2011

A Framework for Analysis of Java-Based XACML Engines

Ildar Rakhmatulin

A Framework for Analysis of Java-Based XACML Engines

(Spine Title: A Framework for Analysis of Java-Based XACML Engines)

(Thesis format: Monograph)

by

Ildar Rakhmatulin

Graduate Program
in
Computer Science

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science

The School of Graduate and Postdoctoral Studies
The University of Western Ontario
London, Ontario, Canada

THE UNIVERSITY OF WESTERN ONTARIO
THE SCHOOL OF GRADUATE AND POSTDOCTORAL STUDIES

CERTIFICATE OF EXAMINATION

Supervisor:	Examination committee:
Dr. Sylvia Osborn	Dr. M. Katchabaw
	Dr. R. Solis-Oba
	Dr. A. Ouda

The thesis by

Ildar Rakhmatulin

entitled:

A Framework for Analysis of Java-Based XACML Engines

is accepted in partial fulfillment of the
requirements for the degree of
Master of Science

Date	Chair of the Thesis Examination Board

Abstract

A lot of applications enhance their security via access-control systems. XACML (eXtensible Access Control Markup Language) is a standardized policy language, which has been widely used in access-control systems. In an XACML-based access-control system, policies, requests, and responses are encoded in XACML. An XACML implementation provides functionalities to evaluate XACML requests against XACML policies.

There are many XACML libraries implemented in the Java programming language which are supposed to provide a set of Java classes that understand the XACML language, as well as the rules about how to process requests and how to manage attributes and other related data.

This thesis focuses on the performance analysis of such libraries. We first implement a framework for analysis of Java-based XACML engines. This is accomplished by creating the hierarchy of Java classes representing the main functionality of XACML engines. We then conduct experiments by means of our framework investigating performance features of such XACML engines as Sun XACML, XEngine, and Enterprise Java XACML. The proposed approach differs from previous work in the engines chosen as well as the variety of experiments conducted and their parameters.

Keywords: Java, XACML, PDP, PEP, Sun XACML, XEngine, Enterprise Java XACML

Acknowledgements

First and foremost, I would like to thank my supervisor, Prof. Sylvia Osborn, who has provided me with an unbounded amount of attention and guidance throughout my work on this thesis. Without her support and valuable suggestions, the work could not have been completed.

I am also thankful to my friends and lab colleagues at Department of Computer Science for their help and participating in interesting discussions where I received a few beneficial ideas.

Above all, I acknowledge the support of my family and relatives. The work would not be possible without their inspiration and encouragement.

Contents

List of Figures

List of Tables

List of Appendices

Glossary

CPU - Central Processing Unit

ISO/IEC - International Organization for Standardization/International Electrotechnical Commission

ITU - International Telecommunication Union

LDAP - Lightweight Directory Access Protocol

OASIS - Organization for the Advancement of Structured Information Standards

PAP - Policy Access Point

PDP - Policy Decision Point

PEP - Policy Enforcement Point

PIP - Policy Information Point

RAM - Random Access Memory

SAML - Security Assertion Markup Language

SOA - Service-Oriented Architecture

XACML - eXtensible Access Control Markup Language

XML - eXtensible Markup Language

Chapter 1

Introduction

1.1 Setting

Currently, security is one of the most important requirements that should be considered in distributed systems. Security and authorization systems must provide not only strong protection, but should be flexible enough as well. On the other hand, flexibility may increase the complexity of such systems significantly. Hence, the access control components of a security system must be able to work together in heterogeneous environments, but at the same time they must be flexible enough to be integrated with various applications [22].

Some the access control systems, which are implemented in a proprietary way, are limited to certain applications and hardly can be used in a networked environment that often implies interaction among isolated administrative domains. In other words, there is a need for additional management for sharing authorization information between autonomous domains. eXtensible Access Control Markup Language (XACML) was ratified by the Organization for the Advancement of Structured Information Standards (OASIS) in order to address the above-mentioned problems [19]. XACML is a standard and very flexible general purpose language for modeling access rights defined using XML. The OASIS XACML Technical Committee includes members from the main vendors such as Oracle, Cisco, IBM, RedHat, Boeing, U.S.

D.H.S., etc. Besides a policy language, XACML defines a syntax for managing access control to resources. Even though XACML does not provide an entire authentication solution, it specifies how components of such a solution should interact among each other [22].

One of the features of the XACML standard is that it provides a way to separate policy definition from its implementation in the applications. Thus, it implies the existence of many available engines for the management and evaluation of XACML policies. The first official XACML implementation was Sun's XACML library [9], which has become the industrial standard. Afterwards, a number of other XACML engines were created, such as XEngine [16], XACMLight [10], Enterprise Java XACML [2], and others. Such XACML libraries are supposed to provide functionalities that understand the XACML language. They also process XACML requests and manage attributes and other related data. Most of them are written in the Java programming language [7], although a .Net-based implementation, XACML .NET [11], is known as well. These engines in general were intended for the correct evaluation of policies and compliance with the standards. That is why most of them may lack optimizations in case of a large number of policies and/or a large amount of content. However, there is no known comprehensive analysis of such limitations for most of them.

1.2 Our Contribution

With the growth of web applications using XACML, the performance of XACML engines becomes a crucial issue. Namely, if a web server has to handle a great number of XACML requests and enforce an XACML policy with a large number of rules, the performance of the whole online application may totally depend on the XACML implementation used, which eventually might become the performance bottleneck at peak demand. Because of the fact that in modern enterprises the number of clients using web technologies as well as resources increases dramatically, the size of XACML policies rises respectively, making them larger and more complex. Undoubtedly, a scalable and fast XACML library which can cope with unbalanced workloads along with analysis of its limitations is necessary.

At the same time, little work has been done to evaluate and compare various XACML engines. Analysis of applicability of such implementations for certain policy size or workload is needed. The framework for automatic XACML libraries performance analysis is of special interest. Prior research work on the performance of XACML policy evaluation engines does not provide unified software which could be used for any XACML libraries.

The previously discussed problem can be stated as the need for a framework for XACML engines analysis which allows importing various libraries written in Java with open API and running unified tests. Available open-source Java-based XACML libraries that are not obsolete are proposed to be used as objects of research.

This thesis mainly focuses on the performance analysis of Java-based XACML libraries, which is a critical issue, by means of a specially implemented framework. In order to research the performance of XACML engines, we propose to run a number of experiments which include XACML policies of different size, various workloads of XACML requests as well as synthetically generated and real-life policies from numerous sources.

An additional task exists that is testing with the above mentioned framework should be conducted on varying platforms (including different CPUs, RAM, and operating system types).

Overall, our research goal is to determine the efficiency of existing XACML engines under different conditions.

The rest of the thesis is organized as follows:

In Chapter 2, we give a description of the XACML standard, along with details of core features of XACML, including its architecture, policy language and the interactions of its entities. We look at software libraries which support XACML that were chosen for our experiments, and illustrate some APIs and corresponding functionalities. We show some industry practices for using XACML as well. Details of previous work that has been done are presented at the end of the Chapter.

In Chapter 3, we present our approach to implement the framework for analysis of Java-based XACML engines. We outline details of the technical design and the hierarchy of classes

created.

In Chapter 4 we describe a number of experiments that were conducted using the previously implemented framework. The policies for our tests were collected from a few sources that are described as well.

In Chapter 5 we draw the final conclusions regarding the practical use of selected XACML engines. We also present a number of ideas for future work in this area as well as possible enhancements of our framework.

Chapter 2

Background and Previous Work

2.1 The eXtensible Access Control Markup Language – XACML

XACML defines an XML-based syntax that describes a policy language as well as request and response languages.

According to [15], XACML provides:

- a way to base access control decisions on attributes of both a subject and a resource;
- a mechanism for supporting multiple subjects with multiple roles (addressed by the XACML profile for RBAC [12]);
- a method to share policies in a distributed environment;
- a way to separate policy definition from its implementation (hence, there can be an arbitrary number of different implementations).

There is also an architecture for the decision-making process which is described in the XACML standard; it is based on the ITU Recommendation X.812 [6] and on the standard ISO/IEC 10181-7 [25, 29]. At the same time, the usage of this architecture is optional and can vary in different implementations. What follows is its description in more detail.

2.1.1 The XACML Architecture

XACML consists of many components depicted in Figure 2.1. The authorization process is described as follows [28]:

A subject which wants to take certain action on a certain resource has to submit its query to the entity protecting that resource. In the XACML architecture, this entity is called a Policy Enforcement Point (PEP). The PEP forms an XACML request using the XACML request language based on the attribute values of the subject, the resource, the action, and the environment. Afterwards, the created request is sent to the Policy Decision Point (PDP). The PDP receives and examines the request. When the access decision is made, the PDP forms an XACML response using the XACML response language and sends it back to the PEP. The response informs whether access should be permitted or denied, with the appropriate obligations. Finally, the PEP performs the obligations using an optional obligation service and, depending on the decision made by the PDP, either permits or denies access.

The PDP retrieves the available policies written in the XACML policy language by means of the Policy Access Point (PAP), which is basically an interface for writing policy sets. Depending on the situation, the PDP may submit a query to the Policy Information Point (PIP) in order to receive attributes related to the subject, the resource, or the environment.

Evaluating the applicable policies and the rules constitutes the decision making process. If there are many policies, the PDP selects only those which are relevant, based on the policy target, containing information about the subject, the action, and other environmental properties.

2.1.2 The XACML Policy Language

The XACML policy language model consists of several hierarchical objects. XACML has three mandatory components: a policy, a PEP, and a PDP. Figure 2.2 illustrates how they are linked to each other.

The main components of the XACML policy language are described as follows [15]:

XACML policies are represented by XML documents rooted in a Policy or PolicySet ele-

Figure 2.1: XACML Main Components [28].

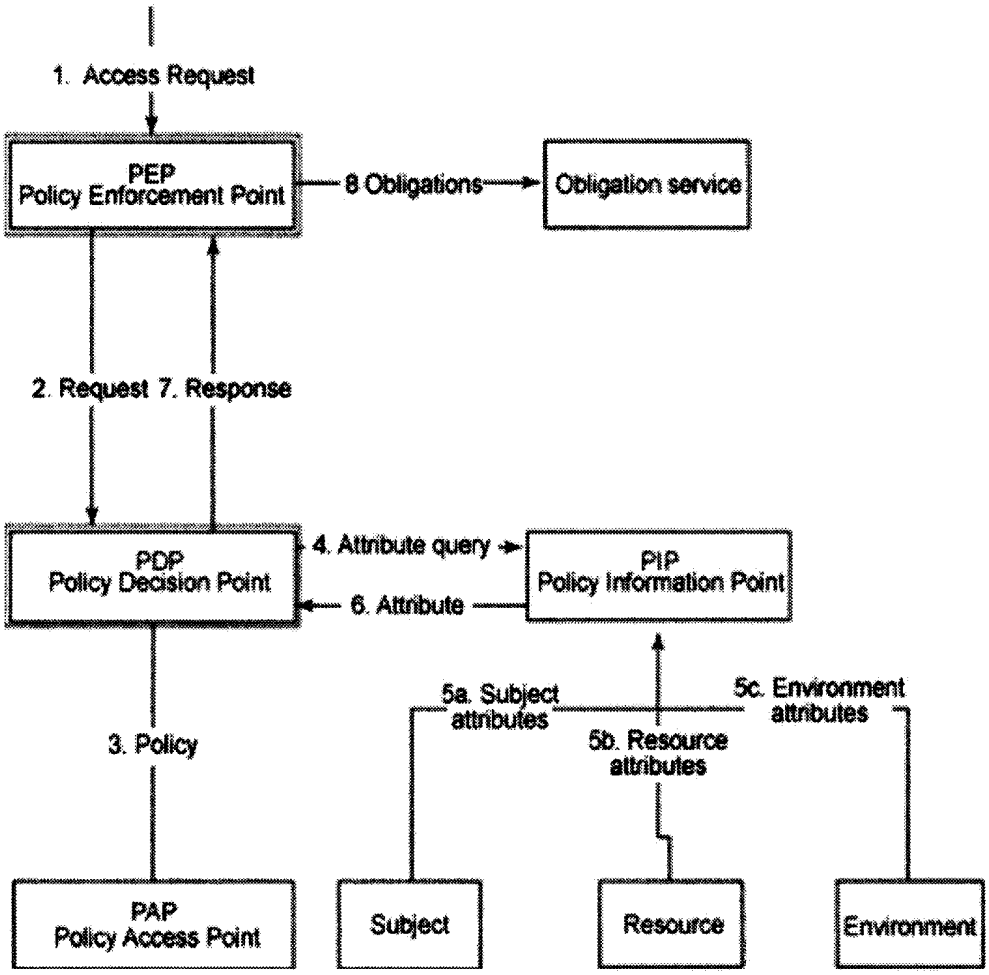
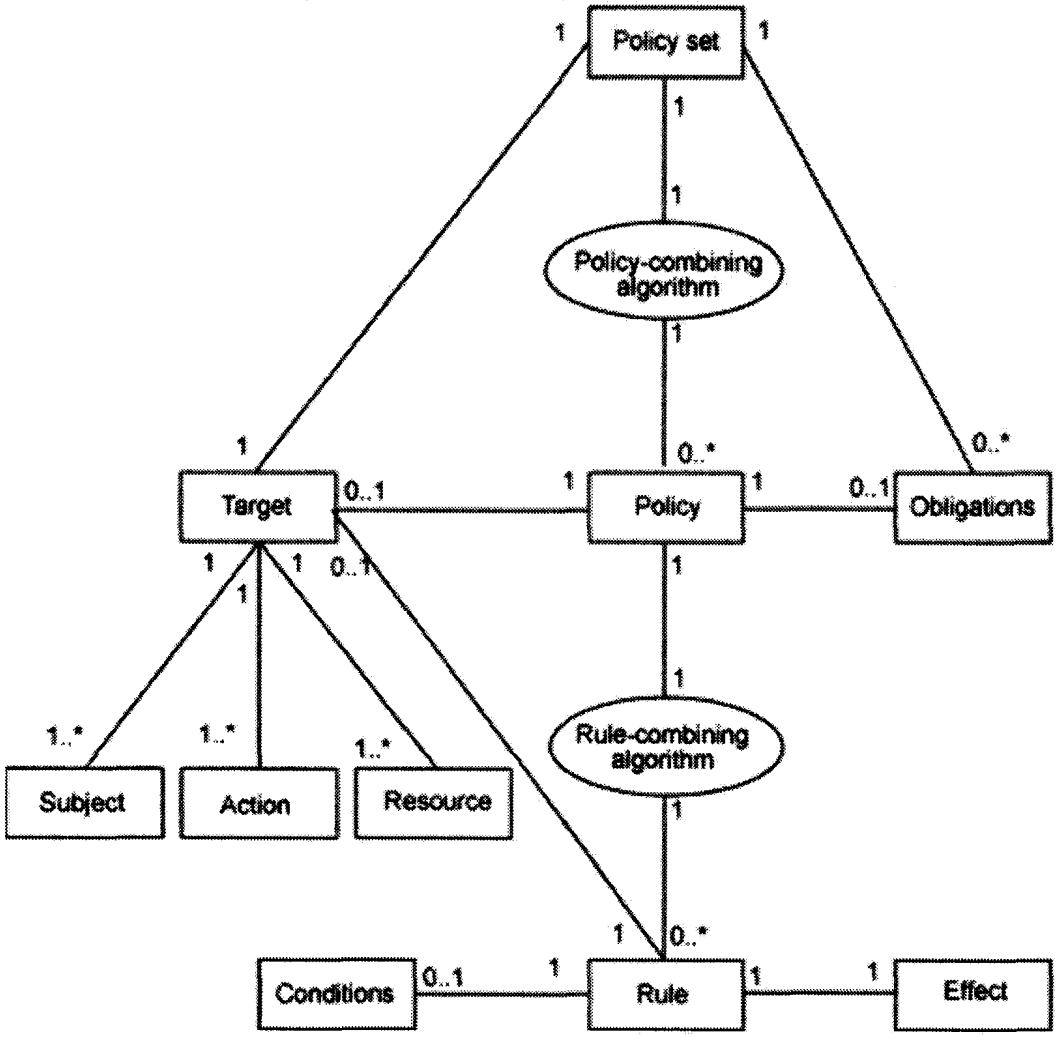


Figure 2.2: Policy Language Model [28].



ment. A PolicySet is a container with other Policy or PolicySet elements.

A policy consists of rules, rule-combining algorithms, obligations, and a target. In fact, XACML policies are the most essential aspect of the whole XACML infrastructure.

It is worth considering subcomponents of XACML policies in detail:

Target: There is only one target per policy. The target aims to find the relevant policy for the certain request. Usually, the target has attribute values of subject, resource, and action, even though they are optional. The comparison of these values with the values of the same attributes in the request allows the PDP to determine whether the policy is

considered relevant to the request or not.

Rules: Multiple rules can be associated with a policy. Each rule consists of a condition, an effect, and a target.

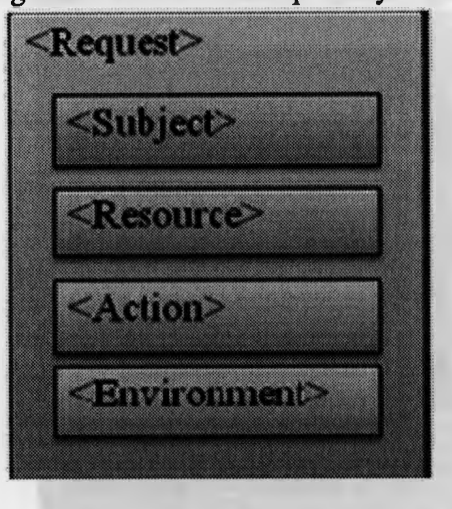
- Conditions are represented by statements about attributes that upon evaluation return either True, False, or Indeterminate.
- Effect is the intended consequence of the rule which was satisfied. It either returns the value Permit or Deny.
- Target, as in the case of a policy, aims to find a relevant rule for a certain request. This is achieved by means of the analogous mechanism, similar to the one for the target for a policy.

