EXAM-S: an Analysis tool for Multi-Domain Policy Sets

Rodolfo Ferrini

Technical Report UBLCS-2009-06

March 2009

Department of Computer Science

University of Bologna

Mura Anteo Zamboni 7
40127 Bologna (Italy)

Dottorato di Ricerca in Informatica
Università di Bologna, Padova

Settore scientifico disciplinare: INF/01
Ciclo XXI

EXAM-S: an Analysis tool for Multi-Domain Policy Sets

Rodolfo Ferrini

March 2009

Coordinatore:
Prof. Simone Martini

Relatore:
Prof. Elisa Bertino, Prof. Antonella Carbonaro

Abstract

As distributed collaborative applications and architectures are adopting policy based management for tasks such as access control, network security and data privacy, the management and consolidation of a large number of policies is becoming a crucial component of such policy based systems. In large-scale distributed collaborative applications like web services, there is the need of analyzing policy interactions and integrating policies. In this thesis, we propose and implement EXAM-S, a comprehensive environment for policy analysis and management, which can be used to perform a variety of functions such as policy property analyses, policy similarity analysis, policy integration etc. As part of this environment, we have proposed and implemented new techniques for the analysis of policies that rely on a deep study of state of the art techniques. Moreover, we propose an approach for solving heterogeneity problems that usually arise when considering the analysis of policies belonging to different domains. Our work focuses on analysis of access control policies written in the dialect of XACML (Extensible Access Control Markup Language) [71]. We consider XACML policies because XACML is a rich language which can represent many policies of interest to real world applications and is gaining widespread adoption in the industry.

Acknowledgements

In this section I would like to thank all the many people that helped me during this three years.

First of all I would like to thank my advisors: Prof. Antonella Carbonaro and Prof. Elisa Bertino:

Prof. Antonella Carbonaro gave me something more than advises for my PhD experience: she is the friend with which I have shared happiness, sadness and the personal troubles I had during this three years. Thank you Antonella.

Prof. Elisa Bertino gave me the chance to work in an extraordinary research group. She taught me how to overcome my limits working harder and better. I will never forgive her excellent help and the CERIAS Group at Purdue University.

Moreover, I would like to thank Prof. Simone Martini for its fundamental meta-tutoring activity and for the time he spent helping me. I would like also to thank my external reviewers who accepted to evaluate this work, dedicating to it part of their valuable time.

A special thank goes to my family and my friends that always supported me in this three years and during my stay at Purdue.

Most of all, I would like to thank you Romina... because you are like you are.

Contents

List of Figures

Chapter 1

Introduction

In the last decades, Internet has become the *de facto* standard in communication. On top of Web technologies, collaborative applications such as distributed systems and complex SOA architectures, have been developed for resource provisioning and data sharing. Such systems introduce however several security issues. Grids, federations as well as simple service providers and intra-organization systems, need to find the best trade-off between flexibility in data providing and reliability of the access control model.

Security in these scenarios is usually managed by exploiting policies-based access control models. This approach is widespread adopted nowadays since allow users and system administrators to decouple the access control management from the application logic. Particularly important class of such security policies is represented by access control policies which determine whether requests to protected resources are permitted or denied. Various types of access control models and mechanisms have emerged, such as PolicyMaker [12], the ISO 10181-3 model [1] and the eXtensible Access Control Mark-up Language (XACML) [72].

However, even for simple scenarios, the maintenance of consistent policy sets is not a trivial task. Moreover, due to the dynamism and complexity of those collaborative systems, tools that support the analysis of security policies are crucial.

Finally, the scenario under consideration becomes even more complicated when taking into account set of policies belonging to different domains.

For these reasons, analysis tools need to support powerful analysis services able to be executed over multi-domain policy sets. In the remaining part of this chapter we discuss the motivations underlying our work and a sketch of the proposed solutions along with an outline of the following chapters.

1.1. Policy Analysis: Requirements

Security policies can be considered as rules stating whether some subjects have the privileges to access some resources. By this perspective, a policy-based access control model is a set of rules defined over an application domain. Usually, it is a good practice to decouple the security policies from the application logic in order to enhance modularity and simplify management. Policy languages fulfill such requirement by offering formalisms and related tools supporting the specification and analysis of such rule sets.

In this thesis we focus our attention on XACML-based policies. XACML (eXtensible Access Control Mark-up Language) [72] is the OASIS standard language for the specification of access control policies. As suggested by the name, XACML is an XML language able to express a policy in terms of rules over different kind of entity attributes. Rules are then collected into policies and combined with rule combining algorithms. Such algorithms are used to define precedence in the application of rules if more than one of them applies for a single request. Table 1.1 shows a simplified example of a XACML rule[1]. XACML has gained widespread adoption as an industry standard and a detailed description of its features will be presented in Chapter 2.

However, even if the creation of XACML policies is well supported[2], the analysis and verification of properties that a policyset has to satisfy is still an ongoing

[1]Some syntactical details have been omitted.

[2]http://sunxacml.sourceforge.net/javadoc/index.html

```
<Rule RuleId="examplerule" Effect="Permit">
  <Target>
    <DisjunctiveMatch>
      <ConjunctiveMatch>
        <Match MatchId="function:string-equal">
          <AttributeValue>PhDStudent</AttributeValue>
          <AttributeDesignator
            AttributeId="action-type"
            Category="...attribute-category:subject"
            DataType="...#string"/>
        </Match>
        <Match MatchId="function:string-equal">
          <AttributeValue>Read</AttributeValue>
          <AttributeDesignator
            AttributeId="action-type"
            Category="...attribute-category:action"
            DataType="...#string"/>
        </Match>
        <Match MatchId="function:string-equal">
          <AttributeValue>Tech_Paper</AttributeValue>
          <AttributeDesignator
            AttributeId="action-type"
            Category="...attribute-category:resource"
            DataType="...#string"/>
        </Match>
      </ConjunctiveMatch>
    </DisjunctiveMatch>
  </Target>
</Rule>
```

Table 1.1: A XACML rule.

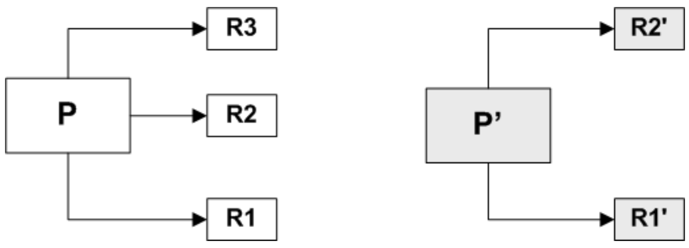


Figure 1.1: Simple policy example

work and no standard approaches have been widely accepted. The reason is related to the fact that the specification of XACML is not based on a formal semantics. This problem is crucial since the properties that security administrators should verify are strongly related to the semantics of the involved policies. In the following, some of the most interesting properties that a policy analysis tool should support will be described and discussed in detail.

1.1.1 Policy Comparison

Consider a collaborative scenario in which system partners need to compare their access control policies in order to understand if similar kind of users have similar capabilities. This kind of requests are of particular interest in federated systems where users belonging to partners organizations may have the rights to access shared resources. Consider the policies shown in Figure 1.1 where policy P is composed by the rules R_1, R_2 and R_3 and P' is composed by the rules R'_1 and R'_2. Policy comparison is the problem of verifying whether two (sets of) policies yield to similar results.

Policy comparison represents an important issue in real case scenarios. For example, as policies are increasingly being deployed at various levels within a distributed system - network devices, firewalls, hosts, applications - an important issue is to determine whether all the deployed policies authorize the same set of requests. Hence a strong requirement in the development of an analysis environment is devising techniques and tools for assessing policy similarity.

1.1.2 Heterogeneous Policy Analysis

The comparison of policies rely on a fine-grained inspection of attribute names and values within the rules. However, we cannot expect that policies belonging to different organizations are based on the same vocabulary. Such heterogeneity in names and values may result in considering as incomparable two policies even when their effects are semantically equivalent. We classify these heterogeneities in two different categories:

- *Terminological heterogeneity*: there is terminological heterogeneity when the same concept is expressed used different terms;
- *Domain heterogeneity*: there is domain heterogeneity when similar application domains are modeled by taking into account different perspectives;

When considering the policy analysis in a relaxed scenario with multi-domain policies, additional issues arise. The most challenging problems are related to the development of a common understanding between the vocabularies under considerations and extracting and formalizing an application domain based on a (possibly partial) knowledge of the domain itself. As an example, consider the policy introduced in Table 1.1. The policy specifies a permit rule allowing student to read technical papers. We have used the term **Tech_Paper** as a representation of the concept **Technical Paper**. However, **Tech_Paper** is just a shortcut, a list of characters used for convenience. In order to effectively compare policies belonging to different organizations a string-based match is not enough.

In the last decade, due to the introduction of the Semantic Web paradigm [9], ontologies have been adopted for the formal representation of application domains. On top of ontologies several techniques have been developed for the alignment and the integration of heterogeneous data. These approaches can be adopted also in our context improving and optimizing state of the art strategies taking advantage of the specific features of XACML policies.

1.1.3 Reasoning on Application Domains

XACML is essentially based on Attribute Based Access Control (ABAC) in which access control is enforced depending by the values of certain resource and subject attributes. However, it is often important to take into account also the semantic relationships among the attributes characterizing the resources and subjects. XACML, natively, is not able to exploit this kind of information and even its specialized profiles give support to just a small portion of all the possible scenarios[3]. Moreover, due to the rigid structure and features of the underlying data model, XACML is not able to represent important security constraints such as dynamic separation of duty (SoD) constraints.

In order to address such shortcomings XACML needs to be improved and this improvement requires extensions to both the XACML language, namely the introduction of specialized functions, and the XACML reference architecture and engine. Therefore, the main issue is to identify a specialized data structure to couple with the reference XACML architecture and engine able to support both the maintenance of past records and the reasoning capabilities. For the same motivations introduced in the above section, ontologies[4] represent the most suitable tool to be coupled with XACML to address its limitations. However, enhancing XACML with such specialized features raises policy analysis to a higher level. This means that the analysis techniques developed for the standard language are not enough and new methodologies needs to be devised.

1.2. EXAM-S: Motivations

So far, we have described some of the issues arising from the management of a policy-based access control model. To date, no comprehensive environments exist supporting the management of heterogeneity issues arising by considering multi domain policy datasets and a large variety of query analysis. Specialized

[3]List of profiles currently available in XACML: http://docs.oasis-open.org/xacml/2.0/

[4]From now on the term ontology is used to represent a Semantic Web ontology.

techniques and tools have been proposed, addressing only limited forms of analysis (detailed discussion is presented in Chapter 3). Common limitations concern:

- *policy conditions*: only policies with simple conditions can be analyzed [35];
- *relationship characterization*: in that for example one can only determine whether two policies authorize some common request, but no characterization of such request is provided.
- *policy comparison*: in Section 1.1.1 we defined policy comparison as: *the problem of verifying whether two (sets of) policies yield to similar results*. It is important to outline the use of the word *similar* with respect to *same*. Some approaches (as for example [56]) consider policy comparison as a boolean problem. In contrast, we think that even if two policies are different, it is important to have a solution that gives a measure of that difference. Maybe two policy sets could be evaluated as different but they may be very similar and this is an information that a system administrator could take into account;
- *reasoning services*: the definition of analysis services based on application domain reasoning is a new approach never taken into account in existing approaches. A sketch of the problem along with some possible analysis proposals is introduced in [55];
- *heterogeneous domain*: at the best of our knowledge no tools have been developed for the management of multi-domain policy sets.

1.2.1 Lack of Semantics

The fact that no one of the tools proposed so far have been widely accepted is due to the fact that XACML lacks formal semantics. In the last years, several proposals have been made to fill such gap [35, 47, 56] each of them taking into account different subsets of the XACML features, translating the language into different logics[5] and providing reasoning services with different complexity. Each of them has both strong and weak points. For example services in [56] are defined on a subset of XACML bigger than the one considered in [35]. However, considering less expressivity yield to the implementation of systems with better performance when reasoning on large policy data sets.

1.2.2 Ontologies and Interoperability

The Semantic Web [9] has been proposed in 2001 as an extension of the current Web in which resources are enriched with a well-defined, machine-processable meaning. One of the main features of the Semantic Web paradigm is the use of domain ontologies in order to describe the semantics of the data. The actual W3C Recommendation for the development of Web ontologies is OWL an XML-based language modeled on top of the Description Logic ($\mathcal{DL}$) family of representation languages.

In the last years, a number of different ontology-based techniques have been proposed for addressing interoperability issues. The most interesting one is Ontology Matching, that is, the process whereby two ontologies are semantically related at conceptual level; source ontology entities are mapped onto the target ontology entities according to those semantic relations. When two heterogeneous systems define their terminology according to the knowledge modeled within an ontology, an ontology mapping procedures allows one to find relations between the semantic schemas and thus between the data managed by those systems. We can take advantage of Ontology Mapping and semantic technologies for solving the problem of policy heterogeneity.

1.3. Aims of the Thesis

The aims of this thesis are twofold.

[5]The approaches proposed in [35,47,56] provide mapping respectively with First Order Logic, MTBDD (Propositional Logic) and Description Logic.

On the one hand, we introduce XACML+OWL a framework in which the XACML language, architecture and engine are extended with ontology reasoning. The approach proposed in XACML+OWL allows to decouple the definition of the main actors and constraints of the access control model from the definition of the permissions that can be associated with them. This solution improve the flexibility of XACML in realizing complex security systems and it also overcomes the XACML limitations outlined in Section 1.1.3. In Chapter 5 we shows the feasibility of our approach modeling the Role Based Access Control (RBAC). Specifically, we give support to a particular version of RBAC named *constrained hierarchical RBAC* in which roles are organized in a hierarchy and several constraints such as static and dynamic separation of duties can be defined over such roles.

On th other hand, we design EXAM-S (Environment for XACML policy Analysis and Management with Semantic technologies), a new analysis tool, combining state of the art technologies and offering new solutions based on the introduction of novel analysis services. We also compare EXAM-S with existing approaches in order to evaluate both correctness and efficiency of the model. We provide a prototype implementation of the EXAM-S architecture along with experimental evaluations and a detailed discussion of the results. Finally, we discuss the applicability of the ideas developed in EXAM-S in other contexts such as privacy policies and digital identity management.

The realization of EXAM-S is the main contribution of the thesis. The major issues we have addressed in EXAM-S can be categorized as follows:

Figure 1.2 depicts a conceptual representation of the aims of this dissertation. The starting point of our work is state of the art policy analysis services (Red Box). Then, this thesis has three different purposes:

1. *Heterogeneity between policies*: to deal with the problem of policy heterogeneity we propose a stack of semantic integration techniques. Usually, reasoning systems adopt the *unique name assumption*, that is, that different names have different meanings. In our scenario such assumption cannot be made

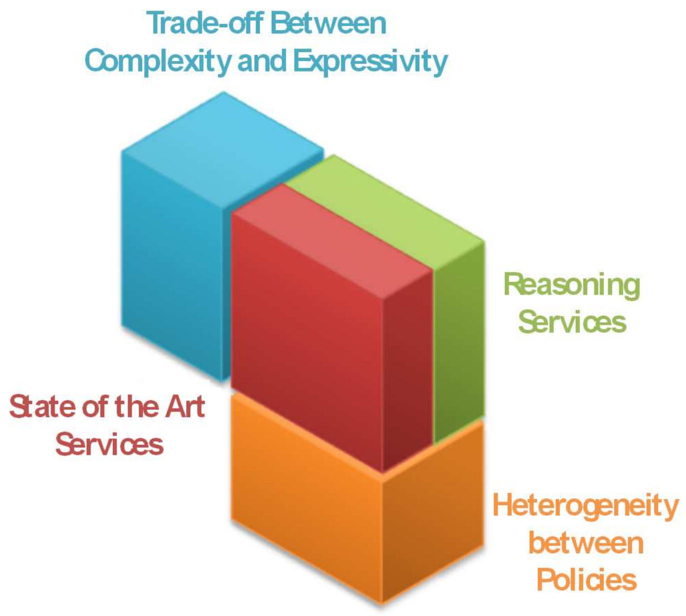

Trade-off Between Complexity and Expressivity
Reasoning Services
State of the Art Services
Heterogeneity between Policies

Figure 1.2: Thesis purposes

since we are working considering policies belonging to different domains. For this reason, we create an ontology-based process that solves the heterogeneity problems by creating a knowledge base of unique names exploited by the upper-level procedures.

2. *Trade-off between Complexity and Expressiveness*: to analyze the trade-off between complexity and expressiveness about the approaches presented so far. This is a crucial point in the development of EXAM-S. An extensive analysis of the formalisms currently adopted is necessary in ensuring a flexible design of the system. According to such analysis we then develop an hybrid set of services that represent a trade-off between expressivity and complexity.

3. *Reasoning Services*: to define new services that take into consideration the policy terminology enriched with semantic data. We add new, original services that exploit semantic techniques supporting powerful analysis services. We designed such technologies as an alternative solution to stan-

dard state of the art approaches. At the best of our knowledge no existing tools support analysis services based on reasoning over domain knowledge bases.

The work of this thesis can be logically grouped in four different steps:

- *Preliminaries*: policy analysis is a complex task and there are several issues that must be taken into consideration before the design of EXAM-S. For this reason we discuss related work and provides theoretical background along with preliminary definitions that are exploited in the following parts of the thesis.
- *The Model*: this part concerns the design of the EXAM-S system. The modules related to the policy analysis are the main contributions of this work and for this reason they have been treated in more details.
- *The Implementation*: the EXAM-S architecture, implementation and experimental evaluations are provided in this part. Firstly, we provide the architecture of EXAM-S considering how the analysis services described in the previous part can be organized together. Secondly, we describe the details about the system implementation and discuss the experimental results providing comparisons with related works.
- *Appendix*: interoperability is an important issue not only regarding security. In this section we provide detailed descriptions on how the solutions developed for EXAM-S can be exploited in other fields of research.

1.4. Outline

This thesis is organized as follows:

- ***Chapter* 2**: in this chapter we introduce background information and necessary preliminaries needed to understand the technical contributions of the dissertation. Specifically, we provide a detailed description of XACML, Description Logics and Ontology Mapping.

- ***Chapter* 3**: in this chapter we survey work related to this dissertation. Specifically, we describe existing access control policy models and related languages, and we review policy analysis and verification approaches.

- ***Chapter* 4**: in this chapter we develop the theory behind EXAM-S. We present the problem of policy heterogeneity providing the definitions that are used in Chapter 6 in the development of the algorithms. We provide the details of the most relevant approaches in the field, discussing the differences between them and pointing out which of the proposed methodologies are suitable to be combined in our model. Finally, we introduce the theory behind the definition of reasoning services over domain ontologies.

- ***Chapter* 5**: in this chapter we present an approach for extending XACML with new functions for the management of ontology driven access control models. We introduce both the theory behind the new functions and the solutions to the issues arising from the application of our approach in practice. Even if the development of this framework is not directly associated with EXAM-S, the functionalities introduced in this Chapter are taken into consideration in Chapter 7 when we define the core EXAM-S analysis services. Our services are able to deal with both standard and semantic functions.

- ***Chapter* 6**: in this chapter we present the technologies adopted for solving the heterogeneity problem. These technologies are organized in a comprehensive process that can be applied to a large set of real case scenarios. In this section we propose both definitions and the algorithms that we have developed.

- ***Chapter* 7**: in this chapter we describe the policy analysis services. We provide the details of two different modules namely: *policy filtering* and *policy similarity analyzer* (PSA). Whereas the PSA provides a precise framework for the analysis of policies, the filtering module has been developed as a support tool for computing quickly the similarity between two policies. When the set of policies involved into the analysis process is very large, the combination of the filtering module and PSA provide a good balance between performance and precision in the results.

- ***Chapter 8***: P3P Policy Similarity. P3P [23] is a W3C Recommendation for the development of privacy policy. Since many websites nowadays implement P3P policies [28], P3P policies represents another domain that is suitable for applying our similarity measure. We describe briefly the P3P language and the formalization of a similarity function. Experimental evaluations are reported and described.

- ***Chapter 9***: in this chapter we present the architecture of EXAM-S. The purpose of this chapter is to analyze how the services previously introduced can be combined together. We address this requirement by providing a detailed description of the relationship between the main services along with the design of the dedicated modules.

- ***Chapter 10***: we provide the experimental evaluations and a comparative evaluation between the obtained results and related approaches.

- ***Chapter 11***: we present concluding remarks as well as possible areas of future work.

- ***Appendix A***: Interoperability in Digital Identity Management. Interoperability is an important issue associated with several research topics. In this section we details the results of the application of an ontology-based approach for dealing with heterogeneities in a Digital Identity Management scenario.

Chapter 2

Background

In this chapter we provide the background which is necessary to understand the remainder of the thesis. First of all, we recall the basis of the XACML standard. Furthermore, we introduce notions about Description Logics, ontologies and related techniques such as ontology mapping.

2.1. XACML

XACML (eXtensible Access Control Mark-up Language) [Moses, 2003] is the OASIS standard language for the specification of access control policies. It is an XML language able to express a large variety of policies, taking into account properties of subjects and protected objects as well as context information. In general, a subject can request an action to be executed on a resource and the policy decides whether to deny or allow the execution of that action. Several profiles such as a role profile, a privacy profile etc. have been defined for XACML. An XACML policy consists of three major components, namely a *Target*, a *Rule set*, and a *rule combining algorithm* for conflict resolution.

- The *Target* identifies the set of requests that the policy is applicable to. It contains attribute constraints characterizing subjects, resources, actions, and environments.

- Each Rule in turn consists of another optional *Target*, a *Condition* and an *Effect* element. The rule Target has the same structure as the policy Target. It specifies the set of requests that the rule is applicable to. The Condition specifies restrictions on the attribute values in a request that must hold in order for the request to be permitted or denied as specified by the Effect. The Effect specifies whether the requested actions should be allowed (*Permit*) or denied (*Deny*). The restrictions specified by the target and condition elements correspond to the notion of attribute-based access control, under which access control policies are expressed as conditions against the properties of subjects and protected objects. In XACML such restrictions are represented as Boolean functions taking the request attribute values as input, and returning true or false depending on whether the request attributes satisfy certain conditions. If a request satisfies the policy target, then the policy is applicable to that request. Then, it is checked to see if the request satisfies the targets of any rules in the policy. If the request satisfies a rule target, the rule is applicable to that request and will yield a decision as specified by the Effect element if the request further satisfies the rule condition predicates. If the request does not satisfy the policy(rule) target, the policy(rule) is ”Not Applicable” and the effect will be ignored.

- The *Rule combining algorithm* is used to resolve conflicts among applicable rules with different effects.

Figure 2.1 gives an overview of a policy structure. Whereas Table 2.1 gives the syntax of the simplified format that will be used in Chapter 7 in the definition of our similarity measure.

2.1.1 Rule Combining Algorithm

Because a `Policy` or `PolicySet` may contain multiple policies or Rules, each of which may evaluate to different access control decisions, XACML a mechanism to combine access decisions. This is accomplished using a collection of combining

```
POLICY: <policy policy-id = "policy-id combining-algorithm = "combining-algorithm >
    (TARGET ELEMENT)?
    < permitrules >
        (RULE ELEMENT)*
    </permitrules>
    <denyrules>
        (RULE ELEMENT)*
    </permitrules>
</policy>

RULE ELEMENT:
<rule rule-id="rule-id effect="rule-effect>
    (TARGET ELEMENT)?
    <condition>PREDICATE</condition>
</rule>

TARGET ELEMENT:
    <target>
        <subject>PREDICATE</subject>
        <resource>PREDICATE</resource>
        <action>PREDICATE</action>
    </target>

PREDICATE:
(attr_name ⊕ (attr_value)+)*
```

attr_name denotes attribute name, *attr_value* denotes attribute value and
$\oplus$ denotes any operator supported by the XACML standard.

Table 2.1: A XACML simplified policy structure.

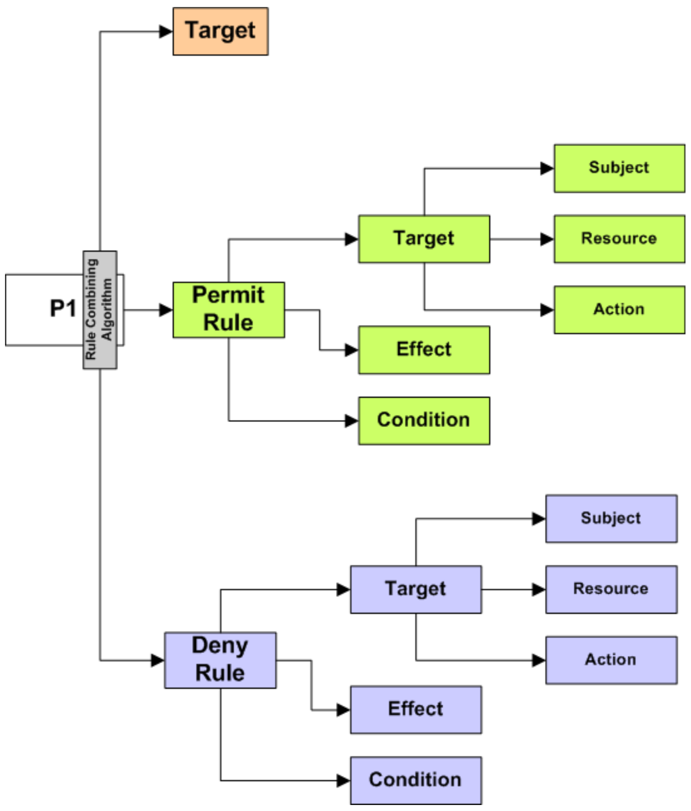


Figure 2.1: A XACML Policy Structure

algorithms, where each algorithm represents a different way of combining multiple access decisions into a single one. Following is a list of the most common combining algorithms:

- `Permit-overrides`. If any rule evaluates to Permit, then the combined decision is also Permit.
- `Deny-overrides`. If any rule evaluates to Deny, then the combined decision is also Deny.
- `First-applicable`. The effect of the first rule that applies is the decision of the policy. The rules must be evaluated in the order that they are listed.
- `Only-one-applicable`. If more than one rule is applicable, return Indeterminate. Otherwise return the access decision of the applicable rule.

In this dissertation, we use the following notation: for a XACML policy element P, we refer to its `Target`, `Effect` (in cases of Rules), its ordered list of children policy elements, its parent policy element and combining algorithm using *P:target*, *P:effect*, *P:children*, *P:parent*, *P:comb* respectively. *P:pos* is used to refer to the position of *P* w.r.t its sibling policy elements.

2.1.2 Hierarchical and Multiple Resource Profile

The policy evaluation performed by a XACML PDP is defined in terms of a single requested resource, with the authorization decision contained in a single Result element in the response. However, A Policy Enforcement Point, or PEP, may wish to submit a single request context for access to multiple resources, and may wish to obtain a single response context that contains a separate authorization decision (Result) for each requested resource. Such a request context might be used to avoid sending multiple decision request messages between a PEP and PDP, for example. Alternatively, a PEP may wish to submit a single request context for all the nodes in a hierarchy, and may wish to obtain a single authorization decision that indicates whether access is permitted to all of the requested nodes. The Multiple Resource Profile provides a mechanism such that a PEP can request authorization decisions for multiple resources in a single request context. It is important to note that the Multiple Resource Profile does not affect the policy itself. It deals with the XACML Access Requests, introducing syntactic shorthand so that multiple requests contexts can be merged into one. The Hierarchical Resource Profile allows users to specify one policy that applies to an entire subtree of a hierarchy, rather than having to specify a separate policy for each node of the subtree. In this Profile, a resource organized as a hierarchy may be a with a single root (tree) or multiple roots (forest), however cycles are not allowed. The nodes in a hierarchical resource are treated as individual resources. An authorization decision that permits (or denies) access to an interior node does not imply that access to its descendant nodes is permitted (or denied).

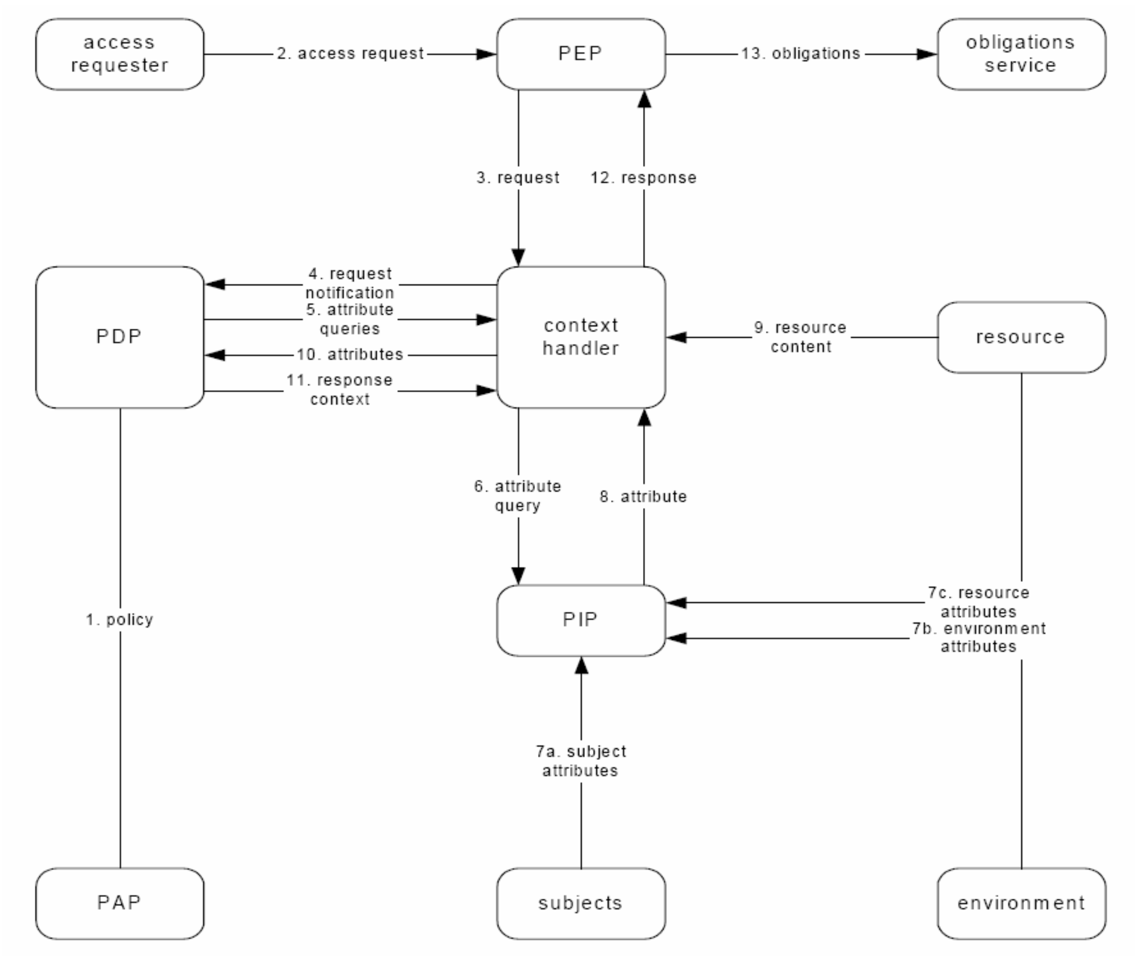


Figure 2.2: The XACML Data Flow

2.1.3 The XACML architecture

The XACML architecture (Figure 2.2) consists of four main components: (i) the Policy Administration Point (PAP); (ii) the Policy Decision Point (PDP); (iii) the Policy Enforcement Point (PEP); (iv) the Policy Information Point (PIP). The PAP creates the policies evaluated by the PDP. The PDP evaluates applicable policies against the incoming requests and sends the resulting authorization decision back to the PEP. The PEP performs access control, making decision requests, and enforcing the authorization decision provided by the PDP. Finally, the PIP is the component that provides the attribute values required by the PDP during the evaluation phase. There is another important component in XACML architecture: the context handler. This component acts as a mediator between the PEP, the PDP, and the PIP by receiving and dispatching information to the appropriate component.

2.2. Description Logics

Description Logics ($\mathcal{DL}$) is a family of knowledge representation languages which can be used to represent the terminological knowledge of an application domain in a structured and formally well-understood manner [6]. The name comes from the facts that, on the one hand, the application domains are described using concept descriptions and, on the other hand, they possess formal, logic-based semantics which can be given by a translation into first-order logic ($\mathcal{FOL}$). The rest of this Section is organized as follows: in Subsection 2.2.1 we give an overview of Description Logics. $\mathcal{DL}$ syntax and semantics are introduced in Subsection 2.2.2 whereas the reasoning services that come with a $\mathcal{DL}$ formalism are proposed in Subsection 2.2.3.

2.2.1 $\mathcal{DL}$ Overview

Each $\mathcal{DL}$ consists of the following building blocks: atomic concepts, atomic roles and individuals. Atomic concepts correspond to unary predicates in $\mathcal{FOL}$ (e.g., $\mathsf{Student}(x)$), atomic roles correspond to binary predicates in FOL (e.g., $\mathsf{enrolledIn}(x; y)$) and individuals represent constant terms in $\mathcal{FOL}$. Atomic concepts and roles are elementary descriptions of objects; complex concepts and roles can be built on top of them using $\mathcal{DL}$ constructors. For example, applying a concept disjunction constructor ($\sqcup$) on the atomic concepts Male and Female, we retrieve the set of all individuals who are either Male or Female: $\mathsf{Male} \sqcup \mathsf{Female}$. In addition to disjunction, $\mathcal{DL}$ typically provides the standard boolean operators as constructors: concept conjunction ($\sqcap$) and concept negation ($\neg$). Most $\mathcal{DL}$ languages also provide a restricted quantification, in terms of universal and existential restrictions on roles. In addition to constructors that allow one to form complex concepts and roles, $\mathcal{DL}$ also provides means for expressing axioms (logical relations) involving concepts and roles. For example, we can specify concept inclusion of the form $\mathsf{Student} \sqsubseteq \mathsf{Person}$ stating that every student is a person, and role inclusion such as $\mathsf{isBrother} \sqsubseteq \mathsf{isRelated}$ stating that if two individuals are brothers, then they are related.

A DL knowledge base (KB) typically consists of the following components:

- A $\mathcal{T}Box$ containing intensional knowledge (axioms and concepts) that form the basic terminology of the KB. The axioms in the TBox are concept inclusions of the form $C_1 \sqsubseteq C_2$ where C_1 and C_2 are concepts (not necessarily atomic).
- An $\mathcal{R}Box$ containing role inclusion axioms of the form $R_1 \subseteq R_2$ where R_1 and R_2 are DL Roles.
- An $\mathcal{A}Box$ containing extensional knowledge about the individuals in the domain. Axioms in the ABox are of the form `C(a)`, called concept (or type) assertions, and `R(a,b)`, called role assertions, where a,b are individual names, `R` is a role, and `C` is a concept.

There are different types of TBoxes depending on the nature of the concepts occurring in their axioms. The simplest TBox type consists of a restricted form of concept inclusion axioms called concept *definitions*: sentences of the form $A \sqsubseteq C$ or $A \equiv C$, where `A` is atomic. Restricting a TBox to concept definitions which are both *unique* (each atomic concept occurs only once on the LHS of an inclusion axiom) and *acyclic* (the RHS of an axiom cannot refer, directly or indirectly, to the concept in the LHS) yields a definitorial TBox. On the other hand, if a TBox contains axioms of the form $C \sqsubseteq D$ where `C` is non-atomic, then the axiom is called a *general concept inclusion axiom* (GCI) and the TBox is called a *general* TBox. The distinction between definitorial and general TBoxes is important since a definitorial TBox greatly reduces reasoning complexity. A common example of DL concept constructors is represented by DL number restrictions. The most expressive form is *qualified* number restrictions, which allow one to build the concepts $\geqslant nR.C$ and $\leqslant nR.C$ from a role R, a natural number n and a concept C. For example, qualified number restrictions can be used to represent a father of exactly two sons:

$$\mathit{Male} \sqcap \leqslant 2\ \mathit{hasChild}.\mathit{Male} \sqcap \geqslant 2\ \mathit{hasChild}.\mathit{Male}$$

More restricted forms are unqualified number restrictions that do not allow one to specify what kind of concept is used as role filler in the restriction. For example, unqualified number restrictions can be used to denote a father of exactly two children:

$$\mathit{Male} \sqcap \leqslant 2\ \mathit{hasChild} \sqcap \geqslant 2\ \mathit{hasChild}$$

Finally, an important feature of DL is the support for datatype, i.e., support for describing concepts using numbers, strings, regular expressions, IP addresses, etc. The main approach is to equip DL with an interface to concrete domains, together with a set of built-in predicates which are associated with that interface. The interface is created by using a new type of roles, called datatype (or concrete) roles, which links abstract objects from the DL domain with datatype predicates from the concrete domain. Also, new concept constructors related to these datatype roles is added. For example, we can denote the set of all people who are more than 18 years old using a datatype role: $\exists \mathit{age}. \geqslant 18$. Since concrete domains are important for the purposes of this thesis, in the next section we formally present their definition and properties.

2.2.2 Syntax and Semantics

In the following the syntax and the semantics of the $\mathcal{ALC}$ description logics is proposed [6].

$$C, D ::= A \mid \top \mid \bot \mid \neg C \mid C \sqcap D \mid C \sqcup D \mid \forall R.C \mid \exists R.C$$

The Description Logics $\mathcal{ALC}$ is based upon a set **CN** of concept names A (including $\top$ and $\bot$), a set **RN** of role names R, and a set **IN** of individual names a. The concepts of the language are constructed by concept names A, role names R, the connectives $\sqcap$, $\sqcup$ and $\neg$, and the quantifiers $\forall$ and $\exists$. Every concept name $A \in$ **CN** is an $\mathcal{ALC}$-concept. If R is a role name and C, D are $\mathcal{ALC}$-concepts, then $\neg C, C \sqcap D, C \sqcup D, \forall R.C, \exists R.C$ are $\mathcal{ALC}$-concepts.

In order to define a formal semantics of $\mathcal{ALC}$-concepts, we consider *interpretations* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ that consist of a non-empty set $\Delta^{\mathcal{I}}$ (the domain of the interpretation) and the interpretation function $\cdot^{\mathcal{I}}$ which assigns to every atomic concept A a set $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, to every atomic role R a binary relation $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ and to every individual a an element $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$. The interpretation function is extended to concept descriptions by the following inductive definitions:

$$
\begin{array}{rcl}
\top^{\mathcal{I}} & = & \Delta^{\mathcal{I}} \\
\bot^{\mathcal{I}} & = & \emptyset \\
(\neg C)^{\mathcal{I}} & = & \Delta^{\mathcal{I}} \setminus A^{\mathcal{I}} \\
(C \sqcap D)^{\mathcal{I}} & = & C^{\mathcal{I}} \cap D^{\mathcal{I}} \\
(C \sqcup D)^{\mathcal{I}} & = & C^{\mathcal{I}} \cup D^{\mathcal{I}} \\
(\forall R.C)^{\mathcal{I}} & = & \{a \in \Delta^{\mathcal{I}} \mid \forall b.(a,b) \in R^{\mathcal{I}} \rightarrow b \in C^{\mathcal{I}}\} \\
(\exists R.C)^{\mathcal{I}} & = & \{a \in \Delta^{\mathcal{I}} \mid \exists b.(a,b) \in R^{\mathcal{I}}\}
\end{array}
$$

Table 2.2: $\mathcal{ALC}$ semantics.

2.2.3 Reasoning Services

When a knowledge engineer models a domain, she constructs a terminology, say $\mathcal{T}$, by defining new concepts, possibly in terms of others that have been defined before. During this process, it is important to determine whether a newly defined concept makes sense or whether it is contradictory. From a logical point of view, a concept makes sense if there is some interpretation that satisfies the axioms of $\mathcal{T}$ (that is, a model of $\mathcal{T}$) such that the concept denotes a nonempty set in that interpretation. A concept with this property is said to be satisfiable with respect to $\mathcal{T}$ and *unsatisfiable* otherwise. Checking satisfiability of concepts is a key inference. As we shall see, a number of other important inferences for concepts can be reduced to the (un)satisfiability. For instance, in order to check whether a domain model is correct, or to optimize queries that are formulated as concepts, we may want to know whether some concept is more general than another one: this is the *subsumption* problem. A concept C is subsumed by a concept D if in every model of $\mathcal{T}$ the set denoted by C is a subset of the set denoted by D. Algorithms that check subsumption are also employed to organize the concepts of a TBox in a taxonomy according to their generality. Further interesting relationships between concepts are *equivalence* and *disjointness*. These properties are formally defined as follows. Let $\mathcal{T}$ be a TBox.

- *Satisfiability*: A concept C is satisfiable with respect to $\mathcal{T}$ if there exists a model $\mathcal{I}$ of $\mathcal{T}$ such that $C^{\mathcal{I}}$ is nonempty. In this case we say also that $\mathcal{I}$ is a model of C.
- *Subsumption*: A concept C is subsumed by a concept D with respect to T if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for every model $\mathcal{I}$ of $\mathcal{T}$. In this case we write $C \sqsubseteq_{\mathcal{T}} D$ or $\mathcal{T} \models C \sqsubseteq D$.
- *Equivalence*: Two concepts C and D are equivalent with respect to T if $C^{\mathcal{I}} = D^{\mathcal{I}}$ for every model $\mathcal{I}$ of $\mathcal{T}$. In this case we write $C \equiv_{\mathcal{T}} D$ or $\mathcal{T} \models C \equiv D$.
- *Disjointness*: Two concepts C and D are disjoint with respect to $\mathcal{T}$ if $C^{\mathcal{I}} \cap D^{\mathcal{I}} = \emptyset$ for every model $\mathcal{I}$ of $\mathcal{T}$.

2.3. The Semantic Web and OWL

The current Web is built on HTML, which describes how information is to be displayed. While computers are able to parse Web pages for layout and routine processing, they are unable to process the meaning of their content. The Semantic Web [9] can be defined as an extension of the current Web in which meaning is added to resources so that machines are allowed to understand them better. This new architecture is based on the annotation of web documents with additional semantic data. In these last years a number of new languages have been proposed in order to carry out this task. XML is the basic language on which the Semantic Web is based. The most important Recommendations proposed by W3C specifically designed for the next generation of the Web are the Resource Description Framework (RDF) [62] and the Web Ontology Language (OWL) [25] which are build respectively on top of XML and RDF itself. Briefly:

- *RDF* is the W3C recommendation for the creation of metadata about resources. With RDF one can make statements about a resource in the form of a subject-predicate-object expression. The described resource is the subject of the statement, the predicate is a specified relation that links the subject and the object that is the value assigned to the subject through the predicate.

- *OWL* is the W3C recommendation for the creation of new ontology optimized for the web. The Web Ontology Language OWL is a semantic markup language for publishing and sharing ontologies on the Web. OWL is developed as a vocabulary extension of RDF and it is derived from the DAML+OIL Web Ontology Language. Essentially, with OWL one can describe a specific domain in terms of class, properties and individuals. It has three increasingly-expressive sublanguages: *OWL Lite*, *OWL DL* and *OWL Full*. An overview of the OWL language is presented in Section 2.3.1.

A number of other different languages have been developed around RDF and OWL such as SPARQL for asking and answering queries against RDF graphs,

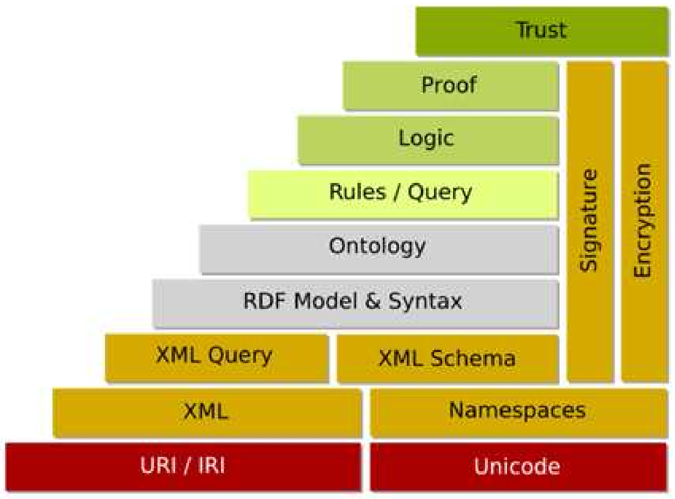


Figure 2.3: Semantic Web Layer Cake.

