_j$ are then of opposite effects and their conditions are mutually satisfiable. Let r_i be the

meta-rule with the `Permit` atomic rule, r'_j the meta-rule with the `Deny` atomic rule, and I_c the set of condition attributes in $r.x$ and $r'.x'$ that have overlapping domain. Let us build from Γ_{sao} and $\overline{\Gamma}_{sao}$ an array T similar to the one of *Figure 3.6*, and run Algorithm 3 on T. Since Algorithm 3 iterates through all the elements of T, and $r_i \in \Gamma_{sao}$ and $r'_j \in \overline{\Gamma}_{sao}$, it must visit at some step k r_i and r'_j simultaneously. Because $I_c \neq \emptyset$, then $H[i,j] \neq \emptyset$. Therefore, Algorithm 3 detects indeed that r_i and r'_j are conflicting; which contradicts our initial assumption. Thus, Algorithm 3 is complete.

Assume to the contrary that Algorithm 3 is not sound i.e., there exist two meta-rules r_i and r'_j that Algorithm 3 claims are conflicting but that are not. If r_i and r'_j do not conflict, then w.r.t *Definition 3.6.5*, r_i and r'_j are of a same effect, or $cond(r_i)$ and $cond(r'_j)$ are not mutually satisfiable. Then, either $r_i \in \Gamma_{sao}$ **and** $r'_j \in \Gamma_{sao}$ or, $r_i \in \overline{\Gamma}_{sao}$ **and** $r'_j \in \overline{\Gamma}_{sao}$. If $cond(r_i)$ and $cond(r'_j)$ are not mutually satisfiable, then $dom(l,c) \cap dom(l',c) = \emptyset, \forall\ l \in L_i, l' \in L'_j$, thus $I_c = \emptyset$. Since Algorithm 3 compares only the domains of attributes of rules of opposite effects, and since r_i and r'_j conflict only if $I_c \neq \emptyset$ (*line 19*), it cannot claim that r_i and r'_j are conflicting; r_i and r'_j cannot then belong to the same set. Thus, Algorithm 3 is also sound. □

Algorithm 4: **Conflicts Resolution by Means of the Annotations.**

Input : T, H obtained from Algorithm 3, n and m.

Output: $\mathcal{P}$, $\mathcal{D}$, $\mathcal{F}$, and E. Initially, $\mathcal{P} = \mathcal{D} = \mathcal{F} = E = \{\}$.

$i \leftarrow 1; j \leftarrow 1$. *Let K be an array. Set its elements to 0*;

while $i \leq n$ **do**

  foreach $I \in \bigcup_{k\geq 1}^{m} H[i,k]$ **do**

    foreach $a \in I$ **do**

      $K[a] \leftarrow K[a] + 1$;

  foreach $I \in \bigcup_{k\geq 1}^{m} H[i,k]$ **do**

    $J \leftarrow I$;

    repeat

      $c \leftarrow \arg\max_{a \in I} K[a]$; $I_d \leftarrow dom(r_{[i]}, c) \cap dom(r'_{[j]}, c)$;

      $I \leftarrow I \setminus \{c\}$; $bool \leftarrow (r'_{[j]}.\delta'_{[j]}(c) \not\supseteq I_d \wedge r_{[i]}.\delta_{[i]}(c) \not\supseteq I_d) \vee$
      $(isMandatory(c, r'_{[j]}.\delta'_{[j]}(c)) \wedge (isMandatory(c, r_{[i]}.\delta_{[i]}(c))$;

    until $bool = false$ **or** $I = \emptyset$;

    if $I = \emptyset$ **and** $bool = true$ **then**

      $E \leftarrow E \cup \{r'_{[j]}\}$; $\mathcal{P} \leftarrow \mathcal{P} \cup \{r_{[i]}\}$;

      $\forall\, c \in J, \mathcal{F}[c] \leftarrow \mathcal{F}[c] \cup \{(r_{[i]}, r'_{[j]})\}$;

    $V \leftarrow dom(r_{[i]}, c) \setminus I_d$; $V' \leftarrow dom(r'_{[j]}, c) \setminus I_d$;

    `/* use annotations attached to` $r_{[i]}$`. */`

    if $isRecommended(c, r_{[i]}.\delta_{[i]}(c))$ **and** $r'_{[j]}.\delta'_{[j]}(c) \not\supseteq I_d$ **then**

      $L(r_{[i]}) \leftarrow L(r_{[i]}) \setminus \neg\psi(c, dom(r_{[i]}, c))$; $L(r_{[i]}) \leftarrow L(r_{[i]}) \cup \neg\psi(c, V)$;

      let r be a permit meta-rule on (s,a,o) s.t $r.\mathcal{A} = r_{[i]}.\mathcal{A}_{[i]}, cond(r.x) = \bigwedge_{l \in L(r_{[i]})} \neg l$;

      $\mathcal{P} \leftarrow (\mathcal{P} \setminus \{r_{[i]}\}) \cup \{r\}$; $\mathcal{D} \leftarrow \mathcal{D} \cup \{r'_{[j]}\}$;

    `/* use annotations attached to` $r'_{[j]}$`. */`

    if $isRecommended(c, r'_{[j]}.\delta'_{[j]}(c))$ **and** $r_{[i]}.\delta_{[i]}(c) \not\supseteq I_d$ **then**

      $L(r'_{[j]}) \leftarrow L(r'_{[j]}) \setminus \neg\psi(c, dom(r'_{[j]}, c))$; $L(r'_{[j]}) \leftarrow L(r'_{[j]}) \cup \neg\psi(c, V')$;

      let r' be a deny meta-rule on (s,a,o) s.t $cond(r'.x') = \bigwedge_{l' \in L(r'_{[j]})} \neg l', r'.\mathcal{A}' = r'_{[j]}.\mathcal{A}'_{[j]}$;