The final result of the rule depends on the condition evaluation. If the condition returns Indeterminate, the rule also returns Indeterminate. If the condition returns False, the rule returns NotApplicable. Finally, if the condition returns True, the value of the Effect element is returned.

Rule-combining algorithm: Providing that a policy can have multiple rules, conflicting results may be produced. Rule-combining algorithms help to resolve such conflicts to achieve one result per policy per request. Only one rule-combining algorithm is applicable per policy. XACML defines five standard rule-combining algorithms (users can define their own algorithms too):

- Deny-overrides: If any rule evaluates to Deny, then the final outcome is also Deny.

Figure 2.3: XACML Request Syntax.



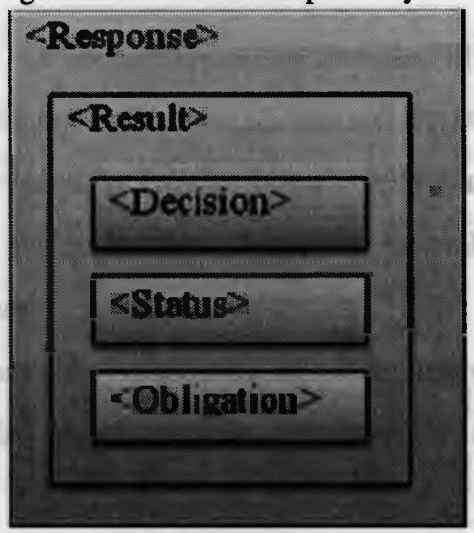
- Ordered-deny-overrides: Same as the previous one, except the order in which relevant rules are evaluated is the same as the order in which they are added in the policy.
- Permit-overrides: If any rule evaluates to Permit, then the final outcome is also Permit.
- Ordered-permit-overrides: Same as the previous one, except the order in which relevant rules are evaluated is the same as the order in which they are added in the policy.
- First-applicable: The final outcome is the result of the first relevant rule encountered.

Obligations: Obligations help to achieve a much finer-level of access control than simple permit/deny values. They indicate the actions that must be fulfilled by the PEP along with the enforcement of an authorization decision.

2.1.3 General Syntax of XACML Request and Response

XACML also defines the format for expressing authorization requests/responses [15]. This format is called the XACML Context and is described in a XML Schema (Figures 2.3, 2.4

Figure 2.4: XACML Response Syntax.



show the structure of a request and a response respectively).

Attributes of the requesting subjects, the resource, the action, and the environment constitute a Request Context.

The Subject element is the entity submitting the access request (e.g., human user, workstation, etc.). The Resource element represents the protected resource (e.g., file, email service, etc.). The Action element defines the action that the Subject wants to perform on the resource (e.g., open, update or delete).

The Environment element contains information about the resource environment (e.g., date, time or place).

It is possible that Subject, Resource, Action and Environment contain multivalued attributes.

A Response Context contains one or more Results which represent the decision that the PDP made. The Decision values can be Permit, Deny, Not Applicable (if no applicable policies or rules were found), or Indeterminate (if some error occurred or data missed). The Status returns optional information which may help to determine the errors if there were any. A Response Context may also include Obligations.

2.1.4 Policy Evaluation

According to Figure 2.1, the PDP receives the request from the PEP containing the XACML Context with information about Subject, Resource, Action and Environment [15]. In order to make a decision, the PDP retrieves all the Policies through the PAP and picks up those which match the Target. Afterwards, the comparison of attributes in the Request Context against attributes in the policy takes place.

XACML defines four standard policy-combining algorithms (users can define their own algorithms as well): Deny-overrides, Permit-overrides, First-applicable and Only-one-applicable [12].

2.2 XACML Entities Interaction

As mentioned above, there are several entities such as PDP, PEP, PAP and PIP that interact in the XACML workflow. Even though those entities are defined fully in the XACML standard, their collaboration is not standardized; this can be beneficial for users who get an opportunity to implement their own systems and adjust the interaction of these entities according to the system's needs.

In the following sections these entity interactions will be discussed more in detail, following [15].

2.2.1 PDP-PEP

There is not any mechanism in the XACML specification for transmitting requests and responses between the PDP and the PEP except the situation when they both are on the same system.

Usage of all other configurations is described in SAML (Security Assertion Markup Language), another OASIS standard [26]. There is also a specially implemented SAML profile for XACML [14]. It defines two general elements that manage transmitting requests and responses: a Query (an extension of the SAML Request element) and a Statement (the response

to the Query giving one or more results). For PEP-PDP interaction, the profile defines both of them. XACMLAuthzDecisionQuery is a query used in a Request and XACMLAuthzDecisionStatement is a statement used in a Response. Respectively, XACMLAuthzDecisionQuery transmits the XACML request from the PEP to the PDP, while XACMLAuthzDecisionStatement transmits the XACML Response that the PDP sends back to the PEP [15].

Technically, SAML does not provide message confidentiality; it gives message integrity only. If data need to be protected, such protocols as SSL or TLS[1] must be used over a network. Alternatively, when data encryption is not necessary and SAML is being used without SSL/TLS, all requests and responses may be signed appropriately in order to authenticate both points – the PEP and the PDP.

2.2.2 PDP-PAP

The XACML 2.0 Core Specification does not provide any information about making policies available to the PDP [12]. However, a XACML 2.0 entity, referred to as a PAP, is described as "a system entity that creates a Policy or PolicySet". Overall, it can be treated as an interface for writing policies and policy sets.

At the same time, there is an explanation of two methods that can be used for interaction between the PAP and the PDP. One is a SAML-based request-response protocol, and another one is a simple SAML Assertion-based storage format [15].

The first protocol describes a method which allows the PDP to retrieve policies from the PAP [14]. This method defines XACMLPolicyQuery – a format to query a policy, and XACMLPolicyStatement – a format to carry the requested policy. Namely, XACMLPolicyQuery is used when the PDP queries policies from the PAP. This element is an extension of the SAML Request element. Respectively, XACMLPolicyStatement is used when the PAP sends a response containing applicable policies (if there are any) to the PDP. This element is an extension of the SAML Statement element.

[1] Transport Layer Security (TLS) and its predecessor, Secure Sockets Layer (SSL), are cryptographic protocols that provide communication security over the Internet

As mentioned above, the PAP also may use a simple SAML Assertion-based storage format. In this case, it just stores policies in a generic repository which may be accessed directly by the PDP.

2.2.3 PDP-PIP

In order to make a decision about access, the PDP compares attributes in a request against attributes in the applicable policies where the request's Target matches. Sometimes the request may miss such attributes. In this case the PDP queries the PIP.

There is an explanation of two methods that can be used for obtaining attributes from the PIP: the Attribute Designator and the Attribute Selector. The first one allows the PDP to obtain attributes from a request, while the other one helps the PDP to search for them in some external source such as a database or over a network, for instance, using an XPath [17] query.

There are four kinds of Attribute Designator according to types of attributes in a request: Subject, Resource, Action, and Environment. Attributes can also be divided into different categories, which can be defined arbitrarily by users. In this case, Attribute Designators can also point out a category to look in.

The Attribute Designator and the Attribute Selector can return multiple values. In order to help the PDP serve such situations, a special attribute type called a Bag is defined. It represents an unsorted collection which allows duplicate values. Empty collection is allowed as well.

Besides the above discussed methods, interaction between the PDP and the PIP can also be organized using SAML Query and Statement extensions: AttributeQuery and AttributeStatement [14].

2.3 XACML Engines

As XACML provides a way to separate policy definition from its implementation in the applications, this naturally implies the existence of many available engines for the management and evaluation of XACML policies. The first official XACML implementation was Sun's

XACML library, presented in 2004 [9]. Afterwards, a number of other XACML engines were created, such as XEngine, XACMLLight, Enterprise Java XACML, and others.

Overall, XACML libraries are supposed to provide a set of Java classes that understand the XACML language, as well as the rules about how to process requests and how to manage attributes and other related data. Using such libraries, software developers can write applications that use XACML to manage their own policy or that hook into existing infrastructure components like LDAP or SAML.

At the same time, with the growth of web applications using XACML, the performance of XACML engines becomes a crucial issue. Namely, if a web server has to handle a great number of XACML requests and enforce an XACML policy with a large number of rules, the performance of the whole online application may totally depend on the XACML implementation used, which eventually might become the performance bottleneck at peak demand. Because of the fact that in modern enterprises the number of clients using web technologies as well as resources increases dramatically, the size of XACML policies rises respectively, making them larger and more complex.

Several groups have worked on XACML libraries analysis, although most of this work limited the experiments to test the correctness of policy evaluation and investigated specific engines only. Martin et al. developed a tool for automated test generation for access control policies [24]. Hu et al. introduced a policy-based segmentation technique to identify policy anomalies and derive effective anomaly resolutions; they developed a tool implementing their method as well [20]. Liu et al., while developing their XACML library XEngine, presented experiments comparing the performance of XEngine and Sun PDP [21]. Their results show that in some cases XEngine is orders of magnitude faster than Sun PDP. This library will be examined in more detail in this section. At last, Turkmen and Crispo tested evaluation time of Sun's XACML, XACMLight, and Enterprise Java XACML with various policy and request parameters, even though they did not conduct experiments with different workloads; they did not provide any details of the software used for analysis of XACML engines either [27]. For

the libraries they tested, Enterprise Java XACML demonstrated the best results in terms of evaluation time.

Overall, this section will describe core APIs of the first and the most popular Sun's XACML engine as well as give a description of two other XACML libraries which were created in order to outperform Sun's implementation. Finally, some industry practices will be presented.

2.3.1 Sun's XACML

Sun's XACML implementation is developed by the Internet Security Research Group (ISRG) within Sun Microsystems Laboratories. It is the first and the most widely deployed XACML evaluation engine. This implementation has become the industrial standard. The APIs are broken into several packages [9]:

com.sun.xacml is the core package. It contains the logic for target matching, rule evaluation, policy and policy set handling, and other related features. The main class of this package is the PDP class, which can be considered the entry point for most code.

com.sun.xacml.attr is the package that supports all the standard XACML attribute data types, as well as designators, selectors, and the factories used to create new attribute values. Standard interfaces and abstract classes are provided to define new attributes types.

com.sun.xacml.combine is the package that defines all the standard XACML combining algorithms as well as the factory for accessing those algorithms. There are also standard interfaces which can be used to define new combining algorithms.

com.sun.xacml.ctx is the package that supports all the types defined in the XACML context schema, i.e. the request and response formats. All of the classes in this package can be encoded and parsed in order to simplify creating a PEP.

com.sun.xacml.cond is the package that supports all of the condition and function logic. There are also standard interfaces and classes which can be used to define new func-

tions.

com.sun.xacml.finder is the package that supports retrieving information that the PDP may need. Classes for searching policies, obtaining attributes outside of the request, resolving resource identifiers are contained in this package.

com.sun.xacml.finder.impl is the package that provides basic implementations of mechanisms that are described in the previous package. Even though they provide enough functions for basic needs, any of these classes can be replaced with alternative implementations.

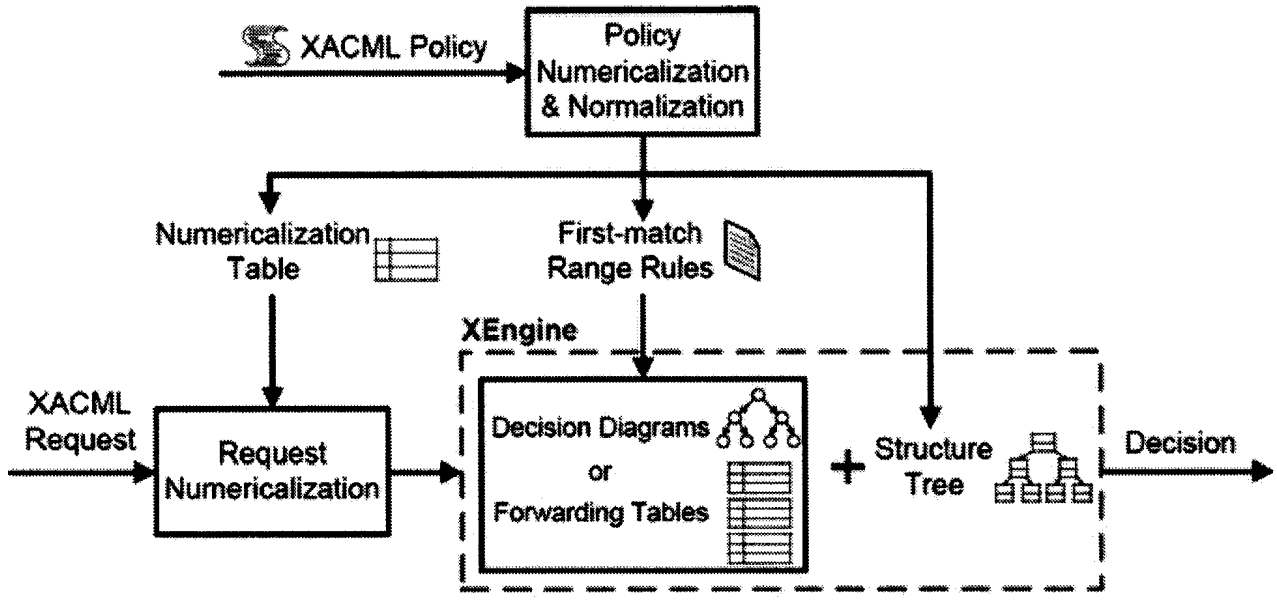
2.3.2 XEngine

Even though Sun's XACML implementation seems to be the most popular one, there are situations when it may lack optimizations [21].

First of all, real-life XACML policies often have complex structures and can be specified recursively. The next common situation is that real-life XACML policies often have conflicting rules, which can be reconciled by the four algorithms which were discussed before. Some XACML engines may simply examine all the rules in an XACML policy before making the final decision, even though it is quite time-consuming. Finally, XACML requests and XACML rules can contain multiple values; this may dramatically increase the complexity of searching and comparing that the PDP needs to perform.

XEngine has three key ideas [21]. First of all, XEngine converts all strings in an XACML policy to numerical values. XACML requests are also converted in the same manner. Obviously, storing and comparing integers is more efficient and less time and memory consuming than operations with strings. Liu et al. call this technique XACML policy numericalization. The second technique is normalization. Namely, a numericalized XACML policy with a hierarchical structure and several potentially complex conflict resolution mechanisms is converted

Figure 2.5: XEngine System Architecture [21].



to an equivalent policy with a flat structure and only one conflict resolution mechanism, which is First-Applicable. This reduces the time that the PDP needs to determine applicable policies, because Sun's PDP, for instance, just tries each policy against the request. Finally, in order to provide fast search, XEngine further converts a numericalized and normalized policy to a tree structure. Figure 2.5 [21] shows the system architecture of XEngine.

However, XEngine has some features that can be considered as shortcomings. Namely, even though it is an open source project so far, it does not provide a well-documented usual API in terms of Java standards; most of the classes and methods seem to be developed for purposes of testing only. It also misses appropriate functional documentation. The next important issue is that XEngine relies on Sun's XACML implementation, and as a result there may be dependence on its potential changes in the future. Finally, it has a pre-processing step when XACML policies are encoded offline using above mentioned techniques.

2.3.3 Enterprise Java XACML

Enterprise Java XACML was created while working on some SOA [23] projects. It fully implements OASIS XACML 2.0, and provides a high performance and good usability in an

enterprise environment. It is worth noting that this is a totally independent implementation. It does not rely on Sun's XACML implementation or any other implementations.

As stated before, Sun's PDP has many shortcomings. Besides those mentioned above, it does not have any cache mechanism for the retrieved policies or the evaluation result. The next point to be mentioned is that Sun's implementation does not define the situation when multiple policies match a single request. Even though the XACML standard does not describe this situation either, it often occurs in real-life enterprises. What comes next is that Sun's engine reads policies from a local file only; in order to use another policy store, the implementation needs to be modified. Finally, there are no extension mechanisms, such as attribute retriever, in Sun's implementation.