SKOS that provides a model for expressing the basic structure and content of concept schemes (such as taxonomies and controlled vocabularies) and SWRL a proposal for a semantic web rules language combining sublanguages of OWL (specifically OWL DL and OWL Lite) with the rule markup language RuleML. The technologies specifically developed for Semantic Web have been applied in many different contexts, as e-learning. The architecture of the Semantic Web is depicted in 2.3 and is usually referred to as *Semantic Web Layer Cake*.

2.3.1 The Ontology Web Language

The Web Ontology Language (OWL) [25] is a set of eXtensible Markup Language (XML) elements and attributes, with well defined semantics, that are used to define a vocabulary in terms of classes and their relationships. There are three species of OWL:

- OWL-Lite has limited expressiveness and is suitable for simple class hierarchies and constraints. Cardinality is restricted to values of 0 or 1.

- OWL-DL supports description logics [Baa03] and automated reasoning and is the OWL species used throughout this thesis. OWL-DL has maximum expressiveness while maintaining computational completeness (i.e., all conclusions are guaranteed) and decidability (i.e., all conclusions finish in a finite time). OWL-DL includes all language constructs but certain constructs can only be used under certain restrictions (e.g., a class cannot be an instance of another class).

- OWL-Full has the most expressiveness and syntactic freedom (e.g., a class may be treated simultaneously as a collection of individuals and as an individual itself), but offers no computational guarantees and therefore, does not support automated reasoning.

2.3.2 OWL-DL Ontologies

In this section we provide an overview of the OWL-DL ontology language since it is the one that is currently considered standard language in the development of Ontologies. All the images provided in this Section have been presented in [42].

Components of OWL-DL Ontologies

An OWL ontology consists of three main components: (i) Individuals, (ii) Properties and (iii) Classes

An *Individual* (i.e., Instance) is a constant in Description Logic and represents an object in the domain of discourse in OWL. OWL-DL, being based on Description Logics, does imposed the Unique Name Assumption. This means that just because two names are different in OWL-DL does not mean they refer to different individuals. Two different OWL-DL class names, for example, may refer to the same individual. Therefore, it must be explicitly stated in OWL-DL whether individuals are the same as each other or different from each other. An example of some individuals is shown in Figure 2.4.

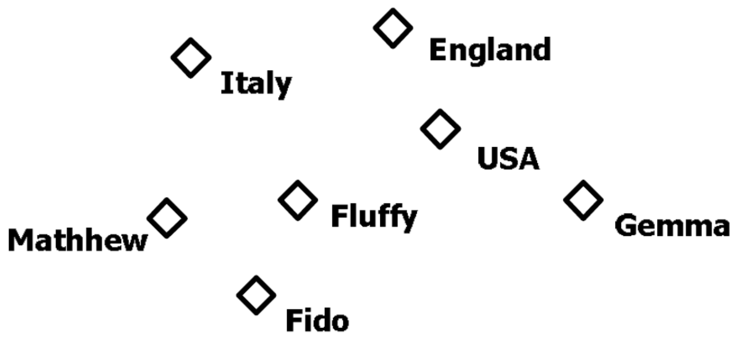


Figure 2.4: Individuals in OWL.

A *Property* is role or binary predicate in Description Logic and represents a binary relation in OWL that links two individuals together. Every OWL property has a specified domain and a range, where a property links an individual from its domain to an individual in its range.In Figure 2.5, the property `hasSibling` links the individual `Matthew` to the individual `Gemma`. Properties can have an inverse. For example, the inverse of `hasOwner` is `IsOwnerOf`. Functional properties are limited to having a single value. Properties can also be symmetrical or transitive. There are three types of properties in OWL:

1. Object properties link an individual to an individual. For example, Figure 2.6(a) shows an object property `hasSister` linking the individual `Bob` to the individual `Alice`.

2. Datatype properties link an individual to an XMLS datatype value or an RDF literal. For example, Figure 2.6(b) shows a datatype property `hasAge` linking the individual `Bob` to the data literal '28' which has type xsd:Integer.

3. Annotation properties meta-data on the model which may be added to classes, individuals or properties and are not instantiated with individuals. For example, Figure 2.6(c) shows an annotation property `dc:creator` linking the individual `Thesis` to the data literal (string) "Bob".

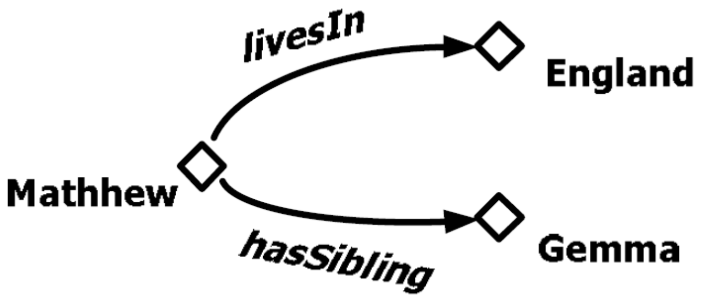


Figure 2.5: Properties in OWL.

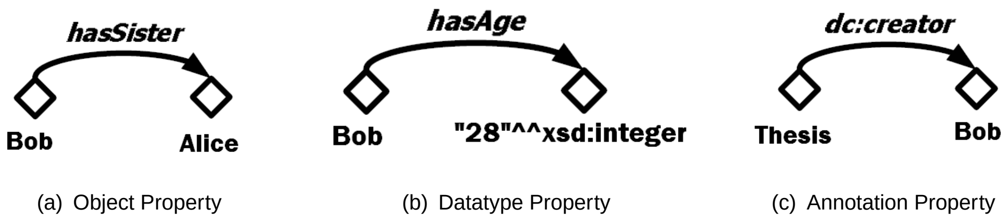


Figure 2.6: Properties in OWL.

A *Class* is a concept or unary predicate in Description Logic and represent a set of individuals in OWL that are defined using formal descriptions that state precisely the requirements for membership in the class. For example, in Figure 2.7, the class `Person` contains the individuals `Matthew` and `Gemma`, the class `Pet` contains `Fluffy` and `Fido`, and the class `Country` contains `Italy`, `England` and `USA`. Classes are concrete representations of concepts and may be organized in a `superclass`, `subclass` hierarchy called taxonomy. Subclasses specialize, or are subsumed by, their superclasses. Consider, for example, the classes `Animal` and `Cat` where `Cat` may be a subclass of `Animal` so that `Animal` is the superclass of `Cat`. This says that: (1) all cats are animals, (2) all members of the class Cat are also members of the class Animal, and (3) being a `Cat` implies being an `Animal`. One of the key features of OWL-DL is that this subsumption relationship can be automatically computed by a reasoner. OWL classes are essentially descriptions that specify the conditions that must be satisfied by an individual

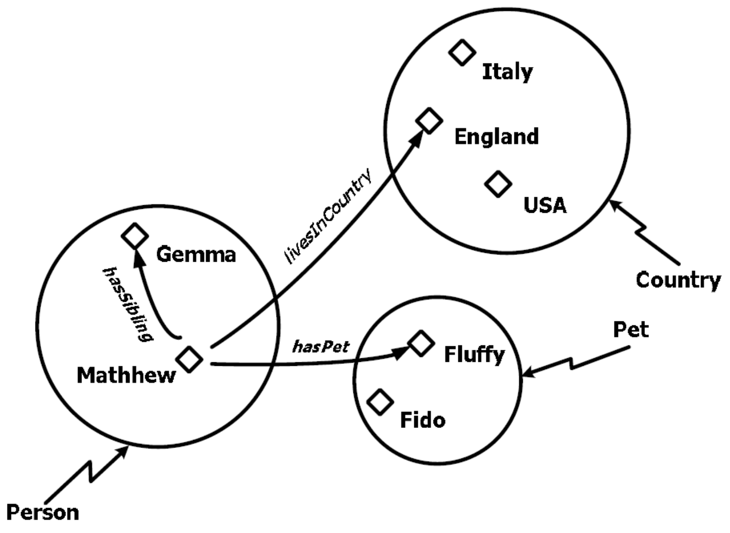


Figure 2.7: Properties in OWL.

for that individual to be a member of a particular defined or primitive class.

Reasoning in OWL-DL

Reasoning in OWL-DL is based on the Open World Assumption (OWA), and is often referred to as Open World Reasoning (OWR). OWA means that it cannot be assumed that something does not exist until it has been explicitly stated that it does not exist. In other words, just because something has not been stated as true, it cannot be assumed to be false. Instead, it must be assumed that the knowledge has just not yet been added to the knowledge base.

2.3.3 An Example of Semantics Supporting Interoperability: Semantic Web Services

Current XML-based Web service technology provides limited support in mechanizing service discovery and invocation, and their integration and composition. The vision of semantic Web services is to describe the various aspects of a Web service using explicit, machine-understandable semantics, enabling the automatic location, combination and use of Web services. Approaches from the area of semantic Web are being applied to Web services in order to keep the intervention of the human user to the minimum. Semantic markup can be exploited to automate the tasks of discovering services, executing them, composing them and enabling interoperation between them [63].

A number of different initiatives are growing around the Semantic Web service architecture i.e. W3C Semantic Web Services Interest Group[1], Semantic Web Services Initiative Architecture Committee (SWSA)[2], Large Scale Distributed Information System project[3] and Web Service Modeling Ontology working group[4].

Moreover, different frameworks have been proposed to design the overall conceptual model. The most interesting projects addressing this point are the ones proposed in [66] [15] [32]. All these projects identify the following major phases:

- *Automatic Web service discovery*: Automatic Web service discovery involves automatically locating Web services that provide a particular service and that accomplish to requested properties. A user might require, for example, "Find a service that sells train tickets between Rome and Milan and that accepts payment by VISA credit card." Currently, a human must perform this task by first using a search engine to find a service and then either reading the Web page associated with that service or executing the service to see whether it accomplishes to the requested properties. With semantic markup of services, one can specify the information necessary for Web service discovery as computer-interpretable semantic markup at a service registry and an (ontology-enhanced) search engine can automatically locate appropriate services.

- *Automatic Web service execution*: Automatic Web service execution involves a

[1]http://www.w3.org/2002/ws/swsig/
[2]http://www.daml.org/services/swsa/
[3]http://lsdis.cs.uga.edu/
[4]http://www.wsmo.org/

computer program or agent automatically executing an identified Web service. A user could request, "Buy me a train ticket from www.worldtrain.com from Rome to Milan on 20 February 2009". To execute a particular service on todays Web, such as buying a train ticket, a user generally must go to the Web site offering that service, fill out a form, and click a button to execute the service. Alternately, the user might send an http request directly to the service URL with the appropriate parameters encoded. Both cases require a human to understand what information is required to execute the service and to interpret the information the service returns. Semantic markup of Web services provides a declarative, computer-interpretable API for executing services. The markup tells the agent what input is necessary, what information will be returned, and how to execute, and potentially interact with, the service automatically.

- *Automatic Web service composition and interoperation*: Automatic Web service composition and interoperation involves the automatic selection, composition, and interoperation of appropriate Web services to perform some task, given a high-level description of the tasks objective. A user might say, "Make the travel arrangements for SACMAT 2009 conference". Currently, if some task requires a composition of Web services that must interoperate, the user must select the Web services, manually specify the composition, ensure that any software for interoperation is custom-created and provide the input at choice points. With semantic markup of Web services, the information necessary to select, compose, and respond to services is encoded at the service Web sites. One can write software to manipulate this markup, together with a specification of the tasks objectives, to automatically execute the task. Service composition and interoperation leverage automatic discovery and execution.

In order to achieve such goals some general assumption has to be introduced. First of all, services can access and interpret Web published ontologies, and can communicate using messages whose content is represented, or can be interpreted, in terms of published ontologies. Service providers publish semantic descriptions of their service capabilities and interaction protocols that can be interpreted by prospective consumers.

The potential benefits of Semantic Web services have led to the establishment of an important research area of interest to both academia and industry. Several initiatives have been proposed for semantically annotating Web services extending the conceptual model given above. Each of such approaches provides different descriptions of Web services and their related aspects that bring a different kind of support for discovery and composition.

2.4. Ontology Matching

In this Section we provide background notions about the ontology matching problem and the most interesting ontology matching techniques. We conclude the Section with a comparative table (Table 2.3) which lists the features of each one of the reviewed approach. The information provided in this Section are taken into account in Chapter 6 when considering the ontology matching algorithm that is most eligible in our context. This Section is organized as follows: in Section 2.4.1 we provide the motivations behind ontology matching, in Section 2.4.2 we give a formalization of the problem whereas in Section 2.4.3 we propose a review of state of the art techniques.

2.4.1 Overview

Schema matching aims at determining relations between entities of two different input schemas. This task is a research area in a number of different domains. Examples are data warehouses, e-commerce, database integration and more recently Semantic Web. Like the Web, the semantic web [9] will be distributed and heterogeneous. In addiction, well defined information is added in order to allow interoperability between software agents involving a minimal human efforts. The semantic interoperability over the next Web generation will be grounded in the use of ontologies. An ontology typically provides a vocabulary that describes a domain of interest and a specification of the meaning of terms used in the vocabulary.

As it has been introduced above, in the idea of its supporters, the Semantic Web will be distributed and heterogeneous. Consequently, the aim of the Semantic Web community is not to create a huge, comprehensive knowledge base but to obtain a number of different, partially overlapping ontologies. Each system based on Semantic Web technologies is expected to develop its own ontology either created from sketch or composed by exploiting some other ontology. In order to grant semantic interoperability between all this systems, it is necessary to provide mapping techniques between ontologies that are defined on overlapping domains.

2.4.2 Ontology Matching: Formalizing the Problem

Following the approach proposed in [51], it is necessary to clarify the meaning of mapping with respect to other concepts such as aligning or merging. Alignment is the process whereby two ontologies are aligned one to the other. This means that the ontologies (or at least one of the ontologies involved into the alignment process) will be modified. Mapping could be used to solve problems as alignment but it does not aim to modify the ontologies. Merging, on the contrary, is the task that taking in input two schemas returns a reconciled schema. Even for merging, a mapping provide the basic building on which a technique can be created but the aims are different. An ontology mapping is the association between entities that belong to different ontologies along with a confidence score representing their similarity.

Following the definition given in [80] we formally define a mapping element as follows:

Definition 2.1 *A mapping element is the 5-tuple* $\langle id, e, e', n, R\rangle$ *where:*

- id*: is a unique identifier of the given mapping element;*
- e *and : are the entities of the first and the second ontology respectively;*
- n*: is a confidence measure in some mathematical structure (typically in the [0, 1]);*
- R*: is a relation holding between the entities e and.*

A mapping element maps the element e belonging to O to element e' belonging to O'. An *ontology matching* between O and O' is a set of mapping elements in which $e \in O$ and $e' \in O'$. The *ontology matching problem* is the problem of finding the set of mapping element between some input ontologies that detect all the overlapping entities.

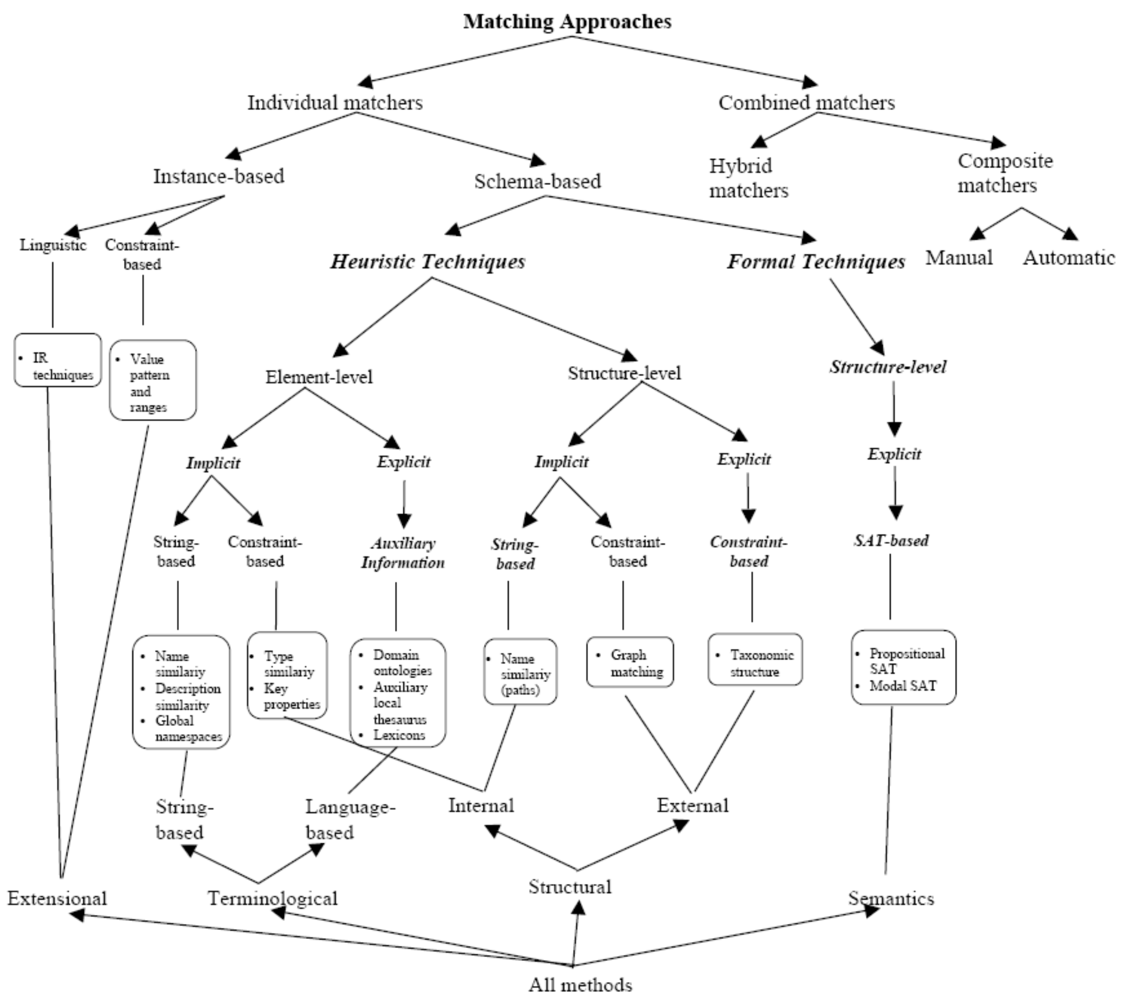


Figure 2.8: Categories of schema matching techniques

In these years a number of different approaches have been proposed to solve the ontology matching problem. Here is defined a classification model so that each involved technique can be categorized. By this way an implicit structure in which the reviewed approaches can be plugged in is offered. The just presented classification model was proposed in [77] and enhanced in [80]. It is reported here in Figure 2.4.2.

The overall classification can be read both in descending (following on how the techniques interpret the input information) and ascending manner (focusing on the kind of manipulated objects). Moreover, ontology matching approaches could be divided into *local methods* and *global methods*. Local methods are the basic ones and enable to measure the similarity correspondence at a local level i.e. given a matching element $\langle id, e, e', n, R\rangle$ for the specific R, the value of n is computed taking into account only the features of the elements and not working at the global scale of ontologies. On the contrary, global methods work on the results returned by local methods combining them and considering the ontology in its totality.

For these reasons a description of principal local methods is given and only the ascending classification of the presented model is considered. An explanation of the main methods is given below. Semantic methods are not taken into account because they are not used in the reviewed approaches. *Terminological methods* compare strings. They can be applied to the name, the label or the comments concerning entities in order to find the ones with which are similar. This class could be divided into:

- *String-based methods*: these approaches take advantage of the structure of the string (as a sequence of letters). A simple string-based method example is the string equality which returns 0 if the strings are not the same and 1 if they are the same. Formally, String equality is a similarity $\sigma : S \times S \longrightarrow [0, 1]$ such that $\forall x, y \in S, \sigma(x, x) = 1$ and if $x \neq y, \sigma(x, y) = 0$. Where S is the set of strings. A more meaningful example is *Substring similarity*. Substring similarity is a similarity $\sigma : S \times S \longrightarrow [0, 1]$ such that $\forall x, y \in S$ let t be the largest common substring of x and y and $\sigma(x, y) = \frac{2 \mid t \mid}{\mid x \mid + \mid y \mid}$. This definition can be used for building function based on the largest common prefix or largest common suffix.

- *Language-based methods*: this approach relies on using Natural Language Processing techniques to find associations between instances of concepts or classes. An example of such method is the similarity measure proposed by Resnik [citation]. Resnik *semantic similarity* makes use of an external resource in which semantically relations among terms are defined. Given two terms s and t and a partially ordered synonym resource $\langle \Sigma, \leqslant \rangle$ provided with a probability π, Resnik semantic similarity is a similarity $\sigma : S \times S \rightarrow [0, 1]$ such that: $\sigma(s, t) = \mathtt{max}_{k \in S(s,t)}(-\log(\pi(k)))$. Where k is a concept of Σ and $S(s, t) \cap \Sigma$ so that contains the concepts that subsume both s and t. Intuitively $\pi(k)$ provides the probability of encountering such concept k. Basically, this equation simply means that we search for a concept k, with maximum information content i.e. with an high value of $\pi(k)$, that subsumes both s and t.

Structural methods compare the entity structure instead of comparing their names or identifiers. This comparison can be extended considering methods that take into account the internal structure of an entity (attributes or properties) and whose that consider the similarity between related entities.

- Methods based on the *internal structure* of entities use criteria such as the range of their properties (attributes and relations), their cardinality, and the transitivity and/or symmetry of their properties to calculate the similarity between them.

- Concerning methods based on the *external structure*, the similarity comparison between two entities (belonging to different ontologies) can be based on the position of entities within their hierarchies. If two entities from two ontologies are similar, their neighbors might also be similar.

Extensional-based methods compare the extensions of classes. The easiest way to compare classes A and B when they share features it is to test their intersection. The intersection between two classes is defined as the set of common features i.e. their set of instances. Basically, classes are very similar when $A \cap B = A = B$ while when $A \cap B = 0$ the classes are very different. An example of extensional based similarity measure is Jaccard Similarity: Given two sets A and B, let $P(x)$ the probability of a random instance to be in set X, the Jaccard similarity is defined as: $\sigma(A, B) = \frac{P(A \cap B)}{P(A \cup B)}$.

Now that an overview of the mapping problem is given, in Figure 2.4.2 the implicit structure introduced above is presented. A similar table was firstly proposed in [77]. Figure 2.4.2 is a simplified version of the classification proposed there.

2.4.3 Reviewed approaches

In this section we provide some review of state of the art ontology matching tools. Figure 2.3 shows the classification of the reviewed approaches according to the categories introduced in previous section.

Anchor-PROMPT.

Anchor-PROMPT presented in [74] is an ontology merging and alignment tool with a sophisticated prompt mechanism for possible matching terms. The anchor-PROMPT is a hybrid alignment algorithm which takes as input two ontologies, (internally represented as graphs) and a set of anchors-pairs of related terms, which can be identified using string-based techniques, using user definition, or another matcher computing linguistic similarity. Then the algorithm refines them by analyzing the paths of the input ontologies limited by the anchors in order to determine terms frequently appearing in similar positions on similar paths. Finally, based on the frequencies and a user feedback, the algorithm determines matching candidates.

COMA

COMA (COmbination of MAtching algorithms) proposed in [26] is a composite schema matching tool. It provides an extensible library of matching algorithms, a framework for combining obtained results and a platform for the evaluation of the effectiveness of the different matchers. Matching library is extensible, and as from it contains 6 elementary matchers, 5 hybrid matchers, and one reuse-oriented matcher. Most of them implement string-based techniques as a background idea; others share techniques with Cupid (thesauri look-up, etc.); and reuse-oriented is a completely novel matcher, which tries to reuse previously obtained results for entire new schemas or for its fragments. Schemas are internally encoded as DAGs, where elements are the paths. This aims at capturing contexts in which the elements occur. Distinct features of COMA tool in respect to Cupid, are a more flexible architecture and a possibility of performing iterations in the matching process

Cupid

Cupid proposed in [59] implements a hybrid matching algorithm comprising linguistic and structural schema matching techniques, and computes similarity coefficients with the assistance of a domain specific thesauri. Input schemas are encoded as graphs. Nodes represent schema elements and are traversed in a combined bottom-up and topdown manner. The matching algorithm consists of three phases and operates only with tree-structures to which non-tree cases are reduced. The first phase (linguistic matching) computes linguistic similarity coefficients between schema element names (labels) based on morphological normalization, categorization, string-based techniques (common prefix, suffix tests) and a thesauri look-up. The second phase (structural matching) computes structural similarity coefficients which measure the similarity between contexts in which basic schema elements occur. The third phase (mapping elements generation) computes weighted similarity coefficients and generates final alignment by choosing pairs of schema elements with weighted similarity coefficients value higher than a fixed threshold.

GLUE

GLUE proposed in [27] employs machine learning techniques to find mappings. Given two ontologies, for each concept in the first ontology GLUE finds the most similar concept in the second one using probabilistic definitions of several practical similarity measures. In addition to this, GLUE also uses multiple learning strategies, each of which exploits a different type of information either in the data instances or in the taxonomic structure of the ontologies. The employed similarity is the joint probability distribution of the concepts involved, so instead of committing to a particular definition of similarity, GLUE calculates the joint distribution of the concepts, and lets the application use the joint distribution to compute any suitable similarity measure. The overall process could be divided into three main steps:

1. *learning distributions*: it learns the joint probability distributions of classes of each ontologies;
2. *similarity estimation*: the system estimates the similarity between two classes in function of their joint probability distributions;
3. *relaxation*: produces an alignment from the similarity matrix by using heuristic rules in order to choosing the more likely correspondences.

Quick Ontology Mapping

The approach presented in [30] is based on the efficiency / efficacy trade off evaluation: the quality of matching algorithm could be slight reduce to obtain a great improvement of efficiency. This fact allows QOM to produce mapping elements fast, even for large-size ontologies. However, due to the efficiency requirements the use of some rules is restricted. The overall process can be divided as follows:

1. QOM uses RDF triples as features;
2. Instead of comparing all entities of the first ontology with all entities of the second ontology, QOM uses heuristics to lower the number of candidate mappings, which is a major problem for run-time complexity. In this dynamic programming approach it only chooses promising candidate mappings.

3. The actual similarity computation is done by using a wide range of similarity functions [30]. The similarity computation was disburdened by removing extremely costly feature-measure combinations such as the comparison of all subclasses of two concepts.

4. These individual measures are all used as input to the similarity aggregation. Instead of applying linear aggregation functions, QOM applies a sigmoid function, which emphasizes high individual similarities and de-emphasizes low individual similarities.

5. From the similarity values we derive the actual mappings. A threshold to discard spurious evidence of similarity is applied. Further mappings are assigned based on a greedy strategy that starts with the largest similarity values first.

6. Through several iteration the quality of the results rises considerably. The returned output is a mapping table between the input ontologies.

2.4.4 Falcon-AO

Falcon-AO [48], [45] is defined in [48] as an automatic tool for aligning ontologies. There are two matchers integrated in Falcon-AO: one is a matcher based on *Linguistic Matching for Ontologies*, called LMO; the other is a matcher based on *Graph Matching for Ontologies*, called GMO and introduced in [44] by the same authors. In Falcon-AO, GMO takes the alignments generated by LMO as external input and outputs additional alignments. Reliable alignments are gained through LMO as well as GMO according to the concept of reliability. The reliability is obtained by observing the linguistic comparability and structural comparability of the two ontologies being compared. Figure 2.9, depicts the Falcon-AO main components.

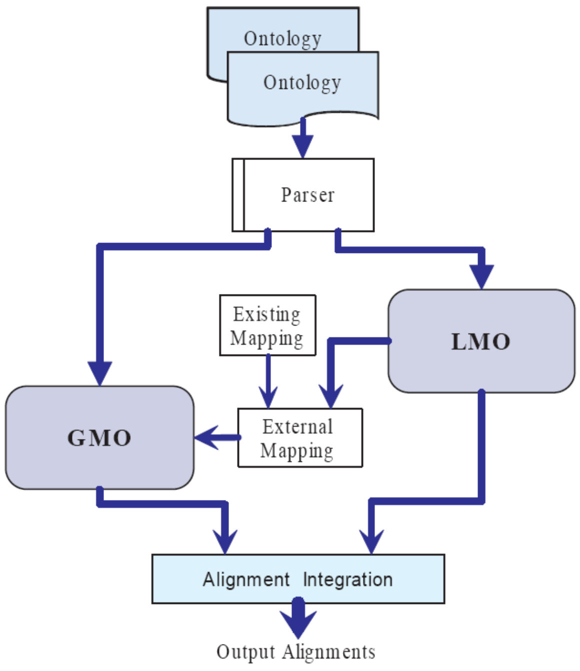


Figure 2.9: The FALCON-AO architecture.

Linguistic Matching for Ontologies

As is known, linguistic matching plays an important role in matching process. Generally, linguistic similarity between two entities relies on their names, labels, comments and some other descriptions. The LMO combines two different approaches to gain linguistic similarities: one is based on lexical comparison; the other is based on statistic analysis.

In lexical comparison, we calculate the edit distance between names of two entities and use the following function to capture the string similarity (denoted by SS):

$$SS = 1/e^{\left(\frac{ed}{s_1.len + s_2.len - ed}\right)}$$

Where ed denotes the edit distance between s_1 and s_2; $s_1.len$ and $s_2.len$ denote the length of the input strings s_1 and s_2, respectively. In statistic analysis, we use the algorithm of [76] based on the Vector Space Model (VSM) in our implementation. Given a collection of documents, we denote N the number of unique terms in the collection. In VSM, we represent each document as a vector in an N-dimensional space. The components of the vector are the term weights assigned to that document by the term weighting function for each of the N unique terms. Clearly, most of these are going to be 0, since only a few of the N terms actually appear in any given document. In our scenario, we construct a virtual document for each of the ontology entities (classes, properties and instances). The virtual document of an entity consists of "bag of terms" extracted from the entity's names, labels and comments as well as the ones from all neighbors of this entity. Once the virtual document is constructed it is possible to define a similarity between documents, and thus between ontology elements, using the cosine distance by taking the vectors' *dot* product.

$$DS = N \cdot N^t$$

The two methods described above will both take effect in ontology matching. In our implementation, we combine them together, and use the following equation to calculate the final linguistic similarity

$$LinguisticSimilarity = w_{DS} DS + w_{SS} SS$$

Where w_{DS} and w_{SS} are weights associated to respectively the statistical distance (or document similarity) and the string similarity. In [48] the author assign to w_{DS} the value 0.8 and the value 0.2 to w_{SS}.

Graph Matching for Ontologies

Another important component in Falcon-AO is GMO, which is based on a graph matching approach for ontologies. It uses directed bipartite graphs to represent ontologies and measures the structural similarity between graphs by a new measurement. Details of the approach are described in [44] The main idea of GMO is as follows. Similarity of two entities from two ontologies comes from the accumulation of similarities of involved statements (triples) taking the two entities as the same role (subject, predicate, object) in the triples, while the similarity of two statements comes from the accumulation of similarities of involved entities of the same role in the two statements being compared.

Usually, GMO takes a set of matched entity pairs, which are typically found previously by other approaches, as external mapping input in the matching process, and outputs additional matched entity pairs by comparing the structural similarity. Our previous experiments showed that GMO were irreplaceable when there was little gain from lexical comparison. In addition, GMO can be integrated with other matchers. While using GMO approach to align ontologies, there should be another component to evaluate reliability of alignments generated by GMO.

Linguistic vs. Structural Comparability

Given two ontologies to be aligned, GMO always tries to find all the possible matched entity pairs. However, how to evaluate the reliability of these matched entity pairs is still a problem. As mentioned above, another component is needed to select more reliable matched entity pairs by using other information. In Falcon-AO, we use a simple approach to observe the reliability of matched entity pairs output by GMO, and select more reliable matched entity pairs to the users. The approach is based on the measure of linguistic comparability (*LC*) and structural comparability (*SC*) of two ontologies to be aligned. Given two ontologies O_1, O_2 to be aligned, the linguistic comparability (*LC*) of O_1 and O_2 is defined as follows:

$$LC = \frac{M}{\sqrt{N_{O_1} \cdot N_{O_2}}}$$

Where M denotes the number of entity pairs with similarity larger than c and c is an experience value; N_{O_1} and N_{O_2} represent the number of named entities in O_1 and O_2, respectively.

System	Terminological	Structural	Extensional
Anchor-PROMPT	√	√	√
COMA	√	√	
Cupid	√	√	
GLUE			√
QOM	√	√	√
Falcon-AO	√	√	√

Table 2.3: Classification of reviewed approaches

We use VSM method to observe the structural comparability. The vectors V_1, V_2 represent the frequency of built-in properties used in O_1 and O_2 and the element v_{ij} denotes the number of occurrence of built-in property p_j in O_i. The structural comparability of O_1 and O_2 is the cosine similarity of V_1 and V_2:

$$SC = \frac{V_1 \cdot V_2}{|V_1| \cdot |V_2|}$$

Part I

Related Work and Preliminary Definitions

Chapter 3

Related Work

The techniques proposed in this thesis are closely related to work in the area of access control policy analysis and policy integration.

3.1. Policy Analysis

Approaches to policy analysis can be broadly classified into two categories: (i) those that deal with verification of properties of a single policy and (ii) those that deal with policy similarity analysis that may involve verification of different relationships such as equivalence, refinement, redundancy etc among two or more policies.

3.1.1 Single Policy Analysis

Most approaches for single policy property analysis are based on model checking techniques [3], [39] and [93]. Ahmed et al. [3] propose a methodology for analyzing four different policy properties in the context of role-based CSCW (Computer Supported Cooperative Work) systems; this methodology uses finite-state based model checking. Since they do not present any experimental results, it is not clear if their state exploration approach can scale well to policies with a very large set of attributes and conditions. Guelev et al. propose a formal language for expressing access-control policies and queries [39]. Their subsequent work [93] proposes a model-checking algorithm which can be used to evaluate access control policies written in their proposed formal language. The evaluation includes not only assessing whether the policies give legitimate users enough permissions to perform their tasks, but also checking whether the policies prevent intruders from achieving some malicious goals. However, the tool can only check policies of reasonable size.

3.1.2 Policy Similarity Analysis

Existing approaches to the policy similarity analysis are mostly based on graph, model checking or SAT-solver techniques [35], [2], [7], [53], [58] and [69]. Koch et al. [53] use graph transformations to represent policy change and integration, which may be used to detect differences among policies. Such an approach supports an intuitive visual representation which can be very useful in the design of a customized access control policy. However, it can only be used as a specification method but not as an execution method. Backes et al. [7] propose an algorithm for checking refinement of enterprise privacy policies. However, their algorithm only identifies which rule in one policy needs to be compared with the rules in the other policy. They do not provide an approach to the evaluation of condition functions. A more practical approach is by Fisler et al. [35], who have developed a software tool known as Margrave for analyzing role-based access-control policies written in XACML. Margrave represents policies using the Multi-Terminal Binary Decision Diagram (MTBDD), which can explicitly represent all variable assignments that satisfy a Boolean expression and hence provides a good representation for the relationships among policies. Policy property verification is then formulated as a query on the corresponding MTBDD structures. For processing a similarity query involving two policies, the approach proposed by Fisler et al. is based on combining the MTBDDs of the policies into a CMTBDD (change-analysis MTBDD) which explicitly represents the various requests that lead to different decisions in the two policies. The MTBDD structure has been credited with helping model checking scale to realistic systems in hardware verification. The major shortcoming of Margrave is that it can only handle simple conditions, like string equality matching. A direct consequence of such limitation is an explosion of the MTBDD size when conditions on different data domains (e.g. inequality functions) have to be represented. For example, to represent the condition time is between 8am to 10pm, the MTBDD tool needs to enumerate all possible values between 8am to 10pm(e.g., time-is-8:00am, time-is-8:01am, time-is-8:02am, ...).

Other relevant approaches are the ones based on SAT-solver techniques. Most such approaches [58], [69] however only handle policy conflict detection. A recent approach by Agrawal et al. [2] investigates interactions among policies and proposes a ratification tool by which a new policy is checked before being added to a set of policies. In [65], McDaniel et al. carry out a theoretical study on automated reconciliation of multiple policies and then prove that this is an NP-complete problem. In [56], Kolovski et al. formalize XACML policies by using description logics and then employ logic-based analysis tools for policy analysis. These SAT-solver based approaches formulate policy analysis as a Boolean satisfiability problem on Boolean expressions representing the policies. Such approaches can handle various types of Boolean expressions, including equality functions, inequality functions, linear functions and their combinations. By construction, the SAT algorithms look for one variable assignment that satisfies the given Boolean expression, although they may be extended to find all satisfying variable assignments. For each round of analysis or query, SAT algorithms need to evaluate the corresponding Boolean expression from scratch. They cannot reuse previous results and are not able to present an integrated view of relationships among policies. Most recently, Mazzoleni et al. [64] have investigated the policy similarity problem as part of their methodology for policy integration. However, their method for computing policy similarity is limited to identifying policies referring the same attribute. Unlike aforementioned works that focus on a special case or a certain type of policy analysis, our approach aims at providing an environment in which a variety of analysis can be carried out. In particular, our environment is able not only to handle conventional policy property verification and policy comparison, but also to support queries on common portions and different portions of multiple policies. Unlike all approaches to policy similarity analysis which require extensive comparison between policies, our proposed similarity measure is a lightweight approach which aims at reducing the searching space, that is, at reducing the number of policies that need to be fully examined. From the view of an entire policy analysis system, our policy similarity measure can be seen as a tool which can act as a filter phase, before more expensive analysis tools are applied. For completeness it is also important to mention that the problem of similarity for documents has been investigated in the information retrieval area. Techniques are thus available for computing similarity among two documents (e.g. [29], [40] and [67]). However, these cannot be directly applied because of the special structures and properties of the XACML policies.

3.2. Semantic Web-Based languages

Recently there has been a great amount of attention to how Semantic Web technologies can be used in policy systems. In particular, there have been a number of proposals that show how to ground or express policies in a Semantic Web framework [91], [50], [49] and [90]. Rei [50] is a policy specification language based on a combination of OWL-Lite, logic-like variables and rules. It allows users to develop declarative policies over domain specific ontologies in RDF and OWL. Rei allows policies to be specified as constraints over allowable and obligated actions on resources in the environment. A distinguishing feature of Rei is that it includes specifications for speech acts for remote policy management and policy analysis specifications like what-if analysis and use-case management. The successor of Rei is Rein [49], which is a policy framework grounded in semantic web technologies that allows for different policy languages and supports heterogeneous policy systems. Rein provides an ontology for describing policy domains in a decentralized manner and provides a reasoning engine built on top of CWM, an N3 rules reasoner. Using Rein and CWM, the authors showed how it is possible to develop domain and policy language specific security systems. Rein has been successfully used as a policy management system in the Policy AwareWeb project [91], which in turn provides an architecture for scalable, discretionary, rule-based access control in open and distributed environments. PeerTrust [37] deals with discretionary access control on the web using semantic web technologies. It provides a mechanism for gaining access to secure information/services on the web by using semantic annotations, policies and automated trust negotiation. In PeerTrust, trust is established incrementally through an iterative process which involves gradually disclosing credentials and requests for credentials. PeerTrusts policy language for expressing access control policies is based on definite Horn clauses. A distinguishing feature of PeerTrust is that it expects both parties to exchange credentials in order to trust each other and assumes that policies are private, which is appropriate for critical resources such as military applications and e-commerce sites. Finally, KaOS Policy and Domain Services [90] use ontology concepts encoded in OWL to build policies. These policies constrain allowable actions performed by actors which may be clients or agents. The KAoS Policy Service distinguishes between authorizations and obligations. The applicability of the policy is defined by a class of situations which definition can contain components specifying required history, state and currently undertaken action.

3.3. Access Control and Ontologies

The most relevant work has focused on strategies for the mapping between RBAC and OWL [34, 52], on the integration between XACML and OWL [24]. Finin et al. [34] have introduced R*OWL*BAC, a representation of RBAC in OWL. They propose two different approaches: mapping roles to classes, and mapping roles to values. In the first case roles are represented as class of users, and the role dominance relation is then mapped to the subsumption relation in OWL. SoD constraints are then mapped onto to class disjointness constraints in OWL. We remark that this solution has several drawbacks and cannot be applied in the general case. Consider two classes D_i and D_j such that $D_i \subseteq D_j$. Now suppose that there is a SoD constraint between D_i and D_j. If SoD constraints are mapped onto class disjointness constraints, we have that $D_i^{\mathcal{I}} \cap D_j^{\mathcal{I}} = \emptyset$. This results in an inconsistency in the ontology since we have at the same time that D_i is included in D_j and that D_i and D_j are no common values. The second solution is to map classes onto individuals and to bind users to classes through the property `role`. This solution is similar to the one adopted in our work but with a substantial difference: in R*OWL*BAC constraints are modeled through specialized properties e.g. `dsod` and `ssod`, binding roles together. This solution is more compact than the one we have adopted in our paper; however a standard DL-reasoner is not able to detect constraint violations and rules must be added to the ontology, thus degrading performance. Knetchel et al. [52] have proposed an approach that exploits OWL for reasoning about RBAC authorizations. The model can support both roles and class hierarchies; however it does not take into consideration SoD constraints. The approach that is closer to our XACML+OWL is the one proposed by Damiani et al. [24]. Their approach directly embeds RDF statements into XACML. However, their approach does not provide any support for OWL reasoning. Crampton [22] has proposed an extended XACML profile for RBAC exploiting obligations and blacklists to support DSoD constraints. This approach is similar to XACML+OWL in the use of obligations; however since in our model we exploit ontologies, we are able to provide more expressive functionalities. Other approaches that have different purposes but are related to the technologies adopted in XACML+OWL are by Kagal et al. [49] and by Kolovski et al. [56]. Rein [49], a general framework based on semantic web technologies, is able to support general purpose policy systems, and for this reason it is well suited for solving mismatches among different policy languages. Kolovski et al. [56] have proposed a DL-based analysis tool for XACML policies. In their approach they propose a mapping between XACML and Description Logics along with reasoning techniques for verifying properties of XACML policies.

Chapter 4

Policy Analysis: Preliminaries

In this chapter we analyze the issues arising from the design and development of analysis services in a multi-domain environment. Usually, policy analysis tools offer a number of different services that are executed on policy sets that belong to the same domain. This assumption makes easier the definition of such services but represents a strong limitation in the applicability of such tools. Therefore, with the increasing popularity of distributed systems and of collaborative applications, there is the need to compare policies from multiple domains. In order to have a better understanding of the problem we will give a formal definition of the main issues and the preliminary notions and terminology that will be used throughout the following chapters.

4.1. Heterogeneity in Policy Analysis

In the analysis of a policy, several information need to be taken into account. Examples of such information are the names of the subjects that are defined into the rules, the resources that may be accessed and the actions that may be performed by the subjects over the resources.

We refer to such data as the $\mathcal{V}ocabulary$ of a policy. For example, the vocabulary of the policy depicted in Figure 4.1, is the set of terms:

$\mathcal{V}ocabulary(P_1) = \{\texttt{PhDStudent}, \texttt{FullProfessor}, \texttt{TechnicalPaper}, \texttt{Read}\}$

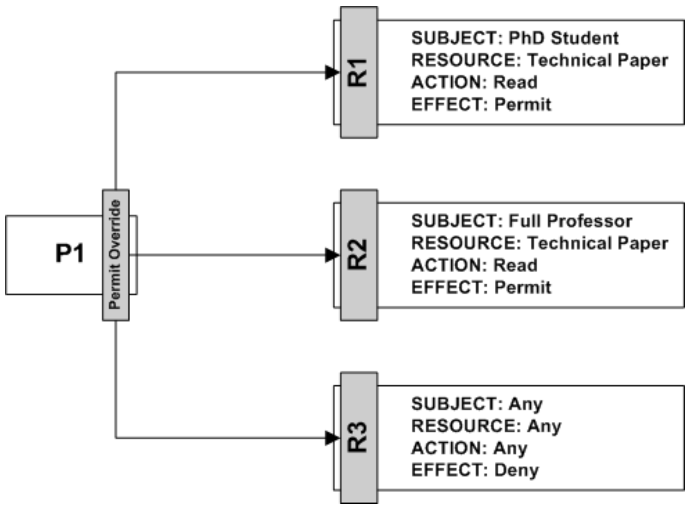


Figure 4.1: An example policy for an university policy-based access control model.