      $\mathcal{D} \leftarrow (\mathcal{D} \setminus \{r'_{[j]}\}) \cup \{r'\}$; $\mathcal{P} \leftarrow \mathcal{P} \cup \{r_{[i]}\}$;

    if $r'_{[j]}.\delta'_{[j]}(c) \supseteq I_d$ **and** $r_{[i]}.\delta_{[i]}(c) \supseteq I_d$ **then**

      if $isRecommended(c, r'_{[j]}.\delta'_{[j]}(c))$ go to 16, if $isRecommended(c, r_{[i]}.\delta_{[i]}(c))$ to 20, if both to either 16 or 20;

    $r'_{[j]} \leftarrow r'$; $r_{[i]} \leftarrow r$; $j \leftarrow j + 1$;

  $i \leftarrow i + 1; j \leftarrow 1$; *set K to 0*;

$\mathcal{D} \leftarrow \mathcal{D} \setminus E$;

return $\mathcal{P}, \mathcal{D}, \mathcal{F}, E$;

Theorem 3.6.2. ***i.***) For all $1 \leq i \leq n, 1 \leq j \leq m : H[i,j] \neq \emptyset$, either Algorithm 4 resolves the conflict between $r_{[i]}$ and $r'_{[j]}$ with the annotations $r_{[i]}.\mathcal{A}_{[i]}$ or $r'_{[j]}.\mathcal{A}'_{[j]}$, ***ii.***)

or $\forall\ c \in H[i,j], \{(r_{[i]}, r'_{[j]})\} \in \mathcal{F}[c]$.

Proof. - ***Theorem 5.2***.

i.) We assume Algorithm 4 cannot resolve the conflict between $r_{[i]}$ and $r'_{[j]}$ with $r_{[i]}.\mathcal{A}_{[i]}$ and $r'_{[j]}.\mathcal{A}'_{[j]}$, and that $r_{[i]}.\delta_{[i]}(c)$ and $r'_{[j]}.\delta'_{[j]}(c)$ can help remove the conflict i.e., $(isRecommended(c, r_{[i]}.\delta_{[i]}(c)) \wedge r_{[i]}.\delta_{[i]}(c) \supseteq I_d) \vee (isRecommended(c, r'_{[j]}) \wedge r'_{[j]}.\delta'_{[j]}(c) \supseteq I_d)$ is true as per *Definition 3.6.3*. Then, the condition $r'_{[j]}.\delta'_{[j]}(c) \not\supseteq I_d \wedge r_{[i]}.\delta_{[i]}(c) \not\supseteq I_d \vee isMandatory(c, r'_{[j]}.\delta'_{[j]}(c)) \wedge isMandatory(c, r_{[i]}.\delta_{[i]}(c)$ in *line 10* is false, which entails condition in *line 16, 20* or *24* is true. In either case, Algorithm 4 resolves the conflict by removing the conflicting space I_d of $r_{[i]}$ and $r'_{[j]}$. ***ii.***) We assume Algorithm 4 cannot resolve the conflict between $r_{[i]}$ and $r'_{[j]}$ i.e., $\forall\ c \in H[i,j], r'_{[j]}.\delta'_{[j]}(c) \not\supseteq I_d \wedge r_{[i]}.\delta_{[i]}(c) \not\supseteq I_d$ or $(isMandatory(c, r_{[i]}.\delta_{[i]}(c)) \wedge (isMandatory(c, r'_{[j]}.\delta'_{[j]}(c))$ (as per *Definition 5.1.3*), and $\exists\ c \in H[i,j], \{(r_{[i]}, r'_{[j]})\} \notin \mathcal{F}[c]$. Then, the condition in *line 12* is true. Thus, $\forall\ c \in H[i,j], \{(r_{[i]}, r'_{[j]})\} \in \mathcal{F}[c]$, which contradicts our previous assumption. □

Level-2 – The common approach used by many conflict resolution strategies in order to resolve conflicts between rules is to order the rules. Precedence-based conflict resolution methods for instance resolve conflicts by specifying conditions under which a rule takes precedence over another rule it conflicts with. XACML combining algorithms combine individual evaluation results of rules that conflict to one final decision. Although these schemes are widely used to resolve conflicts, none of them, however, scales well to our scenario. This, because each entity uses the conflict resolution mechanism it finds more suitable to its own security needs. As a result, a strategy that one entity applies may be inappropriate for another entity.

In this section, we show how to possibly resolve conflicts that we cannot address with the annotations. For this, we use defeasible logic. An important aspect of this formalism is that it allows to reason over inconsistent knowledge. In this logic, valid conclusions are withdrawn in the event that contrary evidence is acquired i.e., a conclusion is only drawn when all the reasons for the contrary conclusion have been

definitely invalidated. To reason with defeasible logic, we must specify a knowledge base, its composing sets of defeasible and defeater rules, and define the inference mechanism. From the conflict resolution standpoint, the inference mechanism is of most practical interest because it helps to establish a precedence relation over rules that conflict, thereby allowing to resolve conflicts.

Below, we decline the method we follow for ordering two rules, r with a `Permit` effect and r' with a `Deny` effect, and we implement it in Algorithm 5. The same method applies when r is a `Deny` rule and r' a `Permit` rule. It suffices just to change r to r' and r' to r. First, we define a concept that we will use shortly.

Definition 3.6.9. - ***Context*** (*Request*).
Let $(s, a, o) \in \mathcal{T}$. The context of a request q to perform a on o is the set of credentials $\mathcal{E} = \{(c_1, v_1), ..., (c_n, v_n)\}$, where $c_k, 1 \leq k \leq n$ are condition attributes and $v_k, 1 \leq k \leq n$ constant values.