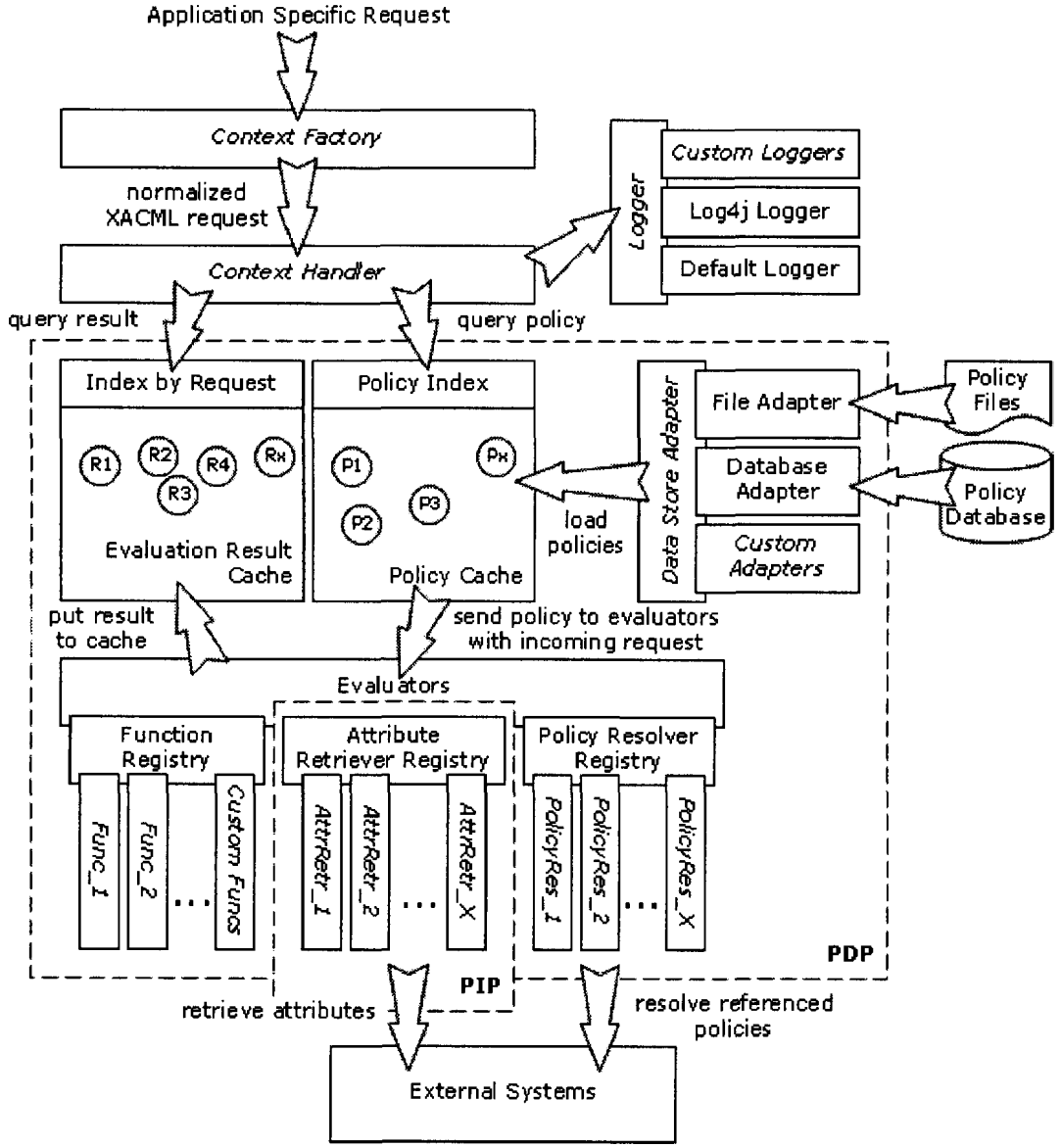
Enterprise Java XACML engine is implemented, considering the above mentioned drawbacks of Sun's XACML library. In a nutshell, it does not introduce sophisticated mathematical methods as XEngine does, but uses several available optimizing techniques from the software development area. Its main point is to provide an extensible and scalable architecture, which could be easily integrated with various applications and thus be used in complex heterogeneous environments. It also addresses the problem of conflicting versions of third party libraries: in fact, Enterprise Java XACML uses log4j only [8]. Moreover, the log4j library can be easily excluded altogether.

The diagram in Figure 2.6 illustrates Enterprise Java XACML implementation's architecture [2]. All components with italics can be customized by users.

The API of the Enterprise Java XACML engine provides:

- classes representing a PDP;
- classes representing a simple PAP;
- an effective target indexing mechanism;
- a cache for decisions and policies;

Figure 2.6: Enterprise Java XACML Architecture [2].



- pluggable mechanisms for data store, context factory and logging (users can implement their own versions and easily replace existing ones);
- extensible mechanisms for attribute retrieving and policy resolving.

2.3.4 Industry Practices

These days the XACML standard is extensively used in many real-life applications and enterprises. In order to adapt it to business requirements, special profiles are introduced (for instance, RBAC Profile, Web Services Profile, Privacy Profile). Also XACML is often used together with an Identity Management System (for instance, LDAP, OpenID) and in various authorization services. Integration of XACML to products was made by Oracle, JBoss, IBM, Cisco, and other main vendors.

Examples of well known real-life applications using XACML include following:

- a general purpose repository system FedoraCommons [3];
- a national Swedish health-Care system (Axiomatics startup) [1];
- Geospatial XACML protecting access to distributed geographic information [4].

Chapter 3

Implementation of the Framework for Analysis of Java-Based XACML Engines

This chapter presents the challenges in implementing the framework for analysis of XACML engines and introduces our approach and details of our implementation.

As was discussed earlier, there is no known system that runs performance tests of different workloads for all possible open-source XACML engines. We introduce the notion of a special framework which allows importing various XACML engines through the implementation of a defined specification via so called middleware. In other words it can be extended to support an arbitrary number of XACML libraries.

The framework should provide the means for quick and easy testing, specification for importing new XACML libraries, extendable testing information, and the ability to work in different heterogeneous environments, i.e., to be platform-independent. The ultimate goal of our framework is to provide users and researchers with a tool which tests all the engines in the same way, while developers will get a single specification for extendable middleware.

3.1 Choosing a Language

In order to create a platform-independent tool, we have chosen the Java programming lan-

guage (JDK 1.6.25) [7]. Moreover, most of the currently available XACML libraries are written in Java as well. Thus, our framework can work under any operating system having the Java Runtime Environment. Furthermore, it is the most convenient way to test engines written in the same language and in the specific environment where they are supposed to be used in a real-life application.

The Java programming language is pure object-oriented, open-source and provides a great number of standard APIs. We will use its notions and terms to describe the architecture of the framework and its components hereinafter.

The language choice in a natural way implies the main limitation of the framework, as it supports Java-based XACML engines only.

3.2 The Framework Architecture

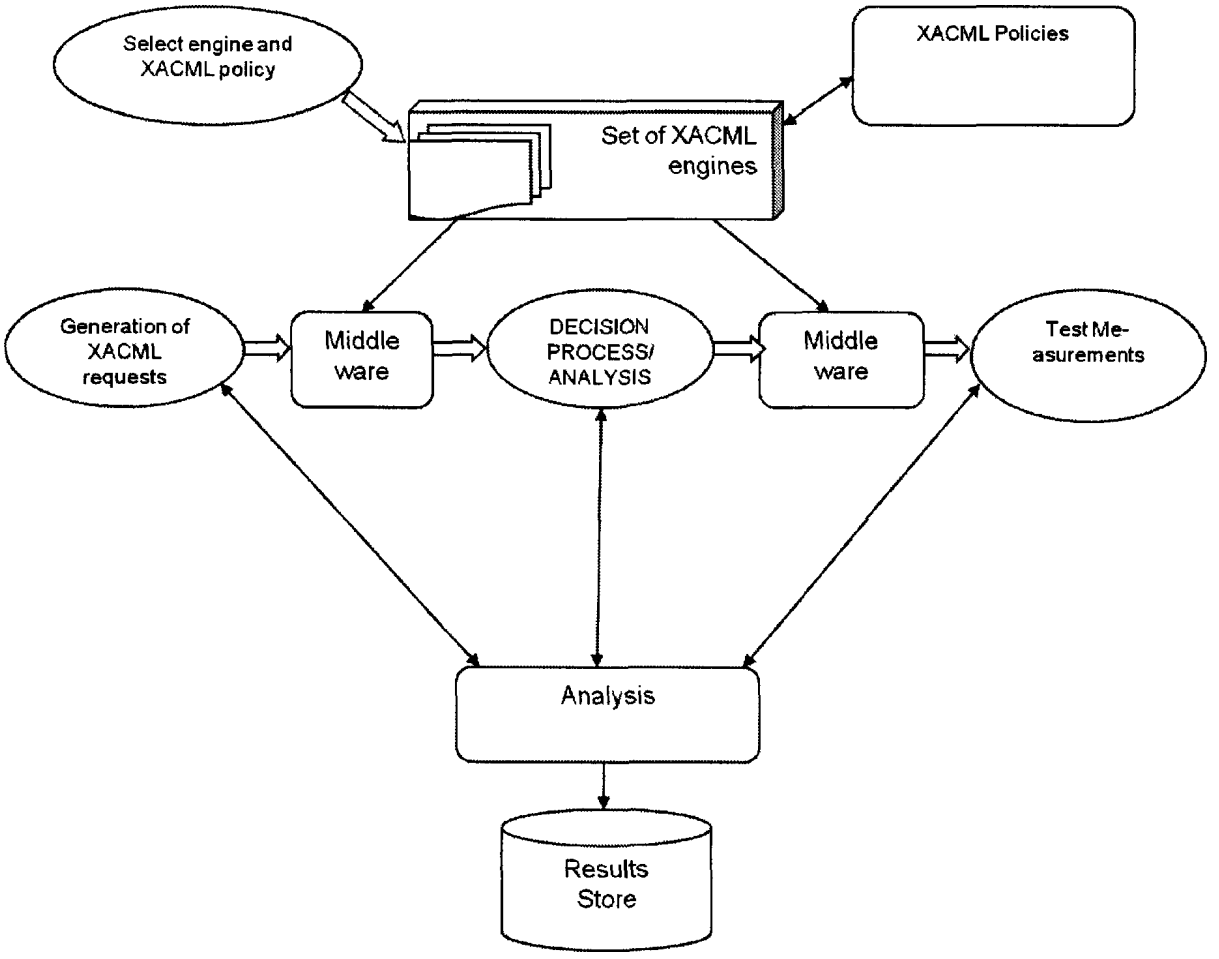
The framework architecture is depicted in Figure 3.1.

In this work we gather XACML policies (we will discuss them in Chapter 4) and write the middleware that represents imported XACML engines and allows us to generate XACML requests for each policy. The framework general workflow is as follows. We first choose an XACML policy and generate a number of XACML requests for it. Then we select an XACML engine we would like to test and pass it along with an XACML policy/requests as parameters to our middleware which performs testing. Technically, the middleware invokes public methods of an XACML engine's API and measures the performance characteristics. Finally, results of experiments are collected by the middleware and stored for further analysis.

3.3 The Software Structure

All the implemented Java classes are contained in the package (in terms of the Java language) called com.rakhmatulin.uwo.jfxt. The software is implemented as a standalone application.

Figure 3.1: The Framework Architecture



The main class is called Main and it contains the entry point of the application – the main() method. There is a parsing of the configuration file (we will discuss this later in this chapter), an instantiation and initialization of the selected XACML engine, and an invocation of the high-level methods for testing in the Main.main() method.

In order to implement the framework, a few subtasks were tackled:

- Introduction of the object-oriented hierarchy of classes, presenting XACML PDPs;
- Implementation of the classes supporting performance measurements and storing their results;
- Importing the PDPs which were chosen for our testing (Sun PDP, XEngine PDP, and Enterprise Java XACML PDP) as well as implementation of the necessary middleware.

There are several ways of providing XACML requests for evaluation. Basically, it can be any storage of data such as a file system, network, or a database. At the same time, storing requests in files allows us to expect the best loading time, whereas all XACML engines provide public methods for evaluating with parameters where files can be indicated as a source of XACML requests. Taking these facts into consideration, in our implementation we use the selected directories for storing XACML requests as regular XML files. This gives an opportunity to vary the number of requests when emulating different workloads.

What comes next is the discussion of how importing of XACML libraries is implemented. Full details of the middleware will be given as well.

3.4 The Hierarchy of Classes

We have applied basic principles of object-oriented programming in order to construct a conceptual architecture of the framework. First of all, we will discuss the PDP importing implementation focusing on creation of its object-oriented model.

In order to maintain the abilty to import arbitrary Java-based XACML libraries (of course, it is supposed that they are open-source), the hierarchy of classes in terms of object-oriented programming is built. Namely, the abstract class AbstractPDP was implemented which is on the top of the hierarchy and it is the superclass for any other classes, presenting concrete PDP implementations of the engines. Correspondingly, all other PDPs which are being imported must extend the AbstractPDP class. It has a single constructor:

```
public AbstractPDP (ConfigPDP config);
```

where ConfigPDP is a class presenting the configuration of the current PDP. While AbstractPDP itself is an abstract class, it has a non-abstract constructor which saves a reference of the ConfigPDP object in its private field.

In order to provide access to the saved configuration, AbstractPDP has the corresponding getter method:

```
public ConfigPDP getConfigPDP();
```

Each subclass of AbstractPDP should override the following abstract methods of their superclass:

```
public abstract void init()
        throws JFXTException;
```

which performs initialization of a PDP, and

```
public abstract ExperimentalResult evaluate (String requestDir)
        throws JFXTException;
```

which loads all the XACML requests located in the folder requestDir, evaluates them, and returns performance measurements in the special object – ExperimentalResult. It is worth noting that both methods may throw a JFXTException which extends the Exception class and represents all the exceptions in the package com.rakhmatulin.uwo.jfxt. Classes ConfigPDP and ExperimentalResult will be discussed in the next section.

3.5 The PDP Importing Implementation

The following classes present implementations of XACML engines and all of them are subclasses of AbstractPDP, overriding its abstract methods: SunPDP, XEnginePDP, and EnterpriseJavaXacmlPDP. We will describe these classes in more detail next.

SunPDP uses Sun APIs, including such packages as:

- com.sun.xacml;
- com.sun.xacml.cond;
- com.sun.xacml.ctx;
- com.sun.xacml.finder;
- com.sun.xacml.finder.impl.

First of all, a few objects, representing different modules, are created in the overridden init() method. There are such objects as FilePolicyModule, PolicyFinder, CurrentEnvModule, and SelectorModule. Then Sun's PDP object is instantiated, using AttributeFinder and PDPConfig objects. In the overridden evaluate() method, an instance of the RequestCtx class is used to deal with files containing generated XACML requests. Finally, evaluation itself is done in the evaluate() public method of the PDP object.

What comes next is XEnginePDP. This class uses the APIs of such packages as:

- ncsu.util;
- xEngineConverter;
- xEngineVerifier;
- commonImpelmentation.

Due to the fact that XEngine is partially based on the Sun API, it also includes some classes from Sun's packages: com.sun.xacml and com.sun.xacml.finder. The overridden init() method creates an instance of PolicyFinder and through conversionTree objects converts given XACML policies to special XEngine format using numericalization and normalization techniques. Eventually, it saves the converted numercalized and normalized policies as regular text files. As to the overridden evaluate() method, XEnginePDP uses the xxAclQuery class and its main method.

Finally, we will take a look at EnterpriseJavaXacmlPDP. This class uses APIs of such packages as:

- an.config;
- an.log;
- an.xacml.adapter.file;
- an.xacml.context;
- an.xacml.engine;
- an.xacml.policy.

Due to the fact that a PDP object of the Enterprise Java XACML engine requires a special configuration file, we first create instances of Configuration and ConfigurationElement objects in the overridden init() method. They aim to load and parse the above mentioned configuration file. Then the LogFactory object is initialized, providing an opportunity to get a reference to an instance of the Logger object. Finally, an instance of a PDP object is created using its singleton method. In the overridden evaluate() method, a static method of the XACMLParser object is used in order to deal with files containing generated XACML requests, which are stored as Request objects. Evaluation itself is done in the handleRequest() method of a PDP object.

3.6 Configuration

The configuration is stored in an XML file, and there are a few classes intended to work with configuration parameters, including loading and parsing XML files. ConfigPDP is the class consisting of public static fields only, which represent the main configuration parameters of any PDP. Some of the fields are used by all the XACML engines, while some may be used by specific ones only. Moreover, not all the fields are used in the current implementation, but they were created considering future extensions of the framework in order to accommodate most of the possible useful parameters. We give a brief description of these fields along with their Java definition.

```
public static String configFile;
// configuration file of the certain PDP
public static String[] policy;
// path to XACML policy
public static String[] convertedPolicy;
// path to converted policy
public static String[] policyReference;
```

```
// path to policy reference
public static String[] requestDir;
// path to directory with XACML requests
public static int[] requestNumber;
// number of XACML requests
public static String responseDir;
// path to directory with XACML responses
public static String logDir;
//  path to directory for logging
public static String logWriter;
//  path to file for logging
```

The instance of ConfigPDP is created through the helper class called ConfigPDPLoader. The main assignment of this class is to load and parse an XML file with the configuration. Its public method aims to do that:

```
public ConfigPDP getConfigPDP(String configFile)
        throws JFXTException;
```

where configFile indicates a path to the configuration file in a local file system. If there is an error during parsing of the XML file, a corresponding exception will be thrown by this method, which is presented as an instance of JFXTException. It is worth mentioning that the low-level parsing is located inside of the private method using the DOM API:

```
private void parseXML(String xml)
        throws ParserConfigurationException, SAXException,
        IOException;
```

All the possibly thrown exceptions (ParserConfigurationException, SAXException, IOException) are wrapped then into an instance of JFXTException in the public method, described above.