The terms `Any`, `Permit` and `Deny` are keywords defined in XACML and for this reason are not considered in the vocabulary of the policy.

Usually, the existent analysis services are defined considering just the vocabulary on the policies involved in the analysis process. The general assumption is that if the policies adopts different terms within a rule, then those rules concerns different concepts.

This assumption is generally true when the policy analysis tools are used in a centralized scenario. In such a case, we say that the policy vocabularies are *coherent* with respect to each other and no further improvements to the analysis tool are necessary. Obviously, this is not the general case. When analyzing policies belonging to heterogeneous owners, it is crucial to make distinctions between such inconsistencies that from now on are referred to as *naming heterogeneity*. We classify the conflicts that can occur in naming heterogeneity into syntactic and terminological variations:

- *Syntactic variations* arise because of the use of different character combinations to denote the same term. An example is the use of *TechnicalReport* and

Tech_Rep to denote a *technical report*.

- *Terminological variations* refer to the use of different terms to denote the same concept. An example of terminological variation is the use of the synonyms *Report* and *Study* to refer a a written document describing some research results.

Usually, the two categories of variations described above can be handled using traditional Natural Language Processing (NLP) techniques such as look up tables and external linguistic resources. Specifically, look up tables enumerate the possible ways in which the same term can be written by using different character combinations and, thus, can be exploited when dealing with syntactic variations. Instead, to detect terminological variations, dictionaries or thesaurus such as WordNet [31] can be exploited. Such kind of external resources are very useful in this case since they allow to retrieve all the synonyms of a given term. Appendix A reports the details of an approach for the application on ontology-based technologies in Digital Identity Management. Moreover, some detailed results on naming heterogeneity issues have been reported in [10][1].

4.1.1 From Vocabularies to Domains

So far, we have introduced the simple notion of policy vocabulary and how heterogeneities issues arising in this scenario can be solved using standard NLP techniques. However, the solution we have introduced is far to be considered exhaustive. It is very difficult, maybe impossible, to effectively determine all the possible ways a concept can be represented in different systems. For this reason, we need to take into account some additional information and thus to exploit more powerful techniques.

A possibility is to consider not only the representation of a concept but also the relations the concept has with other entities. For example, `PhDStudent` and

[1]In the paper an approach is proposed to solve naming heterogeneity in a Digital Identity Management scenario

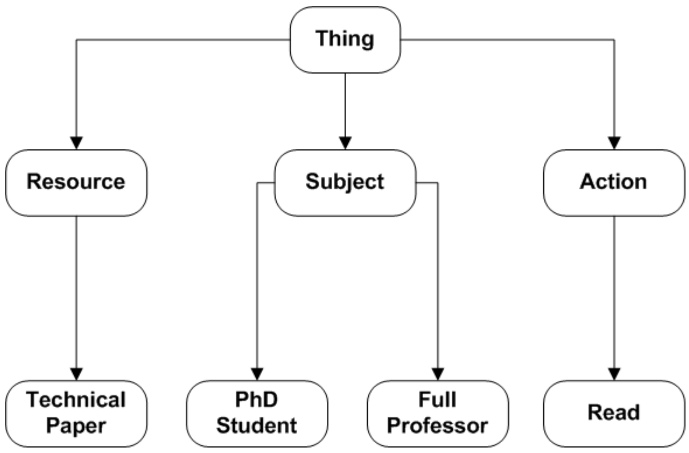


Figure 4.2: The domain of the policy P_1 depicted in Figure 4.1.

`FullProfessor` (see Figure 4.1) are not completely decoupled concepts: they have the common feature of belonging to the more general class of `Faculty` entities. Moreover, assuming the perspective of a security administrator, we may also note that a `FullProfessor` is expected to have a larger set of permissions than a `PhDStudent`. The latter consideration can be mapped to the role subsumption relation in a typical Role Based Access Control (RBAC) scenario [33], [79]. All this information represents important knowledge that must be taken into account in the analysis of heterogeneous policies. We define the Domain of a policy the set of the terms in the policy vocabulary enriched with the relations defined over such terms. The domain of policy P_1 shown in 4.1 is shown in Figure 4.2: the domain is represented as a hierarchy in which each node represent one of the policy concept whereas each arrows is the standard subsumption relation.

Considering the knowledge modeled within a domain gives the possibility of exploiting powerful techniques but also it raises new interesting issues. Specifically, when we have two concepts modeled in two different domains we may have additional conflicts that we refer to as *semantic variations*. Semantic variations are related to the use of two different concepts in different semantic schemas to denote the same term.

Usually, the techniques that take advantage of domain analysis for solving complex issues such as heterogeneity and interoperability are referred to as semantic techniques. Such approaches have been applied in several application domain such as multimedia [46], [18] and [87]; databases [86] and [11]; geographic information systems [36]; connectivity [88] and networks [92], [20]; Web services [15], [63] and [32]; e-learning [73], [81] and [19]; grid [78], [38] and [83]; bio-informatics [16]; recommender systems [68], [17] and [94].

4.1.2 The Formalization of a Policy Domain

The formalization of a domain is essentially a problem of knowledge representation. Due to the issues introduced by the Semantic Web view [9], the adoption of ontologies as a formal knowledge representation system has rapidly grown in popularity. The notion of ontology has been defined in [13] as *the formal specification of a shared conceptualization*. There are several languages for the creation of ontologies such as DAML+OIL [43], RDF Schema [14] and the Ontology Web Language (OWL) [25] which is the W3C Recommendation formalism for the development of ontologies over the Web.

In this thesis we use OWL as the language for formalizing a policy domain. This choice is due to several reasons. On one hand OWL is a XML-based language with a well defined semantics grounded on Description Logics [6]. On the other hand OWL is a widely adopted formalism supported by a number of different reasoning tools. Moreover, several techniques have been developed with the purpose of finding relationships between OWL ontologies modeled on different but related domains. Especially this last feature, known with the name of *ontology matching*, is particularly interesting in our scenario since it allows us to find similarities between concepts belonging to different semantic schemas. In the remaining part of this section we introduce the definition of the entities of our model related to ontologies and ontology matching. The definitions have been defined considering the ones introduced in [61], [29], [80] and [6].

First of all we define the notion of ontology. Recalling the background section about ontologies and ontology mapping, we have that an ontology contains two different parts: the **Terminological Box** (*TBox*) that is the structure of the ontology, and its **Assertional Box** (*ABox*) that is the instances of the entities (classes and relations) defined in the TBox. In Definition 4.1 we define an ontology TBox, the ABox is defined in 4.2 whereas the ontology is defined in 4.3

Concerning the TBox we have defined the relations between concepts in two different sets: $\mathcal{R}$ and $\mathcal{P}$, respectively the set of relations and properties. This distinction intercepts the difference between object properties and datatype properties in OWL.

Definition 4.1 (Terminological Box) *The extensional part of an ontology, or* TBOX, *is the tuple:*

$$O^T := \langle \mathcal{C}, \leqslant_C, \mathcal{DT}, \mathcal{R}, \leqslant_R, \sigma_R, \mathcal{P}, \leqslant_P, \sigma_P \rangle$$

where $\mathcal{C}$ is the set of concepts and $\leqslant_C$ is the hierarchy defined over $\mathcal{C}$. $\mathcal{DT}$ is the set of datatypes. $\mathcal{R}$ is a set of relations, $\leqslant_R$ is the hierarchy defined over $\mathcal{R}$ and $\sigma_R : \mathcal{R} \to \mathcal{C} \times \mathcal{C}$ is the signature of $\mathcal{R}$. Finally, $\mathcal{P}$ is a set of properties, $\leqslant_P$ is the hierarchy defined over $\mathcal{P}$ and $\sigma_P : \mathcal{P} \to \mathcal{C} \times \mathcal{DT}$ is the signatures of $\mathcal{P}$.

An ABox is defined over entities belonging to a TBox.

Definition 4.2 (Assertional Box) *The intensional part of an ontology, or* ABOX, *is the tuple:*

$$O^T_A := \langle \mathcal{C}^\mathcal{T}, \mathcal{DT}^\mathcal{T}, \mathcal{R}^\mathcal{T}, \mathcal{P}^\mathcal{T}, I, V, \iota_{\mathcal{C}^\mathcal{T}}, \iota_{\mathcal{DT}^\mathcal{T}}, \iota_{\mathcal{R}^\mathcal{T}}, \iota_{\mathcal{P}^\mathcal{T}} \rangle$$

where $\mathcal{C}^\mathcal{T}$, $\mathcal{DT}^\mathcal{T}$, $\mathcal{R}^\mathcal{T}$ and $\mathcal{P}^\mathcal{T}$ are respectively the set of concepts, datatypes, relations and properties defined in O^T. I and V are respectively the set of individuals and of values. The sets $\iota_{\mathcal{C}^\mathcal{T}} : \mathcal{C}^\mathcal{T} \to 2^I$, $\iota_{\mathcal{DT}^\mathcal{T}} : \mathcal{DT}^\mathcal{T} \to 2^V$, $\iota_{\mathcal{R}^\mathcal{T}} : \mathcal{R}^\mathcal{T} \to 2^{I \times I}$ and $\iota_{\mathcal{P}^\mathcal{T}} : \mathcal{P}^\mathcal{T} \to 2^{I \times V}$ are the instantiations of $\mathcal{C}^\mathcal{T}$, $\mathcal{DT}^\mathcal{T}$, $\mathcal{R}^\mathcal{T}$ and $\mathcal{P}^\mathcal{T}$ respectively.

Finally, the ontology is defined as a pair of a Tbox and an ABox defined over it.

Definition 4.3 (Ontology) *An ontology is a tuple:*

$$O := \langle O^T, O^T_A \rangle$$

where O^T *is a terminological box and* O^T_A *is the assertional box defined over* O^T*.*

Moreover, we define the set of the *entities* belonging to an ontology O_i as the set:

$$E(O_i) := \mathcal{C} \cup \mathcal{R} \cup \mathcal{P} \cup I$$

In Definition 4.4 we introduce the notion of a mapping element following the approach proposed in [80]:

Definition 4.4 (Mapping Element) *Given two ontologies* O_i *and* O_j*, a mapping element is a tuple:*

$$\langle e_{O_i}, e_{O_j}, s \rangle$$

where e_{O_i} *and* e_{O_j} *are entities of ontologies* O_i *and* O_j *respectively and* s *is a confidence measure in some mathematical structure (typically in the range [0, 1]);*

An ontology matching is then defined as a set of mapping elements.

Definition 4.5 (Ontology Matching) *Given two ontologies* O_i *and* O_j*, an ontology matching is a set* π_{O_i,O_j} *of mapping elements such that: if* $\langle e_{O_i}, e_{O_j}, s \rangle \in \pi_{O_i,O_j} \wedge \langle e_{O_i}, e'_{O_j}, s' \rangle \in \pi_{O_i,O_j}$ *then* $e_{O_j} = e'_{O_j}$ *and* $s = s'$*.*

Definition 4.5 is very general and does not specify a unique procedure for obtaining a mapping between two ontologies. As we have introduced in the background section, several matching techniques have been proposed. Each such approach has some advantages and disadvantages, most of which related to the trade off between performance and mapping accuracy. However, most approaches provide mappings that reduce to the general form introduced in Definition 4.5. We discuss the specific ontology matching algorithm used in EXAM-S in Chapter 6. In the rest of this section we use the symbol π_{O_i,O_j} as one of the possible ontology matching computed between ontologies O_i and O_j regardless to the specific implementation.

4.2. Dealing with Heterogeneous and Partial Knowledge

The computation of a policy vocabulary is a simple task since can be reduced to the enumeration of all attribute names and values appearing in the policy. Instead, when working with the domain of a policy, we have to take into consideration also the relations between those terms. If policy attributes are directly mapped to ontology concepts then the process of obtaining the Domain of a policy is trivial since we just need to access the associated ontology.

4.2.1 Ontology Merging

One of the key feature in the development of modern ontologies is "*reusability*". The reason is that usually, the development of an ontology, is a complex, time-consuming and error-prone task. Moreover, is quite common to have different ontologies that model a partially overlapping domain. Hence, is usually a good practice (when possible) to reuse already defined ontologies instead of creating new ones[2].

For this reason, in considering ontology-based policy, more complicated scenarios may arise. For example we may have heterogeneous knowledge, that is when the same policy use concepts belonging to different ontologies. This may represent an additional problem since it is important to verify the consistency of those concepts and thus to align the group of ontologies exploited verifying additional properties. We define the different ontologies that are exploited in the policy P_i as the set $Ontologies(P_i) := \{O_j \mid \exists\, e_{O_j} \in Vocabulary(P_i)\}$

This problem is usually referred to as *Ontology Merging* [51] and can be considered a general case of the ontology matching problem. Ontology merging is the process of generating a single ontology from two or more existing ontologies. A merged single ontology includes information from all source ontologies but is

[2]This feature is supported in OWL by the command `owl:imports`

more or less unchanged. The original ontologies have similar or overlapping domains but they are unique and not revisions of the same ontology. The intuitive idea is that, given ontologies O_i and O_j we aim to construct the union of entities $e_i \in E(O_i)$ and $e_j \in O_j$ such that if e_i and e_j can be considered as the same element then just one of them is added to the result ontology. For example, we may consider as equivalent the pair of entities e_{O_i}, e_{O_j} belonging to the mapping element $\langle e_{O_i}, e_{O_j}, s\rangle$ such that s is greater than a certain threshold τ. Thus, in building up the merged ontologies we can consider just one of the two entities.

Definition 4.6 (Policy Merged Ontology) *Given a policy P_i its merged ontology $\tilde{O}_{P_i}$ is the ontology recursively defined as:*

$$\tilde{O}_{P_i} := \mathsf{MERGE}_{j=1}^{|\mathsf{Ontologies}(P_i)|}(\tilde{O}, O_j)$$

Where $O_j \in \mathsf{Ontologies}P_i$, $\tilde{O}$ is the initial empty ontology and $\mathsf{MERGE}(O_i, O_j)$ is a function that takes in input to ontologies and add the entities in $e_{O_j} \in O_j$ to O_i such that $\nexists \langle e_{O_i}, e_{O_j}, s\rangle \in \pi_{e_{O_i}, e_{O_j}}$ with s greater than an acceptance threshold τ.

It is worth noting that, according to Definition 4.6, if $\mathsf{Ontologies}(P_i) = \{O_j\}$, that is, the concepts in $\mathsf{Vocabulary}(P_i)$ belong all to the same ontology, then $\tilde{O}_{P_i} = O_j$.

The $\tilde{O}_{P_i}$ represents the result of merging recursively all the ontologies exploited by the policy P_i. Once the merge ontology has been created than it is possible to execute policy analysis services considering just such common knowledge base.

4.2.2 Ontology Extraction

When a policy does not use semantic data, it is necessary to create a new ontology extracting semantic knowledge by the information that can be deduced from the policy itself. The problem of extract meaningful knowledge by unstructured data is usually referred to as *Ontology Extraction* or *Ontology Learning* and has been deeply investigated, especially after the introduction of the Semantic Web paradigm. Interesting approaches have been proposed in [60], [85], [4], [82] and [84]. In [60] the authors proposes a complex machine learning approach for the semi-automatic creation of an ontology from generic unstructured data. In [85], [4] and more recently in [82], the proposed approach is to extract information using the data provided in web pages. In [85] the ontology is created extracting data from general purpose tables and finally in [84] the authors propose an interesting approach based on Fuzzy Formal Concept Analysis for automatic generation of ontologies on uncertainty information.

The automatic extraction of meaningful knowledge by unstructured and uncertain data is a complex task [60]. So, most of the approaches propose some alternative solutions. Usually, if no assumption can be made on the involved data then the extraction process is semi-automatic. On the contrary, whenever the data is organized in a structured fashion it is always possible to obtain more semantic informations and, thus, create a completely automatic process for the extraction of the knowledge base.

In our scenario, we consider data defined in XACML policies; this means that we can exploit the explicit knowledge provided by the policy language to obtain a first classification of terms. In doing so, we adopt the mapping XACML to Description Logics proposed in [56]. The details of the mapping are reported in Section 4.3.3, the key idea is that each *attribute-value* pair in a XACML policy can be translated adding two entities to the extracted knowledge base: given the pair $\langle \mathtt{attribute}, \mathtt{value} \rangle$, the relation `attribute` and the concept `value` are added to the ontology. In [56] the authors propose a translation between XACML and Description Logics, in our case we work with OWL and for this reason we need to check also the data type of the value before translating the attribute in the correct OWL properties. In our approach we deal with Strings and all the data types related to numbers, in future work we plan to extend this mapping for managing more data types. In our mapping, if the $\mathtt{value}$ is a String then it is translated into a new concept and the $\mathtt{attribute}$ become an object property. If the $\mathtt{value}$

is a XML Schema number data type, no concepts are added and the `attribute` become a data type property. Given a policy P_i we refer to its extracted ontology using the symbol $\ddot{O}_{P_i}$.

4.2.3 Hybrid Scenarios

The more complicated scenario is when we have together both heterogeneous domains and partial knowledge. However, we can easily manage such scenario by combining together the approaches defined in Definition 4.6 and in Section 4.2.2. We define the Policy Reference Ontology as follows:

Definition 4.7 (Policy Reference Ontology) *Given a policy* P_i *its reference ontology* $\dot{O}_{P_i}$ *is the ontology defined as:*

$$\dot{O}_{P_i} := \mathsf{MERGE}(\tilde{O}_{P_i}, \ddot{O}_{P_i})$$

It important to underline that $\dot{O}_{P_i}$ is a general case of both $\tilde{O}_{P_i}$ and $\ddot{O}_{P_i}$. And this more general definition can be applied in all of the three cases regardless the specific category[3] of the policy.

where $\square \in \{=, \neq, <, \leqslant, >, \geqslant\}$

4.3. Trade-off between Expressivity and Complexity

In order to define our analysis services, we need to translate an XACML policy to a more convenient formalism. The choice of such formalism is not trivial since it influences the efficiency and the expressivity of the services we define. In this section we discuss in the detail the approach adopted over different formalisms. Specifically, we analyze the translations of XACML into both Propositional Logic

[3]Single ontology, multiple ontology, no ontology and partial knowledge

($\mathcal{PL}$) [35], [93] and Description Logic ($\mathcal{DL}$) [56]. Mappings with First Order Logic ($\mathcal{FOL}$) have been proposed as well, for example using systems like Alloy [47], however due to the complexity of some real case policies[4] $\mathcal{FOL}$-based systems are very inefficient, despite the advantage provided by powerful expressivity. For this reason, in order to find the best trade off between expressivity and complexity, we take into considerations just the mappings with Propositional and Description Logic, in the remaining part of this section we give details about such interpretations providing both weak and strong features.

4.3.1 XACML and Propositional Logic

One of the first attempt in providing efficient reasoning procedure over subsets of the XACML language was made employing Propositional Logic. With $\mathcal{PL}$ it is not possible to express all the features defined in the XACML, however it allow efficient implementation of analysis services using structures such as *binary decision diagrams* (BDD). The execution of services using BDD results to be very useful when dealing with large policy sets. In literature there are essentially two approaches based on BDD and, thus, to Propositional Logic: Margrave [35] and the work proposed by Zhang *et al.* [93]. However, the model proposed by Zhang *et al.* is defined over the language *RW*[5] that is slightly different from XACML. For example in their framework they can obtain two terminal decisions: *Permit* and *Deny*, whereas in XACML it is possible to have also a *NotApplicabile* terminal. This is the reason why they can use BDD conversely to Margrave in which a more powerful structure is employed. For this motivations, in the remaining part of this section we focus on the XACML-PL mapping proposed by Fisler *et al.*.

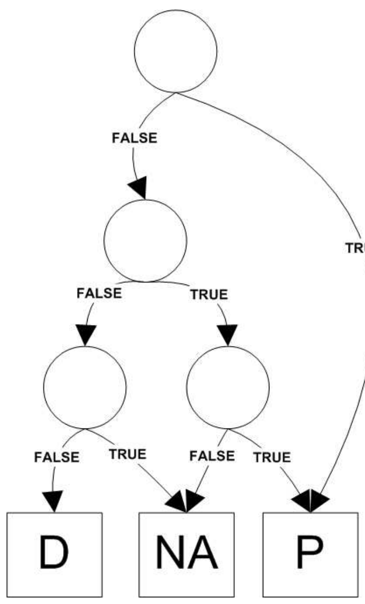


Figure 4.3: An example MTBDD.

A closer look to Margrave

Margrave is a tool for the analysis of XACML policies. The key idea in Margrave is to transform a XACML policy into a boolean function that associates decisions to the incoming requests. Fisler *et al.* represent such boolean functions using Multi Terminal Binary Decision Diagrams (MTBDD). In the following we detail how a boolean function can be represented with an MTBDDs.

Let f be a boolean function such that $f : \mathbb{B}^n \rightarrow \mathbb{B}$ where $\mathbb{B} = \{\texttt{True}, \texttt{False}\}$. The function f can be represented as a Binary Decision Diagram (BDD). Each BDD is essentially a DAG with only one *source* vertex and two *sink* vertexes representing the values `True` and `False`. Each internal node i represents the evaluation of the ith element of the input boolean tuple. If the internal node i is `True` then the remaining $n - i$ elements have to be evaluated according to the subtree rooted in

[4]In [35] Margrave is evaluated against policies with a cardinality of 50 attribute-value pairs. Moreover, the author say that in real case scenarios each of the typical policies has 5 - 20 pairs with an upper bound of about 100 pairs.

[5]Proposed by the same authors.

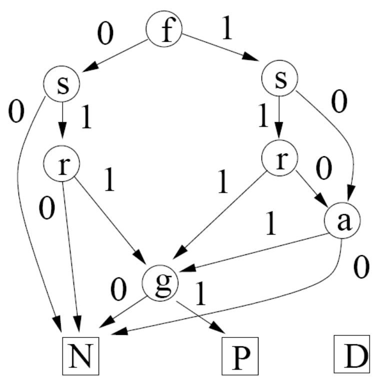


Figure 4.4: An example MTBDD for a simple policy.

the right child element of the ith node. Otherwise, the left branch is considered.

Conversely, if the function f assume the form $f : \mathbb{B}^n \rightarrow \mathcal{P}$, where $\mathbb{B} = \{\texttt{True}, \texttt{False}\}$ and $\mathcal{P}$ is a final set of terminals, then a BDD is no more sufficient. We remark that this is the situation of the XACML policies since they have a number n of input values that have to be evaluated and three terminals Permit (P), Deny (D) and Not Applicable (NA). This kind of functions can be efficiently represented by MTBDDs that are nothing but BDDs in which the cardinality of the set of terminals can be higher than two. An example of MTBDD is shown in Figure 4.3.

Given a policy, Margrave constructs a MTBDD that represents that policy. Each MTBDD is then queried for verifying the properties that the associated policy should satisfy. Figure 4.4 shows an example of an MTBDD representing a simple security policy in which faculty can assign grades and students can receive grades (**f**aculty, **s**tudent, **r**eceive, **a**ssign and **g**rades).

4.3.2 Policy Analysis Services in $\mathcal{PL}$

Analysis services that are usually defined over $\mathcal{PL}$ are property verification queries. A property verification is a procedure that aims to verify if the policies under investigation satisfy a certain property. In this case property verification can be easily implemented updating the MTBDD representing the policies according to the feature to verify. We take ad example the *restriction to decision* property, that is the set of requests that yield the specified decision. To implement restriction to decision, Margrave replaces the terminal for the given decision in the policy MTBDD with logical true and all other terminals with logical false.

Moreover, Fisler et al., define change impact analysis on MTBDDs. Change impact analysis is the process whereby considering together different policies it is possible to check the changes into them effects. This service is very useful when we would like to analyze different version of the same policies and, specifically, the impact of a policy update.

4.3.3 XACML and Description Logic

In [56] Kolovski et al., define a framework for policy analysis providing a mapping with $\mathcal{DL}$. This solution fits in between $\mathcal{FOL}$ and $\mathcal{PL}$ approaches. On one hand it provides more expressiveness than $\mathcal{PL}$ on the other one, being $\mathcal{DL}$ a decidable fragment of $\mathcal{FOL}$, the analysis services come with a lower computation complexity than the ones defined over $\mathcal{FOL}$.

The basic unit in XACML that yields an access decision is a Rule. To capture the behavior of XACML correctly, we need to formalize the prerequisite of the Rule (which is the Target element), and its head (the access decision). We also need to capture how the access decision is propagated upwards toward the root PolicySet - for this, the rule and policy combining algorithms have to be taken into account. While the Target element of Rules and Requests can be mapped to a DL concept expression (we discuss this in more detail below), the interaction of the access decision of various policy elements is difficult, if not impossible, to do using only description logics. This is because of the semantics of the combining algorithms which requires us to use a formalism that supports preferences. To capture the behavior of the XACML combining algorithms, we use Defeasible description logics, which is a formalism that allows for expressing defeasible rules on top of description logics. Only a limited fragment of DDL is needed to formalize the combining algorithm.

Mapping rules and requests

A XACML Rule is translated to a rule in $\mathcal{R}$. The Target element is translated to a $\mathcal{DL}$ concept expression C and becomes the antecedent of the new rule. The Effect is mapped to an effect-literal $L \in \mathcal{L}$ and becomes the conclusion. The effect-literal can be either Permit-P or Deny-P where P is the Policy that contains the Rule. This new rule, denoted $C \rightarrow L$, is added to R. For any policy P and rules r_1, r_2 such that $Permit-P \in Con(r_1)$ and $Deny-P \in Con(r_2)$, we state that r_1 and r_2 are conflicting. The full mapping of the Target element to a DL concept expression is given in Table 4.1. The main idea is that attribute-value pairs are mapped to existential restrictions for example (role Developer) would be mapped to $\exists$ $role.Developer$. We also allow for propositional combinations of attribute-value pairs. Note here that we enforce a one-to-one mapping from attribute names and values used in the XACML policy to their corresponding DL roles and concepts in K (we create a DL role or a concept with the same name as the XACML attribute or value).

For the XACML construct Any, we formalize it as a disjunction where each disjunct corresponds to an attribute. For each attribute, we create another disjunction from all possible attribute values for that attribute. For example, formalizing Any using this mapping would create 15 disjuncts (there are 3 attributes and 5 attribute values). By assuming that the values for role, action and resource attributes are disjoint, we can prune the search space significantly (see how the Any occurring in the running example is mapped below).

Example 4.1 *Consider the policy* P_i *containing the only (natural language) rule* R_j*:* R_j = ***Students cannot write and read technical papers****.* R_j *can be mapped in* $\mathcal{DL}$ *as follows:* $\exists role-type.Students \sqcap \exists res-type.TechnicalReport \sqcap (\exists action-type.read \sqcup \exists action-type.write) \mapsto Deny-P_i$

XACML requests are mapped in the same manner as rules, since they also can be represented as a list of attribute value pairs. To check whether a request r matches a rule with target T, we only need to check whether $K \models \pi(T)(r)$ (equivalent to instance checking in description logics).

Mapping policies and policy sets

To propagate the access decisions from Rules to the root PolicySet, we introduce the following rules in $\mathcal{R}$:

- For each XACML Rule $r :$ ($\texttt{TargetDeny}P$), add an axiom to R, $R = R \cup \{\pi(\mathsf{Target}) \mapsto \mathsf{Deny} - P\}$;
- For each XACML Rule $r :$ ($\texttt{TargetDeny}P$), add an axiom to R, $R = R \cup \{\pi(\mathsf{Target}) \mapsto \mathsf{Permit} - P\}$
- For each policy element P and parent policy element PS introduce the following axioms: $\mathsf{Permit} - P \mapsto \mathsf{Permit} - PS$ $\mathsf{Deny} - P \mapsto \mathsf{Deny} - PS$

A Policy or a PolicySet can also have a Target element. However, we can propagate the constraints specified in Target down to Targets of its children. In this manner, we propagate the constraints to the XACML Rule elements. Thus, without loss of generality, we can assume that Policy and PolicySet elements have empty Target all of the constraints are propagated down to the Target of their XACML Rules descendants. Table 4.1 summarize the mapping details.

4.3.4 Policy Analysis Services in $\mathcal{DL}$

In [56] the following services are provided:

- *Constraints*. We already mentioned separation of duty constraints. In addition, we can also specify more general cardinality constraints; for example, a user cannot be a member of more than 3 security roles at a time. Property

Syntax	π
R ::= (`Rule` T `Effect`)	$\pi(\texttt{T}) \mapsto \pi(\texttt{Effect})$
`Effect` ::= `Permit` | `Deny`	*Permit-P* | *Deny-P*
T ::= ((`Sub`) (`Act`) ((`Res`)))	$\pi(\texttt{Sub}) \sqcap \pi(\texttt{Act}) \sqcap \pi(\texttt{Res})$
`Sub` | `Act` | `Res` ::= `Any` | `Fcn`	$\top \mid \pi(\texttt{Fcn})$
`Fcn` ::= AV | `Fcn` ∩ `Fcn` | `Fcn` ∪ `Fcn` | ¬ `Fcn`	$\pi(\text{AV}) \mid \pi(\texttt{Fcn}) \sqcap \pi(\texttt{Fcn}) \mid \pi(\texttt{Fcn}) \sqcup \pi(\texttt{Fcn}) \mid \neg\pi(\texttt{Fcn})$
AV ::= (attr-id attr-val)	$\exists\pi(attr-id).\pi(attr-val)$
attr-id	$\mathcal{DL}$ property corresponding to $attr-id$
attr-value	$\mathcal{DL}$ property corresponding to $attr-value$

Table 4.1: The mapping function between XACML and $\mathcal{DL}$

(attribute) hierarchies are allowed as well: if X is a brother-of Y, then he is a relative-of Y.

- *Policy Comparison*. For two policies (or policy sets) P_1 and P_2 check if whenever P_1 yields a decision α, P_2 will yield α, too. If not, give a counter example.

- *Policy Verification*. Check if the policy satisfies a particular policy property. If not, give a counterexample.

- *Policy Incompatibility*. If for two policies P_1 and P_2, there cannot exist an access request where both policies apply (yield a decision), then these policies are *incompatible*.

- *Policy Redundancy*. For a policy and an access decision (Permit or Deny), check whether the policy can ever satisfy that decision (or it will be always overridden by some other policy higher up the hierarchy).

- *Policy Querying*. Search for policies in the document based on attribute values.

4.3.5 Results and Discussion

In this section we summarize the key features of Margrave and the $\mathcal{DL}$ approach outlining weak and strong points. In Table 4.2 we briefly represents the differences between the expressivity of the two approaches.

Summarizing, Margrave represents policies using the Multi-Terminal Binary Decision Diagram (MTBDD), which can explicitly represent all variable assignments that satisfy a Boolean expression and hence provides a good representation for the relationships among policies. Policy property verification is then formulated as a query on the corresponding MTBDD structures. The MTBDD structure has been credited with helping model checking scale to realistic systems in hardware verification. The major shortcoming of Margrave is that it can only handle simple conditions, like string equality matching. A direct consequence of such limitation is an explosion of the MTBDD size when conditions on different data domains (e.g. inequality functions) have to be represented. For example, to represent the condition "time is between 8am to 10pm", the MTBDD tool needs to enumerate all possible values between "8am" to "10pm" (e.g., "time-is-8:00am", "time-is-8:01am", "time-is-8:02am", . . .).

On the contrary, the approach proposed in [56] is based on $\mathcal{DL}$. Mapping XACML to Description Logics provide a more interesting subset of the XACML language but it has the obvious drawback of a higher computational complexity. The proposed mapping directly translates XACML targets into $\mathcal{DL}$ concepts and attribute value pairs are translated into existential restriction. The effect of a rule is defined over this basic building blocks and then is propagated to policy and policy sets effects. The advantage of using Description Logic, is that it is possible to gain significantly in expressivity but not loosing too much with respect to performance as we will introduce later. However, it is not clear how their mapping with the different semantic assumptions made in XACML and

Feature	Margrave	Kolovski et al.
Permit-Overrides	yes	yes
Deny-Override	yes	yes
First-Applicable	yes	yes
Only-One Applicable	no	no
Ordered-Permit-Overrides	no	no
Ordered-Deny-Override	no	no
Different attribute datatype	no	yes
Role Hierarchy	no	yes
Resource Hierarchy	no	yes
Continuous Domains	no	yes
Multi-subject request	no	no

Table 4.2: Differences between XACML features supported by the reviewed approaches

Concerning results in [35] the authors say that Margrave take at most 355 milliseconds (ms) for the parsing of a policy and no longer than 10 ms for property verification. Regarding the approach by Kolovski, in [54] it is showed to have comparable time values when considering the translation of a policy and, as expected, slower performance in the property verification. However, the approach by Kolovski et al., appear to be much more scalable with respect to Margrave. This is due to the fact that Margrave need to discretize continuous domains into finite sets of values each one resulting in a new node in the graph. This means obtaining an explosions of nodes in the creation of the MTBDD and problems in the translations of policies.

Finally, we argue that even regardless expressivity, due to the problem of nodes explosion [56] works better in some specific situations:

- policies with predicates defined over continuous domains;
- policies with a large number of attribute value pairs;

However, Margrave is extremely fast when considering policies with a reasonable numbers of pairs and for this reason is most suitable when comparing large dataset of policies.

Part II

EXAM-S: the Model

Chapter 5

Extending XACML with Semantic Functions

In this Chapter we discuss the issue of defining new XACML function for managing information modeled within ontologies. This part of the thesis is not directly related to the development of EXAM-S, however in Chapter 7.5 we introduce new techniques with which we are able to perform policy analysis taking into considerations the functions defined here.

5.1. Introduction

As we have seen in the background section, the management of data in native XACML, is supported with low-level functions such as datatype comparison. Because of such low-level functions it is often difficult to model some important class of applications such as the ones requiring support for Role-Based Access Control (RBAC). Even if XACML cannot support RBAC natively, due to its flexibility, it can be extended to support these specialized access control models. So far, several extensions have been proposed commonly referred to as XACML profiles[1]. In particular, the XACML profile for RBAC [5] provides a mapping between RBAC and XACML introducing new specialized policy sets. However, the current RBAC profile does not provide any support for many relevant constraints, such as static and dynamic separation of duty (SoD). Extending XACML

[1]List of profiles currently available in XACML: http://docs.oasis-open.org/xacml/2.0/

to support such constraints, however, is an issue that requires extensions to the XACML language, namely the introduction of specialized functions, and to the XACML reference architecture and engine. One important requirement for such an extended architecture and engine is to maintain the record of past accesses in order to support dynamic SoD. It is also important to take into account the semantics of role hierarchies with respect to the propagation of authorizations, both positive and negative, along the role inheritance hierarchies. Supporting such propagation and, at the same time, enforcing constraints requires some reasoning capabilities. Therefore, the main issue with respect to the XACML reference architecture and the engine is how to integrate such reasoning capabilities. Because of the wide availability of semantic reasoning tools, such as the ones developed in the context of the semantic Web [9], the most suitable approach is to couple XACML with one such tool.

In this paper we thus propose XACML+OWL, a general and powerful framework that integrates XACML and OWL ontologies for supporting RBAC. Our approach is to decouple the management of constraints, such as SoD, and RBAC hierarchies from the specification and the enforcement of actual XACML policies. Such an approach essentially allows to use any XACML engine without requiring any extensions to it; all interactions between the XACML engine and the OWL ontology are encapsulated inside of specialized functions, referred to as *semantic functions*, that are also defined in this paper. XACML+OWL has thus been designed as a two-layer framework in which constraints are modeled by an OWL ontology, whereas policies are specified by using XACML. The two layers interact through the semantic functions, that basically invoke reasoning operations on underlying ontology and to return result of those operations to the invoking XACML policy. Our approch has several advantages: (i) OWL ontologies have been shown to be expressive enough for expressing security models, including constraints [49], [56], [89] and [34]; (ii) because of the reasoning services provided by an OWL ontology, the design and development of ontology-based XACML policies become easier, thus leading the way to the use of semantic reasoning also in non-RBAC policies; (iii) since the ontology is decoupled from XACML policies, the management of constraints and policies become more flexible; (iv) different XACML-based systems can be based on a single ontology schema, allowing for the development of distributed and more efficient architectures as well as simplifying the interoperation among heterogeneous policy models.

Important contributions of the paper are as follows:

1. We show that OWL ontologies are powerful enough to represent hierarchical RBAC and SoD constraints. As a side effect we obtain a new approach to the mapping of RBAC onto ontologies that addresses major shortcomings of previous mapping approaches [34] and [52].

2. We propose an extension of the XACML standard for integrating into XACML semantic reasoning services based on the OWL ontology. Such extension is based on the semantic functions, to retrieve and reason about the information modeled by the ontology. As part of such extension, we extend the reference architecture of XACML and the XACML data-flow for access control decisions with the invocation of such functions.

The remainder of the paper is organized as follows. In Section 5.2 we provide relevant background notions. In Section 5.3 we introduce a running example to illustrate our approach. The XACML+OWL framework is discussed in Section 5.4. We introduce our strategy for the RBAC and OWL mapping in Section 5.5 whereas details about the extended XACML policies and a complete example of the policy enforcement process are presented respectively in Sections 5.6 and 5.7.

5.2. Background notions about RBAC

We represent a role hierarchy with a tuple $\langle R, \leqslant \rangle$, where R is the set of roles and $\leqslant$ is a partial order relation over the elements of R. We represent users as elements u_i of the set of all users U. For the representation of separation of duty constraints we follow the definition proposed by Crampton [21]: a SoD constraint is a tuple

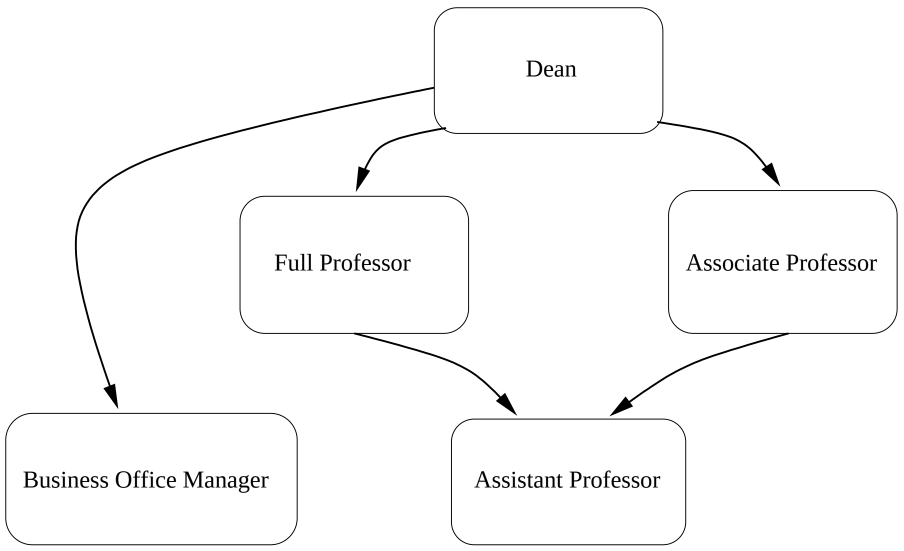


Figure 5.1: Role hierarchy.

$\langle S, (C_i, C_j), \rho\rangle$, where: S is the scope associated with the *constraint set* C_i, called the *antecedent constraint set*; C_j is the *consequent constraint set* to which the constraint is applied; and ρ is the temporal context that can take a value in the set $\{s, d\}$, where s and d stands respectively for *static* and *dynamic* context. The differences between a static (SSoD) and dynamic (DSoD) SoD is that a SSoD limits the entire space of the constraint set, whereas a DSoD limits the assignment to the constraint set during run-time. A constraint $\langle S, (C_i, C_j), \rho\rangle$ is *satisfied* if, whenever $x \in S$ is associated with C_i and y is associated with C_j, then $(x, y) \in \rho$. The general form of a SoD constraint presented above can be applied to different scenarios restricting the scope or the constraint set. For example the SSoD constraint $\langle U, (R_i, R_j), s\rangle$ specifies that the sets of users assigned to R_i and R_j must be disjoint.

5.3. Running Example

To illustrate our approach we consider a simple scenario from an academy domain environment. Specifically, we consider a process for submitting research proposals. Roles in such scenario are organized according to the hierarchy shown in Figure 5.1. Other relevant entities in our scenario are the **Project** resource and the **Submit**, **Review** and **Approve** action. The overall process consists of the following activities: (i) the submission of a project; (ii) its review performed by at least two subjects; and (iii) its approval. Moreover, user-to-role assignments, permissions, and actions must satisfy the following requirements:

- $\texttt{Req}_1$: a subject cannot belong to more than one role.
- $\texttt{Req}_2$: every subject in the hierarchy except the **business office manager** can submit a project.
- $\texttt{Req}_3$: every professor, except **assistant professor** can review a project.
- $\texttt{Req}_4$: a subject cannot review a project he/she has submitted.
- $\texttt{Req}_5$: a professor can review the same project at most one time.
- $\texttt{Req}_6$: only the **business office manager** and the **dean** can approve a project;
- $\texttt{Req}_7$: a subject cannot approve projects he/she has reviewed or submitted;

$\texttt{Req}_2$, $\texttt{Req}_3$ and $\texttt{Req}_6$ constrain the privileges that can be assigned to the roles. $\texttt{Req}_1$, $\texttt{Req}_4$, $\texttt{Req}_5$ and $\texttt{Req}_7$ specify the SoD constraints holding in our scenario. Specifically, $\texttt{Req}_1$ is a SSoD constraint whereas $\texttt{Req}_4$, $\texttt{Req}_5$ and $\texttt{Req}_7$ are DSoD constraints. We finally remark that the scope of the policies is limited to a single instance of the process where the process is composed by the activities (i), (ii) and (ii) introduced above.

5.4. The XACML+OWL Framework

The XACML+OWL framework is organized according to the three major components (see Figure 5.2): (i) *Authoring Environment*, (ii) *Repositories*, and (iii) *Enforcement System*. The Authoring Environment supports the specification of access control policies, role hierarchies, and constraints. The Repositories store the

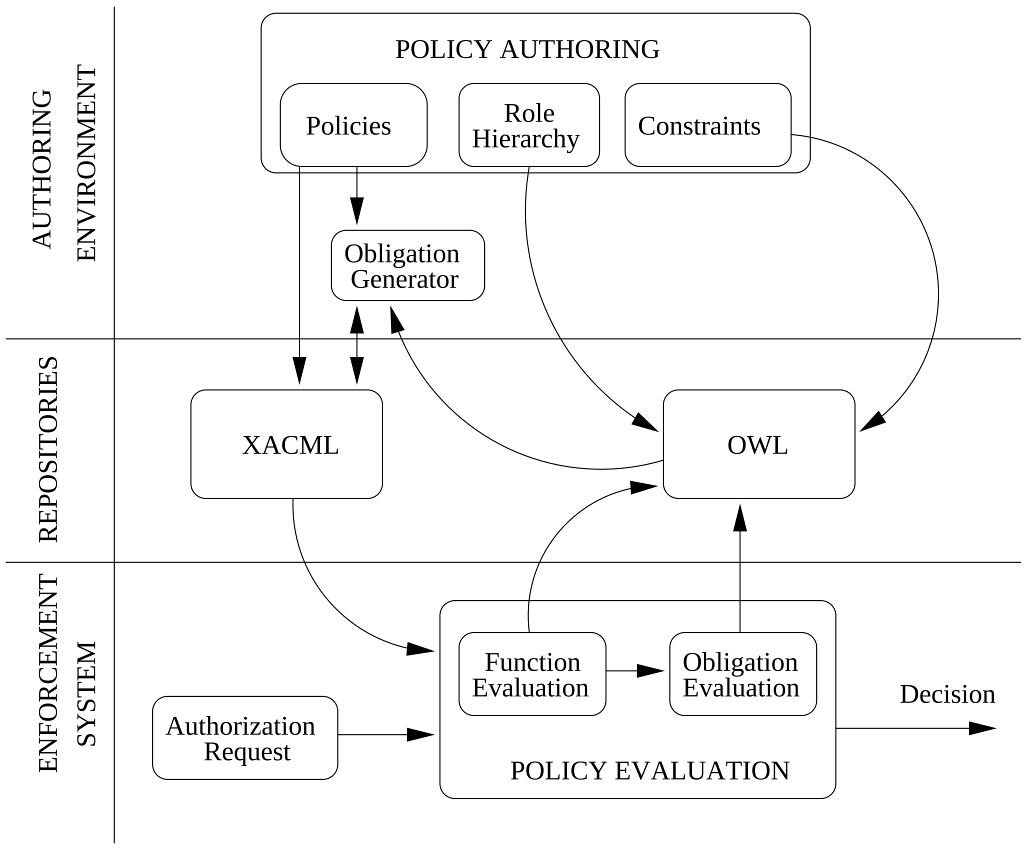


Figure 5.2: The XACML+OWL framework.