If there is $r_i \in \Gamma_{sao}$ such that $c_k \in C(r_i)$ and $v_k \in dom(r_i, c_k), 1 \leq k \leq n$, then the user will be permitted. However, if there is $r'_j \in \overline{\Gamma}_{sao}$ such that $c_k \in C(r'_j)$ and $v_k \in dom(r'_j, c_k), 1 \leq k \leq n$, then the user will be denied.

Inference Process — Given $(s, a, o) \in \mathcal{T}$, $r \in \Gamma_{sao}$ and $r' \in \overline{\Gamma}_{sao}$, let $c_k \in C(y), y \in \{r.x, r'.x'\}$ such that $I_d = dom(r.x, c_k) \cap dom(r'.x', c_k) \neq \emptyset$, and $r.\delta(c_k) \not\supseteq I_d$ and $r'.\delta(c_k) \not\supseteq I_d$. For any request q with the context $\mathcal{E} = \{(c_1, v_1), ..., (c_k, v_k), ..., (c_n, v_n)\}$, where $c_{i \neq k} \in C(y), v_{i \neq k} \in dom(y, c_i)$ and $v_k \in I_d$, r and r' conflict i.e., $\{(r, r')\} \in \mathcal{F}[c_k]$. We resolve this conflict as follows.

- $r \succ r'$ is derived as the final ordering only when all the reasons for the contrary evidence ($\neg(r \succ r') \equiv (r' \succ r)$) have been definitely invalidated.

- $r' \succ r$ is drawn as the final ordering only when all the reasons for the contrary evidence ($\neg(r' \succ r) \equiv (r \succ r')$) have also been definitely invalidated.

In *Table 3.4*, we present the cases for which $r \succ r'$ and $r' \succ r$ hold. To derive $r \succ r'$ or $r' \succ r$, we initially assume both orderings, and we search for any evidence in the knowledge base $\mathcal{KB} = \{\Gamma_{sao}, \overline{\Gamma}_{sao}\}$ that disproves one of them. If any of the cases in *row 1* (*row 2*) holds, then we withdraw the defeated ordering. At the end, when only $r \succ r'$ or $r' \succ r$ holds, the conflict between r and r' is then resolved. Upon a request q with the context $\mathcal{E}$ to which both r and r' are applicable, the evaluation yielded by the dominant rule is issued as the decision to enforce.

There are however cases when both $r \succ r'$ and $r' \succ r$ hold in $\mathcal{KB}$, are disproved, or neither of them can be approved nor disproved, e.g., $\mu(r).f =$`PO` and $\mu(r').f=$`DO`. With such cases, we cannot conclude whether $r \succ r'$ or $r' \succ r$. To resolve then the conflicts between r and r', we propose an approach based on recommendations.

#	$\neg(r \succ r')$	$\neg(r' \succ r)$
1.	$\mu(r) = \mu(r'), \mu(r).f =$ `DO`.	$\mu(r) = \mu(r'), \mu(r').f =$ `PO`.
2.	$\mu(r).f =$ `DO`, $\mu(r').f =$ `DO`, $\exists\ r_i \in \mu(r)$ s.t $r_i \in \overline{\Gamma}_{sao}$.	$\mu(r).f =$`PO`, $\mu(r').f =$ `PO`, $\exists\ r'_j \in \mu(r')$ s.t $r'_j \in \Gamma_{sao}$.
3.	$\mu(r).f =$ `DO`, $\mu(r').f =$ `PO`, $\exists\ r_i \in \mu(r)$ s.t $r_i \in \overline{\Gamma}_{sao}$, $\forall\ r'_j \in \mu(r')$: $r'_j \in \overline{\Gamma}_{sao}$.	$\mu(r).f =$ `DO`, $\mu(r').f =$ `PO`, $\exists\ r'_j \in \mu(r')$ s.t $r'_j \in \Gamma_{sao}$, $\forall\ r_i \in \mu(r)$: $r_i \in \Gamma_{sao}$.

Table 3.4: Defeating cases.

Level-3 – We identify an attribute $c \in C(r), c \in C(r')$ and a range of values in the domain of c in r or r' such that if annotated as recommended for c we can eliminate the conflict between r and r', and possibly between r (r') and other rules (see Algorithm 5).

Algorithm 5: **Conflicts Resolution by Reasoning and by Recommendation.**

Input : $\mathcal{P}, \mathcal{D}, E$ obtained from Algorithm 4.

foreach $r'_j \in E$ **do**
  $I_c \leftarrow \{c' \in C(r'_j) \colon \exists\ r_i \in \mathcal{P}, \{(r_i, r'_j)\} \in \mathcal{F}[c']\}$;
  foreach $c \in I_c$ **do**
    $\mathcal{C} \leftarrow \{r_i \in \mathcal{P} \colon \{(r_i, r'_j)\} \in \mathcal{F}[c]\}$;
    `/* conflict resolution by reasoning */`
    foreach $r_i \in \mathcal{C}$ **do**
      apply the inference on r_i *and* r'_j *over* c *and let* $r = \lambda(r_i, r'_j)$ *be the dominant rule*;
      if $r = r'_j$ **or** $r = r_i$ **then**
        $\mathcal{C} \leftarrow \mathcal{C} \setminus \{r\}$;
        $\mathcal{F}[c] \leftarrow \mathcal{F}[c] \setminus \{(r_i, r'_j)\}$;
    `/* conflict resolution by recommendations */`
    $\forall\ r_i \in \mathcal{C}$ let $\mathcal{H}[r_i] = \{r'_k \in \mathcal{D} \colon \{(r_i, r'_k)\} \in \mathcal{F}[c]\}$;
    sort $\mathcal{C}$ *in a decreasing order based on* $|\mathcal{H}[r_i]|$, *the cardinality of* $\mathcal{H}[r_i], r_i \in \mathcal{C}$;
    foreach $r_i \in \mathcal{C}$ **do**
      $I_d \leftarrow dom(r'_j, c) \cap dom(r_i, c)$;
      find $D = dom(r'_j, c) \cap dom(r_k, c), r_k \in \mathcal{C}$ *s.t* D *is the largest range that contains* I_d;
      find $D' = dom(r_i, c) \cap dom(r'_k, c)$, $r'_k \in \mathcal{H}[r_i]$ *such that* D' *is the largest range that contains* I_d;
      if $|\mathcal{C}| \geq |\mathcal{H}[r_i]|$ **then**
        recommend D' *to be annotated as recommended for* c *in* r'_j; *exit*;
      if $|\mathcal{C}| < |\mathcal{H}[r_i]|$ **then**
        recommend D *to be annotated as recommended for* c *in* r_i;