Here is an example of the configuration file.

```
<?xml version='1.0' encoding='windows-1251'?>
<config>
 <PDP>
  <name>XEnginePDP</name>
  <version>1.0</version>
  <pdpConfigFile></pdpConfigFile>
 </PDP>
 <policies>
  <policy>
   <xacmlPolicy>E:/UWO/XACML/tests/XEngine/
policies/continue-a.xml
   </xacmlPolicy>
   <convertedPolicy>E:/UWO/XACML/tests/
XEngine/policies/continue-a.txt
   </convertedPolicy>
   <policyReference>E:/UWO/XACML/tests/
XEngine/policies/continue-a.xml.fwr.log
   </policyReference>
  </policy>
 </policies>
 <request>
  <requestDir>E:/UWO/XACML/tests/requests/
policies/continue-a/10000/
  </requestDir>
  <requestNumber>10000</requestNumber>
 </request>
```

```
 <response>
  <responseDir>E:/UWO/XACML/tests/XEngine/
responses/continue-a/
   </responseDir>
 </response>
 <log>
  <logDir>E:/UWO/XACML/tests/XEngine/log/
   </logDir>
  <logWriter>log.txt </logWriter>
  <logResults>results.txt </logResults>
 </log>
</config>
```

It is worth mentioning that the only required parameters for all engines are the PDP's name, an original XACML policy, and a directory with XACML requests. All other parameters are either optional, or they are required by specific engines only, or some default values may be used instead if they are empty. So for example if log files are not specified, then the corresponding information from a logger will be displayed on the screen.

3.7 Storing results of experiments

As mentioned in the previous sections, performance measurements are stored in the ExperimentalResult object in the runtime period. Here we suppose that one experiment is the process of evaluating a number of XACML requests contained in one directory against XACML policies by one XACML engine. Consequently, each ExperimentalResult object presents information of one experiment. The ExperimentalResult class contains private fields representing different measurement parameters as well as a description of an engine:

```
private String engine;
```

```
// name of an engine
private long initTime;
// time of initialization
private long requestLoadingTime;
// time of XACML requests loading
private long evalTime;
// time of XACML requests evaluation
private long memory;
// size of the Java virtual memory used
private int requestsNumber;
// number of XACML requests processed
```

There are also public setter methods for all of the fields in the class:

```
public void setInitTime(long initTime);
public void setRequestLoadingTime(long requestLoadingTime);
public void setEvalTime (long evalTime);
public void setMemory (long memory);
public void setRequestsNumber (int requestsNumber);
```

It is worth noting that as in the case of configuration parameters in the ConfigPDP class, not all the fields of ExperimentalResult are used in the current implementation, but they were created considering future extensions of the framework in order to accommodate most of the possible useful parameters of experiments.

The ExperimentalResult class has three overloaded constructors, giving an opportunity to pass different parameters during the construction phase:

```
public ExperimentalResult(String engine, int requestsNumber);
public ExperimentalResult(String engine, long evalTime);
public ExperimentalResult(String engine, long requestLoadingTime,
```

```
long evalTime);
```

Additionally, the toString() method is overridden in order to present overall information about the current experiment. The return values of this method are used in the logging subsystem, providing an easy and efficient way of printing all necessary information as one string.

Chapter 4

Performance Evaluation and Experimental Results

4.1 Performance Benchmarks and Tools

The main goal of the framework discussed in the previous chapter is to evaluate the performance of XACML libraries. Here, we define performance by several characteristics of XACML engines which can be measured during the whole runtime process of XACML policy evaluation, from the engine initialization until the decision is made upon evaluation. While one may reckon the great number of such characteristics, it is natural to select the main ones which affect the performance dramatically. In the current implementation, we measure the time it takes an engine to load XACML requests from the hard disk, the time it takes an engine to evaluate XACML requests against the given XACML policy, and the memory consumed during the engine's work. In this thesis we evaluate performance characteristics of Sun's PDP, XEngine, and Java Enterprise XACML, when they accomplish identical tasks, i.e., when they evaluate identical XACML requests against identical policies under identical workload. Afterwards, we compare the results of these engines.

There is a great variety of test cases used. First of all, XACML policies range from simple ones with a few rules to policies with a large number of rules, both real-life and synthetically generated ones. Additionally, the experiments were executed with a varying number of XACML requests, which ranged from 10 to 10000 in order to simulate different workloads. Worth noting is the fact that all kinds of experiments were conducted enough times until the average outcomes stabilized, providing that following runs did not produce essential variation in the data.

Analysis of XACML engines has been conducted on various platforms (including different CPUs, RAM, and operating system types). The platform which was used for a particular experiment is mentioned in each figure presenting the results. The main reason for choosing various platforms is that there is a well-known fact that the Java virtual machines have different implementations for different platforms, which is why it is necessary to run experiments on a few of them. The sequence of choosing particular platforms depended on the different availability of testbeds during the phase of conducting the experiments.

It is worth mentioning that all performance characteristics in the current implementation were measured using Java methods for determining system time, and methods for determining the amount of memory currently used by the Java virtual machine. Namely, the currentTimeMillis() method of the System object as well as methods of the Runtime object called totalMemory() and freeMemory() were used. During our tests, essential system services only were allowed to run, and all experiments were developed to avoid involuntary garbage collection.

In order to conduct experiments, a few subtasks were tackled:

- Gathering experimental data, i.e., real-life and synthetically generated XACML policies from different sources;
- Generating XACML requests for the gathered XACML policies;

- Writing XML-configurations used by our framework for each test case;
- Running experiments and collecting results.

The following section will describe the general settings of our experiments, such as the experimental data and its sources, the configuration of our testbeds, and parameters of experiments.

4.2 General Settings of Experiments

We have used eight real-life XACML policies from different sources. Among these policies, codeA, codeB, codeC, continue-a, and continue-b are XACML policies used in [18] and [21]; demo1 and demo2 are used in [20]. Policies continue-a and continue-b are designed for a real-life web application that supports a conference management, while pluto is used in the ARCHON system [1]. We present codeA, codeB, and codeC in Appendix A.

Also, we have used four synthetically generated XACML policies which are used in [21]. The main reason for usage of synthetically generated policies is that they can contain an arbitrarily large number of rules. In our experiments we have used policies that contain 400, 800, 1600, and 4000 rules.

We have conventionally divided all of these XACML policies into two groups. The first group contains policies which have fewer than 30 rules. Policies with 200 and more rules belong to the second group. In the following sections we refer to these groups as the small policies group and the large policies group, respectively. Finally, synthetically generated XACML policies constitute the third group.

All the experiments were conducted on three different platforms. We give a brief description of their configuration, specifically, an operating system run, CPU, memory size, and a hard

[1] http://archon.cs.odu.edu/

disk drive model:

- Configuration 1: Intel Core i5 660 @ 3.33GHz, 4.00 GB RAM, Windows 7 Enterprise N 64-bit, Western Digital WDC 500GB Serial ATA III HDD (7200 rpm);
- Configuration 2: AMD Athlon X2 Dual Core Processor L310 @ 1.20 GHz, 4.00 GB RAM, Windows 7 Home Premium 64-bit, Western Digital WDC 250GB Serial ATA III HDD (7200 rpm);
- Configuration 3: Intel Core i3 2100 @ 3.10GHz, 2.00 GB RAM, Ubuntu Linux 10.04 64-bit, Seagate Barracuda ST3 1000GB Serial ATA III HDD (7200 rpm).

In the next section, request generation will be discussed.

4.3 XACML Requests Generation

The next task after gathering XACML policies for our experiments was to generate certain numbers of XACML requests for each policy. For this purpose the RequestGenerator class has been implemented. Technically, it is not a part of the framework discussed in the previous chapter; however, it is worth discussing briefly.

The RequestGenerator class is based on the part of the XEngine API which uses methods of mutations in order to generate a certain number of random XACML requests for the given XACML policy, combining in different ways the attributes of a Subject, an Object, and an Action requested. Most of the API used is part of the reqGen.ncsu package.

Here is the most interesting method of the RequestGenerator class:

```
public static void generateRequests(String xacmlPolicy,
String requestsDir, int requestsNumber)
```

where xacmlPolicy is the path to our XACML policy; requestsDir is the path to the directory where generated XACML requests will be saved; and requestsNumber is the number of XACML requests to be generated.

As it was found, Enterprise Java XACML uses a strict XML schema not only for XACML policies, but for XACML requests as well. That is why we have had to make a few modifications in the XEngine API, also fixing a few minor bugs.

For each of the twelve XACML policies in our experiments 4 sets of XACML requests have been generated: 10, 100, 1000, and 10000 requests in sets. These numbers are chosen due to the fact that the logarithmic scale provides better visualization of time-related benchmarks.

Among the generated policies there are both single-valued and multi-valued policies. We will present one example of a generated XACML request for each group of XACML policies we used for our experiments.

Below there is one of the XACML requests generated for the continue-a policy:

```
<Request xmlns='urn:oasis:names:tc:xacml:2.0:context:schema:os'>
 <Subject SubjectCategory="urn:oasis:names:tc:xacml:1.0:
subject-category:access-subject">
  <Attribute AttributeId="role" DataType="http://www.w3.org/
2001/XMLSchema#string">
   <AttributeValue>pc-member</AttributeValue>
  </Attribute>
 </Subject>
 <Resource>
  <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:
resource:resource-id"
DataType="http://www.w3.org/2001/XMLSchema#string">
   <AttributeValue>DEFAULT RESOURCE</AttributeValue>
  </Attribute>
```

```
  <Attribute  AttributeId="isEq-meetingPaper-resId"
DataType="http://www.w3.org/2001/XMLSchema#string">
    <AttributeValue>true</AttributeValue>
   </Attribute>
 </Resource>
 <Action>
  <Attribute  AttributeId="action-type"
DataType="http://www.w3.org/2001/XMLSchema#string">
    <AttributeValue>read</AttributeValue>
   </Attribute>
 </Action>
 <Environment></Environment>
</Request>
```

This request can be explained as follows: a Subject with a pc-member role requests to read a meeting paper.

Next we will look at one of the XACML requests generated for the codeA policy, where a Subject with a faculty role requests to assign external grades:

```
<Request xmlns='urn:oasis:names:tc:xacml:2.0:context:schema:os'>
 <Subject SubjectCategory="urn:oasis:names:tc:xacml:1.0:
subject-category:access-subject">
  <Attribute  AttributeId="role"
DataType="http://www.w3.org/2001/XMLSchema#string">
    <AttributeValue>Faculty</AttributeValue>
   </Attribute>
 </Subject>
 <Resource>
  <Attribute  AttributeId="resource-class"
```

```
DataType="http://www.w3.org/2001/XMLSchema#string">
    <AttributeValue>ExternalGrades</AttributeValue>
   </Attribute>
  <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:resource:
resource-id" DataType="http://www.w3.org/2001/XMLSchema#string">
    <AttributeValue>DEFAULT RESOURCE</AttributeValue>
   </Attribute>
 </Resource>
 <Action>
  <Attribute AttributeId="command"
DataType="http://www.w3.org/2001/XMLSchema#string">
    <AttributeValue>Assign</AttributeValue>
   </Attribute>
 </Action>
 <Environment></Environment>
</Request>
```

Finally, here is an exapmle of XACML requests generated for the SyntheticPolicy.4000.0 policy:

```
<Request xmlns='urn:oasis:names:tc:xacml:2.0:context:schema:os'>
 <Subject SubjectCategory="urn:oasis:names:tc:xacml:1.0:
subject-category:access-subject">
  <Attribute AttributeId="role"
DataType="http://www.w3.org/2001/XMLSchema#string">
    <AttributeValue>subject_10</AttributeValue>
   </Attribute>
 </Subject>
 <Resource>
```

```
  <Attribute  AttributeId="resource-class"
DataType="http://www.w3.org/2001/XMLSchema#string">
    <AttributeValue>resource_11 </AttributeValue>
   </Attribute>
  <Attribute  AttributeId="urn:oasis:names:tc:xacml:1.0:resource:
resource-id"  DataType="http://www.w3.org/2001/XMLSchema#string">
    <AttributeValue>DEFAULT RESOURCE</AttributeValue>
   </Attribute>
 </Resource>
 <Action>
  <Attribute  AttributeId="action-type"
DataType="http://www.w3.org/2001/XMLSchema#string">
    <AttributeValue>action_0 </AttributeValue>
   </Attribute>
 </Action>
 <Environment></Environment>
</Request>
```

This request can be read as follows: subject_10 requests to perform action_0 on resource_11.

4.4 Testing XACML Requests Loading Time

In this section we look at the XACML requests loading time, which in general does not depend on the policy size. Of course, the main parameter that influences the time is the number of requests. As it was mentioned above, we have used 4 sets of requests for each policy: 10, 100, 1000, and 10000 requests.

For each of the twelve XACML policies we present a figure which depicts the XACML requests loading time for Sun PDP, XEngine, and Enterprise Java XACML engines (see Figures

Figure 4.1: XACML Requests Loading Time. XACML Policy: continue-a; Platform: Configuration 1.

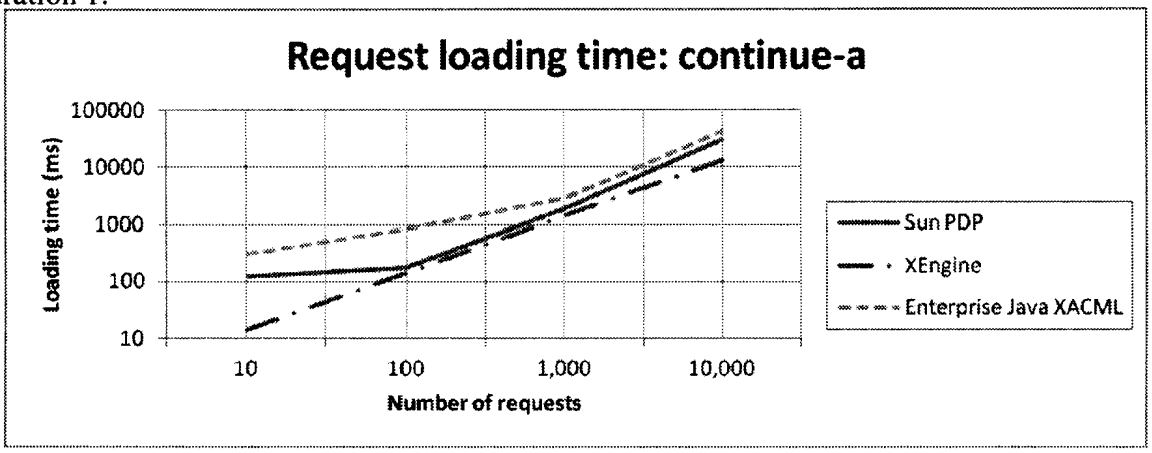
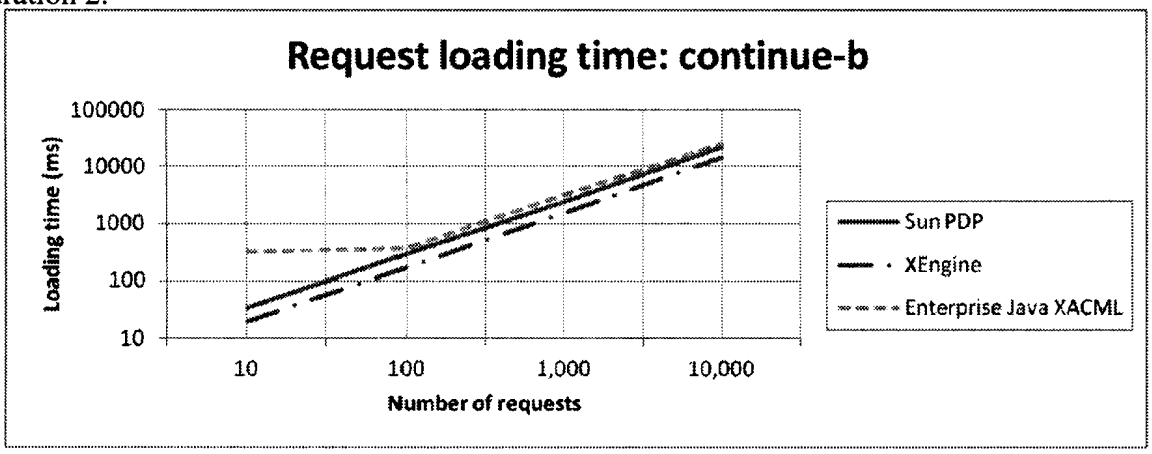


Figure 4.2: XACML Requests Loading Time. XACML Policy: continue-b; Platform: Configuration 2.



4.1-4.12). It is worth noting that the horizontal axes are discrete and present the number of XACML requests loaded, while the vertical axes present the loading time and are in logarithmic scales in terms of milliseconds. We indicate the platform configurations described in the "General settings" section which were used in each experiment.

There are a few evaluations that we can make from the results obtained in the above experiments. First of all, there is a prominent part from 10 to 100 requests loaded where XEngine has the best results, whereas Enterprise Java XACML is an outsider. For instance, loading of 10 XACML requests for SyntheticPolicy.4000 policy took Enterprise Java XACML 294 ms, while Sun PDP did it in 33 ms, and XEngine - in only 14 ms. The main reason for that is

Figure 4.3: XACML Requests Loading Time. XACML Policy: demo1; Platform: Configuration 2.

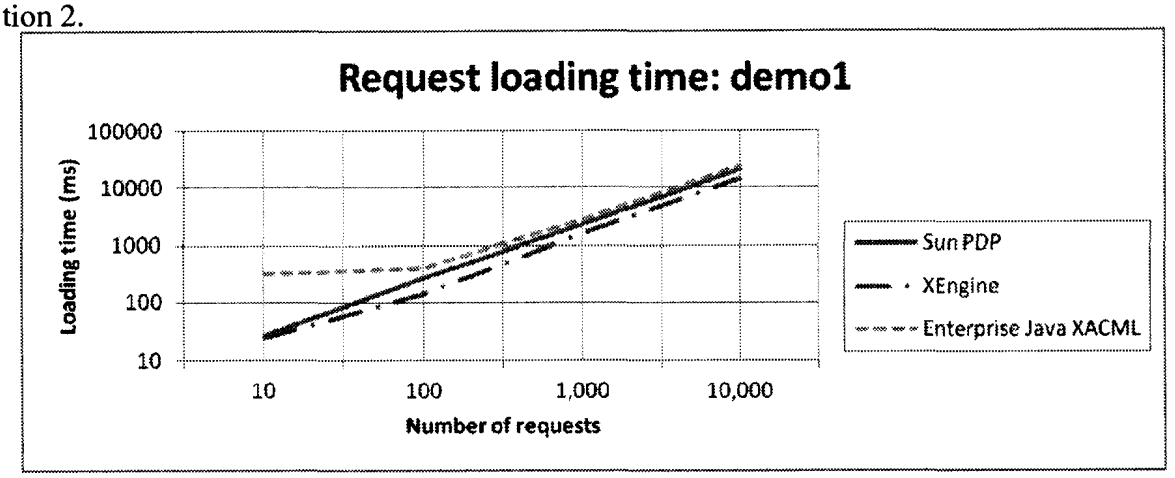


Figure 4.4: XACML Requests Loading Time. XACML Policy: demo2; Platform: Configuration 1.