OWL ontology and the XACML policies. Such repositories are not directly accessible to the user; however the user that can manage both repositories from the Authoring Environment. The policies and the ontology are then exploited by the enforcement level during the enforcement of authorization requests. Finally, the Enforcement System contains the core XACML engine, organized according to the XACML reference architecture, extended with additional modules for handling our semantic functions and obligations. The Enforcement System receives in input an authorization request and performs policy evaluation by querying the ontology through the invocation of the semantic functions and the obligations. The result of the enforcement process is the final authorization decision. A detailed discussion of the extended XACML data flow diagram is presented in Section 5.8.

A critical module in our architecture is the *obligation generator*. Obligations in our model are crucial since they are used to support DSoD constraints. Our approach to manage such constraints can best be illustrated as follows. Consider a DSoD constraint stating that no subject can be granted both permissions p_i and p_j during the same session[2]. Suppose a policy $\mathcal{P}_1$ ($\mathcal{P}_2$) exists which grants permission p_i (p_j); it is obvious that if such a policy is applied and the permission is granted to a subject s, the enforcement system must remember this information, so that in the same session, permission p_j is not granted. Our approach to "remember such information" is to include in the policies ($\mathcal{P}_1$ and $\mathcal{P}_2$ in this case) an obligation that adds such information to the ontology. In our example, the enforcement system will store the information that s has been granted p_i. Now suppose that during the same session, s issues a request to which $\mathcal{P}_2$ applies. The application of such policy would then result in s receiving permission p_j. As for the case of permission p_i, the execution of the obligation associated with $\mathcal{P}_2$ will result in an attempt to update the ontology with the information that s has been granted p_j; such update however will fail, because of an inconsistency in the ontology, and thus such information will not be stored and the authorization will

[2]The specific notion of session is not relevant here.

be denied. Because of the relevance of obligation in enforcing constraints, the XACML+OWL includes an obligation generator that support for the automatic creation of obligations. Whenever a new policy is created, this module generates the necessary obligation and updates automatically the policy.

5.5. SoD Constraints Definition using OWL Ontologies

In this section we discuss the use of OWL ontologies for the representation role hierarchies and constraints. We would like to remark here that OWL ontologies are flexible and the solutions provided here can be easily extended for application to different scenarios. Given a role hierarchy $\langle R, \leqslant \rangle$, we model R as an OWL class `Role`,and all the roles in R as instances of this class. The $\leqslant$ relation is represent through the OWL ObjectProperty `subRoleOf`(`Role`, `Role`). Moreover, we define `subRoleOf` to be transitive, thus making it an instance of `owl:TransitiveProperty` class. Transitivity makes it possible for a reasoner to infer that if `subRoleOf`(R_i, R_j) and `subRoleOf`(R_j, R_k), then `subRoleOf`(R_i, R_k). For example, we model the relation between $FullProfessor$ and $AssistantProfessor$ in our sample scenario by adding to the ontology the property `subRoleOf`(`Assistant Professor`, `Full Professor`). For optimization purposes we also define the property `supRoleOf`(`Role`, `Role`). `supRoleOf`($\mathtt{R_i}$, $\mathtt{R_j}$) represents that $\mathtt{R_i}$ dominates $\mathtt{R_j}$. Like `subRoleOf`, `supRoleOf` is transitive. We also specify that `subRoleOf` and `supRoleOf` are inverse properties by adding the `owl:inverseOf` construct. P_i `owl:inverseOf` P_j means that for every pair $(x, y) \in P_i$, there is $(y, x) \in P_j$ and vice versa. We finally represent the set of subject by the `Subject` class whose instances represent the subjects on which the policies are defined. The association between a subject to a role is obtained by the ObjectProperty `hasRole`(`Subject`, `Role`).

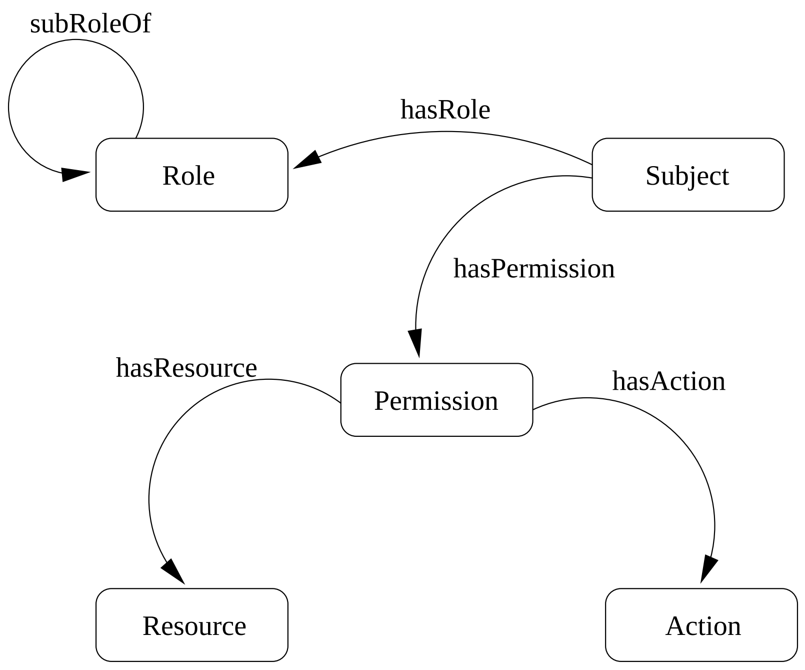


Figure 5.3: OWL schema for RBAC.

5.5.1 Static Separation of Duty Constraints

Let $\langle \mathsf{Subject}, (R_i, R_j), s\rangle$ be a role-based SSoD constraint. Representing this constraint in our ontology means imposing the constraint that the users that are assigned to R_i cannot be assigned to R_j and viceversa. In DL ontologies, given a relation $R(A, B)$, the set of individuals belonging to A that are associated with a specific instance of B, say b, is represented by the class $\exists R : b$ where : is the role filler construct. Thus, in order to model the $\langle \mathsf{Subject}, (R_i, R_j), s\rangle$ constraints, the following axioms are added to the ontology:

- $C' \equiv \exists \mathsf{hasRole} : R_i$.
- $C'' \equiv \exists \mathsf{hasRole} : R_i$.
- C' `owl:disjointWith` C''.

For example, according to Req_1 in our running example, the SSoD constraint concerning full professors and assistant professors is modeled with the axiom $\exists \mathsf{hasRole} : \mathsf{FullProfessor}$ `owl:disjointWith` $\exists \mathsf{hasRole} : \mathsf{AssistantProfessor}$.

5.5.2 Dynamic Separation of Duty Constraints

Let $\langle \mathtt{Subject}, (R_i, R_j), d \rangle$ be a role-based DSoD constraint. We model this kind of constraints by adopting a similar approach to the one introduced for SSoD. The only difference is that at design time not all the assignments between subjects and roles are known. Following [34], we create an other class of roles, called `ActiveRole`. The instances of such class are all the roles that can be activated at run-time (from now on we refer to this kind of roles as *active roles*). Moreover, also the the `hasRole` object properties must be extended for handling also active roles: $\mathtt{hasRole}(\mathtt{Subject}, (\mathtt{Role} \sqcup \mathtt{ActiveRole}))$. For example, to enforce Req_4 from our running example we add to the ontology roles `Submitter` and `Reviewer`, and the following axiom: $hasRole : Submitter$ `owl:disjointWith` $hasRole : Reviewer$. The dynamic assignment of subjects to roles cannot be handled by an OWL ontology by itself. Typically, the assignment of a user to a dynamic role is the result of some specific administrative operations. Instead of introducing a new specific module to handle such operations, we exploit the obligation mechanism of XACML. The details of the approach are discussed in Section 5.6.2.

We can also specify more complex constraints involving not only subjects, but also actions and resources. For example, we may be interested in add a DSoD between the set of users that have reviewed a project and the set of users that can approve a project. It is worth noting that, following the above example, we may add a constraint between the classes $Reviewer$ and $Approver$. However, when the application has a large number of resources and actions, such an approch would potentially result in creating an exponential number of classes each one for each possible action-resource pair. In order to address this issue, we add three new classes: `Resource`, `Action`, and `Permission`. Classes `Resource` and `Action` represent, respectively, resources and actions. The `Permission` class binds a user to an action-resource. This association can be obtained through the two functional ObjectProperties $\mathtt{hasAction}(\mathtt{Permission}, \mathtt{Action})$, $\mathtt{hasResource}(\mathtt{Permission}, \mathtt{Resource})$ linking an action and a resource to a permission and the ObjectProperty `hasPermission`(`Subject`, `Permission`) that links a subject to a permission. For example, Req_7 ca be modeled by adding the following axioms:

- $D' \equiv \exists hasPerm.(\exists hasAct : \{Review\} \sqcap \exists hasRes : \{Project\})$;
- $D'' \equiv \exists hasPerm.(\exists hasAct : \{Submit\} \sqcap \exists hasRes : \{Project\})$;
- D' `owl : disjointWith` D'';

We remark that after the specification of a new constraint at the Authoring environment, the associated axioms are automatically generated and added to the ontology.

5.6. XACML Policies

In XACML a policy is represented by a target, a set of rules, and a rule combining algorithm (see Section 2). The applicability of the policy to a request is then determined by evaluating the list of attribute-value pairs provided in the request against the conditions in the rule target. The XACML profile expresses role as a subject attributes. Other attributes can also be associated with the subject in order to support fine-grained policies. In XACML+OWL subjects are represented as OWL individuals. The attributes with which a subject is associated are represented with both the OWL individuals and data type values related to that subject through properties. Since the subject role is an individual associated with the subject through the `hasRole` property, in our model roles are represented through subject attributes as defined in the XACML RBAC profile. However, instead of specifying all the possible subject attributes, the request specifies just the actual individual that represents the subject in the ontology. Then, we let the policies specify which are the attributes they require by specifying the name of the properties associated with the individual. Specifically, we first use the attribute id to retrieve the subject in the request context, then we apply to such subject the semantic functions able to access the additional attributes required for the policy evaluation. An example of a function is `hasObjectValue`(s_i, P_j) that takes in input the individual s_i and the object property P_j and retrieves the set of individuals associated with s_i through P_j. Such approach does not require modifications to the policy structure nor to the attribute retrieval mechanism.

5.6.1 Semantic Functions for instances

Given an individual, there are essentially three kinds of information that can be extracted: (i) the class the individual belongs to; (ii) the class instances with which the individual is related through an object property; and (iii) the data values associated with the individual through a datatype property. In order to retrieve such information we define the following functions:

- `hasObjectValue`(s_i, P_j): it returns the set of instances associated with s_i through the object property P_j;
- `hasStringValue`(s_i, P_j): it returns the set of string values associated with s_i through the datatype property P_j of type String;
- `hasIntValue`(s_i, P_j): it returns the set of integer values associated with s_i through the datatype property P_j of type Integer;
- `relatedTo`(s_i, P_k): it returns all the individuals associated with s_i through the object property P_k;
- `transitiveRelatedTo`(s_i, P_k): it is the same as the `relatedTo` function but the property given in input is the transitive object property P_k. This means that if $(s_i, s_{j'}) \in P_k$ and $(s_{j'}, s_j) \in P_k$ then `transitiveRelatedTo` returns both s_j and s_j'.

Functions `instanceOf` and `hasObjectValue` are associated respectively with categories (i) and (ii) introduced above. `hasStringValue` and `hasIntValue` provide a way to retrieve the datatype values associated with the input individual. The only datatype we support are String and Integer. We plan to extend our

```
<Policy>
 <Subject>
  <SubjectMatch MatchId=‘‘string-equal’’>
   <AttributeValue>Ri</AttributeValue>
  <Apply FunctionId:‘‘transitiveRelatedTo’’>
    <Apply FunctionId:‘‘hasObjectValue’’>
     <SubjAttrDesignator AttrId:‘‘subject-id’’>
      <AttributeValue>hasRole</AttributeValue>
    </Apply>
    <AttributeValue>supRoleOf</AttributeValue>
   </Apply>
  </SubjectMatch>
 </Subject>
 </Subject>
 <Resource>
  <AttributeValue>Project</AttributeValue>
  <ResAttrDesignator AttrId=‘‘resource-id’’>
 </Resource>
 <Action>
  <AttributeValue>Submit</AttributeValue>
  <ActttrDesignator AttrId=‘‘action-id’’>
 </Action>
</Policy>
```

Table 5.1: A XACML policy with ontology elements.

model to other datatypes in future work. The boolean functions `relatedTo` and `transitiveRelatedTo` are both related to category (ii). `transitiveRelatedTo` is particularly useful in our scenario because allow us to check the dominance relation between two roles. An example, is shown in Table 5.1 in which we provide an abstraction of a policy that exploits with our functions. Consider the subject match of the policy, in order to check role dominance we first extract the role of the subject with the function `hasObjectValue`, then we retrieve all the roles subsuming the role R_i. Finally, we compare the obtained bag of roles with the role R_i with the standard function $string - equal$. It is worth noting that to support RBAC we just need the functions `transitiveRelatedTo` and `hasObjectValue`. We provide however the functions `hasStringValue`, `hasIntValue` and `relatedTo` for dealing with some attributes that, even if not supplied in the standard RBAC, that may be associated with the subject in some specialized implementations.

Notice that instead of introducing new functions there are some alternative solutions to retrieve subject attributes from an XACML policy. For example we may embed the name of the property that binds the individual to the required attribute directly within the attribute id. For example, if we want to determine the role of a subject, we may define the attribute id `urn:subject:subject-id:hasRole`. However, this solution requires an additional module in the context handler able to "understand" the attribute id and retrieve the intended values. Moreover, if we want to exploit the full power of DL reasoning, we have to be able to embed complex expression in the attribute id creating an ad hoc DL-like syntax. Another solution is to define a function supporting the specification an ontology query (for example with the SPARQL query language[3]). However, such an approach has the problem of the run-time customization of the queries with the actual values provided in the request.

[3]http://www.w3.org/TR/rdf-sparql-query/

```
<Obligation
    Id=‘‘addPermission’’, FulfillOn=‘‘Permit’’>
 <AttributeAssignment
  AttributeId:‘‘urn:addpermission:subject’’>
  &lt;SADesignator AttrId:‘‘subject-id’’/&gt;
 </AttributeAssignment>
 <AttributeAssignment
  AttributeId:‘‘urn:addpermission:resource’’>
  &lt;RADesignator AttrId:‘‘resource-id’’/&gt;
 </AttributeAssignment>
 <AttributeAssignment
  AttributeId:‘‘urn:addpermission:action’’>
  &lt;AADesignator AttrId:‘‘action-id’’/&gt;
 </AttributeAssignment>
</Obligation>
```

Table 5.2: The `addPermission` obligation.

5.6.2 XACML Obligations

In our approach, constraints are modeled within the ontology, and the PDP is not aware of which constraints hold for the policies. This a critical issue when considering dynamic constraints because a policy, in order to grant an authorization, must be aware whether such authorization violates come constraints. In XACML+OWL we solve such issue by exploiting the XACML obligation mechanism. An obligation is an operation that the PDP can send to the PEP along with the policy evaluation result. The PEP should fulfill the obligation it receives during the enforcement of the authorization decision. Depending by the semantics of the obligation, a failure of its execution may influence the enforcement process. In XACML+OWL, we exploit the obligation mechanism for managing the DSoD constraints. Specifically, we define the new obligation `addPermission`. `addPermission` is considered as a mandatory operation; this means that even if the PEP should allow access to the required resources, if it cannot fulfill the obligation, then the access shall be denied. The effect of the `addPermission` is to add to the ontology a list of axioms if the permission specified in the associated policy is granted. As shown in Table 5.2, `addPermission` is sent to the PEP only if the effect of the associated policy is evaluated as `Permit`. In the following, we list the axioms that are added to the ontology by such obligation:

$Ax\_1:$ `Individual(p_i` $type$`(Permission))`

$Ax\_2:$ `hasAction(p_i, a)`

$Ax\_3:$ `hasResource(p_i, s)`

$Ax\_4:$ `hasPermission(u, p_i)`

Those axioms are related to the permissions that are allowed by such policy. For example, consider a policy that permits to a role r to perform the action a on the resource s. If a request satisfy the requirements of the policy, the above axioms are instantiated with the actual values and added to the ontology. Specifically, Ax_1 creates the new permission p_i. Ax_2 and Ax_3 associate p_i respectively with the action `a` and the resource `s`. Finally, Ax_4 associates the subject u specified in the request with the permission p_i. The addition of new axioms to the ontology is exploited for checking constraints violation. As we have discussed in Section 5.5, in our approach we reduce constraint evaluation to the problem of checking the consistency of an ontology. This means that if the addition of new axioms to a consistent ontology yields an inconsistency, the new axioms that have been introduced violate some of the constraints defined in the ontology. Obviously, if an inconsistency is detected the new axioms are deleted from the ontology. This ensure the consistency of the ontology after each execution of the obligation.

Notice that there are some alternative solutions for managing DSoD constraints. For example, we may let each policy checks if the enforcement of the permissions it grants violates the constraints in the ontology. This means to replicate the constraints in each policy the constraints are involved. An obvious drawback is that when we have to change a constraint we have to update both the ontology and the involved policies. Another possibility is to introduce a dedicated module specifically designed for acting as a mediator between XACML policies and the OWL ontology. The role of this new module is to interact with the XACML components to provide the necessary information for the enforcement of the requests. For example we may associate the id of a policy with a certain constraint in the ontology and, whenever that policy applies, the module checks if the granted permission violates the constraints. This approach has several drawbacks: first of all we have to update the XACML enforcement process for dealing with the new module. Moreover, when a policy or a constraint is changed we have to update (if necessary) also the associations maintained by the mediator.

5.6.3 Policy Semantics

A XACML policy is a set of rules combined through a *rule combining algorithm*. The semantics of a policy defined over a role hierarchy depends from the evaluation of its rules and the algorithm by which the rules are combined. Given a policy $\mathcal{P}_i$ we categorize the rules belonging to a policy into two sets DR and PR defined as follows:

```
$f_{deny-override}(r, s, a) =$
IF
    $\exists r': r \leqslant r' \wedge (r', s, a) \in DR$;
THEN Deny;
ELSE IF
    $\exists r'': r'' \leqslant r \wedge (r'', s, a) \in PR$;
THEN Permit;
ELSE NotApplicable;
```

Figure 5.4: The function $f_{deny-override}$

```
$f_{permit-override}(r, s, a) =$
IF
    $\exists r': r' \leqslant r \wedge (r', s, a) \in PR$
THEN Permit;
ELSE IF
    $\exists r'': r \leqslant r'' \wedge (r'', s, a) \in DR$;
THEN Deny;
ELSE NotApplicable;
```

Figure 5.5: The function $f_{permit-override}$

$$DR = \{Rule_j \mid Rule_j \in Pol_i \wedge EffectRule_j = \texttt{Deny}\}$$
$$PR = \{Rule_j \mid Rule_j \in Pol_i \wedge EffectRule_j = \texttt{Permit}\}$$

where $\mathsf{Effect}\{Rule_j\}$ extracts the effect of rule $Rule_j$. Given the role hierarchy $\langle R, \leqslant \rangle$ and an authorization request (r, s, a), such that $r \in R$, $s \in S$ and $a \in A$, the semantics of a policy is specified according to functions $f_{deny-override}$ and $f_{permit-override}$ shown respectively in Figures 5.6.3 and 5.6.3.

$f_{deny-override}$ grants the access request $(r,\ s,\ a)$ if: (i) an applicable rule $\in$ PR exists defined on a role r'' that is dominated by r; and (ii) no applicable rule $\in$ DR exists defined on a role r' that dominates r. Conversely, given the authorization request $(r,\ s,\ a)$, if an applicable rule $\in$ PR exists defined on a role r' that is domi-

1: $A(\mathrm{Rule}_k, r) \wedge P(\mathrm{Rule}_k) \rightarrow \langle e, (f^{\geqslant}(r), s, a)\rangle \in XAB$

2: $A(\mathrm{Rule}_k, r) \wedge D(\mathrm{Rule}_k) \rightarrow \langle e, (f^{\leqslant}(r), s, a)\rangle \in XAB$

Table 5.3: Formalization of authorization rules.

nated by r, then $f_{permit-override}$ grants the access request without checking for the presence of deny rules.

5.6.4 Automatic Creation of XACML policies

In this section, we discuss how we translate the authorizations defined by using the Authoring Environment into the XACML+OWL policies. We remark that the actual version of XACML+OWL does not provide an automatic procedure for inferring a rule combining algorithm. The user must thus specify the appropriate algorithm to use for combining the authorization rules into a policy. For simplicity, we first consider the case in which authorization rules are defined on a single subject. We discuss multi-subject rules later in this section.

At the level of the Authoring Environment an authorization rule is seen as a tuple $\langle \mathrm{Effect}, (r, s, a)\rangle$, where $\mathrm{Effect} \in \{\texttt{Permit}, \texttt{Deny}\}$, $r \in R$, $s \in S$ and $a \in A$. We define the predicate $A(\mathrm{Rule}_i, r)$ as a Boolean predicate that checks whether the Rule_i applies to the role r. We define also the new Boolean predicates $D(\mathrm{Rule}_i)$ and $P(\mathrm{Rule}_i)$ that check respectively whether the Rule_i has a `Deny` or `Permit` effect. We also refer to the XACML Authorization Base, that is the repository containing the rules used for integration of XACML and OWL, as XAB. A policy is then defined over the rules in XAB by selecting a combining algorithm and defining the policy target. Given a role hierarchy $\langle R, \leqslant\rangle$, the interpretation of a Rule_k is given in Table 5.3. The first formula states that if a `Permit` Rule_k is applicable to role r, then the tuple $\langle e, (f^{\geqslant}(r), s, a)\rangle$ is added to XAB. $f^{\geqslant}(r)$ is a shortcut for specifying the applicability of $\mathtt{Rule}_k$ also to roles that dominates `r`. In XACML+OWL $f^{\geqslant}$ is obtained through the combination of the functions `transitiveRelatedTo` and `hasObjectValue` as shown in Table 5.1. Similarly, the second formula states that if a rule `Deny` $\mathtt{Rule}_k$ is applicable to the role `r`, then the tuple $\langle e, (f^{\leqslant}(r), s, a)\rangle$ is added to XAB, where $f^{\leqslant}$ has the inverse semantics of $f^{\geqslant}$.

It is worth noting that the mechanism introduced so far for the automatic creation of policies, is consistent to the XACML standard even if we have a rule with multiple subjects. Consider a rule $\mathtt{Rule}_i$ and the roles r and r' such that $A(\mathtt{Rule}_i, r) \wedge A(\mathtt{Rule}_i, r')$ are verified. Then in order for $\mathtt{Rule}_i$ to be applicable to a certain subject s_k, s_k must have both roles r and r'. This means to have two `<subject>` element each one specifying: (i) a different role, in our case r and r'; (ii) the combination of function `hasObjectValue` and `transitiveRelatedTo` for retrieving the role of the s_k. First, we retrieve all the roles dominated by the role of s_k through the functions. Then, this bag of roles is evaluated against each one of the `<subject>` element of the rule.

5.6.5 Policy evaluation

In this section we show how the formalization of the policies given in Section 5.6.4 is sound with respect the policy semantics defined in Section 5.6.3. The XACML policy evaluation process checks whether the target of a policy is applicable, if this is the case all the rules in the policy are evaluated and the policy value if finally determined according to the rule combining algorithm. Given a policy $\mathcal{P}_i$, let $XAB_i = \{\mathtt{Rule}_j \in XAB \mid \mathtt{Rule}_j \in \mathcal{P}_i$ be the set of all rules in $\mathcal{P}_i$. The rules in the policy and the policy are then evaluated according to the pseudo-code shown in Figure 5.6.5. Given a role hierarchy $\langle R, \leqslant\rangle$, the correspondence with the function $f_{deny-override}$ is verified by considering that: (i) according to the semantics defined in Section 5.6.4, $r \in f^{leq}(r')$ and $r \in f^{geq}(r')$ are evaluated `True` respectively when $r \leqslant r'$ and $r' \leqslant r$; and (ii) given $\mathtt{Rule}_k$ if $\mathtt{Effect}(\mathtt{Rule}_k) = \mathtt{Deny}$ then $\mathtt{Rule}_k \in DR$

```
$deny - override(r, s, a) =$
IF
   $\exists \langle e, (f^{\leqslant}(r'), s, a) \rangle : r \in f^{\leqslant}(r') \wedge e =$ Deny;
THEN Deny;
ELSE IF
   $\exists \langle e, (f^{\geqslant}(r'), s, a) \rangle : r \in f^{\geqslant}(r') \wedge e =$ Permit;
THEN Permit;
ELSE NotApplicable;
```

Figure 5.6: Policy evaluation

whereas if $Effect(Rule_k) =$ `Permit` then $Rule_k \in PR$. The case for $permit - override$ follows the approach proposed for $deny - override$.

5.7. A Complete Example of Policy Enforcement

In this Section we provide a complete example to illustrate the enforcement process in XACML+OWL. Suppose we have the request $REQUEST_i$, in which `Alice` requires to `Review` a `Project`:

```
REQUEST_i:
<Request>
  <Subject>
    <Attribute AttributeId=''subject-id''>
    <AttributeValue>Alice</AttributeValue>
  </Subject>
  <Resource>
    <Attribute AttributeId=''resource-id''>
    <AttributeValue>Submit</AttributeValue>
  </Resource>
  <Action>
    <Attribute AttributeId=''action-id''>
    <AttributeValue>Project</AttributeValue>
  </Action>
</Request>
```

Suppose now the policy POLICY_k that contains the rule RULE_j. RULE_j allows the access to perform the `Review` action on the `Project` resource to subjects having a role that subsumes the role `Associate Professor`:

```
RULE_j
<Rule Effect=‘‘Permit’’>
 <Subject>
  <SubjectMatch MatchId=‘‘string-equal’’>
   <AttrValue>Associate Professor</AttrValue>
   <Apply FunctionId:‘‘hasTransitiveObjectValue’’>
    <Apply FunctionId:‘‘hasObjectValue’’>
     <SubjAttrDesignator AttrId:‘‘subject-id’’>
     <AttributeValue>hasRole</AttributeValue>
    </Apply>
    <AttributeValue>supRoleOf</AttributeValue>
   </Apply>
  </SubjectMatch>
 </Subject>
 <Resource>
  <AttributeValue>Project</AttributeValue>
  <ResAttrDesignator AttrId=‘‘resource-id’’>
 </Resource>
 <Action>
  <AttributeValue>Review</AttributeValue>
  <ActttrDesignator AttrId=‘‘action-id’’>
 </Action>
</Rule>
```

Moreover, POLICY_k has the $\mathsf{addPermission}$ obligation introduced in Table 5.2. Now, in order to evaluate RULE_j against the individuals in $\mathsf{REQUEST}_i$ we need to retrieve the role of `Alice`. This is done by combining the functions `transitiveRelatedTo` and `hasObjectValue` as is shown from rows 3 to 12. According to the hierarchy in Figure 5.1, the request is authorized and through the obligation the following axioms are added to the ontology.

$Ax\_1:$ $\texttt{Individual}(\texttt{p\_i}\quad type(\texttt{Permission}))$
$Ax\_2:$ $\texttt{hasAction}(\texttt{p\_i},\ \texttt{Submit})$
$Ax\_3:$ $\texttt{hasResource}(\texttt{p\_i},\ \texttt{Project})$
$Ax\_4:$ $\texttt{hasPermission}(\texttt{Alice},\ \texttt{p\_i})$

Suppose now that `Alice` requires again the authorization $\texttt{REQUEST}_\texttt{i}$. $\texttt{RULE}_\texttt{j}$ is evaluated again and the and the authorization is granted. However, when the PEP tries to fulfill the obligation associated with $\texttt{POLICY}_\texttt{k}$, the new axioms violate the constraint Req_5 and an ontology inconsistency is detected. Due to such inconsistency, the new axioms are deleted from the ontology and the authorization is denied.

5.8. Extended XACML architecture

In this Section we propose an extension of the XACML architecture for managing semantic functions. Figure 5.7 shows the XACML architecture extended for dealing with our new semantic functions. Black boxes represents standard XACML components whereas blue boxes represents the new modules we have introduced or extended.

As in the standard XACML model, the Policy Enforcement Point (PEP) receives an access request, extracts the attributes in the request, generates an XACML request and sends it to the Context Handler (CH) for the evaluation. It also makes sure that all the obligations with an authorization decision are executed. The CH receives a request and forward it to the Policy Decision Point (PDP). The PDP in turn fetches the applicable policies from the policy repository in the repository level. Applicable policies are determined by evaluating, among others, the

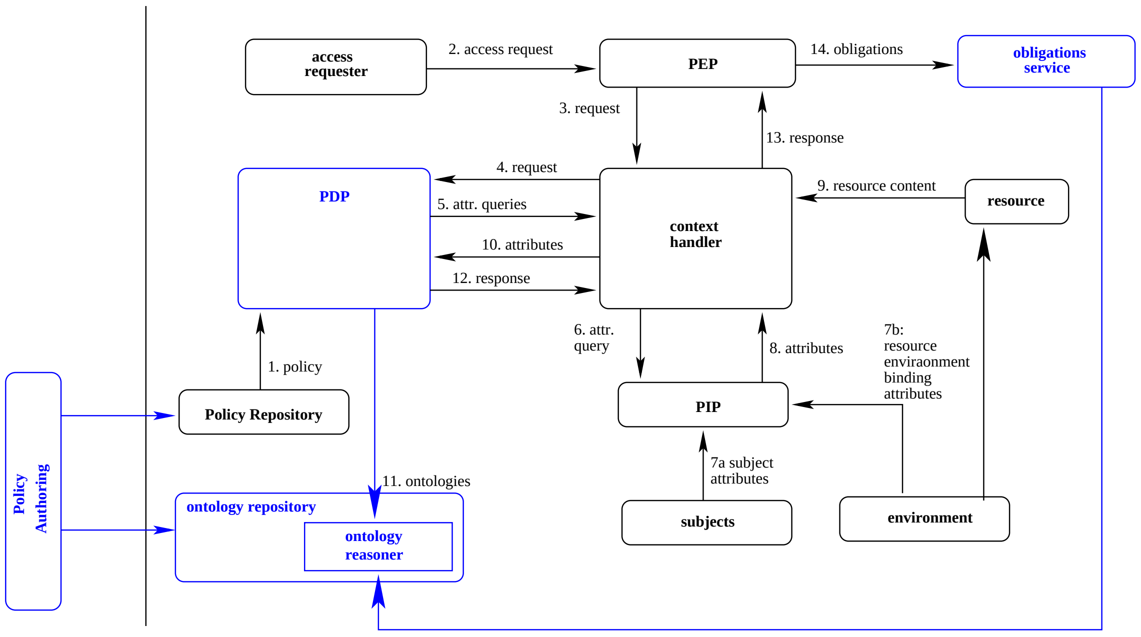


Figure 5.7: Extended XACML Data Flow Diagram.

semantic functions we have defined in XACML+OWL. In doing this we have introduced an ontology reasoner that performs the required operations on the OWL ontology. The PDP returns an authorization decision along with the obligation (if any) that has to be fulfilled by the PEP. Obligations are resolved by the obligation service that interact with ontology reasoner and returns exceptions if the addition of new axioms results in an ontology inconsistency.

Chapter 6

Dealing with Heterogeneous Domains

A key feature in our approach is the ability to execute queries and verify properties on a dataset of heterogeneous policies. This means to create a unified and consistent vocabulary of terms in a way that every policy in the dataset can be expressed with it. In Chapter 4 we have identified three different kind of variations: *syntactical*, *terminological* and *semantic* variations. Usually, Syntactic variations can be identified by using look up tables. Look up tables enumerate the possible ways in which the same term can be written by using different character combinations. Instead, to detect terminological variations, dictionaries or thesaurus such as WordNet [31] can be exploited. Dictionaries are used to retrieve all the synonyms of a given term. Dictionaries and look up tables are typical Natural Language Processing techniques that could be considered as a way to detect the meaning that lies behind the representation of a word. Semantic variations can be determined by using ontology matching techniques. However, it is crucial to adopt much more powerful techniques in order to obtain better mappings and, in turn, a better reference knowledge base for the policy set to be analyzed.

In this chapter we introduce solutions to the issues introduced in Chapter 4. Specifically, our aim is to develop a set of technologies that allows us to create a unified knowledge base for a given policy set. The conceptual architecture of our approach is depicted in Figure 6.1: the basic building block is the Ontology Matching process that is used by all the other methods, on top of ontology

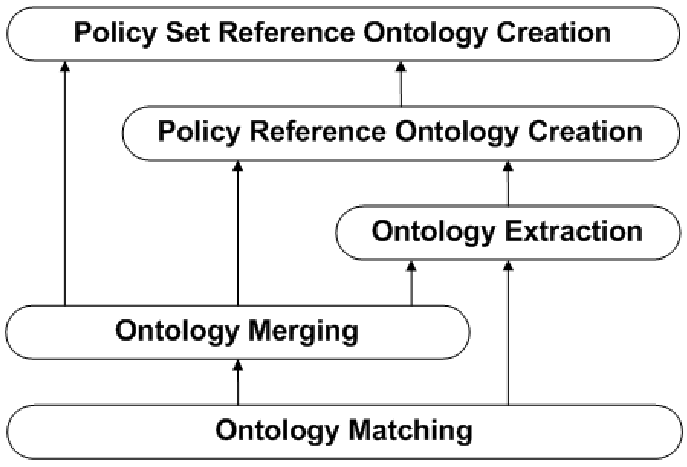

Policy Set Reference Ontology Creation
Policy Reference Ontology Creation
Ontology Extraction
Ontology Merging
Ontology Matching

Figure 6.1: The stack of technologies developed for solving domain heterogeneity.

matching we have Ontology Merging and Ontology Extraction. These processes are not completely decoupled approaches since in the general case, the result of an ontology merging can be exploited during the extraction of an ontology form the unstructured policy data. Final blocks are represented by the creation of the policy reference ontology and in turn reference ontology of the policy set.

In the remaining part of this chapter we describe the architecture in Figure 6.1 adopting a bottom-up approach. In Section 6.1 we describe the ontology matching technique that we use in EXAM-S. Section 6.2 and Section 6.3 are dedicated to Ontology Merging and Ontology Extraction respectively. Finally Sections 6.4 and 6.5 describe how such technologies are combined in building the Policy Reference Ontology and the Policy Set Reference Ontology.

6.1. Ontology Matching

An ontology is a formal representation of a domain in terms of concepts and properties with which those concepts are related. It is used to define the domain and reason about its features. Ontology matching is the process whereby two ontologies are semantically related at conceptual level; concepts belonging to the source ontology are mapped onto the target ontology concepts according to those semantic relations [80]. As introduced in Definition 4.5, an ontology matching function takes as arguments two ontologies O_i and O_j, and returns a set of tuples of the form $\langle e_{O_i}, e_{O_j}, s\rangle$, where e_{O_i} is a concept belonging to ontology O_i, e_{O_i} is a concept belonging to ontology O_j that matches concept e_{O_i}, and s is a confidence score, that typically is, a value between 0 and 1, indicating the similarity between the matched concepts. In Section 2.4 we have reviewed several approaches for solving the ontology mapping problems. In our approach we use a modified version of Falcon-AO [48]. The 2007 Ontology Alignment Evaluation Initiative(OAEI 07) results indicate Falcon to be the best performing ontology matcher available.

Policy View of an Ontology

The Falcon-AO tool is very accurate in finding mappings but when considering large ontologies the matching function is quite slow. Moreover, as we discussed in Section 4.2.1, we may have policies that exploit several ontologies in the definition of the vocabulary. However, not all the entities defined in an ontology are useful for our purposes. Especially when considering large ontologies it is reasonable to take into consideration only the subset of the entities that are relevant for the policy vocabulary creation.

For this reason, we introduce the notion of policy view of an ontology. Given a policy P_i and an ontology O_j such that some elements e_{O_j} are referred by P_i, the view of policy P_i of the ontology O_j is an ontology $\Omega_{O_j}^{P_i}$ such that: (i) $\Omega_{O_j}^{P_i} \subseteq O_j$; (ii) for each $e_{O_j} \in P_i$, e_{O_j} is added to $\Omega_{O_j}^{P_i}$ along with all the axioms related to it e.g. if e_{O_j} is a concept we extract the relation defined over it, and all the entities that belong to the hierarchy in which e_{O_j} appear.

In Chapter 10 we discuss experiments that show that the use if $\Omega_{O_j}^{P_i}$ instead of the whole O_j provides a trade-off between performance and mapping accuracy that is good enough for our purposes.

6.2. Ontology Merging Process

Ontology merging is to process whereby it is possible to obtain a reconciled ontology merging the elements belonging to different semantic schemas. In our model, ontology merging plays a crucial role since is one of the building block in the architecture of our model. Conceptually, the ontology merging process is something more than just adding elements belonging to decoupled schemas to an empty knowledge base. When comparing two ontologies it is possible to have some overlapping elements, if this is the case the ontology merging algorithm have to detect this similarities in order to avoid redundancies in the resulting schema.

Formally, we have to take in consideration the intersection of the elements of the input ontologies.

Definition 6.1 *According to Definition 4.1 and 4.2 we define the intersection of two ontologies* O_i *and* O_j *as the set:*

$$\begin{aligned}\bigcap\nolimits^{T}_{O_i,O_j} &:= O^{T}_{i} \cap O^{T}_{j}\\ \bigcap\nolimits^{A_T}_{O_i,O_j} &:= O^{A_T}_{i} \cap O^{A_T}_{j}\\ \bigcap\nolimits_{O_i,O_j} &:= \bigcap\nolimits^{T}_{O_i,O_j} \cup \bigcap\nolimits^{A_T}_{O_i,O_j}\end{aligned}$$

However, $\bigcap_{O_i,O_j}$ may be difficult to obtain in case of semantic variations. This is the reason why we exploit the ontology matching algorithm in the merging process. Since ontology matching provides mappings between entities belonging to different schemas, we can check if such mappings are *"good enough"* to be trustable and in such a case we can consider those mapped elements as the same entity avoiding to repeat them in the merged ontology.

In order for the elements of a mapping to be considered as the same entity, we evaluate the score of the mapping against a threshold τ. A mapping element is then considered trustable if its score is greater then the threshold. Obviously, the higher is the value of the threshold the higher is the probability to have trustable mappings. The drawback is that assigning an high value to τ we may discard good mappings obtaining redundancy in the merged ontology. In order to have the best performance we have executed several experiments considering both accuracy and recall of our model according to different values of τ. The details of such experiments are reported in Chapter 10.

6.2.1 Ontology Merging Process

The ontology merging process is composed by two different algorithms:

- `MERGE` *(Algorithm 1)*: is the base Merge algorithm that takes in input two ontologies and returns the reconciled knowledge base. This algorithm implements the function MERGE defined in 4.6.
- `ONTOLOGY_MERGING` *(Algorithm 2)* : is the function that applies `MERGE` for the number of ontologies exploited by policy P_i. This algorithm implements all the components defined in 4.6;

In the rest of this section we provide the details of the two algorithms.

`MERGE` (*Algorithm 1*)

The process starts with the creation of a new empty ontology, then we match the two input ontologies and the result of the matching is stored in the variable *mapping* (line 2). Now there are two important issues to deal with:

1. Given the mapping element $\langle e_{O_i}, e_{O_j}, s\rangle$ such that $s \geqslant \tau$, which one of the mapped elements e_{O_i} and e_{O_j} we add to the resulting ontology?
2. In an ontology an entity can be associated to several other entities. For example a concept may be the root of a hierarchy or maybe is in the range of some property. For this reason, after one of the two elements is chosen, say e_{O_i}, and added to the merged ontology, all the axioms in O_j associated to e_{O_j} have to be modified accordingly.

The first issue is described later in section 6.2.2. Concerning the second one, there are several solutions depending by the adopted implementation. For clarity, in Algorithm 1, we deal with the problem with the generic O_j.`update()` function that rename the specified ontology entity with the input name. In the implementation Chapter we describe alternative solutions and we give detail of the one that is currently adopted in our prototype.

In lines 3-6, for each mapping element in the mapping list, if the score is greater than the threshold τ the value belonging to ontology O_i is added to the result ontology. In lines 7-8 and 9-10, the algorithm check whether there are elements belonging to input ontologies that are not added to the result ontology and if this is the case the $\tilde{O}_{i,j}$ is updated with those new elements. The algorithm terminates returning the merged knowledge base (line 11).

`ONTOLOGY_MERGING` ***(Algorithm 2)***

The `ONTOLOGY_MERGING` algorithm iteratively applies the `MERGE` procedure to all the ontologies in the $\mathtt{Ontologies}(P_i)$ set. However, one may have the doubt that changing the order of the ontologies in the input set we may obtain a different result. Section 6.2.2 addresses this issue offering some interesting considerations about the `MERGE` procedure and, more in general, about an ontology mapping process.

6.2.2 Order of the Ontologies in the Merging Algorithm

There is one question that may arise when considering the two proposed algorithms: if we change the order of the ontology in $\mathtt{Ontologies}(P_i)$ the result of Algorithm 2 will be always the same? In general the reply to such a question is “no”. However, in our case the probability of having different merging processes with a different order of the same ontologies is very low. This is related to the fact that we take into consideration just the mappings with a score greater of the threshold τ. We discuss the value that may by assigned to τ in Chapter

10, anyway a general consideration is the following: the meaning of τ is that if a mapping has a score greater than τ then the elements in such mapping element can be considered exactly the same. Moreover, if we have a mapping element $me_{1,2}$ between ontologies O_1 and O_2 that maps e_{O_1} and e_{O_2} with a score greater than a high value of τ, then when mapping O_1 and O_2 with O_3 the probability to have e_{O_1} and e_{O_2} mapped to the same element e_{O_3} is very high. This yield to the obvious consequence that mappings between ontologies are transitive: if e_{O_1} maps e_{O_2} with $s_{1,2} > \tau$ and e_{O_2} maps e_{O_3} with $s_{2,3} > \tau$ than we have also that with e_{O_1} maps e_{O_3} with $s_{1,3} > \tau$.