3.7 P2CR Composite Policy Enforcement Process

In the previous section, we presented three different levels for resolving policy conflicts. In *level-1*, we attempted to remove conflicts by using the annotations provided by the entities. Conflicts that cannot be removed in *level-1* are addressed in *level-2*, and those that cannot be resolved in *level-2* are subsequently addressed in *level-3*. *Level-3* provides recommendations to the entities as to which assertions in their policies to annotate as recommended in order to remove definitely the conflicts.

Assumption and Objective – In this section, we assume, if any, that the entities follow the recommendations provided by *level-3*, and we provide the framework for

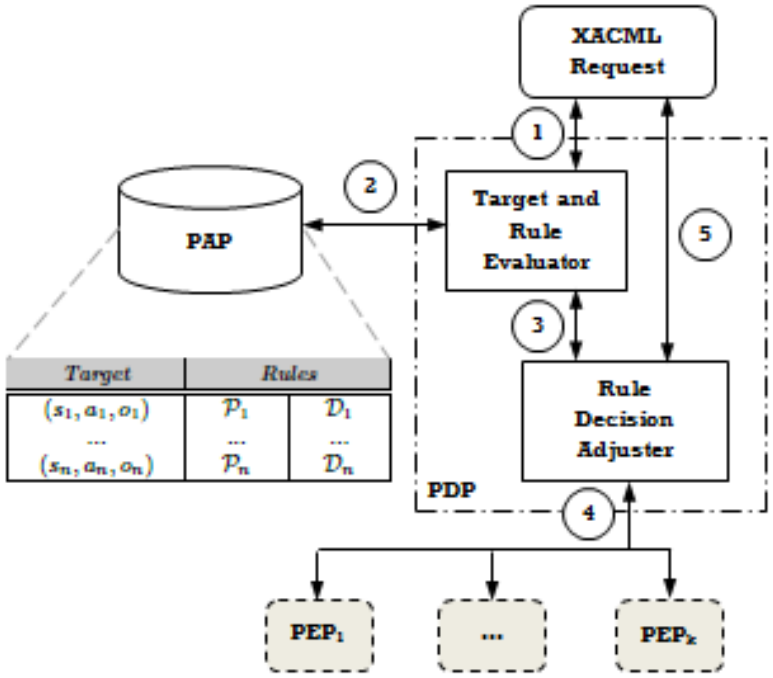

Figure 3.8: P2CR Composite Policy Enforcement.

enforcing P2CR conflict-free composite policy. For this, we design a policy decision point (PDP) to evaluate requests by using P2CR composite policy. The key aspect of this PDP is its ability to adjust access decisions according to the states of the resources or systems of each entity involved in the collaboration. More formally, it is designed to address dynamic conflicts. The purpose of this PDP as we stated initially is to enable the entities to have full control over their resources i.e., upon a request to decide w.r.t to the status of their resources or systems the actions to take. In the following, we explain what dynamic conflicts are and how our PDP deals with them.

3.7.1 Dynamic Conflicts

Dynamic conflicts occur when the amount of target objects available is limited because of outages that an entity may face or as a result of high demands it cannot handle in particular times. To prevent additional overheads that could stem from users' requests, and to guarantee at the same time a minimum level of availability of its resources to its users, the entity may consider to adjust temporarily some of

its authorization rules. For example, an entity that is confronted with bandwidth bottlenecks may change all *download* actions to *read only*, it may *block* access to resources that are of *size* greater than a certain threshold. After the outage, the entity may extend the *access time* it has set before in order to compensate activity losses. For such changes to take effect in a system though, they must be dictated by the underlying environment e.g., *status of a system, resources availability, usage peaks*. With XACML, this contextual information can precisely be encoded as attributes and stored in the XACML policy information point (PIP) of each entity for later use during the policy evaluation process by our PDP.

3.7.2 Leveraging Policy Annotations to Address Dynamic Conflicts

To enable an entity that is confronted with dynamic conflicts to indicate to our PDP its current requirements, we use annotations. The class Rule in *Figure 3.4* on page 47 models such requirements. It is associated with the properties *hasNewCondition* which has as range the class `Conditions`, *hasNewEffect* which has the class `Effect` as range, and *hasNewAction* which has as range the class `Action`. To annotate a rule, it suffices to define its *RuleId* as a `URI` pointing to an `OWL` individual in an ontology. For example, information in *Table 3.5* refers to a rule of *RuleId* r_1 in the ontology *onto_orga.rdf*. Annotations attached on r_1 can be queried by means of the `RDQL` query language [24]. Below, is an example of an `RDQL` query that retrieves the property *hasNewAction* of r_1.