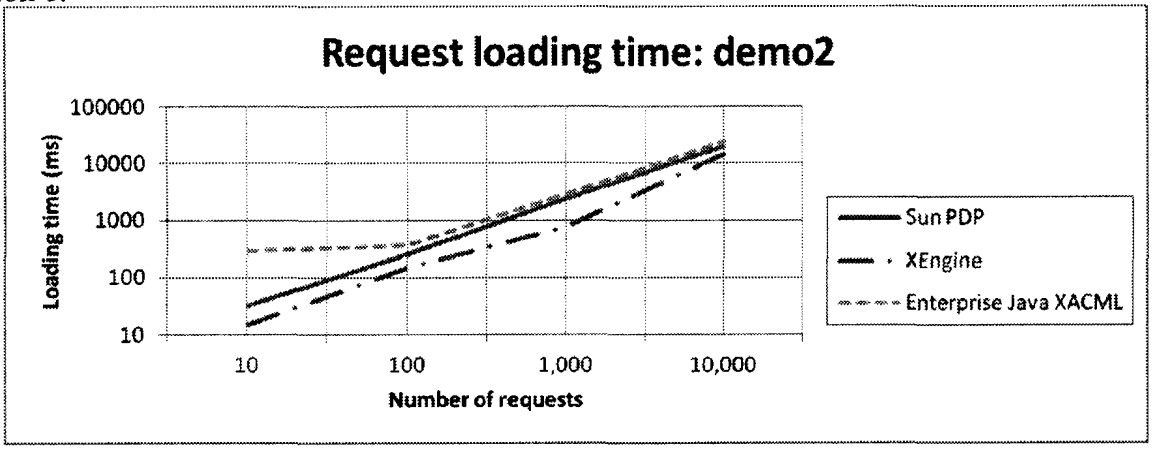


Figure 4.5: XACML Requests Loading Time. XACML Policy: pluto; Platform: Configuration 3.

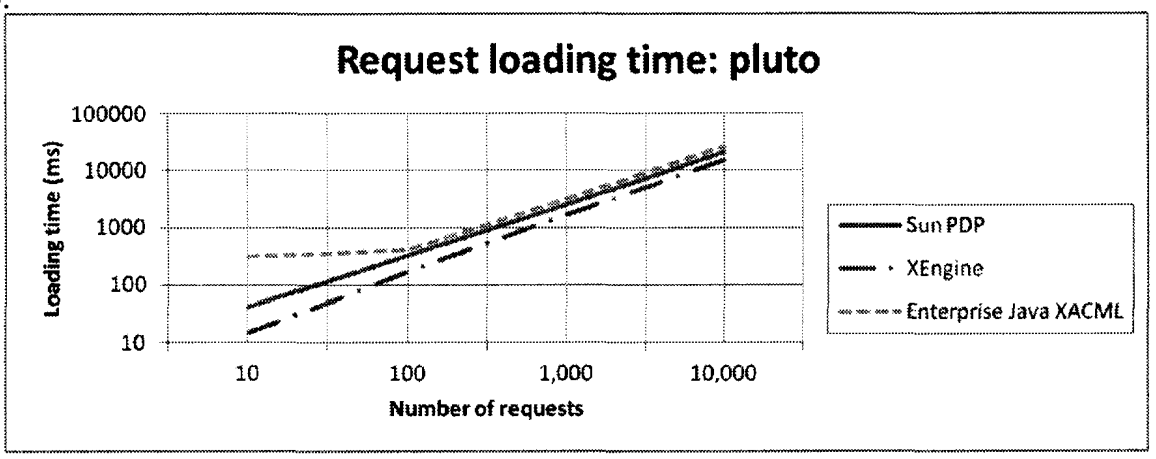


Figure 4.6: XACML Requests Loading Time. XACML Policy: codeA; Platform: Configuration 1.

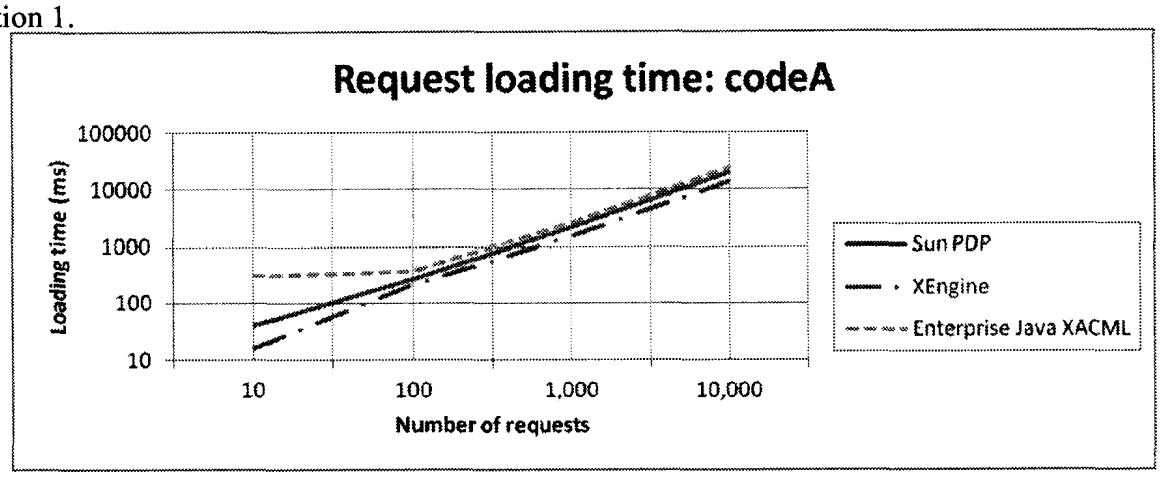


Figure 4.7: XACML Requests Loading Time. XACML Policy: codeB; Platform: Configuration 1.

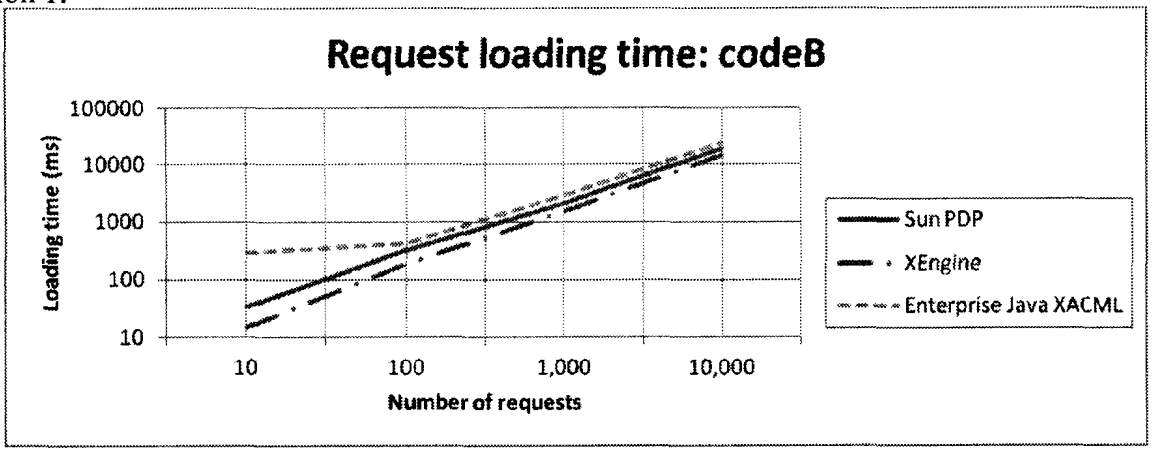


Figure 4.8: XACML Requests Loading Time. XACML Policy: codeC; Platform: Configuration 2.

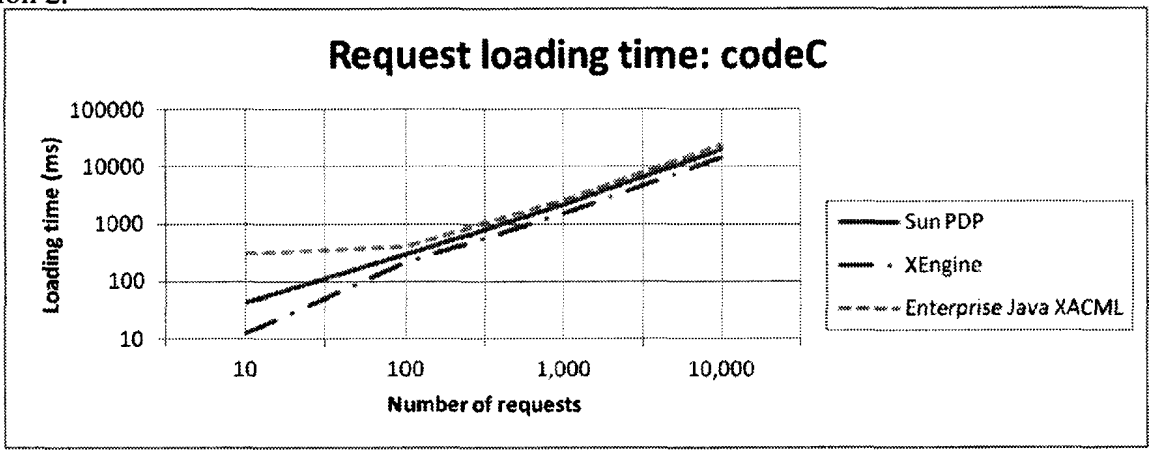


Figure 4.9: XACML Requests Loading Time. XACML Policy: SyntheticPolicy.400; Platform: Configuration 2.

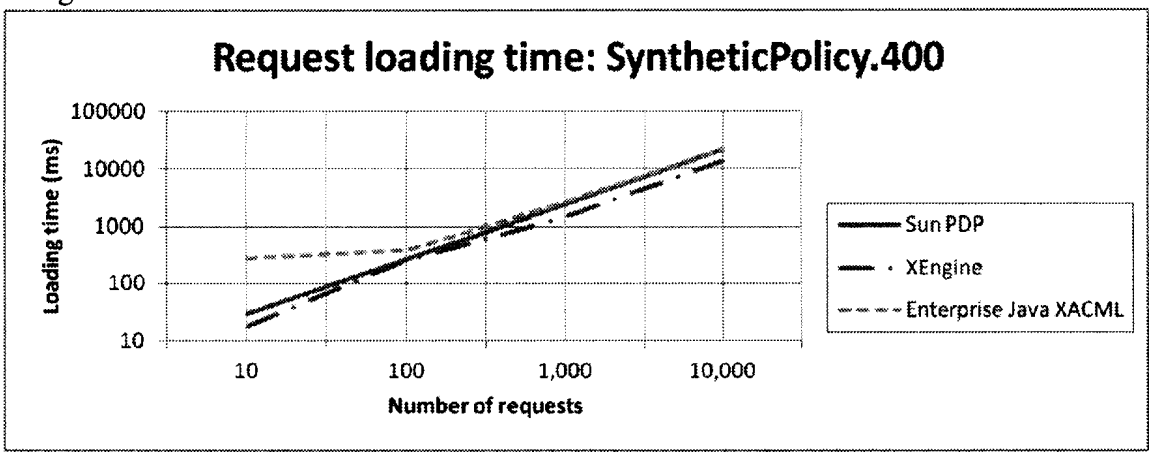


Figure 4.10: XACML Requests Loading Time. XACML Policy: SyntheticPolicy.800; Platform: Configuration 2.

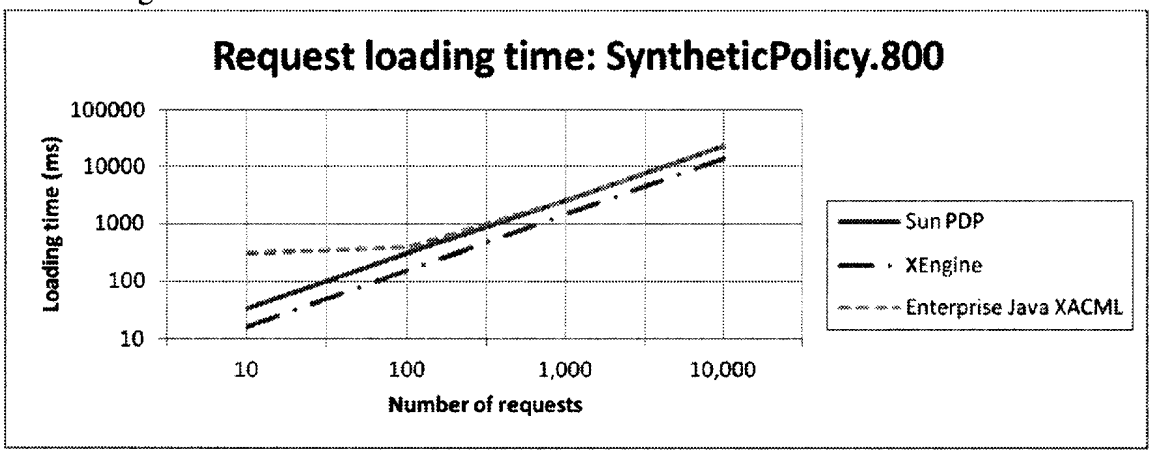


Figure 4.11: XACML Requests Loading Time. XACML Policy: SyntheticPolicy.1600; Platform: Configuration 3.

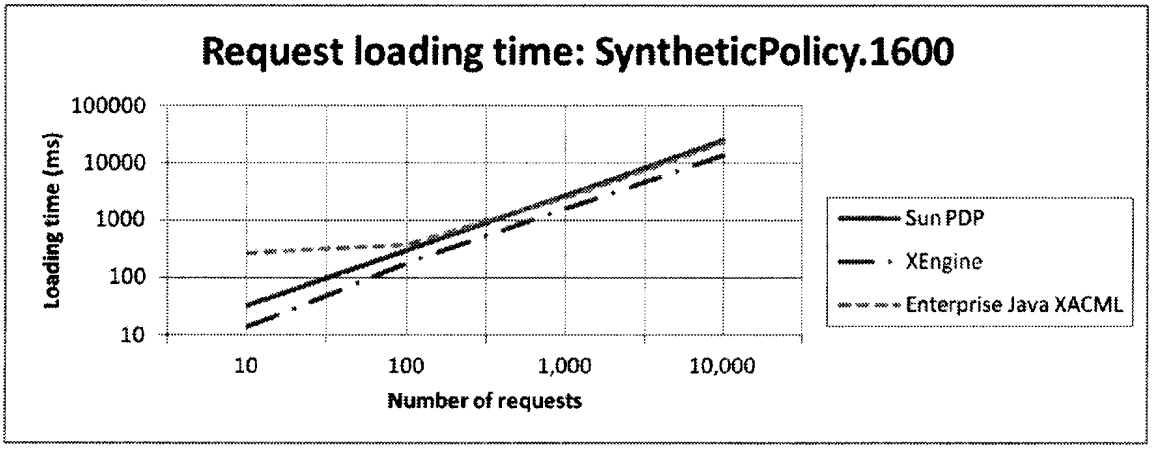
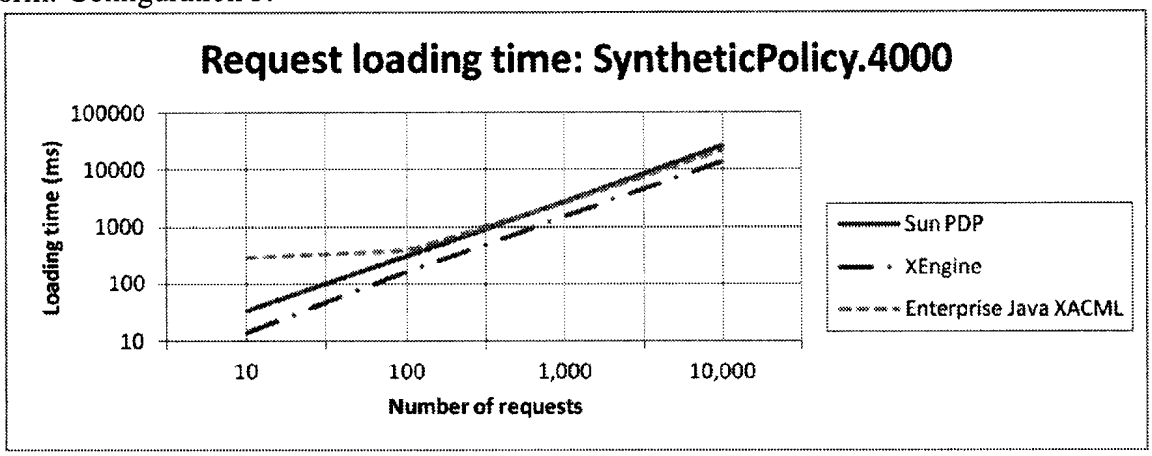


Figure 4.12: XACML Requests Loading Time. XACML Policy: SyntheticPolicy.4000; Platform: Configuration 3.



that Enterprise Java XACML does special indexing and applies a caching mechanism, which consumes certain time at the beginning. Our diagrams show that this time is similar whether loading 10 or 100 requests. For this reason we see almost horizontal lines in that interval. Beyond the first interval, all the engines tend to show approximately the same results, increasing almost linearly, even though XEngine performs slightly better.