Algorithm 1: MERGE

Input:

O_i: The first ontology to be merged

O_j: the second ontology to be merged

Output:

$\tilde{O}_{i,j}$: The ontology resulted by merging the input ontologies

```
(1)  $\tilde{O}_{i,j} = \texttt{new}(Ontology());$
(2)  $mapping = \texttt{MAP}(O_i, O_j);$
(3)  FOR EACH $me_k \in mapping$
(4)      IF $me_k.s > \tau$
(5)          $O_j.\texttt{update}(me_k.e_{O_j}, me_k.e_{O_i});$
(6)          $\tilde{O}_{i,j}.\texttt{add}(me_k);$
(7)  FOR EACH $e_{O_i} \notin \tilde{O}_{i,j}$
(8)      $\tilde{O}_{i,j}.\texttt{add}(e_{O_i});$
(9)  FOR EACH $e_{O_j} \notin \tilde{O}_{i,j}$
(10)     $\tilde{O}_{i,j}.\texttt{add}(e_{O_j});$
(11) return $\tilde{O}_{i,j};$
```

Algorithm 2: `ONTOLOGY_MERGING`

Input:

$Ontologies(P_i)$: The ontologies exploited in the policy P_i

Output:

$\tilde{O}_{P_i}$: The ontology resulted by merging the ontology exploited in the policy P_i

```
(1)    $\tilde{O}_{P_i} = \texttt{new}\ Ontology();$
(2)    FOR EACH $O_j \in Ontologies(P_i)$
(3)        $\tilde{O}_{P_i} = \texttt{MERGE}(\tilde{O}_{P_i}, O_j);$
(4)    return $\tilde{O}_{P_i};$
```

6.3. Ontology Extraction Process

Ontology extraction is the process whereby it is possible to build an ontology from either structured or unstructured data with no explicit semantics relations. We have discussed the theoretical background of this problem in Section 4.2.2, in this section we provide an algorithm for extracting an ontology having in input the vocabulary of the policy and some additional semantic data.

6.3.1 From a Vocabulary to a Domain: the Model

The algorithm improves the mapping in Section 4.3.3 by refining the resulting ontology through a hierarchical organization of the new entities. The motivation for such refinement is that after the extraction of the entities based on the mapping between XACML and DL , we obtain a simple list of new properties and concepts. However, the extracted ontology may be involved in some ontology matching processes exploited in further steps. Since ontology matching takes advantage of both the entity names and their organization in the ontology, a knowledge base without a structure i.e. without hierarchies between entities, is useless for our purposes. For this reason, we combine the mapping in Section 4.2.2 with the subsumption relation between the terms that we retrieve from a lexical databases, such as WordNet [31].

In WordNet, terms are organized in hierarchies following a standard subsumption relation. We extract the list of concepts[1], or path, that bound the term under consideration to the root of the hierarchy. Then, given two terms t_i and t_j, we intersect WordNet paths $WNPath(t_i)$ and $WNPath(t_j)$ and we pick up the least common subsumer. Three different scenarios may arise:

1. the common least subsumer is t_j (t_j subsumes t_i): we add to the ontology a subclass relation between concept t_i and t_j;

2. the common least subsumer is t_i (t_i subsumes t_j): we add to the ontology a subclass relation between concept t_j and t_i;

3. the common least subsumer is the term t_k: if a concept t_k does not belong to the ontology, we add the new concept t_k along with the subclass relations between t_k and t_i and t_j, respectively;

4. no common subsumers are found: no relations are added to the ontology;

For example consider the policy shown in Figure 4.1. The least common subsumer between t_1 (`Associate Professor`), t_2 (`Full Professor`) and t_3 (`Student`) is the WordNet synset represented by the term `Person`. Moreover, by just considering together t_1 and t_2, we obtain as the least common subsumer the term `Professor`. The hierarchy returned by the refinement function will have `Person` as root. `Person` then subsumes `Student` and `Professor` that in turn subsumes the concepts `Full Professor` and `Associate Professor`.

[1]Named *synset* (set of synonyms) in WordNet.

6.3.2 Ontology Extraction Algorithm

For simplicity we show just the creation of object properties, the case for data type properties is straightforward because we just need to add the new property without creating any concept. In lines 2-5 we create a new property and a new concept given a certain attribute-value pair. In line 6 we update the range of the property with the new concept, finally in line 7 we return the $\ddot{O}_{P_i}$ enriched with the hierarchies added by the CREATE_HIERARCHY function.

Algorithm 3: ONTOLOGY_EXTRACTION

Input:

$Pairs(P_i)$: The attribute-value pairs of the policy P_i

Output:

$\ddot{O}_{P_i}$: The ontology extracted by the policy P_i

```
(1)  $\ddot{O}_{P_i} = \texttt{new}\ Ontology();$
(2)  FOR EACH $\langle attribute_j, value_k \rangle \in Pairs(P_i)$
(3)    IF $\ddot{O}_{P_i}.\texttt{not\_contains}(attribute_j)$
(4)      $\ddot{O}_{P_i}.\texttt{add}(\texttt{new}(ObjectProperty(attribute_j)));$
(5)    $\ddot{O}_{P_i}.\texttt{add}(\texttt{new}(Concept(value_k)));$
(6)    $\ddot{O}_{P_i}.attribute_j.\texttt{range} = \ddot{O}_{P_i}.value_k;$
(7)  return CREATE_HIERARCHY($\ddot{O}_{P_i}$);
```

6.4. Policy Reference Ontology

In this section we describe the process whereby we create the Reference Ontology of an input policy. The process, detailed in *Algorithm 4*, combine the procedures introduced above. The input of the process is the policy P_i whereas its output is the Reference Ontology $\dot{O}_{P_i}$. In lines 3-4 we create $\tilde{O}_{P_i}$ merging up all the ontologies exploited in P_i and retrieved by the function $Ontologies(P_i)$. In line we apply `ONTOLOGY_EXTRACTION` obtaining the reconciled Knowledge Base for those terms that does not belong to any ontology. It is worth noting that if all terms used in the policy belong to a semantic schema then $Vocabulary(P_i)$ is the empty set. Finally, we build $\dot{O}_{P_i}$ merging $\tilde{O}_{P_i}$ and $\ddot{O}_{P_i}$.

Algorithm 4: `POLICY_REFERENCE_ONTOLOGY`

Input:

P_i: The Policy

Output:

$\dot{O}_{P_i}$: The Reference Ontology Associated with P_i

```
(1)  $\tilde{O}_{P_i}$ = new $Ontology()$;
(2)  $\ddot{O}_{P_i}$ = new $Ontology()$;
(3)  IF $| Ontologies(P_i) | > 1$
(4)          $\tilde{O}_{P_i}$ = ONTOLOGY_MERGING($Ontologies(P_i)$);
(5)  $\ddot{O}_{P_i}$ = ONTOLOGY_EXTRACTION($Vocabulary(P_i)$, $\tilde{O}_{P_i}$);
(6)  $\dot{O}_{P_i}$ = MERGE($\tilde{O}_{P_i}$, $\ddot{O}_{P_i}$);
(7)  return $\dot{O}_{P_i}$;
```

6.5. Policy Set Reference Ontology

The creation of the Policy Set Reference Ontology is the final step in the conceptual architecture of our model. The procedure detailed in *Algorithm 5* takes advantage off all the functions defined in the above sections. The main role of this algorithm is thus to coordinate each activity calling the related functions.

Algorithm 5: `Policy Set Reference Ontology creation`

Input:

PS: Policy Set

Output:

$\dot{O}_{PS}$: The Unique Ontology Associated with PS

```
(1)  $\dot{O}_{PS}$ = new $Ontology()$;
(2)  FOR EACH $P_i \in PS$
(3)      $\dot{O}_{P_i} = \texttt{POLICY\_REFERENCE\_ONTOLOGY}(P_i)$;
(4)      $Ontologies.\texttt{add}(\dot{O}_{P_i})$;
(5)  FOR EACH $\dot{O}_j \in Ontologies$
(6)      $\dot{O}_{PS} = \texttt{MERGE}(\dot{O}_{PS}, \dot{O}_j)$;
(7)  return $\dot{O}_{PS}$;
```

Chapter 7

Policy Similarity Analysis

In this chapter we introduce the EXAM-S policy analysis services. In the definition of such services we rely on the preliminary section introduced in Section 4. We propose two different approaches: the core of this chapter (Section 7.5) in which we define an hybrid system that exploits a modified version of MTBDDs. The second one (Section 7.4) presents a similarity technique that can be exploited on large policy data sets for optimization purposes. Whereas in Section 7.5 the approach is very precise and both combines and extends state of the art approaches, the one proposed in Section 7.4 rely on data mining techniques and is used for computing quickly a similarity score between policies. The score is used in a filtering techniques for optimizing the data set maintaining only the policies that are more similar. In both Sections 7.4 and 7.5 we assume that policies involved into the analysis process have been already processed by the technologies introduced in Section 6. Thus, we assume that if the policies belong to heterogeneous domains, all the variations introduced in Section 4.1 have been already solved.

The rest of the sections, Section 7.2 and Section 7.1, introduce respectively the running example and preliminary definitions that will be used in the rest of the chapter.

7.1. Analysis Queries on Heterogeneous Policies

The notion of analysis query is a key notion in our approach. Because one can analyze policies and sets of policies from different perspectives, it is important to also devise a comprehensive categorization of such queries. In our work, we have thus identified three main categories of analysis queries, which differ with respect to the information that they query. These categories are: policy metadata queries, policy content queries, and policy effect queries. Policy metadata queries analyze metadata associated with policies, such as policy creation and revision dates, policy author, and policy location. A policy content query, by contrast, extracts and analyzes the actual policy content, such as the number of rules in the policy, the total number of attributes referenced in the policy, the presence of certain attribute values.

A policy effect query determines and analyzes the requests allowed or denied by policies and interactions among policies. The category of the policy effect queries is the most interesting one among the query categories we have identified. The processing of policy effect queries is also far more complex than the processing of queries in the other two categories, and thus we address its processing in details (see next section). The policy effect query category can be further divided into two subcategories: (i) queries on single policy; and (ii) queries on multiple policies. The first subcategory contains one type of query, referred to as property verification query. The second subcategory contains two main types of queries, namely common property query and discrimination query. Figure 3.2 provides a taxonomy summarizing the various query types.

In the following, we first introduce some preliminary notions, and then present more details for each type of policy effect query (query for short), including their definitions and functionalities.

7.2. An Illustrative Example

To illustrate the discussion we consider a scenario from a content delivery network (CDN) system built on P2P network in which parties can replicate their data in storage made available by third party resource providers. Real systems adopting this model are for instance Lockss [8] and LionShare [70]. In such scenario, there are usually two types of parties: data owner and resource owner. A data owner owns some data, whereas a resource owner manages some resources for storing data and processing queries on data. A data owner typically needs to determine which resource owners can be more suited for storing its content. Examples of such CDN systems can be found in Grid computing systems and P2P systems. Each such party in a CDN typically has its own access control policies. The policies of a data owner specify which users can access which data, among these owned by the data owner, under which conditions. The access control policies of the resource owners specify conditions for the use of the managed resources. In large dynamic environments, we cannot expect such policies to be integrated and harmonized beforehand, also because policies may dynamically change. Therefore, a subject wishing to run a query has to comply with both the access control policy associated with the queried data and the access control policy of the resource to be used to process the query. Because such parties may not have the same access control policies, in order to maximize the access to the data, it is important to store the data at the resource owner having access control policies similar to the access control policies associated with the data. Furthermore, besides determining the common parts of the access control policies shared by the data owner and resource owner, the data owner may also be interested in checking if certain key requests can be successfully handled, if the data were to be located at a given resource owner. In other words, the data owner may want to know if the difference among the multiple access control policies has a negative effect on some important tasks. We now introduce two example policies from the above scenario.

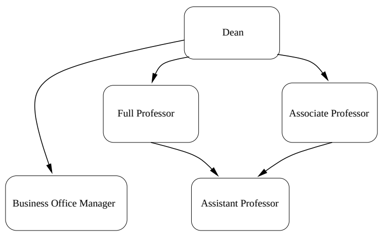


Figure 7.1: The Faculty role hierarchy.

Example 7.1 *Pol1 (Data Owner): Any user with role the is subsumed by "Full Professor" is authorized to access the data from **8am** to **10pm** everyday.*
*Pol2 (Resource Owner): Any user with role that subsumes "PhD Student" or affiliated with "Purdue University" is authorized to access the resource from **6am** to **8pm** everyday.*

With the attribute Role associated with the role hierarchy depicted in Figure 7.1. We assume that the Role attribute is associated to the same ontology. If this is not the case, the procedures introduced in Chapter 6 are applied.

In order for the data owner to decide whether to store the data at the resource owner, it is crucial to determine which kinds of requests will be permitted by both policies and which will not. Because in this case we are dealing with only two policies, the filtering phase is not required and the policies can be directly transmitted to the PSA. The PSA then returns the following characterization of the similarity for the input policies:

- When the role is between "Full Professor" and "PhD Student" and time is in the range of [8am, 8pm], such requests are permitted by Pol1 and Pol2.
- When the role is subsumed by "Full Professor" and time is in the range of (8pm, 10pm], such requests are permitted by Pol1, denied by Pol2.
- When affiliation is "Purdue University" and time is in the range of [6am, 8pm], such requests are denied by Pol1, permitted by Pol2.

- When (the role subsumes "PhD Student" or affiliation is "Purdue University"), and time is in the range of [6am, 8am), such requests are denied by Pol1, permitted by Pol2.

By using such characterization, the data owner can check if most requests (or some important requests) are satisfied and then decide whether to send his data to such resource owner. More specifically, if the data owner knows that a large percentage of requests are issued during the time interval [8am, 8pm], sending the data to this resource owner would be a good choice, as both policies yield same effect for the requests during that time period. If, instead, it is crucial that the data be also available at other times, the data owner may determine if there are other resource owners, whose policies are closer to its own, or use some data replication strategies to make sure that at any point of time there is at least one resource owner whose policies allow the data queries to be processed. The data owner may also investigate additional properties concerning the policies: for example the data owner may also issue queries like "when are the requests of users with a role that subsumes "PhD Student" permitted by both policies?". For such query the PSA will return the time interval in which both policies are satisfied.

7.3. Prelimary Notions

In our work, we assume the existence of a finite set A of names. Each attribute, characterizing a subject or a resource or an action or the environment, has a name a in A, and a domain, denoted by $\mathrm{dom}(a)$, of possible values. The following two definitions introduce the notion of access request and policy semantics.

Definition 7.1 (Access Request) *Let* $a_1, a_2, \ldots, a_k$ *be attribute names in policy* P*, and let* $v_i \in \mathrm{dom}(a_i)$ $(1 \leqslant i \leqslant k)$. $r \equiv \{(a_1, v_1), (a_2, v_2), \ldots, (a_k, v_k)\}$ *is a request, and* e_r^P *denotes the effect of this request against* P*.*

Definition 7.2 (Policy Semantics) *Let* P *be an access control policy. We define the semantics of* P *as a 2-tuple* $\{B_{permit}, B_{deny}\}$*, where* B_permit *and* B_deny *are Boolean expressions corresponding to permit and deny rules respectively.* B_permit *and* B_deny *are defined as follows.*

$$rs_{1i} = \begin{cases} B_{permit} = T_P \wedge ((T_{PR_1} \wedge C_{PR_1}) \vee \cdots \vee (T_{PR_k} \wedge C_{PR_k})) \\ \\ B_{deny} = T_P \wedge ((T_{DR_1} \wedge C_{DR_1}) \vee \cdots \vee (T_{DR_j} \wedge C_{DR_j})) \end{cases} \tag{7.1}$$

where, T_P *denotes a Boolean expression on the attributes of policy target;* T_{PR_i} *and* C_{PR_i} $(i = 1, \ldots, k)$ *denote the Boolean expressions on the attributes of the rule target and rule condition of permit rule* PR_i*; and* T_{DR_i} *and* C_{DR_i} $(i = 1, \ldots\ldots, j)$ *denote the Boolean expressions on the attributes of the rule target and rule condition of deny rule* DR_i.

Example 7.2 *Consider policy Pol2 in Example 7.1. An example of request to which this policy applies is that of a user affiliated with "Purdue University" wishing to access the data at 9am. According to Definition 7.1, such request can be expressed as* $r \equiv \{(affiliation, \text{"PurdueUniversity"}), (time, 9am)\}$.

Example 7.3 *According to Definition 7.2, policy Pol1 in Example 7.1 can be represented as follows.*

$$rs_{1i} = \begin{cases} B_{permit} = (f^{sub}(O_k, Role, \mathtt{FullProfessor})) \wedge (8am \leqslant time \leqslant 10pm) \\ \\ B_{deny} = NULL \end{cases}$$

The Boolean expressions (B, T and C) that frequently occur in policies can be broadly classified into the following five categories, as identified in [13] :

- Category 1: One variable equality constraints. $x = c$, where x is a variable and c is a constant.

- Category 2: One variable inequality constraints. $x \rhd c$, where x is a variable, c is a constant, and $\rhd \in \{<, \leqslant, >, \geqslant\}$.

- Category 3: Real valued linear constraints. $\sum_{i=1}^{n} a_i x_i \rhd c$, where x_i is variable, a_i, c_i are constants, and $\rhd \in \{<, \leqslant, >, \geqslant\}$.

- Category 4: Semantic constraints. $f^{sub}(O_k, C, D)$, where O_k is the ontology with respect to the subsumption relation is checked and C, D are concepts belonging to O_k;

- Category 5: Regular expression constraints. $s \in L(r)$ or $s \notin L(r)$, where s is a string variable, and $L(r)$ is the language generated by regular expression r.

- Category 6: Compound Boolean expression constraints.

This category includes constraints obtained by combining Boolean constraints belonging to the categories listed above. The combination operators are $\vee$, $\wedge$ and $\neg$. By using $\neg$, we can represent the inequality constraint $x \neq c$ as $\neg(x = c)$. It is worth noting that Boolean expressions on the attributes of policy targets or rule targets (T_P, T_{PR}) usually belong to Category 1. The domains of the attributes that appear in the above Boolean expressions belong to one of the following categories :

- Integer domain : The attribute domains in Boolean expressions of categories 1,2 and 6 can belong to this domain.

- Real domain : The attribute domains in Boolean expressions of categories 1,2,3 and 6 can belong to this domain.

- String domain : The attribute domains in Boolean expressions of categories 1,5 and 6 can belong to this domain.

- Semantic domain: The attribute domains in Boolean expressions of categories 4 belong to this domain.

- Tree domain : Each constant of a tree domain is a string, and for any constant in the tree domain, its parent is its prefix (suffix). The X.500 directories, Internet domain names and XML data are in the tree domain. For example, an Internet domain constant .edu is the parent of an Internet domain constant purdue.edu. The attribute domains in Boolean expression of categories 1 and 6 can belong to this domain.

We define a *query constraint* f_q as a Boolean function that represents constraints on a set of requests. Currently, our system supports two types of f_q functions and their combinations: (i) *true*, which means there is no constraint; (ii) $|R| \lhd x (\lhd \in \{<, \leqslant, =, \neq, >, \geqslant\})$, where $|R|$ is the number of requests and x is a constant. For example, $|R| > 0$ is a query constraint which checks if the corresponding query returns at least one request.

Note that in the following discussions on query definitions, the Boolean expression B_q may belong to any of the five categories mentioned above.

7.4. Policy Filtering

In this section we propose a similarity measure for XACML policies based on data mining techniques. The same approach have been applied for obtaining the similarity of policies written in another language: the Platform for Privacy Preferences (P3P). This approach is presented in Chapter 8. Concerning XACML, the proposed policy similarity measure is based on the comparison of each corresponding component of the policies being compared. Here, the corresponding component means the policy targets and the same type of elements belonging to the rules with the same effect. For this reason we are able to deal also with the semantic functions that we have introduced in Chapter 5. We use a simplified

XACML format for defining the policy similarity measure. Each XACML policy must be converted to this format when calculating the similarity score.

Table 7.1 gives the syntax of the simplified format[1]. We would like the policy similarity measure between any two given policies to assign a similarity score that **approximates** the relationship between the sets of requests permitted (denied) by the two policies. The similarity score is a value between 0 and 1, which reflects how similar these rules are with respect to the targets they are applicable to and also with respect to the conditions they impose on the requests. For example, in a scenario where a set of requests permitted (denied) by a policy P_1 is a subset of requests permitted (denied) by a policy P_2, the similarity score for policies P_1 and P_2 must be higher than the score assigned in a scenario in which the set of requests permitted (denied) by P_1 and P_3 have very few or no request in common.

7.4.1 Computation of the Mapping

In this subsection we proceed to introduce how to obtain the similarity score of two policies. Given two policies P_1 and P_2, the rules in these policies are first grouped according to their effects, which results in a set of Permit Rules (denoted as PR) and a set Deny Rules (denoted as DR). Each single rule in P_1 is then compared with a rule in P_2 that has the same effect, and a similarity score of two rules is obtained. The similarity score obtained between the rules is then used to find one-many mappings (denoted as Φ) for each rule in the two policies. For clarity, we choose to use four separate Φ mappings Φ_1^P, Φ_1^D, Φ_2^P and Φ_2^D. The mapping Φ_1^P (Φ_1^d) maps each PR(DR) rule r_{1i} in P_1 with one or more PR(DR) rules r_{2j} in P_2. Similarly the mapping Φ_2^P (Φ_2^D) maps each PR(DR) rule r_{2j} in P_2 with one or more PR(DR) rules r_{1i} in P_1. For each rule in a policy P1(P2), the Φ mappings give similar rules in P2(P1) which satisfy certain similarity threshold.

The computation of the Φ mapping will be addressed in the Section 7.4.2. By

[1]We have already presented this Table in Chapter 2

```
POLICY: <policy policy-id = "policy-id combining-algorithm = "combining-algorithm >
    (TARGET ELEMENT)?
    < permitrules >
        (RULE ELEMENT)*
    </permitrules>
    <denyrules>
        (RULE ELEMENT)*
    </permitrules>
</policy>

RULE ELEMENT:
<rule rule-id="rule-id effect="rule-effect>
    (TARGET ELEMENT)?
    <condition>PREDICATE</condition>
</rule>

TARGET ELEMENT:
    <target>
        <subject>PREDICATE</subject>
        <resource>PREDICATE</resource>
        <action>PREDICATE</action>
    </target>

PREDICATE:
(attr_name ⊕ (attr_value)+)*
attr_name denotes attribute name, attr_value denotes attribute value and
⊕ denotes any operator supported by the XACML standard.
```

Table 7.1: A XACML policy structure.

using the Φ mappings, we compute the similarity score between a rule and a policy. We aim to find out how similar a rule is with respect to the entire policy by comparing the single rule in one policy with a set of similar rules in the other policy. The notation $rs_{1i}(rs_{2j})$ denotes the similarity score for a rule $r_{1i}(r_{2j})$ in policy $P_1(P_2)$. The rule similarity score $rs_{1i}(rs_{2j})$ is the average of the similarity scores between a rule $r_{1i}(r_{2j})$ and the rules similar to it given by the Φ mapping. rs_{1i} and rs_{2j} are computed according to Equations 7.2 and 7.3, where S_{rule} is a function that assigns a similarity score between two rules.

Next, we compute the similarity score between the permit(deny) rule sets PR_1 (DR_1) and $PR_2(DR_2)$ of policies P_1 and P_2 respectively. We use the notations $S^P_{rule-set}$ and $S^D_{rule-set}$ to denote the similarity scores for permit and deny rule sets respectively. The similarity score for a permit(deny) rule set is obtained by averaging the rule similarity scores (Equations 7.2 and 7.3) for all rules in the set. The permit and deny rule set similarity scores are formulated by Equation 7.4 and 7.5, where N_{PR_1} and N_{PR_2} are the numbers of rules in PR_1 and PR_2 respectively, N_{DR_1} and N_{DR_2} are the numbers of rules in DR_1 and DR_2 respectively.

$$rs_{1i} = \begin{cases} \dfrac{\sum\limits_{r_j \in \Phi_1^P(r_{1i})} S_{rule}(r_{1i}, r_j)}{\mid \Phi_1^P(r_{1i}) \mid}, r_{1i} \in PR_1 \\ \dfrac{\sum\limits_{r_j \in \Phi_1^P(r_{1i})} S_{rule}(r_{1i}, r_j)}{\mid \Phi_1^D(r_{1i}) \mid}, r_{1i} \in DR_1 \end{cases} \tag{7.2}$$

$$rs_{2j} = \begin{cases} \dfrac{\sum\limits_{r_i \in \Phi_2^P(r_{1j})} S_{rule}(r_{2j}, r_i)}{\mid \Phi_2^P(r_{2j}) \mid}, r_{2j} \in PR_2 \\ \dfrac{\sum\limits_{r_i \in \Phi_2^D(r_{2j})} S_{rule}(r_{2j}, r_i)}{\mid \Phi_2^D(r_{2j}) \mid}, r_{2j} \in DR_2 \end{cases} \tag{7.3}$$

$$S^P_{rule-set} = \frac{\sum\limits_{i=1}^{N_{PR_1}} rs_{1i} + \sum\limits_{i=1}^{N_{PR_2}} rs_{2j}}{N_{PR_1} + N_{PR_2}} \tag{7.4}$$

$$S^{P}_{rule-set} = \frac{\sum_{i=1}^{N_{PR_1}} rs1i + \sum_{i=1}^{NPR_2} rs2j}{N_{PR_1} + N_{PR_2}} \tag{7.5}$$

Finally, we combine the similarity scores for permit and deny rule sets between the two policies along with a similarity score between the $Target$ elements of the two policies, to develop an overall similarity score, S_{policy}. The formulation of S_{policy} is given by the following equation:

$$S_{policy}(P_1, P_2) = w_T S_T(P_1, P_2) + w_p S^{P}_{rule-set} + w_d S^{D}_{rule-set} \tag{7.6}$$

where S_T is a function that computes a similarity score between the $Target$ elements of any two given policies; w_p and w_d are weights that can be chosen to reflect the relative importance to be given to the similarity of permit and deny rule sets respectively. For normalization purpose, the weight values should satisfy the constraint: $w_T + w_p + w_d = 1$. The intuition behind the similarity score assigned to any two policies is derived from the fact that two policies are similar to one another when the corresponding policy elements are similar. In the following sections, we introduce the detailed algorithms for the computation of Φ mappings and rule similarity score S_{rule}. Table 7.4.1 lists main notations used throughout this section.

7.4.2 Computation of Φ Mappings

In this section, we discuss the procedure for determining the Φ mappings for each rule in the permit and deny rule sets in a policy. The one-many Φ mappings determine for each $PR(DR)$ rule in $P_1(P_2)$ which $PR(DR)$ rules in $P_2(P_1)$ are very similar. Intuitively, two rules are similar when their targets and the conditions they specify are similar. Thus we define a Φ mapping as follows:

$$\Phi(r_i) = \{r_j \mid S_{rule}(r_i, r_j) \geqslant \epsilon\} \tag{7.7}$$

Notation	**Meaning**
P	Policy
PR	Permit rule set
DR	Deny rule set
r	Rule
a	Attribute
v	Attribute value
H	Height of a hierarchy
S_{policy}	Similarity score of two policies
S_{rule}	Similarity score of two rules
$S^{P}_{rule-set}$	Similarity score of two sets of permit rules
$S^{D}_{rule-set}$	Similarity score of two sets of deny rules
$S_{\langle Element\rangle}$	Similarity score of elements, $\langle Element\rangle \in \{'T','t','c','s','r','a'\}$
s_{cat}	Similarity score of two categorical values
S_{cat}	Similarity score of two categorical predicates
s_{num}	Similarity score of two numerical values
S_{num}	Similarity score of two numerical predicates
r_s	Similarity score between a rule and a policy
Φ	Rule mapping
$\mathcal{M}_a$	Set of pairs of matching attribute names
$\mathcal{M}_v$	Set of pairs of matching attribute values
N_{PR}	Number of permit rules in a policy
N_{DR}	Number of deny rules in a policy
N_a	Number of attributes in an element
N_v	Number of values of an attribute
$SPath$	Length of shortest path of two categorical values
$w_{\langle Elementi\rangle}$	Weight of similarity scores of elements, $\langle Elementi\rangle \in \{'T','t','c','s','r','a'\}$
ϵ	Rule similarity threshold
δ	Compensating score for unmatched values

Table 7.2: Notations.

where S_{rule} is computed by Equation 7.8 and ϵ is a threshold. The threshold term is important here, since it allows us to calibrate the quality of the similarity approximation. We expect that the actual value of the threshold will be very specific to the policy domain. This procedure takes two rule sets R' and R'' as input and computes a mapping for each rule in R' based on Equation 7.7.

7.4.3 Similarity Score between Rules

Since our similarity measure serves as a lightweight filter phase, we do not want to involve complicated analysis of Boolean expressions. Our similarity measure is developed based on the intuition that rules r_i and r_j are similar when both apply to similar targets and both specify similar conditions on request attributes. Specifically, we compute the rule similarity function S_{rule} between two rules r_i and r_j as follows:

$$S_{rule}(r_i, r_j) = w_t S_t(r_i, r_j) + w_c S_c(r_i, r_j) \tag{7.8}$$

w_t and w_c are weights that can be used for emphasizing the importance of the target or condition similarity respectively. For example, if users are more interested in finding policies applied to similar targets, they can increase wt to achieve this goal. The weights satisfy the constraint $w_t + w_c = 1$. S_t and S_c are functions that compute a similarity score between two rules based on the comparison of their Target and Condition elements respectively. As the Target element in each rule contains the Subject, Resource and Action elements, each of these elements in turn contains predicates on the respective category of attributes. Thus, the Target similarity function St is computed as follows:

$$S_t(r_i, r_j) = w_s S_s(r_i, r_j) + w_r S_r(r_i, r_j) + w_a S_a(r_i, r_j) \tag{7.9}$$

In Equation 7.9, w_s, w_r, w_a represent weights that are assigned to the corresponding similarity scores. Like in the previous equations, weight values need to satisfy the constraint $w_s + w_r + w_a = 1$. Ss, Sr and Sa are functions that return a similarity score based on the Subject, Resource and Action attribute predicates respectively in the Target elements of the two given rules. The computation of functions S_c, S_s, S_r and S_a involves the comparison of pairs of predicates in the given pair of rule elements, which we discuss in detail in the next subsection.

7.4.4 Similarity Score of Rule Elements

Each of the rule elements Subject, Resource, Action and Condition is represented as a set of predicates in the form of $\{attr\_name_1 \oplus_1 attr\_value_1, attr\_name_2 \oplus_2 attr\_value_2, \ldots\}$, where $attr\_name$ denotes the attribute name, $\oplus$ denotes a comparison operator and $attr\_value$ represents an attribute value. Based on the type of attribute values, predicates are divided into two categories, namely categorical predicate and numerical predicate.

- Categorical predicate: The attribute values of this type of predicate belong to the string data type and semantic functions. In case of String data type such values may or may not be associated with a domain specific ontology. Attribute values that belong to semantic functions are always associated with an ontology. If values are associated with more than one ontology, the techniques introduced in Chpater 6 are applied. Example of predicates that belong to categorical predicates are "FileType = Documentation" and "$f^{sub}(O_k, Role, FullProfessor)$".

- Numerical predicate: The attribute values of this type of predicate belong to integer, real, or date/time data types. For example, predicates "FileSize $<$ 10MB", "Time=12:00" are of numerical type.

The similarity score between two rules r_i and r_j regarding the same element is denoted as $S_{\langle Element \rangle}$, where $\langle Element \rangle$ refers to condition, subject, resource or action. The $\langle Element \rangle$ is computed by comparing the corresponding predicate sets in two rules. There are three steps. First, we cluster the predicates for each rule element according to the attribute names. It is worth noting that one attribute name may be associated with multiple values. Second, we find the predicates in the two rules whose attribute names match exactly and then proceed to compute a similarity score for their attribute values. The way we compute similarity score between attribute values differs, depending on whether the attribute value is of categorical type or numerical type (details about the computation are covered in the following subsection). Finally, we summarize the scores of each pair of matching predicates and obtain the similarity score of the rule element. Since not all attributes in one rule can find a matching in the other, we include a penalty for this case by dividing the sum of similarity scores of matching pairs by the maximum number of attributes in a rule. In addition, there is a special case when the element set is empty in one rule, which means no constraint exists for this element. For this case, we consider the similarity of the elements of the two rules to be 0.5 due to the consideration that one rule is a restriction of the other and the 0.5 is the estimation of the average similarity. The formal definition of $S_{\langle Element \rangle}$ is given by Equation 7.10.

$$S_{\langle Element \rangle} = \begin{cases} \dfrac{\sum\limits_{(a_{1k}, a_{2l}) \in \mathcal{M}_a} S_{\langle attr\_typ \rangle}(a_{1k}, a_{2l})}{max(N_{a1}, N_{a2})}, & N_{a1} > 0 \text{ and } n_{a2} > 0; \\ 1, & \text{otherwise.} \end{cases} \tag{7.10}$$

In Equation 7.10, $\mathcal{M}_a$ is a set of pairs of matching predicates with the same attribute names; a_{1k} and a_{2l} are attributes of rules r_{1i} and r_{2j} respectively; $\langle attr\_typ \rangle$ is the similarity score of attribute values of the type $attr\_typ$; and N_{a1} and N_{a2} are the numbers of distinct predicates in the two rules respectively. In addition, the computation of the similarity score of two policy targets S_T is the same as that for the rule targets i.e. S_t.

7.4.5 Similarity Score for Categorical Predicates

For the categorical values, we not only consider the exact match of two values, but also consider their semantic similarity. For example, policy P_1 is talking about

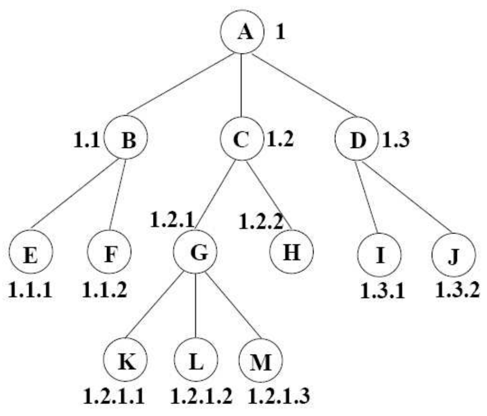


Figure 7.2: An Example Hierarchy

the priority of professors, policy P_2 is talking about faculty members, and policy P_3 is talking about business staff. In some sense, policy P_1 is more similar to policy P_2 than to policy P_3 because “professors” is a subset of ”faculty members” which means that policy P_1 could be a restriction of policy P_2. Based on this observation, our approach assumes that a hierarchy relationship exists for the categorical values. The similarity between two categorical values (denoted as S_{cat}) is then defined according to the shortest path of these two values in the hierarchy. The formal definition is shown below:

$$S_{cat}(v_1, v_2) = 1 - \frac{SPath(v_1, v_2)}{2H} \tag{7.11}$$

where $SPath(v_1, v_2)$ denotes the length of the shortest path between two values v_1 and v_2, and H is the height of the hierarchy. In Equation 7.11, the length of the shortest path of two values is normalized by the possible maximum path length which is 2_H. The closer the two values are located in the hierarchy, the more similar the two values will be, and hence a higher similarity score scat will be obtained. Figure 4.1 gives an example hierarchy, where each node represents a categorical value.

The height of the hierarchy is 3, and the length of maximum path of two values is estimated as 2 3 = 6 (the actual maximum path in the figure is 5 due to the imbalance of the hierarchy). $\mathsf{SPath}(E, B)$ is 1 and $\mathsf{SPath}(E, F)$ is 2. According to Equation 7.11, the similarity score of nodes E and B is 1-1/6 = 0.83, and the similarity score of nodes E and F is $1 - 2/6 = 0.67$. From the obtained scores, we can observe that E is more similar to B than to F. The underlying idea is that the parent-child relationship (B and E) implies that one rule could be a restriction of the other and this would be more helpful than the sibling relationship (E and F). To avoid repeatedly searching the hierarchy tree for the same value during the shortest path computation, we assign to each node a *hierarchy code* (Hcode), indicating the position of each node. In particular, the root node is assigned an Hcode equal to 1, and its children nodes are named in the order from left to right by appending their position to the parents Hcode with a separator ., where we will have Hcodes like 1.1 and 1.2. Then the process continues till the leaf level. The number of elements separated by . is equal to the level at which a node is located.

From such Hcodes we can easily compute the length of shortest path between two nodes. We compare two Hcodes element by element until we reach the end of one Hcode or there is a difference. The common elements correspond to the same parent nodes they share, and the number of different elements correspond to the levels that they need to be generalized to their common parent node. Therefore, the shortest path is the total number of different elements in two Hcodes. For example, the length of the shortest path from node 1.1 to 1.2 is 2, as there are two different elements in the Hcodes. Note that our definition of s_{cat} can also be applied to categorical values which do not lie in a hierarchy. In that case, if two values are matched, their shortest path SPath is 0 and their similarity score will be 1; otherwise, SPath is infinity and their similarity score becomes 0. Having introduced our approach to compare two single values, we now extend the discussion to two sets of values. Suppose there are two attributes $a_1 : \{v_{11}, v_{12}, v_{13}, v_{14}\}$ and $a_2 : \{v_{21}, v_{22}, v_{23}\}$, where a_1 and a_2 are the attribute names belonging to policy P_1 and P_2 respectively, and the values in the brackets are corresponding attribute values. Note that the values associated with the same attribute are different from one another. The similarity score of the two attribute value sets is the sum of similarity scores of pairs $\langle v_{1k}, v_{2l}\rangle$ and a compensating score δ (for non-matching attribute values). Obviously, there could be many combinations of pairs. Our task is to find a set of pairs (denoted as $\mathcal{M}_v$) which have the following properties:

1. If $v_{1k} = v_{2l}$, then $(v_{1k}, v_{2l}) \in \mathcal{M}_v$.
2. For pairs $v_{1k} \neq v_{2l}$, pairs contributing to the maximum sum of similarity scores belong to $\mathcal{M}_v$.
3. Each attribute value v_{1k} or v_{2l} occurs at most once in $\mathcal{M}_v$.

The process of finding the pair set $\mathcal{M}_v$ is the following. First, we obtain the hierarchy code for each attribute value. Then we compute the similarity between pairs of attribute values with the help of the hierarchy code. Next, we pick up exactly matched pairs, which are $\langle v_{11}, v_{21}\rangle$ and $\langle v_{14}, v_{23}\rangle$ in the example. For the remaining attribute values, we find pairs that maximize the sum of similarity scores of pairs. In this example, $\langle v_{12}, v_{22}\rangle$ has the same similarity score as $\langle v_{13}, v_{22}\rangle$, and hence we need to further consider which choice can lead to a bigger compensating score.

The compensating score δ is for attribute values which do not have matchings when two attributes have different number of values. δ is computed as average similarity scores between unmatched values with all the values of the other attribute. For this example, no matter which pair we choose, the compensating score is the same. Suppose we choose the pair $\langle v_{12}, v_{22}\rangle$, and then one value v_{13} is left whose compensating score δ is (0.33+0.67+0.17)/3 = 0.39. Finally, the similarity score for the two attribute a_1 and a_2 takes into account both the similarity of attribute names and attribute values. Specifically, the similarity score for attribute names is 1 as the exact matching of names is used. The similarity score for attribute values is the average scores of pairs and the compensating score. The final score is $\frac{1}{2}[1 + (1 + 1 + 0.67 + 0.39)/4] = 0.88$. The similarity score of two categorical predicates is finally defined as below:

$$S_{cat}(a_1, a_2) = \frac{1}{2}\left[1 + \frac{\sum_{(v_{1k}, v_{2l}) \in \mathcal{M}_v} s_{cat}(v_{1k}, v_{2l}) + \delta}{\max(N_{v1}, N_{v2})}\right] \tag{7.12}$$

$$\delta = \begin{cases} \dfrac{\sum\limits_{(v_{1k}, -) \notin \mathcal{M}_v} \sum\limits_{l=1}^{N_{v2}} s_{cat}(v_{1k}, v_{2l})}{N_{v2}}, & N_{v1} > N_{v2} \\ \dfrac{\sum\limits_{-, (v_{2l}) \notin \mathcal{M}_v} \sum\limits_{k=1}^{N_{v1}} s_{cat}(v_{1k}, v_{2l})}{N_{v1}}, & N_{v2} > N_{v1} \end{cases} \tag{7.13}$$

where N_{v1} and N_{v2} are the total numbers of values associated with attributes a_1 and a_2 respectively.

7.4.6 Similarity Score for Numerical Predicates

Unlike categorical values, numerical values do not have any hierarchical relationship. For computation efficiency, the similarity of two numerical values v_1 and v_2 is defined based on their difference as shown in Equation 7.14.

$$s_{num}(v_1, v_2) = 1 - \frac{| v_1 - v_2 |}{\text{range}(v_1, v_2)} \tag{7.14}$$

s_{num} tends to be large when the difference between two values is small. The computation of the similarity score of two numerical value sets is similar to that for two categorical value sets; we thus have the following similarity definition for numerical predicates:

$$S_{num}(a_1, a_2) = \frac{1}{2}\left[1 + \frac{\sum_{(v_{1k}, v_{2l}) \in \mathcal{M}_v} s_{cat}(v_{1k}, v_{2l}) + \delta}{\max(N_{v1}, N_{v2})}\right] \tag{7.15}$$

$$\delta = \begin{cases} \dfrac{\sum\limits_{(v_{1k},-)\notin \mathcal{M}_v} \sum\limits_{l=1}^{N_{v2}} s_{num}(v_{1k}, v_{2l})}{N_{v2}}, & N_{v1} > N_{v2} \\ \dfrac{\sum\limits_{-,(v_{2l})\notin \mathcal{M}_v} \sum\limits_{k=1}^{N_{v1}} s_{num}(v_{1k}, v_{2l})}{N_{v1}}, & N_{v2} > N_{v1} \end{cases} \tag{7.16}$$

7.5. Policy Similarity Analyzer

In this Section we present the core analysis services provided by EXAM-S. In this section we exploit a modified version of MTBDDs in which we allow to label nodes with Description Logic predicates. This allow us to analyze a large variety of policies: (i) policies with standard functions, (ii) policies with semantic functions, and (iii) policies with both standard and semantic functions. The section is organized as follows. Section 7.5.1 presents our query processing Strategy. Subsections 7.5.1 and 7.5.1 discusses respectively the policy preprocessor and the ratification module. Subsection 7.5.1 describe the approach based on MTBDDs whereas in Subsection 7.5.2 we introduce how to execute queries on the model we have defined.

7.5.1 Query Processing Strategy

For the purposes of this Section, Policies are nothing but Boolean formulae (or constraints) on attributes. The problem of analyzing policies is then translated into the problem of analyzing Boolean formulae. The main task of the PSA module is to determine all variable assignments that can satisfy the Boolean formulae corresponding to one or more policies, and also variable assignments that lead to different decisions for different policies. The Policy Similarity Analyzer module (PSA) uses the ratification module to preprocess the Boolean formulae, and then constructs a MTBDD for each policy and a CMTBDD (change-analysis MTBDD) for each pair of policies to be compared. In order to deal with the semantic function introduced in Chapter 5 we create an MTBDD which nodes labeled with the ontology predicates involved int the policies. Queries on a single policy are carried out on the MTBDD of the policy being queried, whereas queries on multiple policies are carried out on the CMTBDD of corresponding policies.

The technique used for queries on a single policy is a special case of the technique used for queries on multiple policies; thus we only describe the most general technique. It is worth noting that our query processing algorithm applies to all types of queries on multiple policies. Such algorithm includes two main phases. The first phase is the construction of the MTBDD and CMTBDD data structures. Because the MTBDDs and CMTBDDs can be reused for different queries, thus, once these structures are generated, they are cached in the policy repository. When such structures are already in the cache, the first phase of the algorithm reduces to fetching the structures from the cache. The second phase is related to specific queries, and in turn consists of three steps. The first step preprocesses the query, the second step constructs the query MTBDD and performs model checking, and the final step performs some postprocessing. We describe the details of each phase in the following subsection.

Policy Prepocessor

Given a set of input policies, the first step is to translate these policies expressed in XACML into Boolean expressions. Another task of the policy preprocessor is to identify the type of Boolean expressions for all variables in the policies. There are two steps. First, Boolean expressions of the same variables in all policies are clustered. Second, in each cluster, the type of the Boolean expressions is checked. In particular, for a variable x, if the Boolean expressions containing x all belong to category 1, this cluster of Boolean expressions will be labeled category 1; if the Boolean expressions of x belong to either category 1 or 2, the cluster will be labeled category 2; if there is at least one Boolean expression of x belongs to category 3 or 5, the cluster will be labeled category 3 or 5 respectively. If a variable appears in a semantic function then it is labeled with category 4. It is worth noting that, since a semantic function parameter is an entity that belong to an ontology, is not possible to have a variable that belong to category 4 and to the other categories at the same time. Note that we do not need to take special care of Boolean expressions of category 6 since they are just combinations of previous types of Boolean expressions and such combinations are naturally reflected by the MTBDD structure. The labeled Boolean expressions are finally sent to the ratification module for further processing.