Select ?c From ⟨http://www.onto_orga.rdf⟩ Where (?x, ⟨ciae:hasNewCondition⟩, ?c), (?c, ⟨ciae:isRecommended⟩, True) AND ?x = "r_1"

Element	***Annotations***
Rule	*⟨Rule RuleId="uri:http://www.orga.com/ onto_orga.rdf#R1" Effect="Deny"⟩*

Table 3.5: Examples of annotations.

3.7.3 Policy Decision Point Design

We design now our PDP. It uses the composite policy created by P2CR to evaluate requests. With *Figure 3.8*, we extend the XACML data flow model with some components to render the enforcement of the decisions yielded by our PDP distributed. For simplicity, we omit certains components of the XACML model such as the PIP, the PEP (policy enforcement point), and the context handler, although we may refer to them in the following. Our PDP consists of two components: one that, upon a request, evaluates applicable rules to render a decision, and another that adjusts such decision to the status of the system or resources of the entity entitled to enforce the decision.

❶ When a user submits a request, a PEP extracts the attributes, generates an XACML request, and sends it to our PDP. ❷ Our PDP fetches from the PAP that contains our composite policy a rule applicable to the request. Since the rules in any $\mathcal{P}_i$ and $\mathcal{D}_i$ have respectively the same effect, the first rule r in $\mathcal{P}_i$ or $\mathcal{D}_i$ is selected to evaluate the request. This technique is different from the XACML FA combining algorithm evaluation process in that the latter may apply to rules of different targets and effects, and the decision it yields may satisfy only the owner of r but not all the entities in the composition.

To issue a decision, our PDP ensures first that the status of the system of the entity entitled to enforce that decision currently permits it. ❸ Intuitively, it inquires about the presence of dynamic conflicts. In response, it receives from the PEP of the entity, one of the shaded boxes, the status of its system or resources. ❹ Then, by means of the scoping restriction operator (ˆ), it adjusts r w.r.t to that status, and notifies its decision to the user.

The decision is either a Permit, Deny or NotApplicable, or actions the entitled entity authorizes ❺. For instance, in the example about bandwidth bottlenecks in section 7.1, only *read* actions may be permitted. In such a case, the user is notified about this change and asked whether he is interested by the *read* access. Depending on

the response of the user, the PEP of the entitled entity proceeds with the enforcement of the issued decision.

3.8 Related Work

Much of the existing work on policy composition approaches this problem as a two-step process: one in which the policies are compared to check whether they guarantee similar level of security, or policy composition phase, and the policy integration phase during which the policies are combined by means of policy operators.

Relevant approaches that dealt with policy comparison relied mostly on *subject-action-target* solver-techniques [31, 35, 5, 2, 25]. In [31], Lin et al. described a technique based on principles from information retrieval in which they compare access policies written in XACML. Given any two policies P_1 and P_2, each rule in P_1 is compared with a rule in P_2 that has the same effect. A similarity score obtained by using hierarchy and numerical distances is then assigned to the rules. The similarity score of the rules ranges between 0 and 1, and reflects how similar the rules it is assigned to are with respect to the targets they are applicable to and also the conditions they impose on the requests. The higher a score is, the more similar the rules it is assigned to are. The similarity scores are then used to find one-to-many mappings for each rule in the two policies. The main limitation of their approach is that it only reveals the degree of similitude of the policies but not conflicts. In [35], Mazzoleni et al. introduced a policy comparison method by which policies are compared with respect to the sets of requests they authorize. Given two XACML policies, they compute first the similarity between pairs of rules by comparing the sets of values for which the conditions of the rules hold. They distinguished five groups of rules similarity: *Converge, Diverge, Restrict and Extend, Shuffle*. Then, they grouped the rule similarities based on their Effects, and applied some transformation functions to reduce the number of rules similarity to consider. Their method for computing

policy similarity is however limited simply to identifying policies referring to the same attributes. Backes et al. proposed an algebra [4, 6] for composing policies written in E-P3P [3], and an algorithm for checking refinement of enterprise privacy policies [5]. The algebra supports positive decisions, negative decisions, and obligations fulfilled on decisions. The algebra has however some limitations. It does not address conflicts in that it requires policies to be conflict-free. As of the algorithm, it is limited to identifying rules in different policies that need to be compared and does not consider the conditions of the rules. Agrawal et al. studied in [2] the problem of policy ratification – the process by which a new policy is approved before being committed in a system by taking into account its potential interactions with other policies and its deployment environment. The key idea behind the ratification technique is to determine whether a conjunction of two Boolean expressions is satisfiable. They identified four primitive operations that can be used for policy ratification: *dominance or redundancy check, potential conflict check, coverage check* (*default policies*)*, and consistent priority assignment*, and provided algorithms to implement these primitives. Their approach presents however some limitations. The interaction among the policies that require discipline-specific information is not taken into account. Moreover, potential conflicts among policies are resolved manually and by human administrators either by marking (one of) the conflicting policies as inactive or by indicating relative priorities. More pertinent to our work is [25] in which Hu et al. introduced a binary decision diagram (BDD)-based segmentation technique for discovering and resolving anomalies in XACML policies. The policies are parsed to identify their subject, action, resource and condition attributes. Then, the XACML rules of the policies are transformed to Boolean expressions, each being a combination (conjunction or disjunction) of atomic Boolean expressions. Atomic Boolean expressions with overlapping value ranges are further transformed into a sequence of new atomic Boolean expressions. Hu et al. focused on two types of anomalies, namely policy conflicts and

policy redundancy, and examined them at both the XACML policy element level and policy set level. Their policy-based segmentation technique consists of dividing an authorization space defined by an XACML policy or policy set component into a set of disjoint segments. Each segment associated with a unique set of XACML components indicates an overlap relation (either a conflicting or redundant) among those components. Although their approach is intuitive, the main issue is that it assumes a single and monolithic specification of the XACML policies. In other words, their technique works only when the policies are defined by a single entity, or in case they are defined by separate entities, these entities are under the authority of a unique party.