The next point to be mentioned is that it is obvious that most of the parameters of our platforms do not influence the outcome at all. Indeed, the hard disk drive is the part used most intensively in the requests loading phase, and all the configurations have had hard disk drives with approximately the same efficiency.

In the following sections we will look at the time taken to evaluate XACML requests against XACML policies, which in general depends on both the policy size and the number of requests.

4.5 Testing Performance on Small Real-Life XACML Policies

All the general settings of the experiments are the same as above mentioned. Figures 4.13-4.16 illustrate the evaluation time for Sun PDP, XEngine, and Enterprise Java XACML engines for each of the four XACML policies contained in our small policies group. It is worth noting

Figure 4.13: Small Real-Life Policies Evaluation Time. XACML Policy: codeA; Platform: Configuration 3.

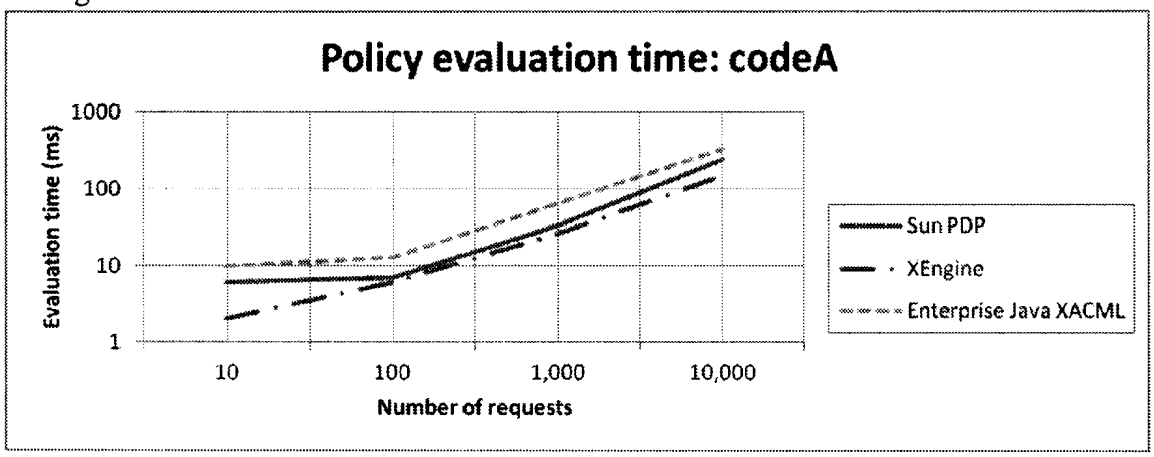
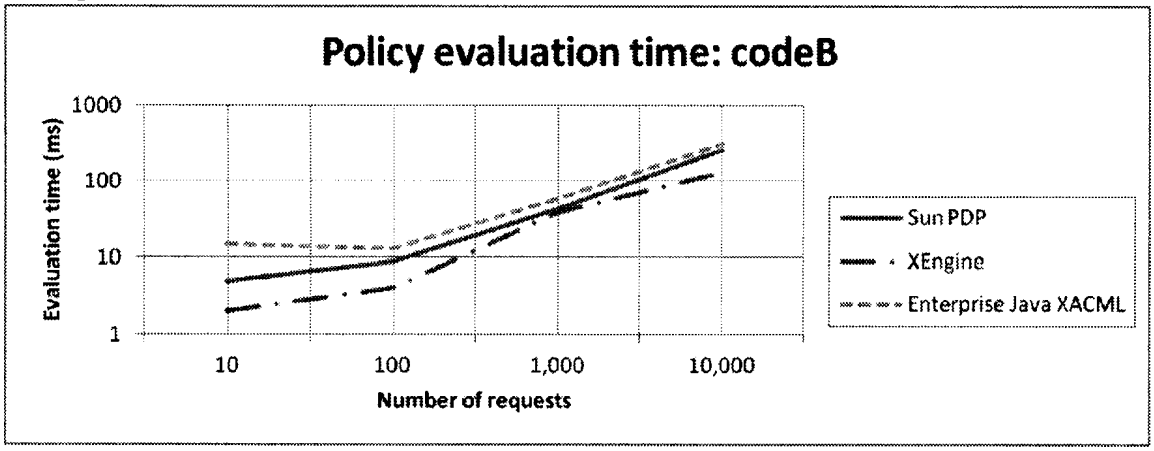


Figure 4.14: Small Real-Life Policies Evaluation Time. XACML Policy: codeB; Platform: Configuration 3.



that the horizontal axes are discrete and present the number of XACML requests evaluated, while the vertical axes present the evaluation time and are in logarithmic scales in terms of milliseconds. We indicate the platform configurations described in the "General settings" section which were used in each experiment. Worth noting is the fact that figures are presented in order of ascending number of rules in policies.

There are a few observations that we can make from the results obtained in the above experiments. First of all, as in the previous set of experiments concerning the request loading time, we can see that in the interval from 10 to 100 requests, Enterprise Java XACML shows approximately horizontal lines. Again, it is due to the fact that Enterprise Java XACML uses a few

Figure 4.15: Small Real-Life Policies Evaluation Time. XACML Policy: codeC; Platform: Configuration 2.

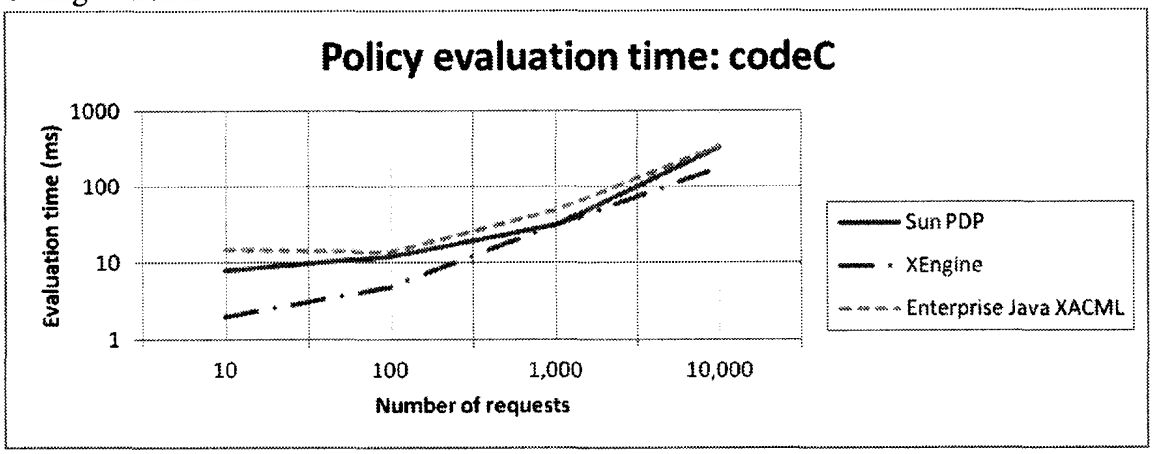
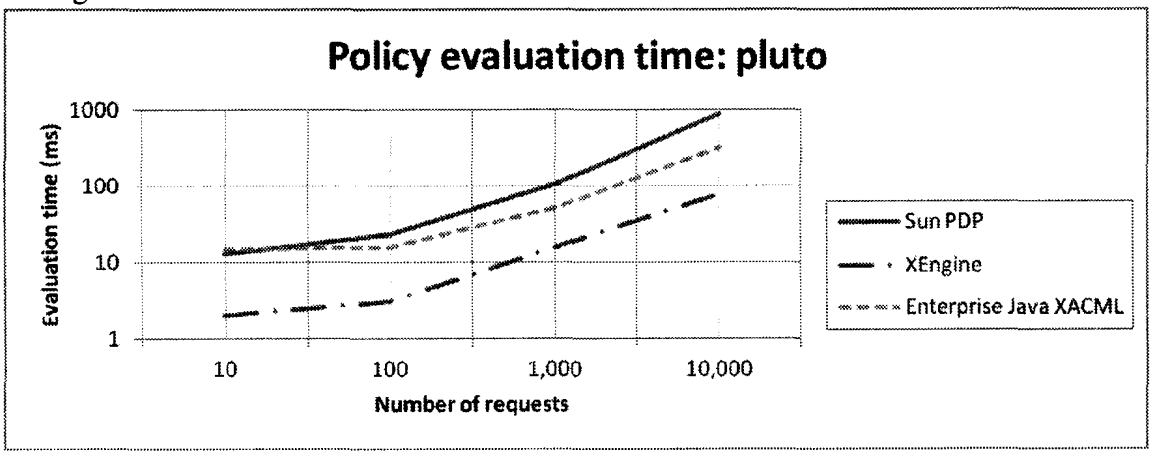


Figure 4.16: Small Real-Life Policies Evaluation Time. XACML Policy: pluto; Platform: Configuration 1.



caching mechanisms, such as decision cache and policy cache, which slow down performance at the beginning.

The next observation is that regardless of which small XACML policy is used and the testbed configuration selected, diagrams of XEngine and Enterprise Java XACML remain almost the same in all experiments. Meanwhile, the performance of Sun PDP decreases correspondingly to the policy size. Namely, it took this engine 239 ms to evaluate 10,000 requests against the codeA policy (2 rules), 258 ms to evaluate 10,000 requests against the codeB policy (3 rules), 326 ms to evaluate 10,000 requests against the codeC policy (4 rules), and finally, 885 ms to evaluate 10,000 requests against the pluto policy (21 rules). For this reason, while for codeA and codeB Enterprise Java XACML has the worst results, for codeC and the workload of 10,000 requests it reaches the performance of Sun PDP, then for pluto and workloads of 100 requests and more, Sun PDP performs worst. Again, the leader is XEngine, even though it is quite close to Sun PDP for the two smallest policies.

4.6 Testing Performance on Large Real-Life XACML Policies

In this section we present the figures which depict the evaluation time for Sun PDP, XEngine, and Enterprise Java XACML engines for each of the four XACML policies contained in our large policies group. These result are plotted on Figures 4.17-4.20.

We still can see that in the interval from 10 to 100 requests Enterprise Java XACML shows approximately horizontal lines due to a few indexing and caching mechanisms. For 100 requests and more, Sun PDP performs worst almost in all cases, even though Enterprise Java XACML gives close results: for instance, for the demo1 policy it took 14 milliseconds to evaluate 100 requests with Enterprise Java XACML and 19 milliseconds with Sun PDP, for 1000 requests the results are 55 milliseconds with Enterprise Java XACML and 78 milliseconds with Sun PDP, and for 10000 requests the results are 351 and 623 milliseconds, correspondingly (see Figure 4.19). XEngine performs best in all cases. Worth noting is the fact that all

Figure 4.17: Large Real-Life Policies Evaluation Time. XACML Policy: continue-a; Platform: Configuration 3.

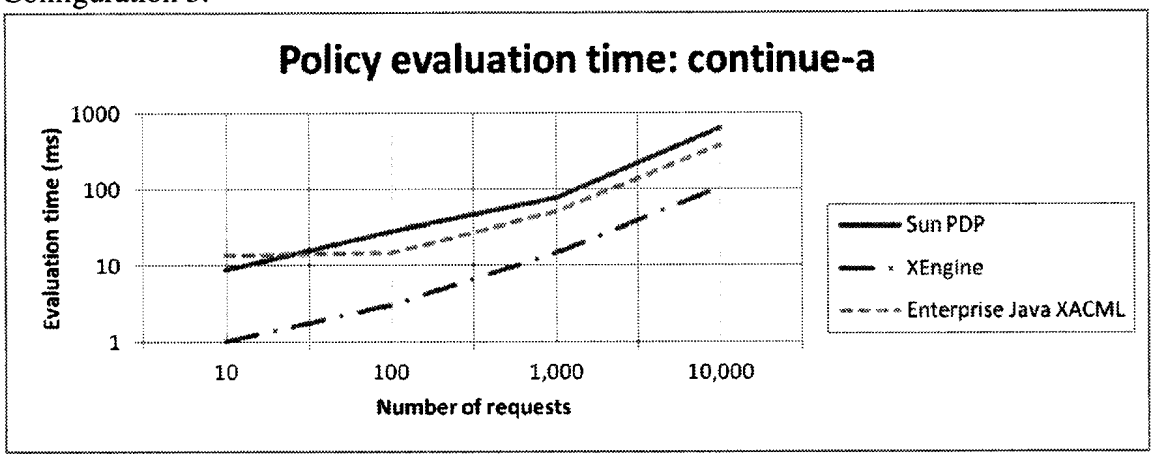


Figure 4.18: Large Real-Life Policies Evaluation Time. XACML Policy: continue-b; Platform: Configuration 3.

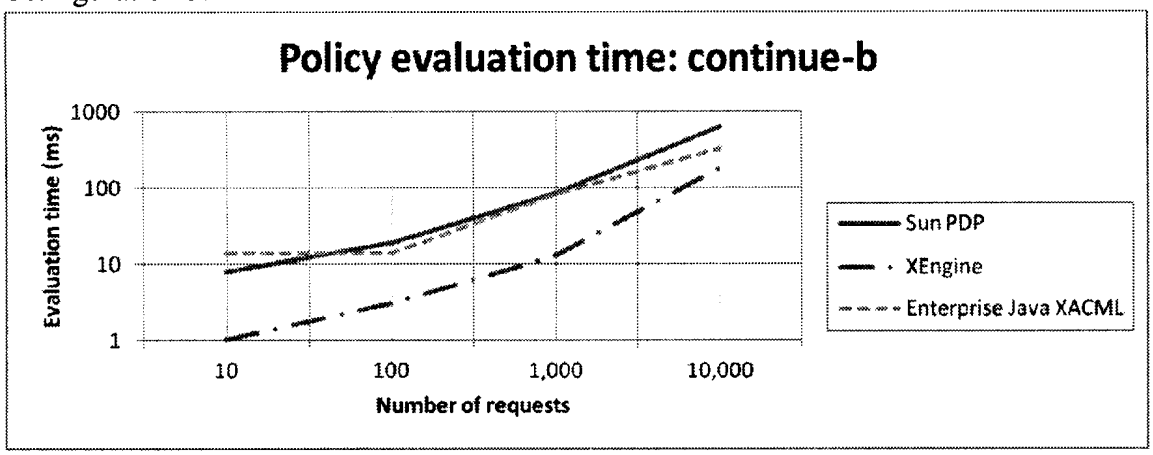


Figure 4.19: Large Real-Life Policies Evaluation Time. XACML Policy: demo1; Platform: Configuration 3.

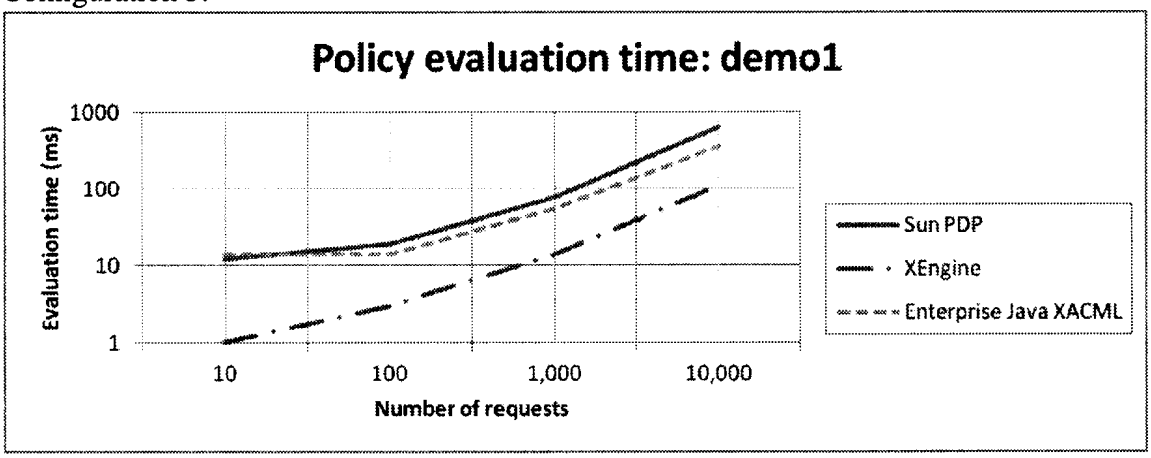


Figure 4.20: Large Real-Life Policies Evaluation Time. XACML Policy: demo2; Platform: Configuration 3.