Ratification Module

The ratification module has two main tasks. First, it needs to generate unified nodes for all the policies. Second, it needs to generate auxiliary rules for additional constraints introduced by the node unification and domain check. The unified nodes and auxiliary rules will later be consumed by the MTBDD module. We now proceed to discuss how the ratification module handles various types of Boolean expressions. The Boolean expressions of category 1 can be directly treated as nodes of the form $N(f(x))$ by the MTBDD module, where N is the name of the node and $f(x)$ is the Boolean expression of x. To generate unified nodes for the Boolean expressions of category 2, i.e. one variable inequality constraints, we need to first find the disjoint domain ranges of the same variable occurring in different policies. Assume that the original domains of a variable x are $[d_1^-, d_1^+]$, $[d_2^-, d_2^+]$, ... , $[d_n^-, d_n^+]$, where the superscript '-' and '+' denote lower and upper bound respectively, d_i^- can be $-\infty$, and d_i^+ can be $+\infty$ can be $(1 \leqslant i \leqslant n)$. We sort the domain bounds in an ascending order, and then employ a plane sweeping technique which scans the sorted domain bounds from left to right and keeps the ranges of two neighbor bounds if the ranges are covered in the original domain. The obtained disjoint ranges: $[d_1'^-, d_1'^+]$, $[d_2'^-, d_2'^+]$, ... , $[d_m'^-, d_m'^+]$ satisfy the following three conditions.

1. $d_i^-, d_i^+ \in D, D = \{d_1^-, d_1^+, \dots d_n^-, d_n^+\}$.

2. $\bigcup_{i=1}^{m}[d_i'^-, d_i'^+] = \bigcup_{j=1}^{n}[d_j^-, d_j^+]$.

3. $\bigcap_{i=1}^{m}[d_i'^-, d_i'^+] = \emptyset$.

It is easy to prove that m is at most $4n-2$. After having obtained disjoint domain ranges, all related Boolean functions are rewritten by using new domain ranges. Specifically, an original Boolean function $d_j'^- \lhd x \lhd d_j^+ \quad (1 \leqslant j \leqslant n, \lhd \in \{<, \leqslant\})$ is reformatted as $\bigvee_{i=1}^{k} \ (d_i'^- \lhd x \lhd d_i^+)$, where $\bigcup_{i=1}^{m}[d_i'^-, d_i'^+] = \bigcup_{j=1}^{n}[d_j^-, d_j^+]$. Then, the ratification module generates unified nodes in the form of $N(f(x))$, where $f(x)$ is an inequality function in the form of $d_i'^- \lhd x \lhd d_i^+$.

Example

Pol3: $(x_1 < 10 \wedge x_1 + x_2 < 20)$, Pol4: $(x_2 < 10 \vee x_1 + x_2 > 10)$. We can see that there are two conjunctions containing linear constraints:
$(x_1 < 10 \wedge x_2 < 10 \vee x_1 + x_2 < 20)$
$(x_1 < 10 \wedge x_1 + x_2 < 20 \vee x_1 + x_2 > 10)$.

We adopt a similar approach for the ratification of Boolean expression belonging to category 4, i.e. semantic functions. In this case we need to find disjoint ranges of classes associated to the same attribute name. Given a variable x we create the set $[c_1^-, c_1^+]$, $[c_2^-, c_2^+]$ where the superscripts assume the same meaning as defined above. Each $[c_i^-, c_i^+]$ represents classes in the ontology specified in the semantic function. Lower and Upper bounds are respectively $\texttt{owl}:\texttt{Nothing}$ and $\texttt{owl}:\texttt{Thing}$. Then we calculate disjoint ranges applying the same algorithm proposed for category 2.

Example

Consider as example the following semantic functions:

1. $subConceptOf(Attr - id, D)$

2. $superConceptOf(Attr - id, D')$

Where D and D' are organized in the ontology O_k as follows: $D \sqsubseteq D'$. After the ratification of the variable C we obtain the set of ranges:

$$[\texttt{owl:Nothing}, D], \quad [D, D'], \quad [D', \texttt{owl:Thing}]$$

Next, we introduce the processing of Boolean functions of category 3, i.e. real value linear constraints. Given a linear constraint in the form of $f(x_1, \ldots, x_k)$, we need to consider it together with other constraints containing variables $x_1, \ldots, x_k$ in all policies. Consider, for example, the following two Boolean expressions of two policies. The ratification module will check if each conjunction can be satisfied. Sometimes linear constraints can be reduced to and processed in the same way as Boolean expressions of category 1 or 2, such as the first conjunction which can be reduced to $(x_1 < 10 \wedge x_2 < 10)$. For other cases, the ratification module generates a node for each linear constraint and an auxiliary rule to indicate whether there is a solution. If there is a solution, the effect of the rule will be *conditional permit*, where *conditional* means the function is satisfied in certain cases. Otherwise, the effect of the rule will be *not applicable*. Note that the number of such conjunctions containing linear constraints is usually very small in real policies, though the disjunctive form of a formula may become exponentially larger than the original formula. For the Boolean function of category 5, i.e., regular expression constraints, finite automata techniques are used to determine satisfiability [41]. An auxiliary rule is then generated to indicate the satisfiability, in the similar way as that for the Boolean function of category 3.

Finally, we will introduce how to construct auxiliary rules for the domain constraint. For a variable x in the integer, real or string domain, an auxiliary rule is generated to indicate that each time only one node of x can be assigned the value true. In other words, this rule tells the MTBDD module that each variable can only have one value or belong to one disjoint range during each round of the assessment. An example is shown in Figure 7.5.1. For a variable x in the tree domain, we collect all its values appearing in the policies. For values along the same path in the tree, an auxiliary rule is needed to guarantee that if a variable cannot be assigned a certain value, then none of its children value can be satisfied. For example, suppose there are two constraints, *domain = .edu* and *domain = purdue.edu*. The auxiliary rule will state that if the node of *domain=.edu* is *false*, the node of *domain=purdue.edu* should also be *false*.

MTBDD Module

The MTBDD represents policies as rooted, directed acyclic graphs whose internal nodes represent Boolean predicates on policy attributes and whose terminals denote policy decisions, i.e. *Permit*, *Deny* and *Not Applicable*. MTBDDs are the same as BDDs (Binary Decision Diagrams) except that they can have more than two types of terminal nodes as compared to BDDs which have only 0 or 1 terminals. While in the worst case the number of nodes in an MTBDD is exponential in the number of variables, in practice the number of nodes is often polynomial or even linear [35]. So far, we have obtained all inputs for the MTBDD module:

1. unified nodes;
2. reformatted Boolean expressions;
3. auxiliary rules;

We proceed now to present the construction procedure used by the MTBDD module. First, the MTBDD module constructs the MTBDD for each policy according to the reformatted Boolean expression. Then, the MTBDD module combines the MTBDDs of policies to be compared and constructs the corresponding initial CMTBDD. Note that our system currently only support CMTBDD constructed from two policies. For multiple policies, we need to construct CMTBDD for each pair of policies to be compared and then aggregate the analysis results. Next, the auxiliary rules are applied to the initial CMTBDD. When the effect of the rule is permit, the terminal function follows the original CMTBDD. When the effect of the rule is conditional permit, the terminal function changes its original decision to conditional decision, e.g., permit will be changed to conditional permit. When the effect of the rule is not applicable, the corresponding terminal function changes to not applicable. Since each MTBDD has five terminals, i.e., Permit, Deny, *ConditionalPermit*, *ConditionalDeny*, *NotApplicable*, a CMTBDD has twenty-five terminals, one for each ordered pair of results from the policies being compared (such as Permit-to-Permit, Permit-to-deny). Here, we can see that our CMTBDD has two more types of terminals indicating conditional permit(CP) and conditional deny(CD) than that in [35]. The CMTBDD contains comparison results of the associated policies which is then used for various types of queries. Algorithm 6 summarizes the construction procedure followed by the PSA module.

Algorithm 6: `MTBDD_CMTBDD_Construction`

Input:

P_i: P_i is a policy and $1 \leqslant i \leqslant n$

Output:

$\mathsf{CMTBDD}(P_i)$: The CMTBDD associated with policy P_i

```
(1)  BF = Translate_Policy_To_Boolean_Formulae($P_1, P_2, \cdots, P_n$);
(2)  FOR EACH variable $bf_i \in BF$
(3)     $[f_1(x), \cdots, f_n(x)] \leftarrow$ atomic Boolean expressions with $x$;
(4)     IF every $f_i(x) (1 \leqslant i \leqslant n)$ belongs to category 1
(5)             construct node $N(f_i(x))$;
(6)     ELSE
(7)     IF $f_i(x) (1 \leqslant i \leqslant n)$ belongs to category 2
(8)        compute disjoint domains of $x$;
(9)        convert every $f_i(x)$ to $f_i'(x)$ by using new domains;
(10)       construct node $N(f_i'(x))$;
(11)       construct an auxiliary rule;
(12)    ELSE
(13)    IF $f_i(x) (1 \leqslant i \leqslant n)$ belongs to category 4
(14)       compute disjoint domains of $x$;
(15)       convert every $f_i(x)$ to $f_i'(x)$ by using new domains;
(16)       construct node $N(f_i'(x))$;
(17)       construct an auxiliary rule;
(18)    ELSE
(19)    IF $f_i(x) (1 \leqslant i \leqslant n)$ belongs to category 3, 5
(20)       construct an auxiliary rule;
(21)       construct an MTBDD for each policy;
(22)       construct an MTBDD for each auxiliary rule;
(23)       CMTBDD = combine($MTBDD_i$);
(24)       combine the CMTBDD with auxiliary rules;
(25) return MTBDD_CMTBDD.
```

To illustrate the above steps, let us consider again the Example 1 but this time from the system's perspective. Policy Pol_1 and Pol_2 are first translated into Boolean formulae. There are three variables occurring in these policies, namely *domain*, *time* and *affiliation*.

$$\mathrm{Pol}_1 = (f^{sub}(\mathrm{Role}, \mathrm{FullProfessor})) \wedge (8 \leqslant \mathrm{time} \leqslant 22)$$
$$\mathrm{Pol}_2 = (f^{sup}(\mathrm{Role}, \mathrm{PhDStudent}) \vee \mathrm{affiliation} = \text{“PurdueUniversity”}) \wedge (6 \leqslant \mathrm{time} \leqslant 20)$$

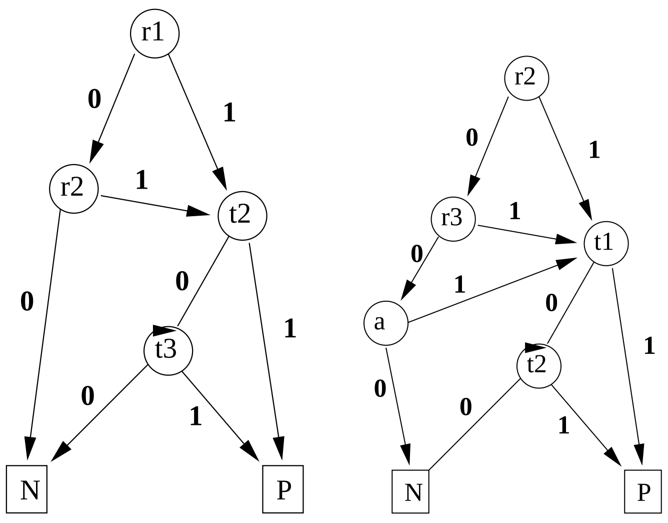
For the variable *domain* and *affiliation*, whose Boolean expressions belong to the first category, the preprocessor generate the node $a(\mathrm{affiliation} = \text{“PurdueUniversity”})$, and sends the node to the MTBDD module. For the Boolean formulae of variable time and role which belong to categories 2 and 4 respectively, the preprocessor sends them to the ratification module. The ratification module computes the disjoint range of the variable time and obtain three nodes: $t_1(6 \leqslant \mathrm{time} < 8)$, $t_2(8 \leqslant \mathrm{time} \leqslant 20)$, $t_3(20 < \mathrm{time} \leqslant 22)$. For the variable Role the following nodes are obtained:

$$r_1(f^{sup}(\mathrm{Role}, \mathrm{owl:Nothing}), f^{sub}(\mathrm{Role}, \mathrm{PhDStudent}))$$
$$r_2(f^{sup}(\mathrm{Role}, \mathrm{PhDStudent}), f^{sub}(\mathrm{Role}, \mathrm{FullProfessor}))$$
$$r_3(f^{sup}(\mathrm{Role}, \mathrm{FullProfessor}), f^{sub}(\mathrm{Role}, \mathrm{owl:Thing}))$$

Correspondingly, Pol_1 and Pol_2 are rewritten as:

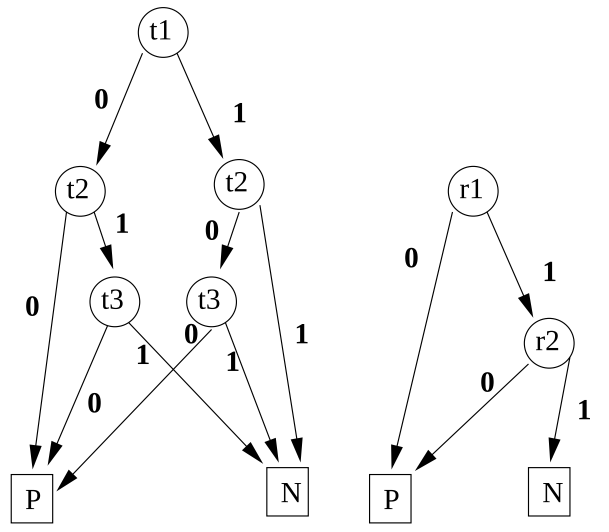
$$\mathrm{Pol}_1 = (r_1 \vee r_2) \wedge (t_2 \vee t_3)$$
$$\mathrm{Pol}_2 = (r_2 \vee r_3 \vee a) \wedge (t_1 \vee t_3)$$

By taking the unified nodes and new Boolean formulae as inputs, the MTBDD module first constructs the MTBDD for each policy and auxiliary rules (as shown in Figures 7.3(a), 7.3(b) and 7.4(a), 7.4(b)). Then it combines these MTBDDs into a CMTBDD (a section of the resulting CMTBDD is shown in Figure 7.5). In the following subsection, we show how the CMTBDD is used to execute our analysis queries.



(a) Pol$_1$ (b) Pol$_2$

Figure 7.3: MTBDD of policies.



(a) Auxiliary Rule (time) (b) Auxiliary Rule (role)

Figure 7.4: MTBDD for the auxiliary rules.

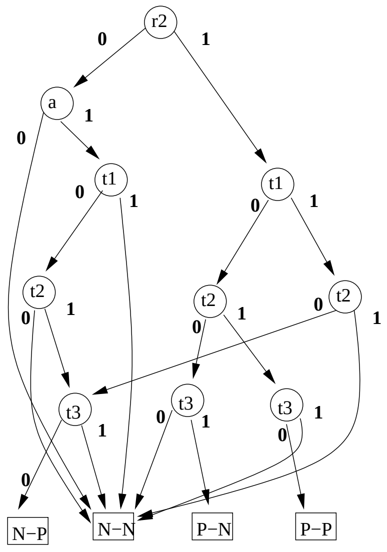


Figure 7.5: CMTBDD.

7.5.2 Query Processing

Recall that each query has three types of components, $\mathcal{B}_q$, e_q and f_q, where $\mathcal{B}_q$ is a Boolean expression on $\mathtt{Attr}_q$, e_q is the desired effect and f_q is a constraint on a set of attributes. For a given query, first we normalize its $\mathcal{B}_q$, map the specified ranges of attributes to the existing unified nodes, and represent the specified ranges as corresponding unified nodes. Then, we construct the query MTBDD. Here, we can treat the normalized $\mathcal{B}_q$ and effect e_q in a query as a rule, and then construct the MTBDD for it. Now consider a query in we would like to know all the possible requests allowed for a role that is subsumed by "PhD Student". Such request can be translated as "given $f^{sup}(\mathtt{Role}, \mathtt{PhDStudent})$, Decision = *permit*, find all possible requests". Figure 7.6 shows the corresponding query MTBDD. After we obtained the query MTBDD, we combine it with the MTBDD or CMTBDD of the policies being queries, where we obtain a temporary structure called Query CMTBDD. By using the model checking technique on the Query CMTBDD, we are now able to find the requests satisfying the $\mathcal{A}_q$ and e_q. As for the example query, we just need to find all paths in the Query CMTBDD which leads to the

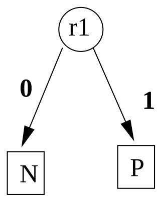


Figure 7.6: Query MTBDD.

terminal named "P-P". Note that for conditional decisions, the nodes along the path may need to be examined by plugging the specific variable values.

As for the policy queries with an empty set of Bq, such as the policy relationship evaluation queries, the processing is even simpler. We only need to check the terminals of the CMTBDD. For example, to check if two policies are equivalent, we check whether there exist only three terminals containing "P-P", "D-D" and "N-N", which means two policies always yield same effects for incoming requests. Finally, a post-processing may be required if there are constraints specified by f_q. This step is straightforward since we only need to execute some simple examinations on the requests obtained from the previous step. The results will then be collected and organized by the result analyzer before being presented to the user.

Chapter 8

P3P Similarity

In this section we provide some results about the application of the data-mining approach developed in Section 7.4 for P3P privacy policies. In Section 8.1 we provide some background information about P3P. We define the similarity function in Sections 8.2 and . We discuss the meaning of a clustering approach for P3P policies in Section 8.4. Finally, in Section 8.5, we report details about the experimental evaluation.

8.1. P3P

The Platform for Privacy Preferences Project (P3P) enables Web sites to express their privacy practices in a standard format that can be retrieved automatically and interpreted easily by user agents [23]. P3P user agents will allow users to be informed of site practices (in both machine- and human-readable formats) and to automate decision-making based on these practices when appropriate. Thus users need not read the privacy policies at every site they visit.

With the widespread use of web services, a wide range of personal information from personal hobby, shopping history to driver license and social security number can be collected by the service providers. There is no doubt that the misuse of such personal information can cause problems like receiving tons of junk mails and suffering from identity theft. To address the concerns about personal information privacy, the Platform for Privacy Preferences (P3P) is proposed as a World Wide Web Consortium (W3C) standard for expressing privacy policies of a website. P3P can help balance the web service providers need for information to provide consumers with desired services and each individuals privacy preferences. P3P version 1.0 specifies a protocol for user agents to locate P3P policies on websites and a syntax for compact policies sent in HTTP response headers. A format is specified for policy reference files that indicate the location of P3P policies on a website and the parts of the website to which they apply. Compact P3P policies is a summary of the sites privacy policy transmitted as a series of tokens in a P3P HTTP header along with a cookie. Its main purpose to enable the web browser to make a quick decision about whether to accept a cookie. In addition, the P3P1.0 recommendation also specifies an XML syntax for privacy policies. In what follows, we will describe the XML syntax in some detail.

A P3P policy consists of an Entity element(`<ENTITY>`) which is used to provide the name and contact information for the website, an Access element(`<ACCESS>`) which indicates whether the site provides access to various kinds of information, a Disputes(`<DISPUTES>`) which describes dispute resolution procedures and one or more Statement elements (`<STATEMENT>`). The Statement element is the most important component of the policy in that it specifies the data and the type of information collected (`<DATA>` and `<CATEGORIES>` elements) along with how the information may be used (`<PURPOSE>` element), how the information may be shared (`<RECIPIENT>` element) and the associated data retention policies (`<RETENTION>` element). Each of these elements may contain elements chosen from a predefined set of options, although human readable fields are also provided for detailed explanations. In addition, the Purpose and Recipient elements may be associated with a required attribute to indicate whether the corresponding elements are always required or opt-in/opt-out policies apply. The `<DATA>` element is associated with an optional attribute to indicate if the user is required to provide this data. As an example, consider an online business "OnlineStore". Their website collects basic information about a visitors computer/connection for administration purposes and aggregate information about visited pages for improving the site. They do not collect identifiable data and they purge any collected within two weeks. Visitors can contact an independent agency "Foo" regarding disputes and will correct any wrongful errors or actions. The P3P policy corresponding to their privacy practices is given in Table 8.1.

8.2. P3P Similarity Measure

P3P policies represents another domain that is suitable for applying our similarity measure. Though many websites nowadays implement P3P policies [28], it is still a headache for consumers to read a long policy literally and fully understand it. Therefore, our goal is to develop a more convenient approach that provides an overview of web service providers privacy practices by ranking all available web services and displaying them in a descending order of the percentage of the difference between the P3P policy and the user privacy concerns. On the other hand, from the view of a service provider, a new web service provider would like to have a good P3P policy that can address most consumers privacy concerns. To obtain the knowledge of common privacy concerns of consumers, the web service provider may need to survey other websites providing similar services and study their P3P policies. In such case, a meta policy or a policy writing guideline for certain type of websites will be very helpful and can save a lot of efforts for a new service provider. In addition, for web service providers looking for potential collaborators, they also need to make sure that their P3P policies are similar so that their collaboration will not affect their existing customers privacy preferences.

Aims at achieving the above goals, we propose a novel and efficient approach to quantify the difference between P3P policies. The basic idea is to assign a similarity score to two P3P policies by summarizing the similarity between each corresponding component in the two policies. The obtained similarity score can be used to rank websites as well as clustering P3P policies. Clustering is an important technique for discovering interesting data patterns and policy clustering

```
<POLICY name="StorePolicy" discuri="..." xml:lang="en">
    <ENTITY>
        <DATA-GROUP>
            <DATA ref="# business.name">OnlineStore
            </DATA>
        </DATA-GROUP>
    </ENTITY>
    <ACCESS><nonident/></ACCESS>
    <DISPUTES-GROUP>
        <DISPUTES resolution-type="independent" service="Foo"/>
            <REMEDIES><correct/></REMEDIES>
        </DISPUTES>
    </DISPUTES-GROUP>
    <STATEMENT>
        <PURPOSE><admin/><develop/></PURPOSE>
        <RECIPIENT><ours/></RECIPIENT>
        <RETENTION><stated-purpose/></RETENTION>
        <DATA-GROUP>
            <DATA ref="# dynamic.clickstream"/>
            <DATA ref="# dynamic.http"/>
        </DATA-GROUP>
    </STATEMENT>
</POLICY>
```

Table 8.1: An Example P3P Policy.

will help to understand the most common statements in policies which helps to find meta-policies or policy writing guidelines for different types of organizations. Our contributions can be summarized as follows.

- We have identified two interesting and important problems. The first problem is helping a consumer quickly select a suitable web service that satisfy his privacy concerns from a long list of available web services. The second problem is helping new web service providers write good P3P policies that can cover most consumers privacy concerns.
- We have proposed an efficient approach for comparing P3P policies and also applied it to clustering P3P policies.
- We have carried out a set of experimental studies which demonstrate both efficiency and effectiveness of our approach.

8.3. P3P Policy Similarity Measure

In this section we apply the similarity measure defined for XACML in the area of P3P policies [1]. The general approach is similar to the one proposed above but there are some important differences that need to be taken into account. First of all, values within a P3P policy are predefined built-in keywords with a fixed semantics. Secondary, it is important to deal with both the structure and the model of a P3P policy. Actually the theoretical model behind a policy is slightly different from the language defined for its formalization: this means that a similarity measure that follows strictly the structure of a P3P document will bring to bad results in the evaluation of the similarity measure.

The P3P policy similarity measure assigns a score between 0 and 1 to quantify the distance(similarity) between two P3P policies. The scores are calculated by comparing the corresponding elements of the given two policies. A score is calculated for each pair of elements being compared. The scores for different types

[1]For the rest of this section the word "policy" will be used to refer to a P3P policy

```
<dynamic.clickstream, {}, {admin, develop}, {ours}, {stated-purpose}>
<dynamic.http,        {}, {admin, develop}, {ours}, {stated-purpose}>
```

Table 8.2: Simplified form of a P3P Policy

of elements are then aggregated to find the overall similarity score between the policies. We use a semantically equivalent but simplified form of a P3P policy when defining the policy similarity measure. In particular, we transform a policy conforming to the P3P syntax into a list of tuples corresponding to the *Statement* depicted in Table 8.2.

elements in the policy. We do not consider the *Entity*, *Access* and *Disputes* elements because these elements contain information like addresses that are specific to the website that owns the P3P policy and are not related to their actual privacy practices as such. The policy similarity measure must only compare elements which are relevant to how a policy uses what data as we are mainly interested in knowing how similar two policies are with respect to how each of them deals with a visitors data. A simplified P3P policy is a list of tuples $t_i(1 <= i <= n)$, where each t_i is of the form : $ti : \langle D_i, C_i, P_i, RT_i, RN_i \rangle$ where D_i corresponds to a Data element, C_i, P_i, RT_i and RN_i correspond to the categories, purposes, recipients and retention policies respectively associated with the data element D_i. The simplified form of the P3P policy in Table 8.1 is shown in Table 8.2. The similarity score between two policies is hence the similarity score between the list of tuples corresponding to the policies. The similarity score between two tuples is calculated as a weighted sum of the similarity between the data, categories, purposes, recipient and retention components of the two tuples. In what follows, we will first describe how the similarity for different types of components are calculated and then using these we will show how the overall similarity score between policies is calculated.

8.3.1 Data Similarity

A Data element in a P3P policy is associated with a uriref attribute whose value specifices a URI reference of data and an optional attribute whose value can be either yes or no and which specifies whether or not the website requires visitors to provide this data to access a resource or complete a transaction. In addition, each data is associated with a Categories element that specifies the intended uses of the data. The P3P 1.0 standard provides a fixed set of categories to which the data might belong. In case, the data does not belong to any of the available categories, then a human readable explanation must be provided. In our work, we only consider the cases where data fits into one of the available categories.

The similarity between two Data elements D_i and D_j is calculated as a weighted sum of the similarity between `uriref/optional` attribute values(S_{dat}) and the categories associated with the data (S_{cat}). The data similarity S_{DATA} is thus given in Equation 8.1.

$$S_{\mathsf{DATA}}(D_i, D_j) = w_{dat}S_{dat}(D_i, D_j) + w_{cat}S_{cat}(C_i, C_j) \tag{8.1}$$

where:

- w_{dat} and w_{cat} are weights such that $w_{dat} + w_{cat} = 1$,
- D_i and D_j are *Data* elements,
- C_i and C_j are *Categories* element associated with D_i and D_j respectively.

The score S_{dat} is a weighted average of the similarity between `uriref` value and optional value and is given by:

$$S_{dat}(D_i, D_j) = w_{uri}S_{uri}(u_i, u_j) + w_{opt}S_{opt}(o_i, o_j) \tag{8.2}$$

where:

- w_{uri} and w_opt are weights such that $w_{uri} + w_{opt} = 1$,

- u_i, u_j and o_i, o_j represents the uriref and optional values respectively of Data elements D_i,D_j.

The score S_{opt} is equal to 1 iff o_i is equal to o_j and is 0 otherwise. The score S_{uri} is calculated as follows:

$$S_{nodes}(u_i, u_j) = \frac{\mid path(u_i) \cap path(u_j) \mid}{max(height(u_i), height(u_j))} \tag{8.3}$$

where:

- $path(u_i)$ represents the set of ordered nodes of the data element referred by u_i,
- $height(u_i)$ represents the number of nodes in the path of u_i.

Finally, the score S_{cat} is given by:

$$S_{cat}(C_i, C_j) = \frac{\mid C_i \cap C_j \mid}{\mid C_i \cup C_j \mid} \tag{8.4}$$

where:

- $\mid Ci \mid$ represents the cardinality of the set C_i.

8.3.2 Purpose Similarity

The Purpose element of a P3P policy consists of a set of elements chosen from a set of pre-defined purposes. Each of these purposes is associated with a required attribute whose value may be one of "always", "opt-in" or "opt-out". The similarity between two Purpose elements is calculated by taking into account the number of common purposes with common value of required attribute between the two elements. It is given by:

$$S_{\mathsf{PURPOSE}}(P_i, P_j) = \frac{\sum_{p \in P} S_{att}(p)}{\mid P_i \cup P_j \mid} \tag{8.5}$$

$$S_{att}(p) = \begin{cases} 1, & \text{if } att_1(p) = att_2(p) \\ \sigma, & \text{otherwise} \end{cases} \tag{8.6}$$

where:

- $P = P_i \cap P_j$,
- $att_i(p)$ returns the value of the required attribute associated with the purpose p in P_i,
- $0 \leqslant \sigma \leqslant 1$ is the penality returned by $Satt(p)$ if p has different value for `required` attribute in P_i and P_j [2].

8.3.3 Recipient Similarity

The similarity between two *Recipient* elements is calculated in a manner similar to that used for the Purpose elements. It is given by:

$$S_{\texttt{RECIPIENT}}(RT_i, RT_j) = \frac{\sum_{r \in RT} S_{att}(r)}{\mid RT_i \cup RT_j \mid} \tag{8.7}$$

$$S_{att}(r) = \begin{cases} 1, & \text{if } att_1(r) = att_2(r) \\ \sigma, & \text{otherwise} \end{cases} \tag{8.8}$$

where:

- $RT = RT_i \cap RT_j$,
- $att_i(r)$ returns the value of the required attribute associated with the recipient r in RT_i,
- $0 \leqslant \sigma \leqslant 1$ is the penality returned by $Satt(r)$ if r has different value for `required` attribute in RT_i and RT_j [3].

[2]Typically σ may be set to 0.5.
[3]Typically σ may be set to 0.5.

8.3.4 Retention Similarity

The *Retention* element in P3P policy consists of retention values, indicating the retention policies, chosen from a set of predefined retention values. The similarity between two Retention elements is calculated by simply taking into account the intersection of the set of values in the two elements. It is given by:

$$S_{\texttt{RETENTION}}(RN_i, RN_j) = \frac{| RN_i \cap RN_j |}{| RN_i \cup RN_j |} \tag{8.9}$$

8.3.5 Tuple Similarity

The similarity between two tuples t_i and t_j is calculated by taking the weighted sum of the similarity of the corresponding *Data*, *Purpose*, *Recipient* and *Retention* elements in the two tuples. It is given by:

$$\begin{aligned} S_{\texttt{TUPLE}}(t_i, t_j) = w_{data} S_{\texttt{DATA}}(D_i, D_j) + w_{purp} S_{\texttt{PURPOSE}}(P_i, P_j) + \\ w_{rec} S_{\texttt{RECIPIENT}}(RT_i, RT_j) + w_{ret} S_{\texttt{RETENTION}}(RN_i, RN_j) \end{aligned} \tag{8.10}$$

where:

- $t_i = \langle D_i, C_i, P_i, RT_i, RN_i \rangle$,
- $w_{data} + w_{purp} + w_{rec} + w_{ret} = 1$,
- D_i, C_i represent the data and categories belonging to tuple i,
- P_i represents the set of purposes belonging to the tuple i,
- RT_i represents the set of recipients belonging to the tuple i,
- RN_i represents the set of retention values belonging to tuple i.

8.3.6 Policy Similarity

Finally, the similarity between two P3P policies P_i and P_j is calculated by averaging the similarity scores obtained between pairs of tuples corresponding to the two policies. Equation 8.11 is used for the computation of the family of sets Θ_a that represents all the possible combinations of pairs of the form $\langle t_{n_i}, t_{m_j} \rangle$ satisfying the conditions that the similarity between each tuple is greater than the threshold ε [4] and that each tuple is took into consideration at most once. Equation 8.12 define the set Θ that is the set Θ_a that maximize the sum of the similarity between the pairs of tuples it contains. Finally, Equation 8.16, represents the similarity between a pair of policy. Since it is possible that some tuple is not contained in any of the pairs in Θ we take the average similarity between each of those tuples and all the tuples belonging to the paired policy.

$$\begin{aligned}\Theta_a(P_n, P_m) = \{(t_{n_i}, t_{m_j}) \mid t_{n_i} \in P_n \ \wedge \ t_{m_j} \in P_m \ \wedge \ S_{triple}(t_{n_i}, t_{n_j}) > \varepsilon \ \wedge \\ \forall \ (t_{n_{i'}}, t_{m_{j'}}), (t_{n_i} = t_{n_{i'}}) \Rightarrow (t_{m_j} = t_{m_{j'}})\}\end{aligned} \tag{8.11}$$

$$\begin{aligned}&\Theta = \Theta_{a'} \quad \text{so that:} \\ &\forall a \left(\sum_{(t_{m_i}, t_{n_j}) \in \Theta_a} S_{triple}(t_{n_i}, S_{m_j}) \quad \leqslant \quad \sum_{(t_{n_i}, t_{m_j}) \in \Theta_{a'}} S_{triple}(t_{n_i}, t_{m_j}) \right)\end{aligned} \tag{8.12}$$

$$Sum^{\in \Theta} = \sum_{(t_{n_i}, t_{m_j}) \in \Theta} 2 \ S_{triple}(t_{n_i}, t_{m_j}) \tag{8.13}$$

[4]This threshold is used as a filter. We do not take into considerations the similarities between pairs of tuples that do not satisfy the constraint on the treeshold.

$$Sum_n^{\notin\Theta} = \sum_{(t_{n_i},-)\notin\Theta} S_{triple}^{avg}(t_{n_i}, P_m) \tag{8.14}$$

$$Sum_m^{\notin\Theta} = \sum_{(-,t_{m_j})\notin\Theta} S_{triple}^{avg}(t_{n_j}, P_n) \tag{8.15}$$

$$S(P_n, P_m) = \frac{Sum^{\in\Theta} + Sum_n^{\notin\Theta} + Sum_m^{\notin\Theta}}{\mid P_n \mid + \mid P_m \mid} \tag{8.16}$$

8.4. P3P Policy Clustering

Policy clustering aims to separate policies into distinct groups based on the differences in the policies. Each group of policies has its unique characteristics which can be later used to create a meta policy for this group. In what follows, we present the details of the clustering algorithm. We use $1 - S(P_i, P_j)$ as the distance function between any two policies P_i and P_j and employ the well-known K-median clustering algorithm with minor modifications. The algorithm consists of four main steps.

The first step is a preprocessing step which computes the distance between all pairs of policies and stores the values in a matrix. This avoids distance recomputation during the clustering. Second, we randomly select k policies as initial cluster centroids. Third, we assign each policy to the cluster that has the closest centroid. Fourth, when all policies have been assigned, recalculate the k centroids and then repeat Steps 3 and 4 until the centroids no longer change. Note that the computation of the centroid for policies is different from calculating the center point of a cluster of points in Euclidean space since it is impossible to compute an "average" policy. Therefore, we select a policy which has the minimum sum of distance to all the other policies as the centroid.

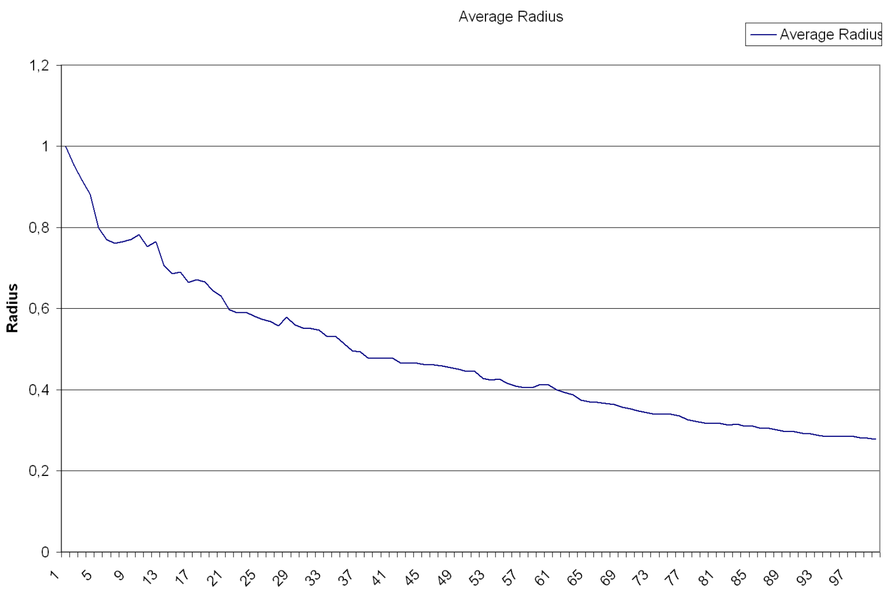


Figure 8.1: Average Radius varying K from 1 to 100 on a dataset of 1000 policies.

8.5. Experiments

We have implemented a Java prototype of the clustering algorithm presented in section above and we have experimented our approach on a subset the TAPPA db containing 23,891 P3P policies [5]. First of all we used the average radius to evaluate the results of the clustering algorithm. The average radius of a cluster measures the average distance between two elements in a cluster (the normalized distance between two policies can be obtained subtracting the value of the similarity between the two policies to 1) and is one of indicators of the *quality* of the clusters and hence in turn the quality of the distance measure (in our case the proposed similarity measure). The smaller the radius the better the clustering in Figure 8.1 we show the average radius for the value of K ranging from 1 to 100 on a dataset of 1000 policies.

Anyway, the results on the quality of the clusters are not enough. Typically, a clustering algorithm is used to identify groups of objects in a certain dataset.

[5]http://cups.cs.cmu.edu/tappa/

The groups are identified according to some specific features of the objects. For example, if we have points in an euclidean space we can clusterize those points taking into consideration their position within the space. In our case we have policies that are complex objects that contains a number of different information and for this reason different perspectives needs to be used in the evaluation step. Average radius is a good choice for understanding the quality of the resulting clusters (and in turn the quality of the clustering algorithm) but we need to take into consideration additional data for understanding the meaning of our clusters. The TAPPA db maintains some interesting meta-data for each policy:

- Top level domain;
- Top level domain type;
- Traffic rank;
- Domain name;
- URL of the policy;
- Owner of the policy;
- Email contact;
- Address of Website;
- Country where site is registered;
- P3P Policy name;

Moreover, in [28] is proposed an interesting categorization of P3P-enabled site. The results depicted in Figure 8.2 show some implicit clusters based on how web-sites (and hence the P3P policies defined for those sites) have been categorized. Since the values reported in Figure 8.2 have been obtained running test on the TAPPA db would be very interesting to retrieve those categories for the policies we already have in our dataset.

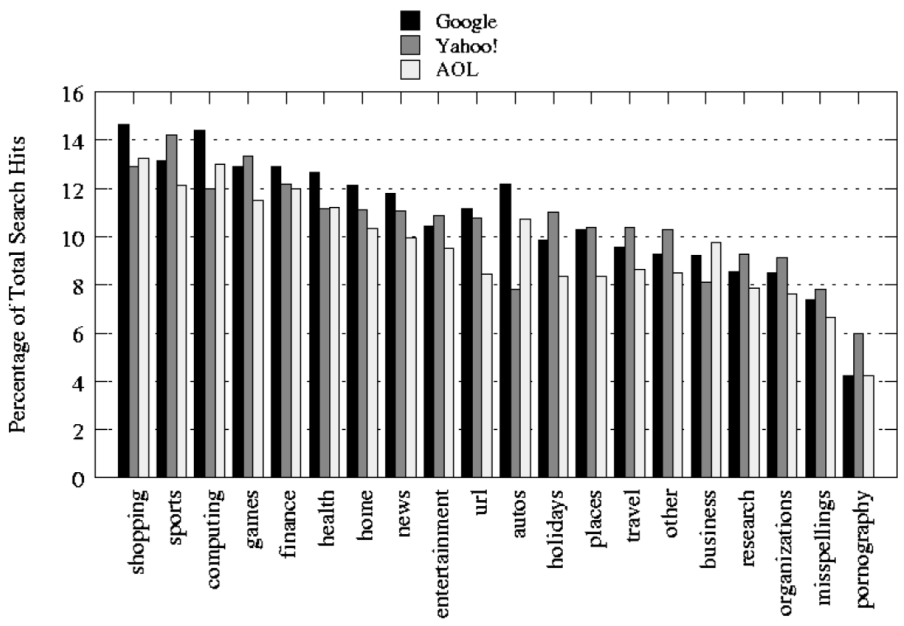


Figure 8.2: Distribution of P3P-enabled search results by search term category.

We are working on all these data in order to come up with interesting test cases for deeper evaluations of both our P3P similarity measure and the clustering algorithm.

Part III

EXAM-S: Implementation

Chapter 9

EXAM-S: the Architecture

In this chapter we discuss the various components in the architecture of EXAM-S. The Chapter is organized as follows: in Section 9.1 we give an overview of the main components. Section 9.2 discuss the main module in which all the analysis components are organized. The entities exploited for solving policy heterogeneity are introduced in Section 9.3. Finally, Section 9.4 discuss the features of the repositories main module.

9.1. Architecture

The EXAM-S environment, an overview of which is shown in Figure 9.1, includes three different modules. The first module is the *analysis module*, which receives policies requests and queries from users, and returns request replies and query results. The second module is the *heterogeneity module* that contains all the components that have been defined in Chapter 6. This module handles policy heterogeneities providing a unified vocabulary for the policies involved into the analysis process. Finally, the third module is the one that contains all the repositories that are exploited by the other modules.

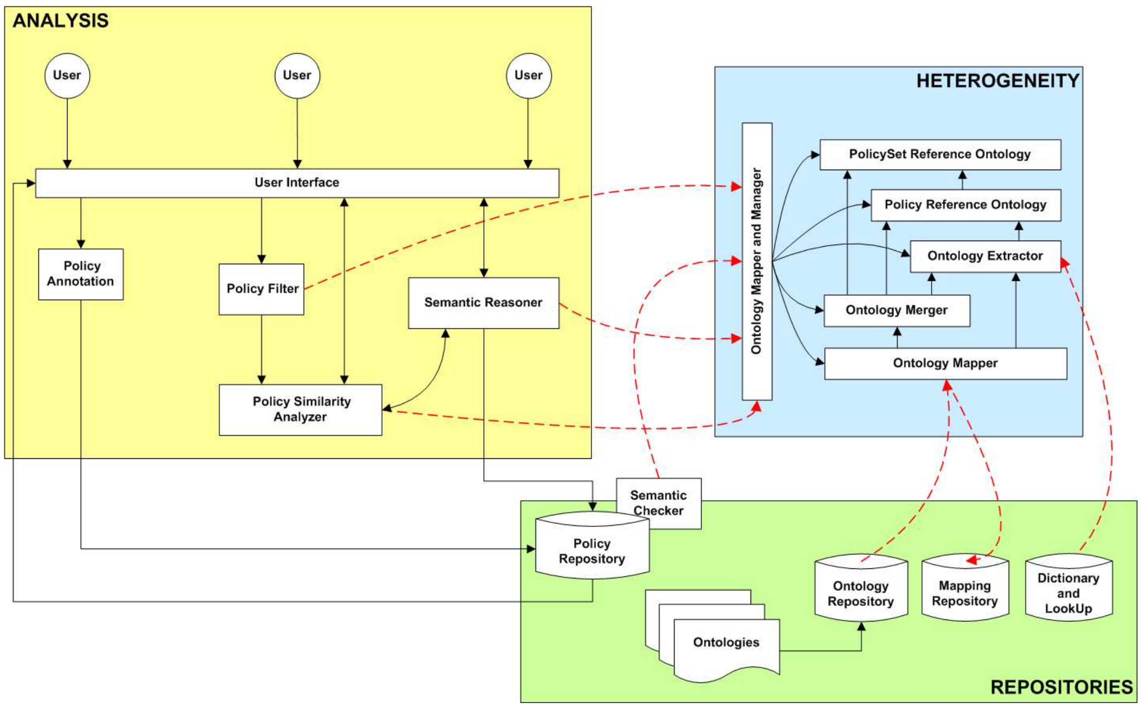


Figure 9.1: The EXAM-S Architecture

9.2. Analysis Module

The Analysis Module, includes three different levels. The first level is the user interface, which receives policies requests and queries from users, and returns request replies and query results. The second level is the request dispatcher, which handles various requests received from the user interface, dispatches them to proper analysis module and aggregates obtained results. The third level is the core level of EXAM and includes four modules supporting different tasks in policy analysis, namely: policy annotation, policy filtering, policy analysis and Semantic Reasoning. The policy annotation module pre-processes each newly acquired policy by adding annotations to it. The annotations explicitly represent the behavior or semantics of each function referred in the policy. Such annotations help in automatically translating policies into Boolean formulae that can then be evaluated by the policy analysis modules. The annotated policies are stored in the policy repository together with the policy metadata. The policy filter module acts as a filter phase for policy similarity analysis when there is a large amount of policies to compare. It is a lightweight approach which quickly evaluates similarity between each pair of policies and assigns them a similarity score. According to the obtained similarity scores, policies with low similarity scores can be safely pruned from further analysis, whereas policies with high similarity scores can be further examined. The main goal of the policy filter module is to reduce the number of policies that need to be analyzed more in details, when dealing with large size policy sets. The filtering approach we use is based on techniques from information retrieval and is extremely fast. The use of filtering in the policy analysis process is however optional. The policy management module can directly send analysis queries to the policy similarity analyzer (PSA), to carry out a fine-grained policy analysis, without performing the filtering. Moreover, the PSA implements the standard policy analysis queries supported by EXAM-S.