In [12], Bonatti et al. addressed the problem of combining authorization specifications (access policies) that may be incomplete, independent and possibly expressed in different languages. The policies are formalized as Horn clauses and combined by means of policy composition operators such as: *addition, conjunction, subtraction, closure, scoping restriction, overriding, and template.* Their framework is expressive, supports a wide range of policies spanning from open to closed policies, and is able to integrate policies of different entities while allowing them to retain their autonomy. However, it supports only positive authorizations. There is no clear distinction between an access that should be explicitly denied and an access that is outside the scope of a policy. Some attempt to support negative authorizations is proposed by using composition subtraction i.e., p-q, which means the set of accesses of p less the set of accesses of q. In this way, denials by q are given higher priority than grants by p, so that conflicts cannot arise. However, one cannot symmetrically state that grants of q should take priority over denials of p. It should be possible to state absolute access rights as well as absolute prohibitions. As part of their technique to compare policies in [35], Mazzoleni et al. described also a method to integrate policies with similar attributes. They introduced *policy integration preferences*, an XACML extension, by

which a party can specify the approach to be taken if its policy is to be integrated with other policies. They specified five different approaches: *converge-override, deny-override, restrict-override, permit-override*. However, they have not specified how an entity can embed its preferences into its policies, nor did they discuss any mechanism for integrating the policies. The work closer to ours is that of Lee et al. [30] in which they explored the concepts of defeasible policy composition wherein policies written in WS-SecurityPolicy [37] and augmented with annotations (also denoted as meta-policies) are combined based on non-monotonic inference rules. WS-SecurityPolicy is a specification that allows providers of xml-based web services to specify the security requirements of their services. The annotations are added by the entities owners of the policies to describe their composition preferences such as whether certain of their policy assertions are required or, if not, under what circumstances the entity owner of the policy is willing to compromise and allow other assertions to take precedence. The annotations, specified along with the policies, enable them not only to use a single composition operator to combine any two meta-policies (in contrast to many proposals) but also to automate the policy-composition process. Lee et al.'s approach is practical, sound and correct within their context. However, it presents two major limitations. First, their approach is not applicable in situations where the policies of the entities are expressed in XACML. WS-SecurityPolicy, in contrast to XACML has a much simpler structure; hence, easier to formalize and to reason with than XACML that presents a hierarchical structure with recursive policy specification. Second, to eliminate policy conflicts in their policy-composition approach, they assumed a partial order relation of the policies of the entities. This greatly limits the applicability of their approach in scenarios where the policies to compose are specified by independent entities that are under the authority of different parties.

3.9 Concluding Remarks

We have proposed in this chapter a simple and intuitive approach to compose access policies stated by separate and independent entities. The approach is expressive and handles a wide range of combinations spanning from closed policies to open policies. It provides formal foundation for reasoning with provable properties about the correctness and consistency of the resulting composite policy. The resulting composite policy is secure and the autonomy and confidentiality requirements of each entity are preserved. The composite policy is easy to enforce and the algorithms used for generating the policy are all tractable.

Our results will be submitted to the International Journal of Secure Software Engineering (IJSSE) in December 15^{th}, 2011.

Chapter 4

Future Work

In our future work, we will focus on the following problems.

First, we will extend the BD-RBAC model to address the problem of reasoning over imprecise or incomplete data. Such a reasoning is important when comparing credentials and task specifications. We will develop ontologies to model domain semantics to aid the reasoning, and investigate ways to refine the exception handling mechanism by using more elaborated techniques for the decision inference. Finally, we intend to use a hybrid bottom-up/top-down approach to derive more accurately the relationship between system roles and tasks.

Second, we will extend P2CR to address more efficiently dynamic conflicts. Our objective will be to render access control systems of the entities involve in the composition pro-active in order to prevent dynamic conflicts by providing alternatives to requests of users that would exacerbate these issues in the entities' systems. For this, we intend first to leverage system alarms that we will model. Based on these alarms and the status of resources that triggered these alarms, we will hierarchize system privileges (accesses). For example, in a system where dynamic conflicts occur, our extended P2CR framework will allow requests that would not cause harm to this system to proceed, and reformulate the others by downgrading the requested system privilege(s) to a minimum acceptable access.

Finally, we will provide a proof of concept implementation to demonstrate the practicality and soundess of our two models.