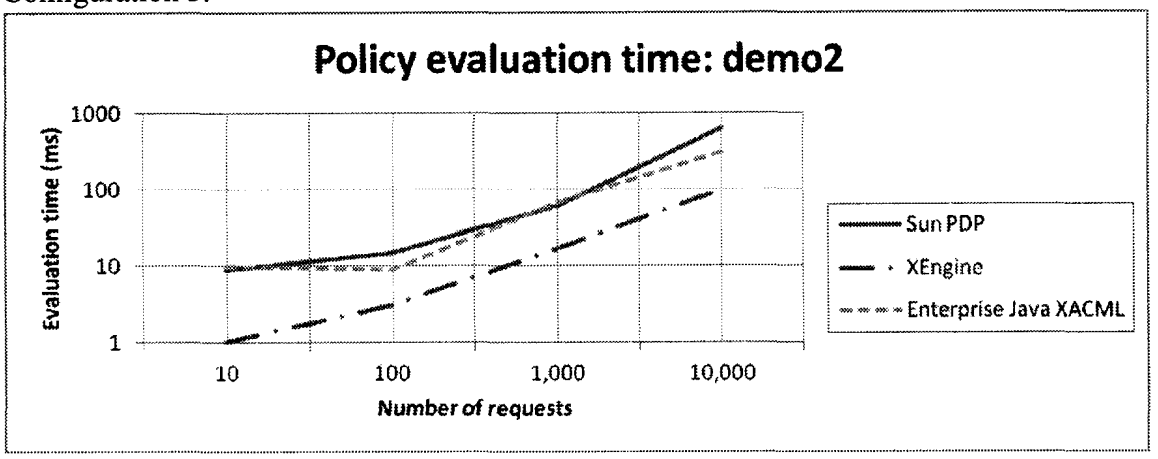
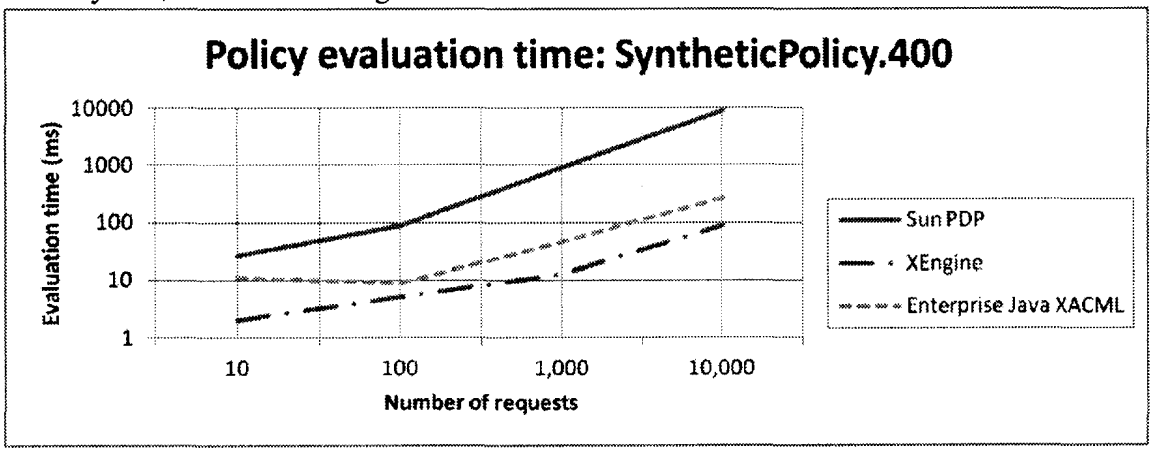


Figure 4.21: Synthetically Generated Policies Evaluation Time. XACML Policy: SyntheticPolicy.400; Platform: Configuration 1.



the experiments in this section were conducted on the platform with configuration 3.

4.7 Testing Performance on Synthetically Generated XACML Policies

In this section we present the figures which depict the evaluation time for Sun PDP, XEngine, and Enterprise Java XACML engines for each of the four synthetically generated XACML policies. We used policies with 400, 800, 1600, and 4000 rules. These results are

Figure 4.22: Synthetically Generated Policies Evaluation Time. XACML Policy: SyntheticPolicy.800; Platform: Configuration 1.

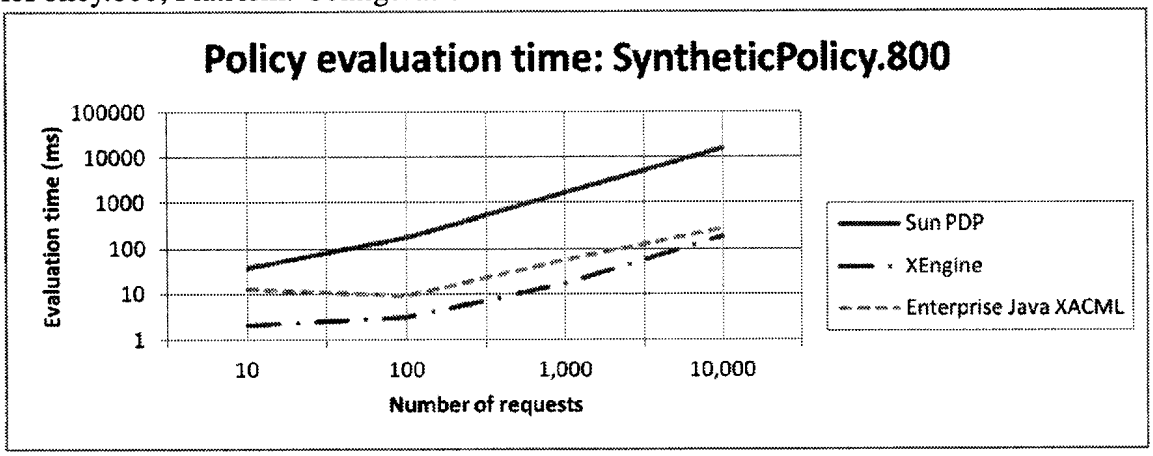


Figure 4.23: Synthetically Generated Policies Evaluation Time. XACML Policy: SyntheticPolicy.1600; Platform: Configuration 1.

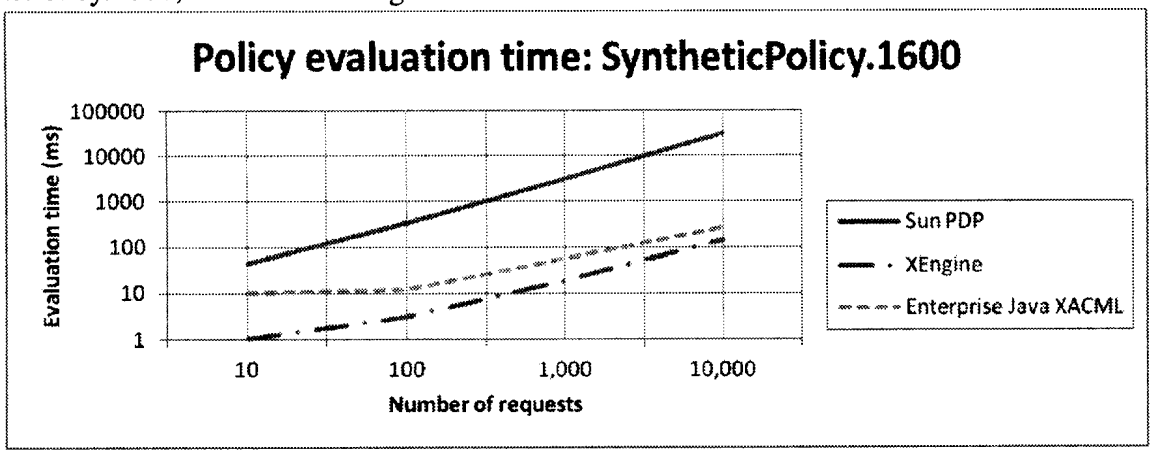
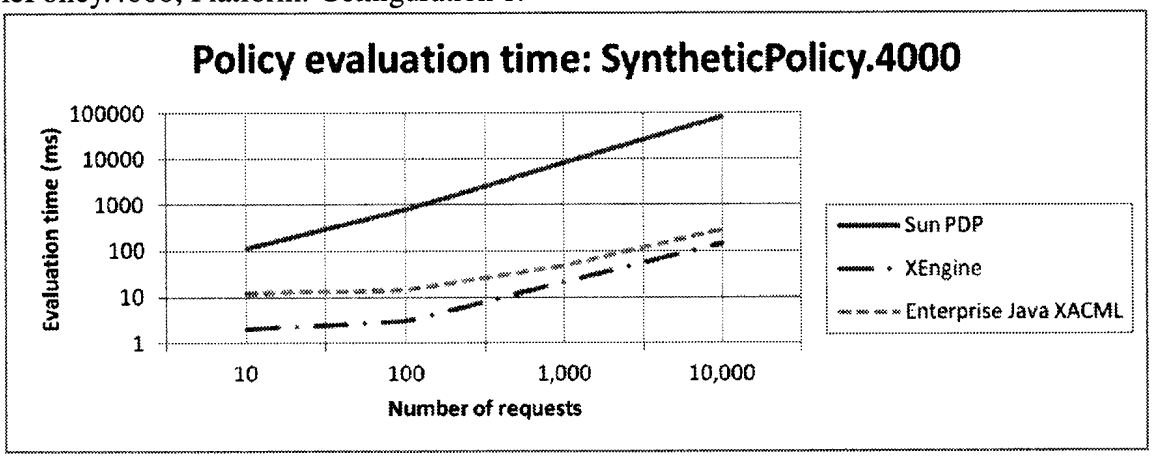


Figure 4.24: Synthetically Generated Policies Evaluation Time. XACML Policy: SyntheticPolicy.4000; Platform: Configuration 1.



shown on Figures 4.21-4.24.

Results of the last experiments are very impressive. They demonstrate that Sun PDP uses brute force searching through all the rules, so the time of evaluation increases proportionally to the number of rules. Another interesting observation is that the time of evaluation with Enterprise Java XACML grows more slowly than that of XEngine. This happens because the advantages of caching and indexing mechanisms eventually come into action for a really great number of rules. For our largest synthetic policy (SyntheticPolicy.4000) it took 141 milliseconds to evaluate 10000 requests with XEngine, 285 milliseconds with Enterprise Java XACML, and 83,033 milliseconds with Sun PDP. Even though in these experiments XEngine is a leader again, it would not be surprising if, for 100000 requests or more, Enterprise Java XACML could beat XEngine due to the effective data structures and caching algorithm, designed for workloads of enterprise scale. Worth noting is the fact that all the experiments in this sections were conducted on the platform with configuration 1.

4.8 Testing Memory Usage

One of the challenging tasks in the Java runtime process is to assess memory intensively used in the external libraries. The main reason is that Java uses the garbage collection mechanism which is quite useful and convenient. However, a programmer cannot force garbage collection in Java; it will only trigger if the Java Virtual Machine evaluates it needs a garbage collection based on the Java heap size. In other words, it is impossible to predict with 100% accuracy when it happens. Even though all our measurements were implemented to avoid involuntary garbage collection, such methods as System.gc () and Runtime.gc () which are used to send request for garbage collection to the Java Virtual Machine do not guarantee that garbage collection will happen. For this reason, all the results in this section are approximate and can be different for other implementations of the Java Virtual Machine and/or other hardware. Moreover, some figures may vary for the same configurations because of the above mentioned methods. The final results are summarized in Table 4.1.

Table 4.1: Memory Usage (Kilobytes).

Policy/Engine	Sun XACML	XEngine	Enterprise Java XACML
codeA	2,528	5,927	1,734
codeB	2,871	5,898	3,003
codeC	2,886	5,906	3,062
pluto	3,869	6,445	3,405
continue-a	4,369	8,078	2,302
continue-b	3,787	7,855	2,238
demo1	4,101	8,058	2,786
demo2	3,803	8,383	2,756
SyntheticPolicy.400	2,309	9,031	2,495
SyntheticPolicy.800	4,076	9,698	3,990
SyntheticPolicy.1600	6,124	15,022	3,989
SyntheticPolicy.4000	20,717	36,309	5,839

Our experiments show that there is no significant difference between the workloads, i.e. the number of XACML requests, mainly because each request is processed sequentially while others are simply located in the cache. For this reason we present the average results from all three platforms for each XACML policy for an engine.

Table 4.1 shows that XEngine has the most memory consumption. While for small and large real-life XACML policies Sun XACML and Enterprise Java XACML have approximately the same results, for synthetically generated policies Sun XACML uses much more memory. Consequently, overall, Enterprise Java XACML performs best, demonstrating low memory consumption and great scalability.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

It is a well-known fact that performance is vital in software, especially in security systems performing access control. Thus, it is crucial to have benchmarking tools that help to evaluate all the subcomponents of a security system. The framework implemented in this work is intended to be an easy-to-use tool that allows testing performance characteristics of arbitrary Java-based XACML engines. It provides a few extendable features that make it possible to add new performance measurements and alternative measuring methods.

We used the framework described in this thesis to evaluate the performance of the three most recently developed and widely used XACML libraries: Sun XACML, XEngine, and Enterprise Java XACML. In our experiments we evaluated the run time of two interconnected procedures: loading XACML requests from disk to memory and then the actual evaluation of the given requests against the XACML policies. The memory consumption of XACML engines was evaluated as well. Our empirical study demonstrated that each XACML engine we tested has its own strengths and weaknesses.

The first characteristic investigated was the XACML requests loading time. Overall, request loading is more expensive with Enterprise Java XACML due to the auxiliary caching and

indexing mechanisms. XEngine performed best of all, while Sun PDP was in the middle.

The next phase was the actual request evaluation time. In our study we selected test cases consisting of three groups of XACML policies: small real-life policies, large real-life policies, and synthetically generated policies. XEngine was a leader in all experiments again. Sun PDP and Enterprise Java XACML showed close results for both the real-life policy groups. However, there was an apparent observation: Sun PDP started with evaluation time values better than Enterprise Java XACML for the smallest policies, and those values began increasing proportionally to the policy sizes, so for the large policy group, Enterprise Java XACML slightly outperformed Sun PDP. The most impressive results were obtained for synthetically generated policies, which contained a great number of rules. Even though XEngine performed best of all again, its evaluation time was increasing faster than that of Enterprise Java XACML, so that they demonstrated close results for a workload of 10000 XACML requests. Sun PDP performed far behind other engines. For instance, evaluation of requests against the largest synthetically generated policy that was used (SyntheticPolicy.4000) with Sun PDP was orders of magnitude slower. The main reason for such a dramatic increasing of evaluation time is that Sun PDP uses brute force to search through all the rules.

The final characteristic we investigated in our empirical study is memory used by the engines during the evaluation phase. XEngine performed much worse than the other two, consuming much more memory. For the small and large real-life policies Sun PDP and Enterprise Java XACML demonstrated approximately the same results. However, for the synthetically generated policies with a great number of rules, Enterprise Java XACML performed better, showing low memory consumption and great potential for scalability.

Overall, summarizing our results, we can make a conclusion that currently XEngine demonstrates the best performance in terms of request loading and evaluation time for these three engines. Memory consumption is the only shortcoming of XEngine revealed in our experiments. Enterprise Java XACML performs quite well for large policies under heavy workloads, demonstrating good scalability and effectiveness. It is also a leader for memory usage. As to

Sun PDP, it performs reasonably well for small policies. When the number of rules grows, its evaluation time increases proportionally. For this reason, Sun PDP is not useful for large policies. Worth noting is the fact that, in general, our findings correspond to the results obtained in [21] and [27].

5.2 Future work

5.2.1 Benchmarking

As discussed earlier, our framework is extendable in a few directions. First of all, it can be extended with importing of new Java-based XACML engines, such as XACMLight [10] and Herasaf [5], for example. Our test cases can be enriched with more complicated real-life policies as well as synthetically generated ones with an even greater number of rules.

The next point to be mentioned is that in this work we focused on measurement of requests loading and evaluation time as well as memory consumption. Apparently, there are many other performance characteristics that could be included in our framework. For instance, it would be interesting to measure such parameters as initialization time and policy loading time. Also, there are many ways of extending the test suite. Additionally to just larger policies, it would be interesting to investigate policies with anomalies and policies with similarity of content.

Finally, in order to evaluate real potential of scalability for those engines which have performed well for workloads up to 10000 XACML requests, experiments with heavier workloads are of special interest.

5.2.2 Software Implementation

Even though the current version of our framework can be considered as a totally functional and easy-to-use tool, there are a few directions for future work. First of all, the current implementation uses a console. That is why a graphical user interface would be beneficial. It should be simple yet to provide full control over the conducting of the experiments as well as to give an opportunity to easily configure all the settings.

Another point for improvement is providing more formats for experimental results. Namely, additionally to a simple text file format, such formats as XML, HTML, and Excel would be useful. Storing the results in a database is a valuable asset as well. Moreover, for some of the parameters investigated, automatic drawing of diagrams with the help of visual aids would significantly improve visualization of data and simplify the results processing.

Finally, our framework is developed as a standalone application. While it has certain benefits, the feature of working in distributed environments can be considered as a great advantage. For this reason, a web version of the framework is needed. It would allow a group of researchers to work together within one project as well as provide an ability to conduct experiments on a cluster or a powerful server. Furthermore, it does not require a lot of modifications in the current software code, as all the crucial methods are isolated well, so they can be transferred easily into an SOA architecture. A simplified web version as an applet or a servlet can be done as well.

Bibliography

[1] Axiomatics. http://www.axiomatics.com/, accessed November 25, 2011.

[2] Enterprise Java XACML Implementation. http://sourceforge.net/projects/java-xacml/, accessed November 25, 2011.

[3] Fedora Commons Repository Software. http://fedora-commons.org/, accessed November 25, 2011.

[4] GeoXACML. http://www.geoxacml.org/, accessed November 25, 2011.

[5] HERASAF. http://herasaf.org/xacmlimpl/index.html, accessed November 25, 2011.

[6] ITU-T Rec X.812 (1995) | ISO/IEC 10181-3:1996 "Security Frameworks for open systems: Access control framework". . http://www.itu.int/rec/T-REC-X.812/, accessed November 25, 2011.

[7] Java SE 6 Documentation. http://download.oracle.com/javase/6/docs/, accessed November 25, 2011.

[8] Log4J Apache Project. http://logging.apache.org/log4j/, accessed November 25, 2011.