9.2.1 Architecture of the Policy Similarity Analyzer (PSA)

Figure 9.2 shows the architecture of PSA. The basic idea underlying its architecture is to combine the functionalities of the policy ratification technique [2] and MTBDD technique [35] by using a divide-and-conquer strategy. Specifically, policies are first passed to a preprocessor which identifies parts to be processed by the ratification module and parts to be directly transmitted to the MTBDD module. The ratification module then generates unified nodes and a set of auxiliary rules that are transmitted to the MTBDD module. The MTBDD module then creates a combined MTBDD that includes policies and additional rules. By using the combined MTBDD, the PSA module can thus process the queries that we introduced in Chapter 7. Queries are first translated by the query preprocessor and then executed by the MTBDD module. Finally, the result analyzer reformats the output of the MTBDD module and reports it to the users.

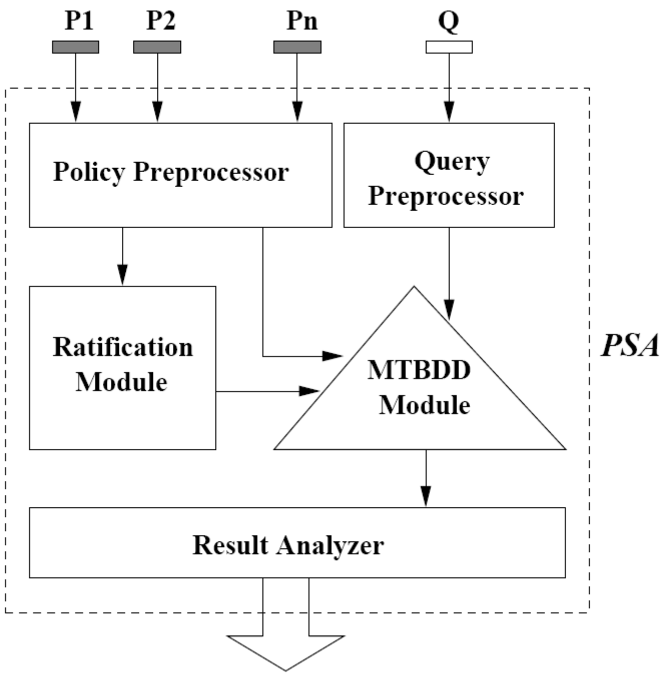


Figure 9.2: Architecture of the Policy Similarity Analyzer (PSA)

9.2.2 Semantic Reasoner submodule

Finally, the Semantic Reasoner component supports the PSA for the reasoning tasks on the specified domain ontology. The main task of this module is to support the ratification module in finding the disjoint ranges over the involved domain ontologies. The Semantic Reasoner, receive the set of ranges and the ontology they belong and then creates the set of disjoints ranges. Semantic reasoning is executed through the use of the Pellet reasoner [75]. The result of the reasoning procedure is then returned to the PSA and then exploited in the remaining steps of the analysis process. The architecture of this module is shown in Figure 9.3.

9.3. Heterogeneity Module

This section discuss the features of the heterogeneity modules. This module contains all the components that have been defined in Chapter 6. We have added an

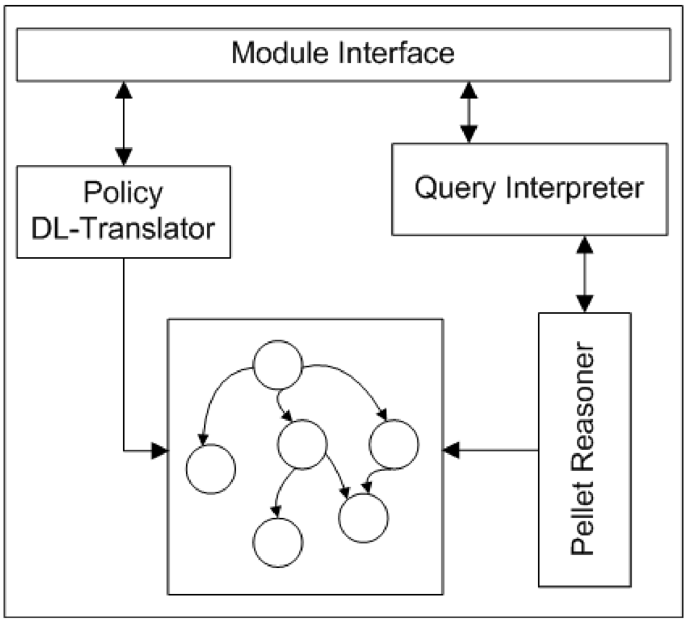


Figure 9.3: Architecture of the Semantic Reasoner Submodule

interface that is dedicated to the dispatch of the requests to the respective module. We have implemented the stack of technologies defined for solving policy heterogeneity has decoupled modules each one able to perform the associated algorithm defined in Chapter 6. Subsection 9.3.1 is dedicated to the Policy Reference Architecture creation submodule.

9.3.1 Policy Reference Ontology creation

Figure 9.4 depicts a conceptual visualization of the Policy Reference Ontology submodule (PSO). The PSO combine the architectural components exploited in the involved procedures. The ontology extraction procedure is implemented with two submodule: the first extracts the taxonomy from the terms in the policy exploiting the dictionary and the lookup tables. We use WordNet [31] as a reference dictionary. The second one, creates the relations between those terms exploiting additional semantic information, such as ontologies and the relations implicitly defined within the policy. If the policy exploits some ontologies them are merged

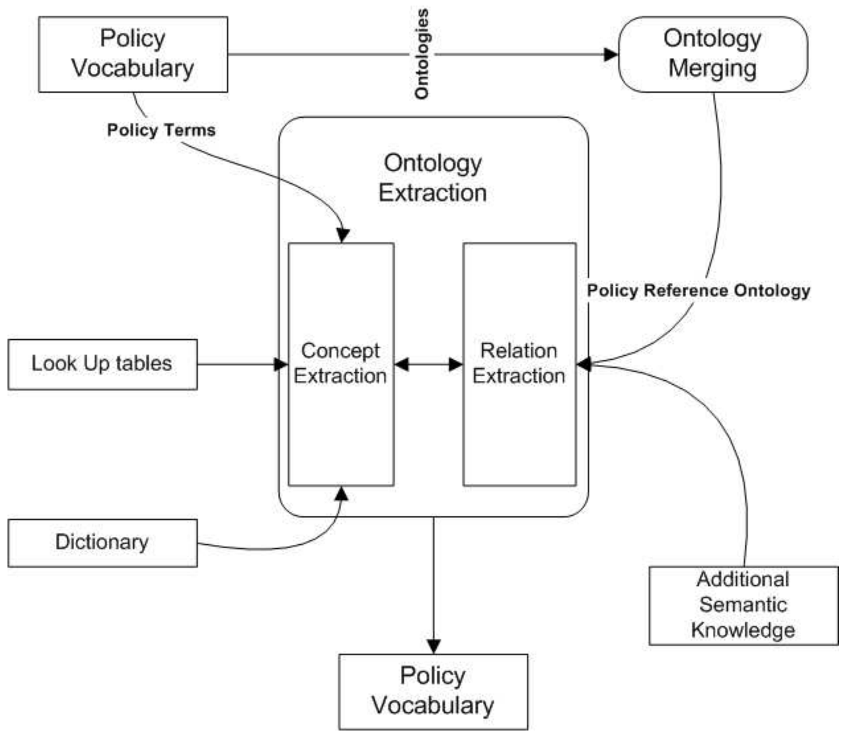


Figure 9.4: Architecture of the Policy Reference Ontology creation procedure.

and used in the extraction procedure.

9.4. Repositories Module

The repository module has been created for optimization purposes. Sometimes, the operation performed over ontologies, can be expensive especially when dealing with large knowledge bases or with complex policies that use several ontologies.

For this reason, whenever an ontology mapping or an ontology extraction is performed, we maintain repositories in which the results of such procedures are stored. Then, when it is necessary to perform again the same kind of operation, we check in the repositories if we have already computed the result. If this is the case then we obviously skip the execution of the operation returning to the following procedures (if any) or to the calling module the result of the operation. Otherwise, the process is carried out as usual. For this reason we have created two different repositories, the first one the maintain ontologies (both the imported from external resources and the one generated by the internal modules) and the second one that store the mapping between them.

Moreover, this is the module in which we maintain the dictionary and the lookup tables. In Figure 9.1 they are depicted in the same repository for brevity. As we have already introduced before, we use WordNet as our reference dictionary. WordNet, comes as an autonomous database with its own interface so no additional components are necessary.

Chapter 10

Implementation and Experimental Evaluation

In this chapter we discuss implementation and experimental evaluation of the approaches developed in Chapters 6 and 7. The Sections are organized as follows: in Section 10.1 we describe the details of the heterogeneity module. Sections 10.2 and 10.3 present the experiments for the filtering technique, the policy similarity analyzer is discussed in Sections 10.5 and 10.5.

10.1. Experimental Results: Heterogeneity Module

We have implemented a JAVA prototype of the heterogeneity module. In the prototype we have exploited the Sun implementation of XACML, the OWL API for loading, updating and creating ontologies, the Falcon-AO library for ontology matching, and the MIT Java WordNet Interface for managing queries on the WordNet database.

For the experimental evaluation, we generated a set of policies in a XACML format which serves as input for the creation of the unified vocabulary. Attributes were randomly selected by a predefined list while semantic data was obtained by randomly selecting entities by a set of ontologies retrieved by using the SWOOGLE ontology search engine.

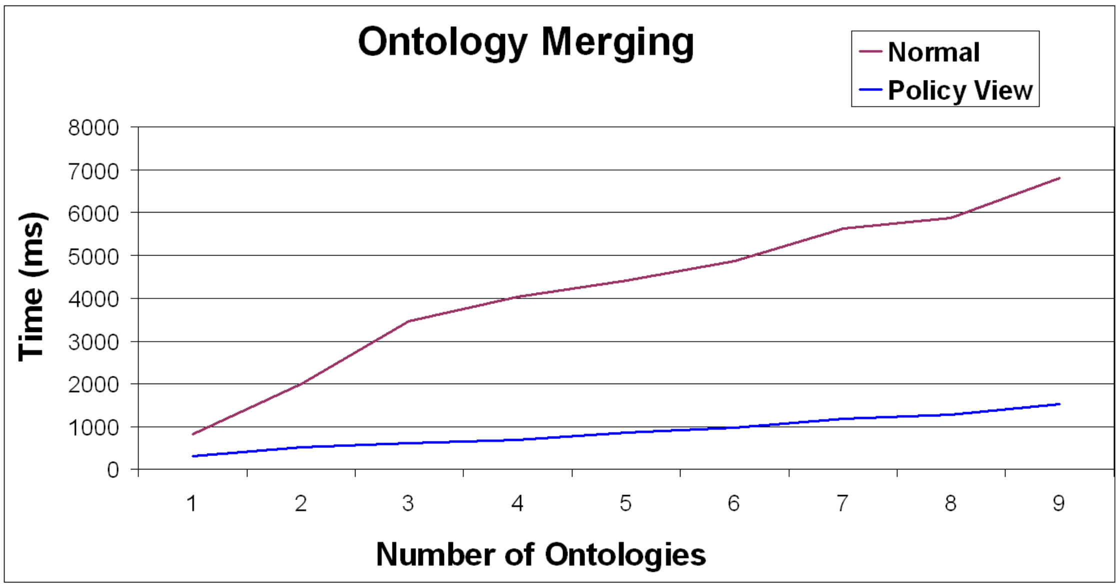


Figure 10.1: Comparison between the times of the Ontology Merging algorithm applied to original referenced ontologies and the Ontology Merging applied to their policy views.

Figure 10.1 shows the comparison between the merge algorithm applied to original referenced ontologies and the merge algorithm applied only to their policy views for increasing values in the number of the ontologies involved. The use of the policy views of an ontology significantly improves the performance of the merging algorithm. Moreover, the accuracy in the policy analysis results is not affected by the adoption of the policy view optimization. The reason is that during the creation of the view, we prune all entities that are not considered by the structural techniques adopted by state of the art ontology matching tools. Figure 10.2 shows the total execution time of our process for increasing values in the number of attributes. We plotted the execution time of the approach for varying values in the number of total attributes[1] between 10 and 50. Each column shows the time for: (i) the merge algorithm, (ii) the extract algorithm and (iii) the

[1]Since in our approach we consider attribute-value pairs, it makes more sense to analyze the times with respect to the number of attributes instead of the number policies.

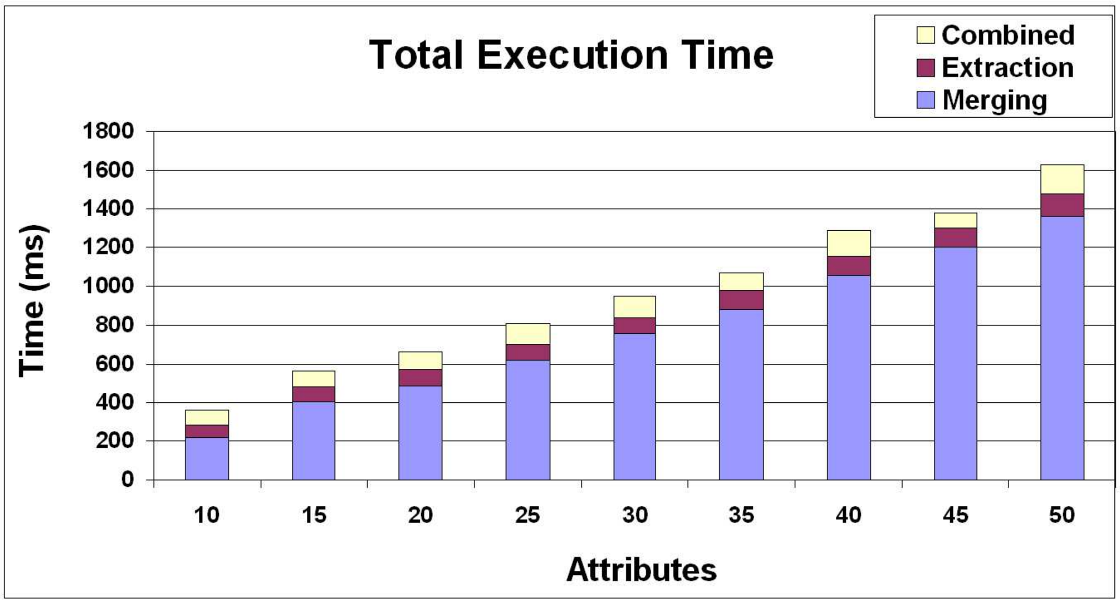


Figure 10.2: Total execution times for increasing values in the number of attributes.

combination of their result. As expected, most of the execution time is spent in merging ontologies. Conversely, the extraction is very quick and even for high number of attributes (not reported in the Figure) e.g. ≈ 100, the execution time is ≈ 150 msec.

In Table 10.1 we report data concerning the accuracy of our model. We evaluate the number of similar concepts detected and the correctness of the related mappings when varying the value of the threshold τ between 0.65 and 0.95. Modifying the value of τ means changing the acceptance threshold during the merge algorithm. The results show that it is important to find a good trade-off between the number of mappings retrieved and their correctness. For this experiment, we run our prototype on a set of policies with an average number of 50 attributes. Attributes have been then randomly associated to concepts belonging to a set of ontologies related to the faculty domain. The results show that for values of τ between 0.75 and 0.80 our model provides a good balance between the correctness of the mappings and the increase in similarities detected. For values of τ

greater or equal than 0.80 we obtain a higher accuracy but the number of mappings seems to be too low for being adopted in practice. Conversely, with values of τ lower or equal than 0.70 we obtain more hits but the correctness is too low to be considered acceptable.

Table 10.1: The accuracy of the model.

τ	Similarities Detected	Correctness
$[0.65, 0.70]$	86,458%	58,823%
$[0.70, 0.75]$	85,416%	64,705%
$[0.75, 0.80]$	**80,208%**	**80,411%**
$[0.80, 0.85]$	58,333%	85,294%
$[0.85, 0, 90]$	52,083%	91,176%
$[0.90, 0.95]$	46,875%	94,117%
> 0.95	42,708%	97,059%

10.2. Implementation: Filtering

We have implemented a prototype of the proposed similarity measure techniques using Java. We have performed extensive testing of the implementation on randomly generated access control policies. We evaluated both the effectiveness and efficiency of our lightweight policy similarity measure in contrast to exhaustive policy comparison techniques which involve Boolean expression analysis.

All experiments were conducted on 3Gz Pentium III processor machine with 500MB RAM.

10.3. Experimental Results: Filtering

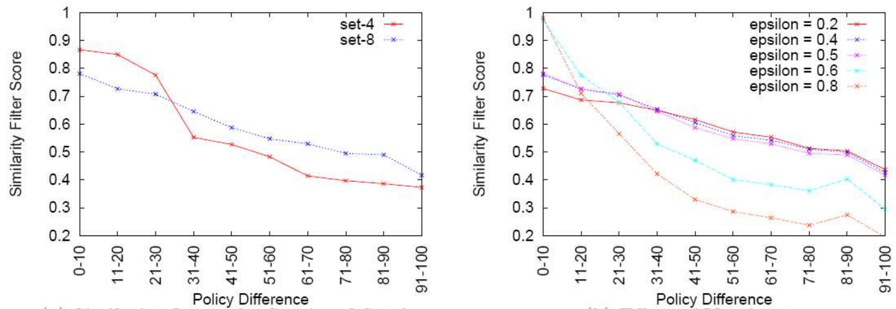
We first evaluated the effectiveness and efficiency of the policy similarity measure. This set of experiments were conducted without considering the ontology matching and dictionary lookup. We then measured the scalability of the implementation for both the variations with and without ontology. Finally we looked in detail the differences obtained with respect to the similarity scores when using the ontology matching and dictionary lookup.

10.3.1 Effectiveness

Since our policy similarity measure is an approximation of the similarity between two policies, in order to demonstrate the effectiveness of the similarity measure, we compared our results with those obtained by the exact policy similarity analyzer proposed in [57] .The output of the exact policy similarity analyzer is a list of requests and effects of the two policies for these requests. Based on this information, we can quantify the differences between two policies using the percentage of the requests for which the two policies have different effects. The higher the percentage of such requests the less similar the policies are.

Each policy pair in *set-4* and *set-8* was input to both the policy similarity measure and the exact policy similarity analyzer. For each policy pair a policy similarity score and a policy difference percentage was recorded. The test sets *set-4* and *set-8* each contained 100 pairs of policies. In *set-4* each policy had 4 rules each and in *set-8* each policy had 8 rules each. The maximum number of attribute predicates in any given policy was 68 for *set-4* and 124 for *set-8*. Considering that for typical policies we have encountered in real world applications the average number of atomic Boolean expressions lies between 10 and 50, our test sets covered a much bigger range.

Figure 10.4(a) shows the policy similarity score and policy difference percentage for policy pairs in *set-4* and *set-8* with the threshold ϵ set to 0.5. We can observe that policy similarity scores decrease when the differences between two policies increase. This indicates that our policy similarity measure provides a good approximation of the similarity between policies. We also explore the effect of the threshold ϵ by varying ϵ from 0.2 to 0.8 for test set *set-8*. The result is shown



(a) Similarity Scores for Set-4 and Set-8 (b) Effect of Varying ϵ

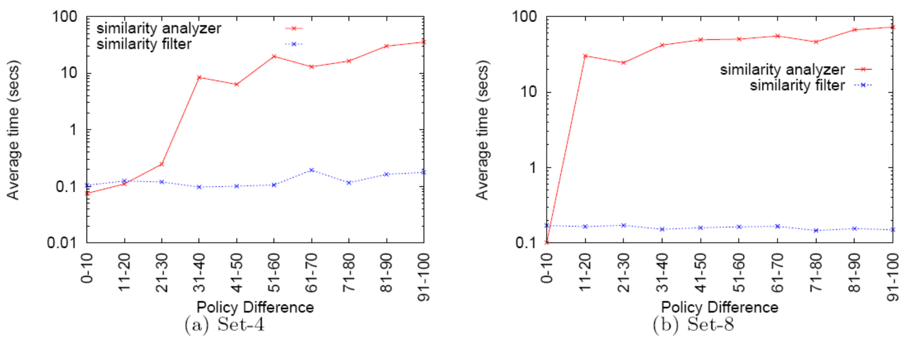
Figure 10.3: Policy Similarity Scores

in Figure 10.4(b). Observe that higher values of ϵ tend to provide a better approximation. This is mainly because that the overall similarity score is the average of the rule similarity scores above ϵ and using higher values of ϵ prunes more rules which are less similar to one another.

10.3.2 Efficiency

The previous set of experiments demonstrate the effectiveness of the policy similarity measure. In order for our technique to be useful as a filter technique that can quickly pruning dissimilar policies, it must also be efficient. We compared the execution time of the policy similarity measure with that of the exact policy similarity analyzer. The same data sets *set-4* and *set-8* were used.

The results for *set-4* and *set-8* are shown in Figure 10.4. Each point in the graphs corresponds to the average execution time for 10 different policy pairs. The value of was set to 0.5. From the figures, we can observe that the policy similarity measure almost remains constant for both *set-4* and *set-8*. This is because the time taken by the policy similarity measure depends on the number of rules and predicates in the policies being compared which is constant for policies in both sets. While for the exact policy similarity analyzer, the more difference between two policies, the more analysis needs to be carried out and hence it requires much more time. Moreover, the average execution time taken by the policy similarity



(a) Set-4 (b) Set-8

Figure 10.4: Execution time

measure is two to three orders of magnitude less than the time taken by the exact similarity analyzer. Such considerable gain is attributed to the quick comparison techniques which avoids complicated Boolean expression analysis. This also indicates that the similarity measure can well serve as a filter phase before invoking the computationally expensive similarity analysis.

10.3.3 Scalability

In this set of experiments, we evaluated the scalability of the policy similarity measure implementation varying the number of attribute predicates across the policies and plotted the average time taken to compute the similarity score. For these experiments the value of the threshold ϵ was set at 0.5. Figure 10.5 reports the average time taken to compute the similarity scores for 10 policy pairs in *set-4* and *set-8*, when varying the number of predicates in each policy from 25 to 400. We can observe that our approach scale reasonably well as the number of predicates per policy increases.

10.4. Implementation: Policy Analyzer

We have developed a prototype of PSA in Java. An implementation of the modified simplex algorithm [2] has been used for processing Boolean expressions with

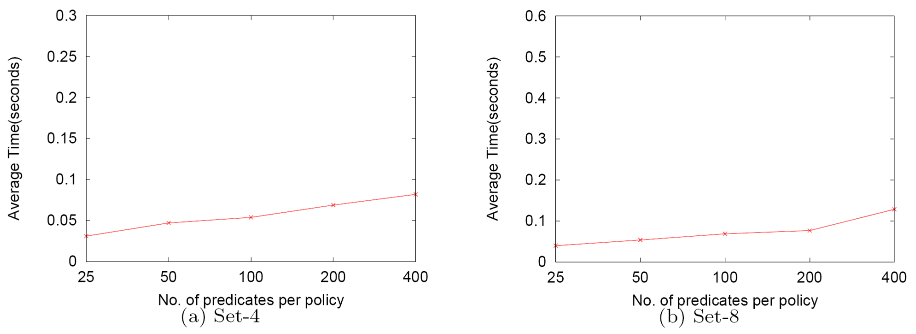


Figure 10.5: Scalability as number of attribute predicates per policy is increased

real value linear constraints. The modified CUDD library developed in [35] has been used for the MTBDD module. In order to test our implementation, we generated XACML policies with a random number of rules. For each policy rule, we first randomly generated atomic Boolean expressions, and then concatenated them with the operator "and" or "or". The atomic Boolean expression (ABE for short) usually contains a pair of attribute name and value except for the atomic linear inequality function which has multiple attributes. The attributes in each atomic Boolean expression were randomly selected from a predefined attribute set. We performed policy similarity analysis between pairs of generated XACML policies with varying number of rules and attributes. The experiments were conducted on a Intel Pentium4 CPU 3.00GHz machine with 512 MB RAM.

10.5. Experimental Results: Policy Analyzer

In the first experiment, we analyzed the total time taken by our policy similarity analyzer as we increase the number of atomic Boolean expressions associated with a pair of policies. The average number of atomic Boolean expressions was varied between 50 and 150 for each policy. The number of rules in each policy was varied between 8 and 32. The results of this experiment are summarized in Figure 10.6. Each policy similarity experiment was repeated 10 times. For visual clarity, we have plotted the time in log scale. The actual minimum and maximum

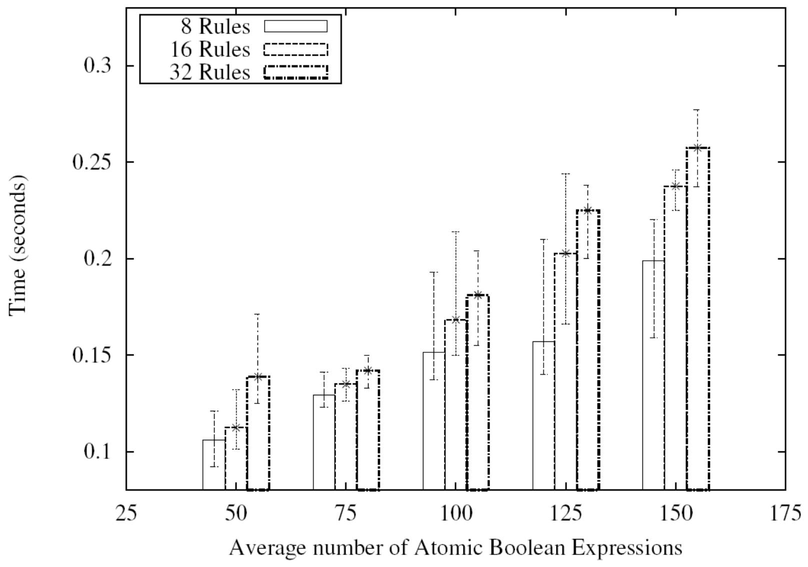


Figure 10.6: Total Response Time for Varying Number of Atomic Boolean Expressions

response time obtained were 0.1s and 50.4s respectively. We observe a clear trend here: increasing the number of atomic Boolean expressions results in an increase in the time needed for similarity analysis. Considering that the number of the attribute value pairs in policies tend to lie in the range of 20 to 100 as reported in [35], we believe that our results indicate a positive trend.

In order to study the characteristics of our proposed preprocessing technique, we examined the average number of average number of MTBDD variables created and the time consumed by the preprocessor module for the same policy pairs used in the first experiment. As shown in Figure 10.7, we can see that the number of variables introduced due to the creation of disjoint intervals and auxiliary rules is less than 1.5 times the average number of atomic Boolean expressions in a policy pair. Next, we plotted the time taken by the preprocessor module in Figure 10.8 for the policy pairs in the previous experiment. We can see that the preprocessor scales well with the increase in the number of rules and atomic Boolean expressions. Specifically, in the case where each policy has 32 rules and

Average number of atomic Boolean expressions per policy	Average number of atomic Boolean expressions per policy pair	Average number of variables		
		8 Rules	16 Rules	32 Rules
50	100	106	106	150
75	150	152	174	169
100	200	194	238	275
125	250	220	294	303
150	300	252	334	388

Figure 10.7: Average number of variables generated for policy pairs

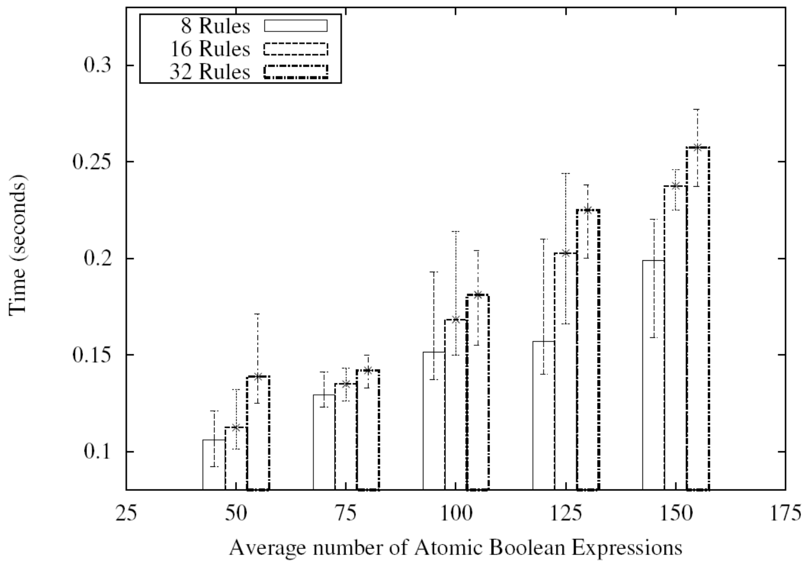


Figure 10.8: Response Time Taken for Preprocessing a Pair of Policies

150 atomic Boolean expressions, the preprocessing time is about 0.5% percent of the overall response time. We can conclude that the overall response time is mainly dominated by the MTBDD module.

In another experiment, results of which are reported in Figure 10.9, we fixed the number of atomic Boolean expressions in each policy and varied the number of pairs of policies to be analyzed for similarity. We considered policies with an average of 100 atomic Boolean expressions, and ran the experiments for policies

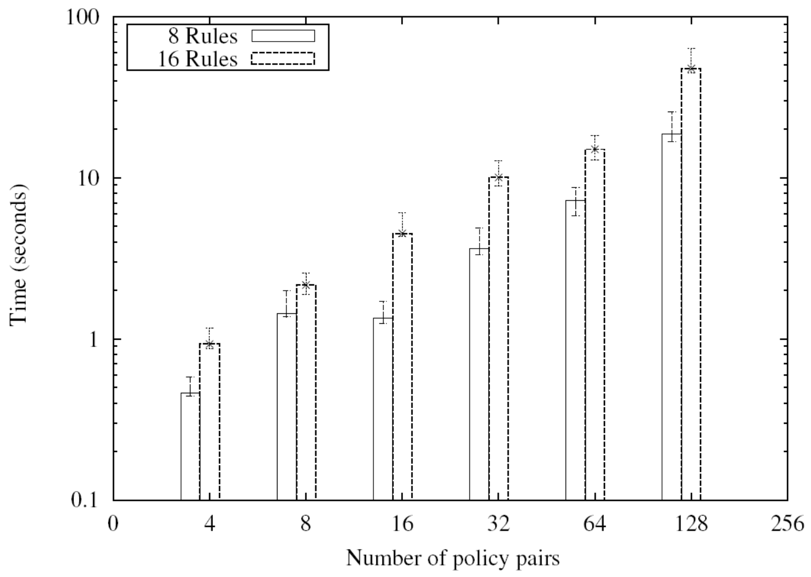


Figure 10.9: Total Response Time for Varying Number of Policy Pairs

with 8 and 16 rules. Each of these experiments were conducted 10 times. The time axis in Figure 10.9 is in log scale. The minimum and maximum response time obtained for these experiments were 0.44s and 63s respectively. From this result, we observe that the current approach can scale reasonably well when deployed in policy integration systems where the number of policies that need to be integrated in a single point in time ranges in a few hundreds.

Chapter 11

Conclusions and Future Work

The use of policy based security management in distributed collaborative applications and architectures has led to the proliferation of policies. A direct consequence of this is the need for tools and techniques to manage and consolidate the large set of policies. In this thesis we have proposed EXAM-S, a comprehensive environment for the analysis and management of access control policies. We consider policies expressed using XACML (Extensible Access Control Markup Language) [72] because XACML is a rich language which can represent many policies of interest to real world applications and is gaining widespread adoption in the industry. We identified and defined three types of basic policy analysis queries which can be combined to perform different advanced analyses.

We have faced the policy analysis problems both theoretically and practically. Concerning theoretical issues, we have compared state of the art approaches in policy analysis trying to find the best trade-off between expressivity and complexity. The result of this study has been taken as input for the definition of our services:

- We have addressed the issues arising in policy heterogeneity analysis. Our approach represents the terminology of a policy through the use of ontologies and consists of a set of functions that allows one to create a unified vocabulary for a multidomain policy set. This vocabulary can be then exploited by policy analysis tools for analyzing and comparing policies. We have implemented a prototype of the proposed approach and demonstrated its effectiveness and applicability in real case scenarios.

- We have proposed a novel policy similarity measure which can be used as a filter approach in policy comparison. The policy similarity measure represents a lightweight approach to quickly analyze similarity of two policies. Detailed algorithms of computation of similarity scores are presented. To the best of our knowledge this is the first work to introduce the notion of a similarity score for access control policies. We have implemented a prototype of the similarity filter and reported experimental results that demonstrate the efficiency and effectiveness of this approach.

- We have proposed a policy similarity analyzer that combines the advantages of MTBDD based model checking and SAT-solver based techniques to provide a precise characterization of the set of requests permitted, denied or not-applicable to the policies being analyzed. The experimental results obtained from the prototype implementation of the analyzer demonstrate the efficiency and scalability of the proposed approach.

- We have proposed and discussed three other projects that are not strictly related to the development of EXAM-S but with which they share a number of common features. In Chapter 5 we have proposed an extension of XACML for dealing with ontology-based access control models. In Chapter 8 we have applied the filtering technique introduced in Section 7.4 to the analysis of P3P policies. Finally, in Appendix A we have proposed an approach for solving interoperability in Digital Identity Management. Even if the main topic is different from the one addressed in this thesis the combination of ontology-based techniques for solving heterogeneity are based on the same assumption and have similar algorithms and results.

Policy Analysis is a challenging task and we plan to improve EXAM-S adopting more powerful techniques. We have performed preliminary test for a tighter integration of MTBDDs and Description Logics in order to provide better analysis services. Moreover, we have implemented a prototype of the extended XACML architecture proposed in Chapter 5. Even if preliminary experimental results are encouraging we plan to investigate this topic in more detail. Related to semantic functions we plan to define new analysis services especially created for dealing with the semantic functions we have proposed. Details about the future work of this thesis are listed below:

- *Semantic Functions*: So far we have created a Java prototype of the extended XACML architecture proposed in Chapter 5 exploiting the the XACML sun libraries [1]. We have then implemented the f^{sub} function exploiting the OWL API[2] and the $\mathcal{DL}$ Pellet Reasoner [75]. The implementation of f^{sup} and f^{same} is still an ongoing work. However, since such functions can be defined in terms of f^{sub} we believe that their realization will be soon available. Open issues in this topic are the variation in performance when modeling equivalent policies by adopting different combination of functions.
- *Improving the PSA*: We will extend the policy similarity analyzer to compare more than two policies at once. Preliminary experiments have indicated that the order in which the policies are compared can result in an order of magnitude difference in the time needed to perform the comparison. We plan to study in detail the cause for such difference and then propose optimization techniques to determine the optimum order in which to compare the policies so as to minimize the comparison time. Moreover,
- *New analysis services*: We have already defined and realized a mapping between XACML and DL that extends the one proposed in [56]. Such extension is motivated by the need to deal with more information than the ones taken into consideration in [56]. Based on this new mapping, we will develop new powerful analysis services for XACML policies extended with

[1]http://sunxacml.sourceforge.net/
[2]http://owlapi.sourceforge.net/

semantic functions. Examples of such services are semantic policy redundancy, semantic policy subsumption and semantic validation.

References

[1] *Iso 10181-3 access control framework.*

[2] D. Agrawal, J. Giles, K. W. Lee, and J. Lobo. Policy ratification. In *IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY)*, pages 223–232, 2005.

[3] T. Ahmed and A. R. Tripathi. Static verification of security requirements in role based cscw systems. In *Proceedings of the 8th ACM Symposium on Access Control Models and Technologies (SACMAT)*, pages 196–203, 2003.

[4] H. Alani, S. Kim, D. E. Millard, M. J. Weal, W. Hall, P. H. Lewis, and N. R. Shadbolt. Automatic ontology-based knowledge extraction from web documents. *IEEE Intelligent Systems*, 18:14–21, 2003.

[5] A. Anderson, editor. *Core and hierarchical role based access control (RBAC) profile of XACML v2.0.* 2005. OASIS Standard, http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-rbac-profile1-spec-os.pdf.

[6] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[7] M. Backes, G. Karjoth, W. Bagga, and M. Schunter. Efficient comparison of enterprise privacy policies. In *Proceedings of the 2004 ACM Symposium on Applied Computing (SAC)*, pages 375–382, 2004.

[8] M. Baker, K. Kimberly, and M. Sean. Why traditional storage systems do not help us save stuff forever. Technical report, HP Labs 2005 Technical Reports, 2005. HPL-2005-120.

[9] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, May 2001.

[10] E. Bertino, R. Ferrini, A. Musci, F. Paci, and K. J. Steuer. A federated digital identity management approach for business processes. In *Proceedings of 4th International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom 2008)*, 2008.

[11] C. Bizer. D2r map - a database to rdf mapping language. In *Posters World Wide Web Conference 2003 (WWW2003)*, 2003.

[12] M. Blaze, J. Feigenbaum, and M.Strauss. Compliance checking in the policymaker trust management system. In *Proceedings of the International Conference on Financial Cryptography*, pages 254–274, 1998.

[13] P. Borst and H. Akkermans. An ontology approach to product disassembly. In *Proceedings of EKAW 1997*, Lecture Notes in Computer Science, pages 33–48. Springer, 1997.

[14] D. Brickley and R. V. Guha. *RDF Vocabulary Description Language 1.0: RDF Schema*, 2004. W3C Recommendation 2004, http://www.w3.org/TR/rdf-schema.

[15] M. Burstein, C. Bussler, T. Finin, M. N.Huhns, M. Paolucci, A. P. Sheth, S. Williams, and M. Zaremba. A semantic web services architecture. *IEEE Internet Computing*, 1089-7801(5):72–81, 2005.

[16] E. Camon, M. Magrane, D. Barrell, V. Lee, E. Dimmer, J. Maslen, D. Binns, N. Harte, R. Lopez, and R. Apweiler. The gene ontology annotation (goa) database: sharing knowledge in uniprot with gene ontology. *Nucleic Acids Research*, 32:262–266, 2004.

[17] A. Carbonaro and R. Ferrini. Concepts-based content analysis for semantic recommendations. In *Proceedings of the ECAI 2006 Workshop on Recommender Systems - 17th European Conference on Artificial Intelligence (ECAI 2006)*, 2006.

[18] A. Carbonaro and R. Ferrini. Ontology-based video annotation in multimedia entertainment. In *Proceedings of the 3rd IEEE International Workshop on Networking Issues in Multimedia Entertainment (NIME'07) - 4th IEEE Communications and Networking Conference (CCNC 2007)*, 2007.

[19] A. Carbonaro and R. Ferrini. The use of concepts to improve content analysis in a distance learning system. *Int. Journal Cont. Engineering Education and Lifelong Learning*, 17(4-5):369–380, 2007.

[20] S. Castano, A. Ferrara, and G. Messa. Islab hmatch results for oaei 2006. In *Proc. of International Workshop on Ontology Matching, collocated with the 5th International Semantic Web Conference ISWC-2006*, Athens, Georgia, USA, November 2006.

[21] J. Crampton. Specifying and enforcing constraints in role-based access control. In *Proceedings of 8th ACM Symposium on Access Control Models and Technologies*, pages 43–50, 2003.

[22] J. Crampton. XACML and role-based access control. *Presentation at DIMACS Workshop on Security of Web Services (DIMACS 2005)*, 2005.

[23] L. Cranor, M. Langheinrich, M. Marchiori, M. Presler-Marshall, and J. Reagle. *The Platform for Privacy Preferences 1.0 (P3P1.0) Specification*, 2002. W3C Recommendation 2002, http://www.w3.org/TR/P3P/.

[24] E. Damiani, S. D. C. di Vimercati, C. Fugazza, and P. Samarati. Extending policy languages to the semantic web. In *Proceedings of ICWE 2004*, pages 330–343, 2004.

[25] M. Dean and G. Schreiber. *OWL Web Ontology Language Guide*, 2004. W3C Recommendation 2004, http://www.w3.org/TR/owl-guide/.

[26] H. H. Do and E. Rahm. Coma a system for flexible combination of schema matching approaches. In *Proceedings of VLDB*, pages 610–621, 2002.

[27] A. H. Doan, J. Madhavan, P. Domingos, and A. Halevy. Ontology matching: a machine learning approach. *Handbook of ontologies, International handbooks on information systems*, pages 385–404, 2004.

[28] S. Egelman, L. F. Cranor, and A. Chowdhury. An analysis of p3penabled web sites among top-20 search results. In *Proceedings of the 8th international conference on Electronic commerce (ICEC06)*, pages 197–207, 2006.

[29] M. Ehrig, P. Haase, M. Hefke, and N. Stojanovic. Similarity for ontologies - a comprehensive framework. In *Workshop Enterprise Modelling and Ontology: Ingredients for Interoperability, at PAKM 2004*, 2005.

[30] M. Ehrig and S. Staab. Qom - quick ontology mapping. In *Proceedings of 3rd International Semantic Web Conference (ISWC 2004)*, 2004.

[31] C. Fellbaum. *WordNet: An Electronic Lexical Database*. MIT Press, 1998.

[32] D. Fensel and C. Bussler. The web service modeling framework (wsmf). *Electronic Commerce Research and Applications*, 1(2):113–137, 2002.

[33] D. Ferraiolo and R. Kuhn. Role-based access controls. In *15th NIST-NCSC National Computer Security Conference*, pages 554–563. IEEE Press, October 1992.

[34] T. Finin, A. Joshi, L. Kagal, J. Niu, R. Sandhu, W. H. Winsborough, and B. Thuraisingham. ROWLBAC - Representing Role Based Access Control in OWL. In *Proceed-*

ings of the 13th Symposium on Access Control Models and Technologies. ACM Press, June 2008.

[35] K. Fisler, S. Krishnamurthi, L. A. Meyerovich, and M. C. Tschantz. Verification and change-impact analysis of acces scontrol policies. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 196–205, 2005.

[36] F. T. Fonseca and M. J. Egenhofer. Ontology-driven geographic information systems. In *Proceedings of the 7th ACM international symposium on Advances in geographic information systems (GIS '99)*, pages 14–19. ACM, 1999.

[37] R. Gavriloaie, W. Nejdl, D. Olmedilla, K. Seamons, and M. Winslett. No registration needed: How to use declarative policies and negotiation to access sensitive resources on the semantic web. In *European Semantic Web Symposium*, 2004.

[38] C. Goble and D. D. Roure. The grid: an application of the semantic web. *ACM SIGMOD Special section on semantic web and data management*, 31(4):65–70, 2002.

[39] D. P. Guelev, M. Ryan, and P. Schobbens. Model-checking access control policies. In *Proceedings of the 7th Information Security Conference (ISC)*, pages 219–230, 2004.

[40] T. Hoad and J. Zobel. Methods for identifying versioned and plagiarized documents. *Journal of the American Society for Information Science and Technology*, 54(3):203–215, 2003.

[41] J. E. Hopcroft and J. D. Ullman. *Introduction to automata theory, languages and computation*. Addison Wesley, 1979.

[42] M. Horridge, H. Knublauch, A. Rector, R. Stevens, and C. Wroe. Building owl ontologies using the protg-owl plugin and co-ode tools (edition 1.0). Technical report, University Of Manchester, 2004. http://www.co-ode.org/resources/tutorials/ProtegeOWLTutorial.pdf.