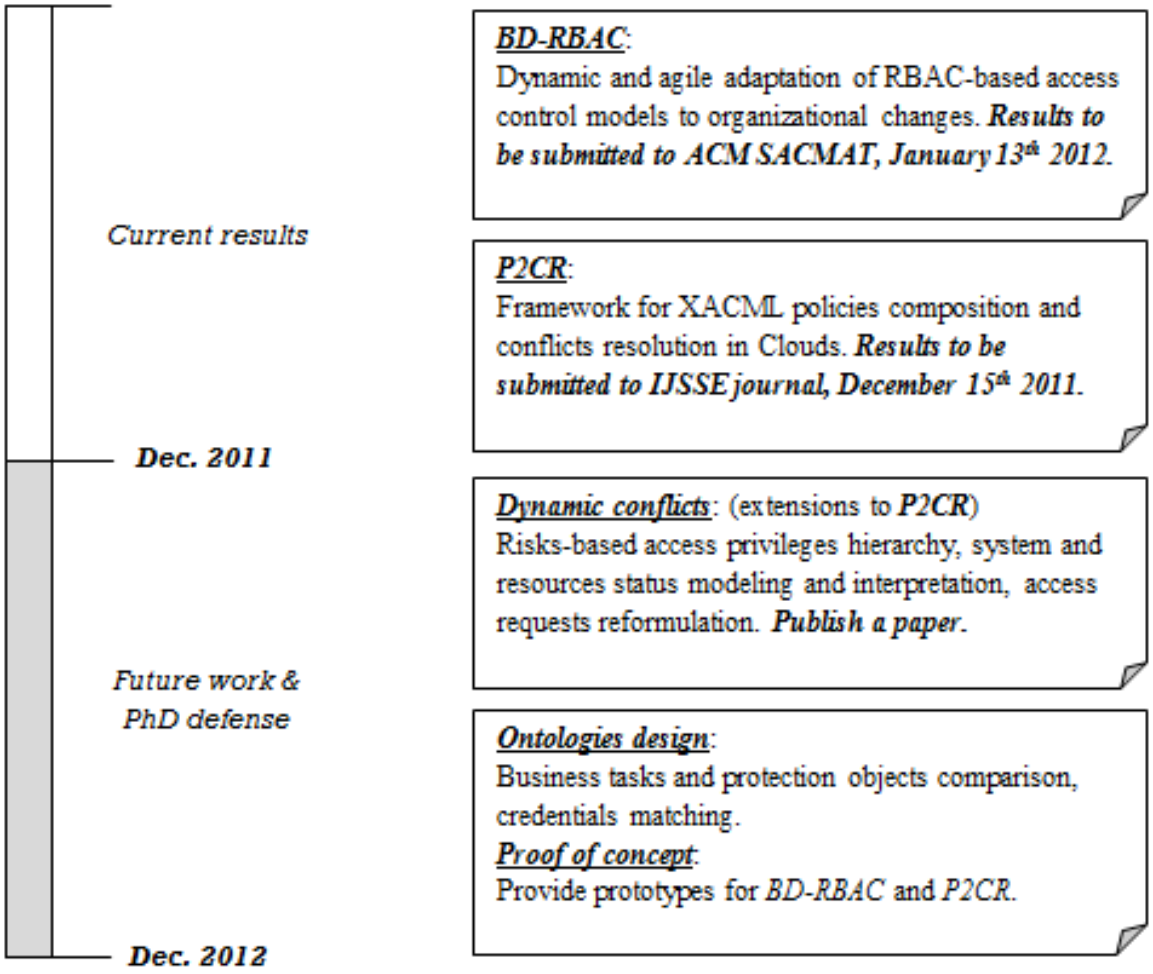

Figure 4.1: Outline.

Chapter 5

Conclusion

In this proposal, we have first analyzed the recurring changes that need to be brought to the access control systems of organizations that adopt the RBAC model. We have examined the security threats that may occur as a result of these business changes. By extending RBAC with the concepts of credentials and capabilities, and by decoupling the business processes from the IT resources they involve, we have shown that specifications and implementation of organizational security requirements can be aligned easily despite these changes, yet ensuring efficiently the confidentiality and integrity of the organizations' resources.

Second, we have addressed the challenging problem of policy composition, more specifically the problem of conflict resolution in the context where several autonomous entities want to collaborate. Ensuring that the access policies of the entities are compatible i.e., that they maintain similar levels of security, is an important step towards a safe collaboration between the entities owners of the policies. Although many proposals have addressed this issue in the past, the challenge lies here in the fact that the policies are independently stated and in the absence of a heading party that has the authority to decide which of two policies that conflict should take precedence over the other. Existing conflict resolutions strategies apply only to single and monolithic policies, and therefore lack extensibility.

Bibliography

[1] M. A. Al-Kahtani, R. Sandhu. A Model for Attribute-Based User-Role Assignment. *ACSAC, ACM* 2002.

[2] D. Agrawal, J. Giles, K. Lee, J. Lobo. Policy Ratification. *In Sixth IEEE International Workhop on Policies for Distributed Systems and Networks*, pages 223-232, 2007.

[3] P. Ashley, S. Hada, G. Karjoth, M. Schunter E-p3p Privacy Policies and Privacy Authorization. In *WPES*, pages 103-109, 2002.

[4] M. Backes, M. Dürmuth, R. Steinwadt. An Algebra for Composing Enterprise Privacy Policies. In P. Samarati, P. Y. A. Ryan, D. Gollman, R. Molva, editors, *ESORICS*, 2004, pp. 33-52 *Springer 2004*.

[5] M. Backes, K. Kimberly, M. Sean. Efficient comparison of enterprise privacy policies. *Proceedings of the 2004 ACM Symposium on Applied Computing (SAC).*, pp. 375-382.

[6] M. Backes, B. Rfitzmann, M. Schunter. A Toolkit for Managing Enterprise Privacy Policies. In E. Snekkenes, D. Gollman, editors, *ESORICS*, 2003, pp. 162-180 *Springer 2003*.

[7] L. Bauer, L. Cranor, R. W. Reeder, M. K. Reiter, K. Vaniea. Effects of access-control policy conflict-resolution methods on policy-authoring usability. *Technical Report CMU-CyLab-09-06-006, Cylab, CMU*, March 2009.

[8] A. Baumgrass, M. Strembeck, S. Rinderle-Ma. Deriving Role Engineering Artifacts from Business Processes and Scenario Models. *SACMAT* 2011, Innsbruck, Austria.

[9] E. Bertino, A. C. Squicciarini. A Fine-grained Access Control Model for Web Services. *SCC IEEE*, 2004.

[10] C. Bertolissi, M. Fernandez. A Rewriting Framework for the Composition of Access Control Policies. *PPDP, ACM*, July 2008.

[11] C. Bolan. Need to know: Security or Liability. *AISM*, 2004.

[12] P. Bonatti, S. De Capitani di Vimercati, P. Samarati. A Modular Approach to Composing Access Control Policies. *CCS, ACM*, 2000.

[13] Break Glass - Granting Emergency Access to Critical ePHI Systems - HIPAA Security. Protecting the privacy and security of health information. `http://hipaa.yale.edu/security/sysadmin/breakglass.html`.

[14] A. D. Brucker, H. Petritsch. Extending access control models with break-glass. *SACMAT, ACM* 2009.

[15] L. Cholvy, F. Cuppens. Analyzing Consistency of Security Policies. *Security and Privacy, IEEE*, 1997.

[16] M. A. Covington. Logical Control of an Elevator with Defeasible Logic. *IEEE Transactions on Automatic Control*, Jul. 2000.

[17] E. Coyne. Role Engineering. *ACM* 1995.

[18] C. Dong, G. Russello, N. Dulay. Flexible Resolution of Authorization Conflicts in Distributed Systems. *DSOM, IFIP/IEEE*, 2008.

[19] OASIS Standard. eXtensible Access Control Markup Language (XACML) Version 2.0. http://docs.oasis-open.org/xacml/2.0.

[20] A. Ferreira, D. Chadwick, P. Farinha, R. Correia, G. Zao, R. Chilro. How to securely break into RBAC: the BTG-RBAC model. *ACSAC - ACM*, December 2009.

[21] M. Frank, A. P. Streich, D. Basin, J. M. Buhmann. A probabilistic approach to hybrid role mining. *CCS*, 2009.

[22] C. Giblin, M. Graf, G. Karjoth, A. Wespi, I. Molloy, J. Lobo, S. Calo. Towards an Integrated approach to Role Engineering. *ACM*, 2010.

[23] G. Governatori. Defeasible Description Logic. *Springer*, 2004.

[24] http://phpxmlclasses.sourceforge.net/rdql.html

[25] H. Hu, G.J. Ahn, K. Kulkarni. Anomaly Discovery and Resolution in Web Access Control Policies. *SACMAT, ACM*, June 2011.

[26] V. Hu, D. F. Ferraiolo, R. Kuhn. Assessment of Access Control Systems. *NIST*, September 2006.

[27] S. Jajodia, P. Samarati, M. L. Sapino, V. S. Subramanian. Flexible Support for Multiple Access Control Policies. *ACM Trans. Database Syst.*, 26 (2) pp 2144-260, 2001.

[28] H. Karl, K. Molitorisz, C. Emig, K. Klinger, S. Abeck. Extending Role-based Access Control for Business Usage. *ESIST, IEEE* 2009.

[29] V. Kolovski, J. Hendler, B. Parsia. Analyzing Web Access Control Policies. *In Proceedings of the International World Wide Web Conference*, p. 677, 2007.

[30] A. J. Lee, J. P. Boyer. Defeasible Security Policy Composition for Web Services. *FMSE*, Nov. 2006.

[31] D. Lin, P. Rao, E. Bertino, J. Lobo. An Approach to Evaluate Policy Similarity. *SACMAT*, June 2007.

[32] H. Lu, J. Vaidya, V. Atluri. Optimal Boolean Matrix Decomposition: Application to Role Engineering. *ICDE* 2008.

[33] E. Lupu, M. Sloman. Conflicts in policy-based distributed systems management. *IEEE Trans Softw Eng* (TSE), 1999.

[34] S. Marinovic, R. Craven, J. Ma. Rumpole: A Flexible Break-glass Access Control Model. *SACMAT, ACM* 2011, Innsbruck, Austria.

[35] P. Mazzoleni, E. Bertino, B. Crispo. XACML Policy Integration Algorithms. *SACMAT, ACM*, June 2006.

[36] R. McGraw. Risk Adaptive Access Control (RAdAC). Privilege Management Workshop. *NIST*, 2009.

[37] A. Nadalin. Web services security policy language. Web Services Specification, 2002. `www.verysign.com/wss/WS-SecurityPolicy.pdf`.

[38] G. Neumann, M. Strembeck. A Scenario-driven Role Engineering Process for Functional RBAC Roles. *SACMAT* 2002, Monterey, California, USA.

[39] Q. Ni, E. Bertino, J. Lobo. D-Algebra for Composing Access Control Policy Decisions. *ASIACCS, ACM*, March 2009.

[40] P. Rao, D. Lin, E. Bertino, N. Lui, J. Lobo. An Algebra for Fine-Grained Integration of XACML Policies. *SACMAT, ACM*, July 2009.

[41] R. Sandhu, D. F. Ferraiolo, R. Kuhn. The NIST model for Role-Based Access Control: Towards a Unified Standard. *NIST*, 2000.

[42] R. Sandhu, E. J. Coyne. Role-Based Access Control Models. *IEEE*, 1996.

[43] D. Shin, G. Ahn, S. Cho, S. Jin. On Modeling System-centric Information for Role. *SACMAT* 2003.

[44] J. Vaidya, V. Atluri, J. Wariner. Roleminer: Mining roles using subset enumeration *ACM, CCS* 2006.

[45] J. Yang, D. Wijesekera, S. Jajodia. Subject Switching Algorithms for Access Control in Federated Databases. *DATABASE AND APPLICATION SECURITY ACM*, 2001.

[46] XACML RBAC profile. `http://xml.coverages.org/`

[47] T. Yu, M. Winslett. Supporting Structured Credentials and Sensitive Policies through Interoperable Strategies for Automated Trust Negotiation. *TISSEC, ACM Journal*, 2003.

[48] D. Zhang, K. Ramamohanarao, T. Ebringer. Role Engineering using Graph Optimization. *SACMAT* 2007.