[43] I. Horrocks. Daml+oil: A description logic for the semantic web. *IEEE Data Engineering Bulletin*, 25(1):4–9, 2002.

[44] W. Hu, N. Jian, Y. Qu, and Y. Wang. Gmo: A graph matching for ontologies. In *Proceedings of K-CAP Workshop on Integrating Ontologies*, pages 41–48, 2005.

[45] W. Hu and Y. Qu. Falcon-ao: A practical ontology matching system. *Web Semantics: Science, Services and Agents on the World Wide Web*, 6(3):237–239, 2008.

[46] J. Hunter. Enhancing the semantic interoperability of multimedia through a core ontology. *Circuits and Systems for Video Technology, IEEE Transactions on*, 13(1):49–58, Jan 2003.

[47] D. Jackson. Alloy: a lightweight object modelling notation. *ACM Transaction on Software Engineering and Methodologies (TOSEM)*, 11(2):256–290, 2002.

[48] N. Jian, W. Hu, G. Cheng, and Y.Qu. Falcon-ao: Aligning ontologies with falcon. In *Proceedings of K-CAP Workshop on Integrating Ontologies*, pages 85–91, 2005.

[49] L. Kagal, T. Berners-Lee, D. Connolly, and D. Weitzner. Using semantic web technologies for policy management on the web. In *21st National Conference on Artificial Intelligence (AAAI)*, 2006.

[50] L. Kagal, T. Finin, and A. Joshi. A policy language for a pervasive computing environment. In *IEEE 4th InternationalWorkshop on Policies for Distributed Systems and Networks*, 2003.

[51] E. Kalfaoglou and M. Schorlemmer. Ontology mapping: the state of the art. *The Knowledge Engineering Review*, 18(1):1–31, 2003.

[52] M. Knechtel, J. Hladik, and F. Dau. Using owl dl reasoning to decide about authorization in RBAC. In *OWLED '08: Proceedings of the OWLED 2008 Workshop on OWL: Experiences and Directions*, 2008.

[53] M. Koch, L. V. Mancini, and F. P.-Presicce. On the specification and evolution of access control policies. In *Proceedings of the 2004 ACM Symposium on Applied Computing (SAC)*, pages 121–130, 2001.

[54] V. Kolovski. A logic-based framework for web access control policies. Technical report, University of Maryland, 2008.

[55] V. Kolovski and J. Hendler. Xacml policy analysis using description logics. Under submission, 2008.

[56] V. Kolovski, J. Hendler, and B. Parsia. Analyzing web access control policies. In *Proceedings of the International World Wide Web Conference WWW 2007*, pages 677–686, May 2007.

[57] D. Lin, P. Rao, E. Bertino, N. Li, and J. Lobo. Exam - a comprehensive environment for analysis of access control policies. Technical report, Department of Computer Science, Purdue University, 2007.

[58] E. Lupu and M. Sloman. Conflicts in policy-based distributed systems management. *IEEE Transactions on Software Engineering (TSE)*, 25(6):852–869, 1999.

[59] J. Madhavan, P. A. Bernstein, , and E. Rahm. Generic schema matching with cupid. In *Proceedings of the 27th International Conference on Very Large Data Bases*, pages 49–58, 2001.

[60] A. Maedche and S. Staab. Ontology learning for the semantic web. *IEEE Intelligent Systems*, 16(2):72–79, 2001.

[61] E. Maedche and S.Staab. Measuring similarity between ontologies. In *Proceedings of the European Conference on Knowledge Acquisition and Management (EKAW*, pages 251–263. Springer, 2002.

[62] F. Manola and E. Miller. *RDF Primer*, 2004. W3C Recommendation 2004, http://www.w3.org/TR/rdf-primer/.

[63] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, T.Payne, E. Sirin, N. Srinivasan, and K. Sycara. *OWL-S: Semantic Markup for Web Services*, 2004. http://www.daml.org/services/owls/1.1/overview/.

[64] P. Mazzoleni, E. Bertino, and B. Crispo. Xacml policy integration algorithms. In *Proceedings of the 11th ACM Symposium on Access Control Models and Technologies (SACMAT)*, pages 223–232, 2006.

[65] P. McDaniel and A. Prakash. Methods and limitations of security policy reconciliation. *ACM Transactions on Information and System Security (TISSEC)*, 9(3):259–291, 2001.

[66] S. A. McLlraith, T. C. Son, and H. Zeng. Semantic web service. *IEEE Intelligent Systems*, 16:46–53, 2001.

[67] D. Metzler, Y. Bernstein, W. B. Croft, A. Moffat, and J. Zobel. Similarity measures for tracking information flow. In *Proceedings of the 14th ACM international conference on Information and knowledge management (CIKM)*, pages 517–524, 2005.

[68] S. E. Middleton, N. R. Shadbolt, and D. C. D. Roure. Ontological user profiling in recommender systems. *ACM Transactions on Information Systems (TOIS)*, 22(1):54–58, 2004.

[69] J. D. Moffett and M. S. Sloman. Policy conflict analysis in distributed system management. *Journal of Organizational Computing*, 1993.

[70] D. Morr. Lionshare: A federated p2p app. In *Internet2 members meeting*, 2007.

[71] T. Moses. *Extensible access control markup language (XACML) version 2.0*, 2005. OASIS Standard.

[72] T. Moses. *Extensible access control markup language (XACML) version 2.0*, 2005. OASIS Standard.

[73] W. Nejdl, B. Wolf, C. Qu, S. Decker, M. Sintek, A. Naeve, M. Nilsson, M. Palmr, and T. Risch. Edutella: A p2p networking infrastructure based on rdf. In *Proceedings of the 11th International Conference on World Wide Web*, pages 604–615, 2002.

[74] N. Noy and M. Musen. Anchor-prompt: using non-local context for semantic matching. In *Proceedings of the workshop on Ontologies and Information Sharing at the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 63–70, 2001.

[75] B. Parsia and E. Sirin. Pellet: An owl dl reasoner. In *Third International Semantic Web Conference - Poster*, 2004.

[76] V. Raghavan and S. Wong. A critical analysis of vector space model for information retrieval. *Journal of the American Society for Information Science*, 37(5):279–287, 1986.

[77] E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *The VLDB Journal*, 10(4):334–350, December 2001.

[78] D. D. Roure and J. A. Hendler. E-science: The grid and the semantic web. *IEEE Intelligent Systems*, 19(1):65–71, 2004.

[79] R. Sandhu, E. J. Coyne, H. L. Feinsteinand, and C. E. Youman. Role-based access control models. *IEEE COMPUTER*, 29(2):38–47, 1996.

[80] P. Shvaiko and J. Euzenat. A survey of schema-based matching approaches. *Journal on Data Semantics IV*, pages 146–171, 2005.

[81] B. Simon, Z. Miklos, W. Nejdl, M. Sintek, and J. Salvachua. Elena: A mediation infrastructure for educational services. In *Proceedings of the 12th International World Wide Web Conference*, 2003.

[82] V. C. Storey, R. H. L. Chiang, and G. L. Chen. Ontology creation: Extraction of domain knowledge from web documents. In *Prooceedings 24th International Conference on Conceptual Modeling (ER 2005)*, pages 256–269, 2005.

[83] H. Tangmunarunkit, S. Decker, and C. Kesselman. Ontology-based resource matching in the grid the grid meets the semantic web. *The SemanticWeb - ISWC 2003, LNCS*, 2780/2003:706–721, 2003.

[84] Q. T. Tho, S. C. Hui, and T. H. C. A. C. M. Fong. Automatic fuzzy ontology generation for semantic web. *IEEE Transactions on Knowledge and Data Engineering*, 18(6):842–856, 2006.

[85] Y. A. Tijerino, D. W. Embley, D. W. Lonsdale, and G. Nagy. Ontology generation from tables. In *Proceedings of the Fourth International Conference on Web Information*

Systems Engineering (WISE 03), page 242, Washington, DC, USA, 2003. IEEE Computer Society.

[86] Q. Trinh, K. Barker, and R. Alhajj. Semantic interoperability between relational database systems. In *IDEAS '07: Proceedings of the 11th International Database Engineering and Applications Symposium*, pages 208–215, Washington, DC, USA, 2007. IEEE Computer Society.

[87] C. Tsinaraki, P. Polydoros, and S. Christodoulakis. Interoperability support for ontology-based video retrieval applications. *Image and Video Retrieval*, 3115/2004:582–591, 2004.

[88] M. Uschold and M. Gruninger. Ontologies and semantics for seamless connectivity. *SIGMOD Rec.*, 33(4):58–64, 2004.

[89] A. Uszok, J. M. Bradshaw, R. Jeffers, N. Suri, P. J. Hayes, M. R. Breedy, L. Bunch, M. Johnson, S. Kulkarni, and J. Lott. Kaos policy and domain services: Toward a description-logic approach to policy representation, deconfliction, and enforcement. In *Proceedings of IEEE Fourth International Workshop on Policy (Policy 2003)*, pages 93–98. IEEE Computer Society, 2003.

[90] A. Uszokand and J. Bradshaw. Kaos policies for web services. In *W3C Workshop on Constraints and Capabilities for Web Servies*, October 2004.

[91] D. J. Weitzner, J. Hendler, T. Berners-lee, and D. Connolly. Creating the policy-aware web: Discretionary, rules-based access for the world wide web. In *In Web and Information Security, E. Ferrari and B. Thuraisingham (Eds), IRM Press*. Press, 2005.

[92] A. K. Y. Wong, P. Ray, N. Parameswaran, and J. Strassner. Ontology mapping for the interoperability problem in network management. *IEEE Journal on Selected Areas in Communications*, 23(10):2058–2068, 2005.

[93] N. Zhang, M. Ryan, and D. P. Guelev. Evaluating access control policies through model checking. In *Proceedings of the 8th Information Security Conference (ISC)*, pages 446–460, 2005.

[94] C. N. Ziegler. Semantic web recommender systems. *Current Trends in Database Technology - EDBT 2004 Workshops, LNCS*, 3268/2004:74–89, 2004.

Appendix A

Interoperability in Digital Identity Management

In this chapter we provide an application of ontology mapping in solving naming heterogeneity in an Identity Management Scenario. The approach described here has been proposed in [10]. The rest of this Chapter is organized as follows. In Section A.1 we introduce the problem related to Identity Management in Business Process. Section A.2 introduces a running example that is used throughout the rest of the Sections to illustrate the discussion. Section A.3 discusses the main issues related to digital identity management for business processes. Section A.4 introduces the notions on which our multi-factor identity attribute verification protocol is based. Section A.5 presents the multi-factor identity attribute verification protocol. Section A.6 discusses the system architecture. Section A.7 reports experimental results. Finally, Section A.8 concludes the Chapter and outlines some future work.

A.1. Introduction

Business processes have gained a lot of attention because of the pressing need for integrating existing resources and services to better fulfill customer needs. A key feature of business processes is that they are built from composable services, referred to as *component services*, that may belong to different domains. In such a context, flexible multi-domain identity management solutions are crucial for increased security and user-convenience. In particular, it is important that during the execution of a business process the component services be able to verify the identity of the client to check that it has the required permissions for accessing the services. Clients identity consists of data, referred to *identity attributes*, that encode relevant-security properties of the clients. The management of identity attributes in business processes raises however a number of challenges. On one hand, to enable authentication, the propagation of client's identity attributes across the component services should be facilitated. On the other hand, identity attributes need to be protected as they may convey sensitive information about a client and can be target of attacks. Moreover, because business processes orchestrate the functions of services belonging to different domains, interoperability issues may arise in client authentication processes. Such issues range from the use of different identity tokens and different identity negotiation protocols, such as the client-centric protocols and the identity-providers centric protocols, to the use of different names for identity attributes. The use of different names for identity attributes, that we refer to as *naming heterogeneity*, typically occurs because clients and component services use a different vocabulary to denote identity attribute names. In this case, whenever a component service requests from a client a set of identity attributes to verify its identity, the client may not understand which identity attributes it has to provide.

To address the problem of multi-domain identity management, we propose a multi-factor identity attribute verification protocol for business processes that assures clients privacy and handles naming heterogeneity. The protocol uses an identity attribute names matching technique based on look-up tables, dictionaries and ontology mapping, to match component services and clients vocabularies and aggregated zero knowledge proofs of knowledge (AgZKPK) cryptographic protocol to allow clients to prove with a single interactive proof the knowledge of multiple identity attributes without the need to provide them in clear.

A.2. Running Example

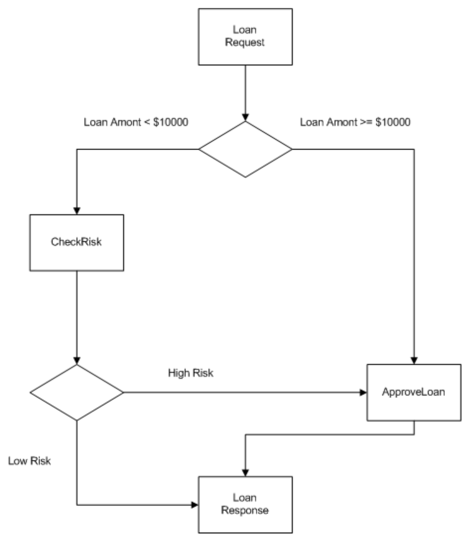


Figure A.1: A loan approval process specification

In this section we introduce an example of business process that implements a loan approval process (see Figure A.1). Customers of the service send loan requests. Once a request is received, the loan service executes a simple process resulting in either a "loan approved" message or a "loan rejected" message. The decision is based on the amount requested and the risk associated with the customer. For amounts lower than $10,000\$$ a streamlined process is used. In the streamlined process low-risk customers are automatically approved. For higher amounts, or medium and high-risk customers, the credit request requires further processing. For each request, the loan service uses the functiona provided by two other services. In the streamlined process, used for low amount loans, a *risk assessment* service is used to obtain a quick evaluation of the risk associated with the customer. A full *loan approval* service (possibly requiring direct involvement of a loan expert) is used to obtain an assessment about the customer when the streamlined approval process is not applicable.

Four main activities are involved in the process:

- `Loan Request` allows a client to submit a loan request to the bank

- `Check Risk` (provided by *risk assessment* service) computes the risk associated with the loan request
- `Approve Loan` (provided by *loan approval* service) determines if the loan request can be approved or rejected
- `Loan Response` sends to the client the result of the loan request evaluation process

risk assessment and *loan approval* services require a set of identity attributes from the client who has submitted the loan request. The risk assessment service asks DrivingLicense, CarRegistration and EmployeeID, whereas the loan approval service requires EmployeeID and CreditCard.

A.3. Identity Management for Business Processes

Managing and verifying clients identity in a business processes raise a number of challenging issues. A first issue is related to how the client's identity attribute have to be managed within the business process. The client of a business process is not aware that the business process that implements the required service invokes some component services. The client thus trusts the composite service but not the component services. Therefore, every time the component services have to verify the client's identity, the composite service has to act as an intermediary between the component services and the client. Moreover, since the client's identity attributes may contain sensitive information and clients usually do not trust the component services, the client's identity attributes should be protected from potential misuse by component services.

Another issue is related to how the identity verification process is performed. Because component services belong to different domains, each with its own identity verification policies, the sets of identity attributes required to verify client's identity may partially or totally overlap. Therefore, the client has to prove several times the knowledge of the same subset of identity attributes. It is thus important to take advantage of previous client identity verification processes that other component services have performed.

Finally, another issue is the lack of interoperability because of naming heterogeneity. Naming heterogeneity occurs when component services define their identity verification policies according to a vocabulary different from the one adopted by clients. Therefore, component services and clients are not able to have "meaningful" interactions because they do not understand each other. Thus, it is also necessary that client identity verification process supports an approach to match identity attribute names of component services and clients vocabularies. In such respect, a first question to be addressed is which matching technique to use, which in turn depends from the types of variation in identity attribute names. A second question is related to the matching protocol to use, that is, by which party the matching has to be performed and whether the fact that a client has already performed a matching with a component service may help in a subsequent matching.

To address such issues we propose a multi-factor identity attribute verification protocol for business processes that supports a privacy usage of clients identity attributes and that guarantees interoperable interactions between clients and component services. In what follows, we provide more details about our approach.

A.4. Preliminary concepts

To enable multi-factor identity attribute verification, clients have to register their identity attributes to a *registrar*. The registrar is an additional component in digital identity management systems that stores and manage information related to identity attributes. For each client's identity attribute m, the registrar records an identity tuple $(\sigma_i, M_i, tag, validity{-}assurance, ownership{-}assurance, \{W_{ij}\})$. Each identity tuple consists of tag, an attribute descriptor, the Pedersen commitment of m, denoted as M_i, the signature of the registrar on M, denoted as σ_i, two types of assurance, namely *validity assurance* and *ownership assurance* and a set of weak identifiers $\{W_{ij}\}$. M_i is computed as $g^m h^r$, where g and h are generators in a group G of prime order q. G and q are public parameters of the registrar and r is chosen randomly from $\mathbb{Z}_q$. Validity assurance corresponds to the confidence about the validity of the identity attribute based on the verification performed at the identity attribute's original issuer. Ownership assurance corresponds to the confidence about the claim that the principal presenting an identity attribute is its true owner.

Weak identifiers are used to denote identity attributes that can be aggregated together to perform multi-factor authentication. The identity tuples of each registered client can be retrieved from the registrar by the component services or the registrar can release to the client a certificate containing its identity record.

We assume that each of the component services define their identity verification policies by specifying a set of identity attribute names that have to be required from the client. Because of naming heterogeneity, clients may not understand component services identity verification policies. The type of variations that can occur in clients and component services identity attribute names can be classified in: *syntactic*, *terminological* and *semantic* variations.

- *Syntactic variations* arise because of the use of different character combinations to denote the same term. An example is the use of "CreditCard" and "Credit_Card" to denote a client's credit card.

- *Terminological variations* refer to the use of different terms to denote the same concept. An example of terminological variation is the use of the synonyms "Credit Card" and "Charge Card" to refer a client's credit card.

- *Semantic variations* are related to the use of two different concepts in different knowledge domains to denote the same term.

Syntactic variations can be identified by using look up tables. A look up table enumerates the possible ways in which the same term can be written by using different character combinations. In detecting terminological variations, dictionaries or thesaurus such as WordNet can be exploited. Finally, semantic variations can be determined by using ontology matching techniques. An ontology is a formal representation of a domain in terms of concepts and properties with which those concepts are related. Ontologies can be exploited to define a domain of interest and for reasoning about its features. Ontology mapping is the process whereby two ontologies are semantically related at conceptual level; source ontology concepts are mapped onto the target ontology concepts according to those semantic relations [3, 7]. Typically an ontology matching algorithm takes in input two ontologies O_i and O_j, and returns a set of triples of the form $\langle c_i, c_j, s\rangle$, where c_i is a concept belonging to ontology O_i, c_j is a concept belonging to ontology O_j that matches concept c_i, and s is a *confidence score*, that is, a value between 0 and 1, indicating the similarity between the matched concepts.

To enable the matching of identity attributes by using the above techniques, we make the following assumptions. Component services' identity verification policies are defined according to their domain vocabulary ontology. Moreover, they track existing mappings with other component services' ontologies. Such mappings are formally represented by tuples of the following form:

$$\langle O_{CS}, CS', O_{CS'}, \{\langle c_1, c_2, s_{1,2}\rangle, \ \ldots \ , \langle c_l, c_m, s_{l,m}\rangle\}\rangle$$

where O_{CS} is the ontology of a component service CS, CS' is a component service whose ontology $O_{CS'}$ matches ontology O_{CS} and $\{\langle c_1, c_2, s_{1,2}\rangle, \ \ldots \ , \langle c_l, c_m, s_{l,m}\rangle\}$ is the set of concepts mappings $\langle c_i, c_j, s_{i,j}\rangle$ where $c_i \in O_{CS}$ and $c_j \in O_{CS'}$.

Moreover, each component service keeps a look up table containing alternative character combinations and store a set of synonyms, denoted as *Synset*, for each of the identity attribute names used for expressing its identity verification policies. Finally, since we want to avoid that the client proves several times the possession of a same set of identity attributes, we assume that component services have a PKI infrastructure that allows them to issue certificates to clients. These certificates (see Definition 1 below) assert that an identity attribute by a client matches an identity attribute by a component service and that the component service has verified that the client owns the attribute. Clients can use these certificates to prove that they own a set of identity attributes without going through the authentication process during the execution of the same business process instance in which the certificates have been released. Instead, clients can use the certificates in business processes different from the one in which the certificate have been issued to prove there is a mapping between a set of client's attributes and a service's ontology. This distinction is motivated by the fact that there is a trust relationship between the component services in the same business process instance, that may not exist with services external to the process.

Definition A.1 (Proof-of-Identity Certificate) *Let* S *be a component service participating to a business process* BP *and* C *be a client. Let* O_S *be the ontology describing the domain of* S *and* $\mathsf{AttrSet}$ *be the set of* C*'s identity attribute names. The proof of identity certificate released by* S *to* C *upon a successful verification is a tuple* $\langle \mathsf{Issuer}, \mathsf{Owner}, \mathsf{OID}, \mathsf{Mappings}, \mathsf{Is}$
where: Issuer *is the identifier of* S*,* Owner *is the identifier of* C*,* OID *is* O_S *ontology identifier,* $\mathsf{Mappings}$ *is a set of tuples of the form* $\langle \mathsf{Attr}, \mathsf{Concept} \rangle$ *where* $\mathsf{Attr} \in \mathsf{AttrSet}$ *and* $\mathsf{Concept} \in O_S$*, and* $\mathsf{IssuanceDate}$ *is the release date of the certificate.*

Besides being stored by the clients, proof-of-identity certificates released during the execution of a business process instance are stored in a local repository, denoted as $\mathsf{CertRep}$, by the composite service for the whole process execution.

A.5. Interoperable Multi-Factor Authentication

In this section, we present a multi-factor authentication protocol for business processes. The protocol takes place between a client, the composite service and a component service. Since the client is not aware of the component services, the composite service has to mediate the interactions between them. The protocol consists of two phases that make use of the notion of proof-of-identity certificate introduced in the previous section (see Figure A.2). In the first phase, the component service matches the identity attributes of clients vocabulary with its own attributes to help the client understand its identity verification policy. In the second phase, the client carries out an aggregate ZKPK protocol to prove to the component service the knowledge of the matched identity attributes. Algorithm 1 summarizes the different phases of the protocol.

A.5.1 Identity attribute matching protocol

The technique that we have developed for matching identity attribute names from different vocabularies is based on the combined use of look-up tables, dictionaries, and ontology mapping.

As we have already mentioned, an important issue is which party has to execute the matching. In our context, the matching can be executed by the client, the composite service or the component services. Performing the matching at the client has the obvious drawback that the client may lie and asserts that an identity attribute referred to in the component services policy matches one of its attribute, whereas this is not the case. The matching process cannot be performed by the composite service because it should have access to information which are local to the component services. Therefore, in our approach, the matching is performed by the component services. Notice that because of the privacy-preserving protocol that we use (see next section), the composite service and the component services will not learn the values of the identity attributes of the client and therefore do not have incentives to lie.

A second issue is how to take advantage of previous interactions that the client has performed with other component services. It is also important to exploit mappings that can exist between ontologies by different component services. To address such issue, the matching protocol relies on the use of the proof-of-identity certificates and matching techniques. We assume that $\mathtt{AttrProof}$ is the set of identity attributes that a component service asks to a client to verify its identity. The identity attribute name matching process is carried out between the client, the component service and the composite service when some attributes in $\mathtt{AttrProof}$

do not match any of the attributes in $AttrSet$, the set of clients' identity attributes. We refer to the set of component service's identity attributes that do not match a client attribute name to as $NoMatchingAttr$.

Algorithm 7: `Multi-factor verification protocol`

```
Input: $CertRep$: proof-of-identity certificates repository
$AttrProof$: set of identity attributes requested from the client $c_i$: proof-of-identity certificate
Output:

(1)  FOR EACH $a_i \in AttrProof$
(2)      IF $\exists c_j \in CertRep$ such that $c_j$ prove the knowledge of $a_i$
(3)          THEN $a_i$ is verified
(4)      ELSE
(5)         Match $a_i$ with client's proof-of-identity certificates
(6)         Verify AgZKPK
(7)         Release new proof-of-identity certificate $c_i$
(8)         Store $c_i$ in $CertRep$
```

The matching process consists of two main phases. The goal of the first phase is to match the identity attributes that have syntactical and terminological variations. During this phase, the component service sends to the composite service, for each identity attribute a_i in the $NoMatchingAttr$, the set $Synset_i$ that contains a set of alternative character combinations and a set of synonyms. Thus, the composite service sends the sets $Synset_i$ to the client. The client verifies that for each identity attribute a_i, there is an intersection between $Synset_i$ and $AttrSet$. If this is the case attribute a_i is removed from $NoMatchingAttr$. Otherwise, if $NoMatchingAttr$ is not empty, the second phase is performed. During the second phase the client sends $CertSet$, the set of its proof-of-identity certificates to the composite service that forwards them to the component service. Thus, in the second phase of the matching process the component service tries to match the

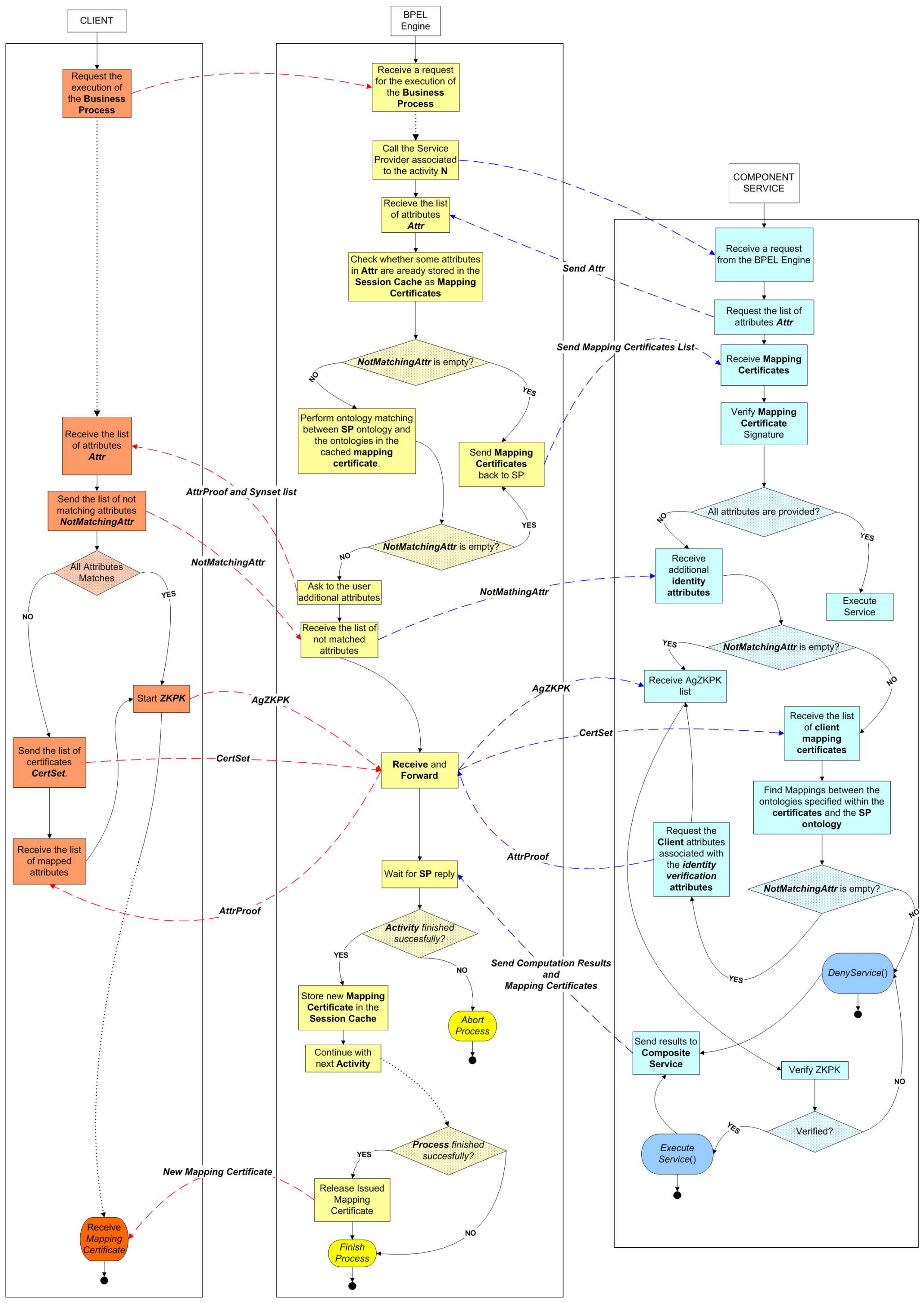

CLIENT
BPEL Engine
COMPONENT SERVICE
Request the execution of the Business Process
Receive a request for the execution of the Business Process
Call the Service Provider associated to the activity N
Recieve the list of attributes Attr
Check whether some attributes in Attr are aready stored in the Session Cache as Mapping Certificates
Receive a request from the BPEL Engine
Send Attr
Request the list of attributes Attr
Send Mapping Certificates List
Receive Mapping Certificates
NotMatchingAttr is empty?
NO
YES
Verify Mapping Certificate Signature
Perform ontology matching between SP ontology and the ontologies in the cached mapping certificate.
Send Mapping Certificates back to SP
Receive the list of attributes Attr
AttrProof and Synset list
Send the list of not matching attributes NotMatchingAttr
All attributes are provided?
NO
YES
NotMatchingAttr is empty?
NO
YES
NotMatchingAttr
All Attributes Matches
Ask to the user additional attributes
Receive additional identity attributes
Execute Service
NotMathingAttr
YES
NO
Receive the list of not matched attributes
NotMatchingAttr is empty?
YES
NO
Start ZKPK
AgZKPK
AgZKPK
Receive AgZKPK list
Receive the list of client mapping certificates
CertSet
Send the list of certificates CertSet.
CertSet
Receive and Forward
Find Mappings between the ontologies specified within the certificates and the SP ontology
Request the Client attributes associated with the identity verification attributes
AttrProof
Receive the list of mapped attributes
Wait for SP reply
NotMatchingAttr is empty?
NO
AttrProof
Activity finished succesfully?
YES
NO
Send Computation Results and Mapping Certificates
YES
DenyService()
Store new Mapping Certificate in the Session Cache
Abort Process
Continue with next Activity
Send results to Composite Service
Verify ZKPK
NO
YES
Verified?
Process finished succesfully?
YES
New Mapping Certificate
Release Issued Mapping Certificate
Execute Service()
Receive Mapping Certificate
NO
Finish Process

Figure A.2: Approach schema

concepts corresponding to the identity attributes the client is not able to provide with concepts from the ontologies of the services which have issued the proof-of-identity certificates. The mappings can be determined by mapping the ontology of the component service with the ontologies of the services that have released the certificates. Those other services are referred to as semantic neighbors of the component service. Only matches that have a confidence score s greater than a predefined threshold are selected. The acceptance threshold is set up by the component service to assess the matches' validity. The greater the threshold, the greater is the similarity between the two concepts and thus higher is the probability that the match is correct. If the component service is able to find mappings for its concepts, it then verifies by using the information in the proof-of-identity certificates that each matching concept matches a client's attribute $Attr$. If this check fails, the component service notifies the composite service that terminates the interaction with the client.

A.5.2 Multi-factor authentication

Once the client receives $Match$, the set of matched identity attributes from the composite service, it retrieves from the registrar or from its $RegCert$ the commitments M_i satisfying the matches and the corresponding signatures σ_i. The client aggregates the commitments by computing $M = \prod_{i=1}^{n} M_i = g^{m_1+m_2+...+m_i} h^{r_1+r_2+...+r_i}$ and the signatures into $\sigma = \prod_{i=1}^{n} \sigma_i$, where σ_i is the registrar 's signature on the committed value $M_i = g^{m_i} h^{r_i}$. According to the ZPK protocol, the client randomly picks y, s in $[1,..q]$, computes $d = g^y h^s \pmod{p}$, and sends d, σ, M, M_i , $1 \leqslant i \leqslant t$, to the composite service that on in turn sends these values to the component service. The component service sends back a random challenge $e \in [1,...,q]$ to the client. Then the client computes $u = y + em \pmod{q}$ and $v = s + er \pmod{q}$ where $m = m_1 + \ \dots\ m_t$ and $r = r_1 + \ \dots\ r_t$ and sends u and v to the composite service.

Algorithm 8: `Verification()`

```
Input:
Output:

(1)  C: Receive($Match$)
(2)  C: $AttrMathces$.Add($Match$)
(3)  C: FOR EACH $\langle Attr, Id_i \rangle \in AttrMatches$
(4)  C:     $\{M_i, \sigma_i\} :=$ Select($RegCert, Attr$)
(5)  C: $M = \prod_{i=1}^{n} M_i$
(6)  C: randomly picks $y$, $s \in [1..q]$
(7)  C: $d = g^y h^s$ (modp)
(8)  C: Send($\{M_1, \ldots, M_n\}, \{\sigma_1, \ldots, \sigma_n\}, M, \sigma, d$)

(9)  CS: Receive($\{M_1, \ldots, M_n\}, \{\sigma_1, \ldots, \sigma_n\}, M, \sigma, d$)
(10) CS: randomly picks $e \in [1..q]$
(11) CS: Send($e$)

(12) C: Receive($e$)
(13) C: $u := y + em$ (mod q) where $m = m_1 + m_2 + +m_n$
(14) C: $w := s + er$ (mod q) where $r = r_1 + r_2 + \ldots + r_n$
(15) C: Send($u, w$)

(16) CS: Receive($u, w$)
(17) CS: IF ($g^u h^w == dM^k$ (mod p) $\wedge \sigma == \prod_{i=1}^{t} \sigma_i$)
(18) CS:     Execute(S);
(19) CS:     IssueCertificate();
(20) CS: ELSE
(21) CS:     Send(Service Denied)
```

The composite service forwards u and v to the component service. The component service accepts the aggregated zero knowledge proof if $g^u h^v = dc^e$. If this is the case, the component service checks that $\sigma = \prod_{i=1}^{n} \sigma_i$. If also the aggregate signature verification succeeds, the component service releases a proof of identity certificate to the client. The certificate states that client's identity attributes in the $Match$ set are mapped onto concepts of the component service ontology and that the client has successfully proved the knowledge of those attributes. The composite service sends the proof-of-identity certificate to the client and stores a copy of the certificate in its local repository $CertRep$. The proof-of-identity certificate can be used by the client to prove the knowledge of an attribute without performing the aggregate ZKP protocol with another component service.

A.6. System architecture and Implementation

In this section we discuss the system architecture that supports our multi-factor identity attributes authentication for business processes. We assume that our processes are implemented as WS-BPEL business processes, that is, as business processes in which each component service is implemented by a Web service. The main components of the architecture are: the BPEL engine, the `Identity Attribute Requester` module, the `Client`, the `Registrar` , the `Identity Verification Handler` module, and the component Web services. The WS-BPEL engine is responsible for scheduling and synchronizing the various activities within the business process according to the specified activity dependencies, and for invoking Web services operations associated with activities. The `Identity Attribute Requester` module extends the WS-BPEL engine's functions by carrying on the communication with the client asking for new identity attributes whenever necessary. The `Identity Attribute Requester` keeps in a local repository the mapping certificate associated with previous clients identity verifications. The `Client` supports the functions to trigger the execution of the WS-BPEL business process, to select the identity attributes matching the ones requested by the component services, and to generate the aggregate ZKP of the matched attributes. The `Registrar` component provides functions for storing the clients' identity records and retrieving the public parameters required in

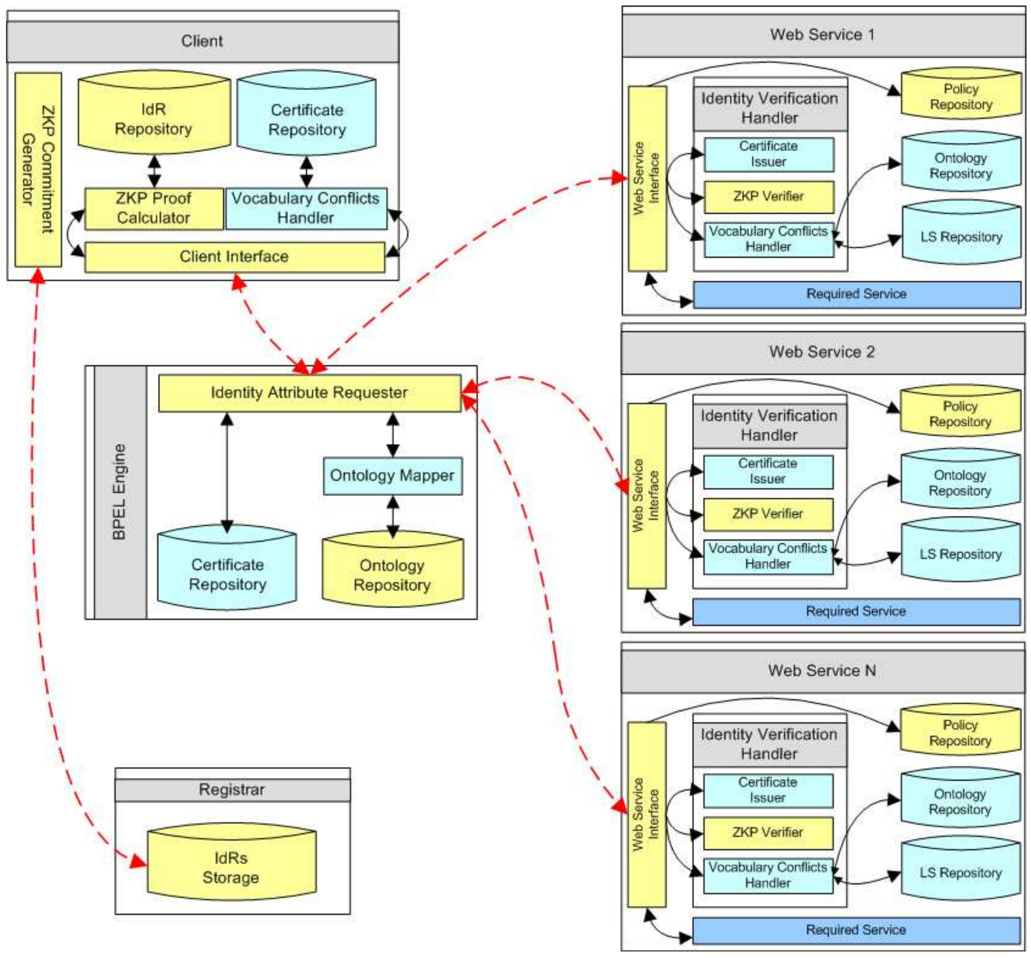


Figure A.3: System architecture

the AgZKPK protocol. The `Identity Verification Handler` intercepts the components services invocation messages and provides functions for matching client identity attribute names and performing the aggregate ZKP verification. Finally, the component Web services support the operations that are orchestrated by the business process.

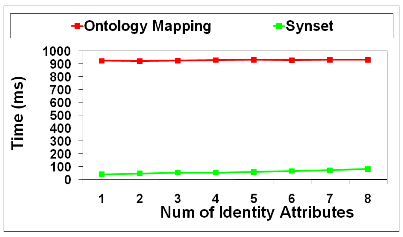
The `Identity Attribute Requester`, the `Identity Verification Handler` modules, and the component Web services have been implemented in JAVA. The `Identity Verification Handler` implements the identity attribute name matching protocol using the Falcon-AO v0.7 [1, 2] ontology mapping API and

WordNet 2.1 English Lexical database [9]. The `Client` application has been implemented in JSP while the `Registrar` has been implemented as a JAVA servlet. As BPEL engine we have chosen ODE. Finally, we have used Oracle 10g DBMS to store clients' identity records, ontology mappings, set of synonyms, session data and mapping certificates.

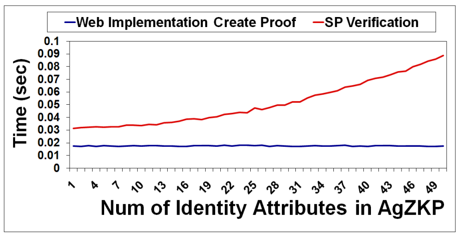
Moreover, we have adopted ODE as BPEL engine. Oracle 10g DBMS is used to store clients' identity records, ontology mappings, set of synonyms, session data and mapping certificates. Among these components the most relevant one is the BPEL Orchestrator in that it provides a number of operations to manage both identity management and naming heterogeneity according to the protocol presented in the above Sections. Such component is a logically independent component, as shown in Figure A.3; it can however be physically instantiated on the same hosts supporting the other components. For example, a host supporting a Web service may also include one such instance. The BPEL Orchestrator provides the following features:

- orchestrates the Web services that are involved into the process;
- carries on the communication with the client asking for new credentials whenever necessary;
- implements the protocol for solving naming heterogeneity using the Falcon-AO v0.7 [1, 2] ontology mapping tool and by exploiting the WordNet 2.1 English Lexical database [9];

All the mapping certificates regarding the client that has instantiated the actual process are maintained within the Session Cache. During the execution of a process activity, all the certificates are retrieved and sent to the associated Web service. When the execution of the activity terminates, the Web service sends back the new mapping certificates, if any, released to the client and the Session Cache is updated with the new data.



(a) Heterogeneity evaluation



(b) AgZKPK Verification versus Creation

Figure A.4: Experimental results

A.7. Experimental Evaluation

We have performed several experiments to evaluate the AgZKPK process that characterize the proposed approach to multi-factor identity verification and the identity attribute names matching process. To execute the tests we have developed a BPEL process composed by four component Web services and we have created a set of ontologies with an average cardinality of 60 concepts. We have carried out the following experimental evaluations:

- we have measured the time taken by a component Web service to perform the two different phases of the identity attribute names matching process by varying the number of identity attributes that have to be matched from 1 to 8. (Figure A.4(a));
- we have measured the time taken by a component Web service to generate the aggregate ZKP by varying the number of identity attributes being aggregated from 1 to 50. (Figure A.4(b));
- we have measured the time taken by a component Web service for aggregate ZKP verification execution time varying the number of identity attributes being aggregated from 1 to 50. (Figure A.4(b));

The execution time has been measured in CPU time (milliseconds). Moreover, for each test case we have executed twenty trials, and the average over all the trial execution times has been computed.

Figure A.4(a) shows the execution times of the two phases of the matching protocol for varying values in the number of identity attributes verified by a component service. The execution time of the first phase (green line) slightly increases and is around 60 ms. Instead, the time of the second phase is constant because even if the number of identity attributes to be match increases, this phase performs always the same operation, that is, matching two ontologies. The execution time for the ontology matching is constant with respect to the number of identity attributes, while the execution time of the mapping path creation increases. Regarding ontology matching, the increase in computational time due to the larger number of attributes is negligible compared to the time required for the matching process. On the contrary, the time for the mapping path computation increases, as expected, because the same basic operations are repeated for a number of times equal to the number of identity attributes considered. This is the reason why the computation of a mapping path is more efficient than ontology mapping when considering a smaller number of attributes. Figure A.4(b) reports the times to create an AgZKP and to verify it for varying values in the number of identity attributes being aggregated. The execution time to generate the AgZKP (represented by the blue line in the graph) is almost constant for increasing values in the number of identity attributes. The reason is that the creation of AgZKP only requires a constant number of exponentiations. By contrast, the time that the component Web service takes to perform identity attributes verification linearly increases with the number of identity attributes to be verified. The reason is that during the verification the component Web service is required to multiply all the commitments to verify the resulting aggregate signature.

A.8. Concluding Remarks

In this Chapter we have proposed a digital identity management approach for business processes. Our approach uses a combination of techniques from the area of semantic web and security protocols. We plan to extend this work in several directions. One direction is related to deal with heterogeneous identity negotiation protocols. The second direction is related to the definition of a language for identity verification policies that would allow service providers to specify conditions on identity attributes. We also plan to extend the AgZKPK protocol to verify that identity attribute's commitments satisfies such conditions.