[9] Sun's XACML Implementation. http://sunxacml.sourceforge.net, accessed November 25, 2011.

[10] XACMLight. http://sourceforge.net/projects/xacmllight/, accessed November 25, 2011.

[11] XACML.NET. http://mvpos.sourceforge.net/, accessed November 25, 2011.

[12] Anne Anderson. Core and Hierarchical Role Based Access Control (RBAC) profile of XACML. Technical report, OASIS, 2004.

[13] Anne Anderson. XACML Profile for Role Based Access Control (RBAC). Technical Report Draft 1, OASIS, February 2004.

[14] Anne Anderson and Hal Lockhart. SAML 2.0 profile of XACML v2.0. Technical report, OASIS, February 2005.

[15] Cataldo Basile, Antonio Lioy, and Piervito Giovanni Scaglioso. Modern standard-based access control in network services: XACML in action. *INTERNATIONAL JOURNAL OF COMPUTER SCIENCE AND NETWORK SECURITY*, 8, no. 12:296–305, 2008.

[16] Fei Chen and Alex X. Liu. XEngine project. http://xacmlpdp.sourceforge.net/, accessed November 25, 2011.

[17] James Clark and Steve DeRose. XML Path Language (XPath). Recommendation, W3C, 1999. http://www.w3.org/TR/xpath, accessed November 25, 2011.

[18] Kathi Fisler, Shriram Krishnamurthi, Leo A. Meyerovich, and Michael Carl Tschantz. Verification and change-impact analysis of access-control policies. In *ICSE*, pages 196–205, 2005.

[19] Simon Godik and Tim Moses. eXtensible Access Control Markup Language (XACML) Version 1.1. OASIS Standard, August 2003.

[20] Hongxin Hu, Gail-Joon Ahn, and Ketan Kulkarni. Anomaly discovery and resolution in web access control policies. In *SACMAT*, pages 165–174, 2011.

[21] Alex X. Liu, Fei Chen, JeeHyun Hwang, and Tao Xie. XEngine: A fast and scalable XACML policy evaluation engine. In *Proc. International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS 2008)*, pages 265–276, June 2008.

[22] Markus Lorch, Seth Proctor, Rebekah Lepro, Dennis G. Kafura, and Sumit Shah. First experiences using XACML for access control in distributed systems. In *XML Security*, pages 25–37, 2003.

[23] Matthew MacKenzie, Ken Laskey, Francis McCabe, Peter Brown, and Rebekah Metz. Reference Model for Service Oriented Architecture 1.0. Technical Report wd-soa-rm-cd1, OASIS, October 2006.

[24] Evan Martin and Tao Xie. Automated Test Generation for Access Control Policies via Change-Impact Analysis. In *Proc. 3rd International Workshop on Software Engineering for Secure Systems (SESS 2007)*, pages 5–11, May 2007.

[25] E. F. Michiels, editor. *ISO/IEC 10181-7:1996 Information technology Open Systems Interconnection Security frameworks for open systems: Security audit alarms framework.* ISO/IEC, Geneva, int. standard edition, 1996.

[26] Nick Ragouzis, John Hughes, Rob Philpott, and Eve Maler. Security Assertion Markup Language (SAML) V2.0 Technical Overview. Technical report, OASIS, 2006.

[27] Fatih Turkmen and Bruno Crispo. Performance evaluation of XACML PDP implementations. In *SWS*, pages 37–44, 2008.

[28] Manish Verma. XML Security: Control information access with XACML. Technical report, IBM, October 2004. https://www.ibm.com/developerworks/xml/library/x-xacml/, accessed November 25, 2011.

[29] R. Yavatkar, D. Pendarakis, and R. Guerin. A Framework for Policy-based Admission Control. RFC 2753, January 2000. http://www.faqs.org/rfcs/rfc2753.html, accessed November 25, 2011.

Appendix A

XACML Policy Examples

A.1 codeA

```
<PolicySet PolicySetId="RPSlist" PolicyCombiningAlgId="urn:oasis:names:tc:xacml:
1.0:policy-combining-algorithm:permit-overrides">
  <Target/>
  <PolicySet PolicySetId="RPSlist.0" PolicyCombiningAlgId="urn:oasis:names:tc:
xacml:1.0:policy-combining-algorithm:permit-overrides">
    <Target>
      <Subjects>
        <Subject>
          <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:
string-equal">
            <AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#string">Faculty</AttributeValue>
            <SubjectAttributeDesignator SubjectCategory="urn:oasis:names:tc:xacml:
1.0:subject-category:access-subject" AttributeId="role"
DataType="http://www.w3.org/2001/XMLSchema#string"/>
          </SubjectMatch>
        </Subject>
      </Subjects>
    </Target>
    <PolicySet PolicySetId="RPSlist.0.0" PolicyCombiningAlgId="urn:oasis:names:
tc:xacml:1.0:policy-combining-algorithm:permit-overrides">
      <Target/>
      <Policy PolicyId="RPSlist.0.0.0" RuleCombiningAlgId="urn:oasis:names:tc:
xacml:1.0:rule-combining-algorithm:permit-overrides">
        <Target/>
        <Rule RuleId="RPSlist.0.0.0.r.1" Effect="Permit">
          <Target>
            <Resources>
              <Resource>
```

```
                <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:
string-equal">
                  <AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#string">ExternalGrades</AttributeValue>
                  <ResourceAttributeDesignator AttributeId="resource-class"
DataType="http://www.w3.org/2001/XMLSchema#string"/>
                </ResourceMatch>
              </Resource>
              <Resource>
                <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:
string-equal">
                  <AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#string">InternalGrades</AttributeValue>
                  <ResourceAttributeDesignator AttributeId="resource-class"
DataType="http://www.w3.org/2001/XMLSchema#string"/>
                </ResourceMatch>
              </Resource>
            </Resources>
            <Actions>
              <Action>
                <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:
string-equal">
                  <AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#string">Assign</AttributeValue>
                  <ActionAttributeDesignator AttributeId="command"
DataType="http://www.w3.org/2001/XMLSchema#string"/>
                </ActionMatch>
              </Action>
              <Action>
                <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:
string-equal">
                  <AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#string">View</AttributeValue>
                  <ActionAttributeDesignator AttributeId="command"
DataType="http://www.w3.org/2001/XMLSchema#string"/>
                </ActionMatch>
              </Action>
            </Actions>
          </Target>
        </Rule>
      </Policy>
    </PolicySet>
  </PolicySet>
  <PolicySet PolicySetId="RPSlist.1" PolicyCombiningAlgId="urn:oasis:names:tc:
xacml:1.0:policy-combining-algorithm:permit-overrides">
```

```
    <Target>
      <Subjects>
        <Subject>
          <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:
string-equal">
            <AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#string">Student</AttributeValue>
            <SubjectAttributeDesignator SubjectCategory="urn:oasis:names:tc:
xacml:1.0:subject-category:access-subject" AttributeId="role"
DataType="http://www.w3.org/2001/XMLSchema#string"/>
          </SubjectMatch>
        </Subject>
      </Subjects>
    </Target>
    <PolicySet PolicySetId="RPSlist.1.0" PolicyCombiningAlgId=
"urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:permit-overrides">
      <Target/>
      <Policy PolicyId="RPSlist.1.0.0" RuleCombiningAlgId="urn:oasis:names:tc:
xacml:1.0:rule-combining-algorithm:permit-overrides">
        <Target/>
        <Rule RuleId="RPSlist.1.0.0.r.1" Effect="Permit">
          <Target>
            <Resources>
              <Resource>
                <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:
string-equal">
                  <AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#string">ExternalGrades</AttributeValue>
                  <ResourceAttributeDesignator AttributeId="resource-class"
DataType="http://www.w3.org/2001/XMLSchema#string"/>
                </ResourceMatch>
              </Resource>
            </Resources>
            <Actions>
              <Action>
                <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:
string-equal">
                  <AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#string">Receive</AttributeValue>
                  <ActionAttributeDesignator AttributeId="command"
DataType="http://www.w3.org/2001/XMLSchema#string"/>
                </ActionMatch>
              </Action>
            </Actions>
          </Target>
```

```
        </Rule>
      </Policy>
    </PolicySet>
  </PolicySet>
</PolicySet>
```

A.2 codeB

```
<PolicySet PolicySetId="RPSlist" PolicyCombiningAlgId="urn:oasis:names:
tc:xacml:1.0:policy-combining-algorithm:permit-overrides">
  <Target/>
  <PolicySet PolicySetId="RPSlist.0" PolicyCombiningAlgId="urn:oasis:names:
tc:xacml:1.0:policy-combining-algorithm:permit-overrides">
    <Target>
      <Subjects>
        <Subject>
          <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:
string-equal">
            <AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#string">Faculty</AttributeValue>
            <SubjectAttributeDesignator SubjectCategory="urn:oasis:names:
tc:xacml:1.0:subject-category:access-subject" AttributeId="role"
DataType="http://www.w3.org/2001/XMLSchema#string"/>
          </SubjectMatch>
        </Subject>
      </Subjects>
    </Target>
    <PolicySet PolicySetId="RPSlist.0.0" PolicyCombiningAlgId="urn:oasis:
names:tc:xacml:1.0:policy-combining-algorithm:permit-overrides">
      <Target/>
      <Policy PolicyId="RPSlist.0.0.0" RuleCombiningAlgId="urn:oasis:names:
tc:xacml:1.0:rule-combining-algorithm:permit-overrides">
        <Target/>
        <Rule RuleId="RPSlist.0.0.0.r.1" Effect="Permit">
          <Target>
            <Resources>
              <Resource>
                <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:
string-equal">
                  <AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#string">ExternalGrades</AttributeValue>
                  <ResourceAttributeDesignator AttributeId="resource-class"
DataType="http://www.w3.org/2001/XMLSchema#string"/>
                </ResourceMatch>
```

```
            </Resource>
            <Resource>
              <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:
string-equal">
                <AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#string">InternalGrades</AttributeValue>
                <ResourceAttributeDesignator AttributeId="resource-class"
DataType="http://www.w3.org/2001/XMLSchema#string"/>
              </ResourceMatch>
            </Resource>
          </Resources>
          <Actions>
            <Action>
              <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:
string-equal">
                <AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#string">Assign</AttributeValue>
                <ActionAttributeDesignator AttributeId="command"
DataType="http://www.w3.org/2001/XMLSchema#string"/>
              </ActionMatch>
            </Action>
            <Action>
              <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:
function:string-equal">
                <AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#string">View</AttributeValue>
                <ActionAttributeDesignator AttributeId="command"
DataType="http://www.w3.org/2001/XMLSchema#string"/>
              </ActionMatch>
            </Action>
          </Actions>
        </Target>
      </Rule>
    </Policy>
  </PolicySet>
  <PolicySet PolicySetId="RPSlist.1" PolicyCombiningAlgId="urn:oasis:names:
tc:xacml:1.0:policy-combining-algorithm:permit-overrides">
    <Target>
      <Subjects>
        <Subject>
          <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:
string-equal">
            <AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#string">Student</AttributeValue>
```

```
            <SubjectAttributeDesignator SubjectCategory="urn:oasis:names:
tc:xacml:1.0:subject-category:access-subject" AttributeId="role"
DataType="http://www.w3.org/2001/XMLSchema#string"/>
          </SubjectMatch>
        </Subject>
      </Subjects>
    </Target>
    <PolicySet PolicySetId="RPSlist.1.0" PolicyCombiningAlgId="urn:oasis:names:
tc:xacml:1.0:policy-combining-algorithm:permit-overrides">
      <Target/>
      <Policy PolicyId="RPSlist.1.0.0" RuleCombiningAlgId="urn:oasis:names:tc:
xacml:1.0:rule-combining-algorithm:permit-overrides">
        <Target/>
        <Rule RuleId="RPSlist.1.0.0.r.1" Effect="Permit">
          <Target>
            <Resources>
              <Resource>
                <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:
string-equal">
                  <AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#string">ExternalGrades</AttributeValue>
                  <ResourceAttributeDesignator AttributeId="resource-class"
DataType="http://www.w3.org/2001/XMLSchema#string"/>
                </ResourceMatch>
              </Resource>
            </Resources>
            <Actions>
              <Action>
                <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:
string-equal">
                  <AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#string">Receive</AttributeValue>
                  <ActionAttributeDesignator AttributeId="command"
DataType="http://www.w3.org/2001/XMLSchema#string"/>
                </ActionMatch>
              </Action>
            </Actions>
          </Target>
        </Rule>
      </Policy>
    </PolicySet>
  </PolicySet>
  <PolicySet PolicySetId="RPSlist.2" PolicyCombiningAlgId="urn:oasis:names:
tc:xacml:1.0:policy-combining-algorithm:permit-overrides">
    <Target>
```

```
      <Subjects>
        <Subject>
          <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:
string-equal">
            <AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#string">TA</AttributeValue>
            <SubjectAttributeDesignator SubjectCategory="urn:oasis:names:
tc:xacml:1.0:subject-category:access-subject" AttributeId="role"
DataType="http://www.w3.org/2001/XMLSchema#string"/>
          </SubjectMatch>
        </Subject>
      </Subjects>
    </Target>
    <PolicySet PolicySetId="RPSlist.2.0" PolicyCombiningAlgId="urn:oasis:
names:tc:xacml:1.0:policy-combining-algorithm:permit-overrides">
      <Target/>
      <Policy PolicyId="RPSlist.2.0.0" RuleCombiningAlgId="urn:oasis:
names:tc:xacml:1.0:rule-combining-algorithm:permit-overrides">
        <Target/>
        <Rule RuleId="RPSlist.2.0.0.r.1" Effect="Permit">
          <Target>
            <Resources>
              <Resource>
                <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:
string-equal">
                  <AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#string">ExternalGrades</AttributeValue>
                  <ResourceAttributeDesignator AttributeId="resource-class"
DataType="http://www.w3.org/2001/XMLSchema#string"/>
                </ResourceMatch>
              </Resource>
              <Resource>
                <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:
string-equal">
                  <AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#string">InternalGrades</AttributeValue>
                  <ResourceAttributeDesignator AttributeId="resource-class"
DataType="http://www.w3.org/2001/XMLSchema#string"/>
                </ResourceMatch>
              </Resource>
            </Resources>
            <Actions>
              <Action>
                <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:
string-equal">
```

```
                    <AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#string">Assign</AttributeValue>
                    <ActionAttributeDesignator AttributeId="command"
DataType="http://www.w3.org/2001/XMLSchema#string"/>
                  </ActionMatch>
                </Action>
                <Action>
                  <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:
string-equal">
                    <AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#string">View</AttributeValue>
                    <ActionAttributeDesignator AttributeId="command"
DataType="http://www.w3.org/2001/XMLSchema#string"/>
                  </ActionMatch>
                </Action>
              </Actions>
            </Target>
          </Rule>
        </Policy>
      </PolicySet>
    </PolicySet>
</PolicySet>
```

A.3 codeC

```
<PolicySet PolicySetId="RPSlist" PolicyCombiningAlgId="urn:oasis:names:tc:
xacml:1.0:policy-combining-algorithm:permit-overrides">
  <Target/>
  <PolicySet PolicySetId="RPSlist.0" PolicyCombiningAlgId="urn:oasis:names:
tc:xacml:1.0:policy-combining-algorithm:permit-overrides">
    <Target>
      <Subjects>
        <Subject>
          <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:
string-equal">
            <AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#string">Faculty</AttributeValue>
            <SubjectAttributeDesignator SubjectCategory="urn:oasis:names:tc:
xacml:1.0:subject-category:access-subject" AttributeId="role"
DataType="http://www.w3.org/2001/XMLSchema#string"/>
          </SubjectMatch>
        </Subject>
      </Subjects>
    </Target>
```

```
    <PolicySet PolicySetId="RPSlist.0.0" PolicyCombiningAlgId="urn:oasis:
names:tc:xacml:1.0:policy-combining-algorithm:permit-overrides">
      <Target/>
      <Policy PolicyId="RPSlist.0.0.0" RuleCombiningAlgId="urn:oasis:
names:tc:xacml:1.0:rule-combining-algorithm:permit-overrides">
        <Target/>
        <Rule RuleId="RPSlist.0.0.0.r.1" Effect="Permit">
          <Target>
            <Resources>
              <Resource>
                <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:
string-equal">
                  <AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#string">ExternalGrades</AttributeValue>
                  <ResourceAttributeDesignator AttributeId="resource-class"
DataType="http://www.w3.org/2001/XMLSchema#string"/>
                </ResourceMatch>
              </Resource>
            </Resources>
            <Actions>
              <Action>
                <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:
string-equal">
                  <AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#string">Assign</AttributeValue>
                  <ActionAttributeDesignator AttributeId="command"
DataType="http://www.w3.org/2001/XMLSchema#string"/>
                </ActionMatch>
              </Action>
              <Action>
                <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:
string-equal">
                  <AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#string">View</AttributeValue>
                  <ActionAttributeDesignator AttributeId="command"
DataType="http://www.w3.org/2001/XMLSchema#string"/>
                </ActionMatch>
              </Action>
            </Actions>
          </Target>
        </Rule>
      </Policy>
      <PolicySet PolicySetId="RPSlist.0.0.1" PolicyCombiningAlgId="urn:oasis:
names:tc:xacml:1.0:policy-combining-algorithm:permit-overrides">
        <Target/>
```

```
        <Policy PolicyId="RPSlist.0.0.1.0" RuleCombiningAlgId="urn:oasis:
names:tc:xacml:1.0:rule-combining-algorithm:permit-overrides">
          <Target/>
          <Rule RuleId="RPSlist.0.0.1.0.r.1" Effect="Permit">
            <Target>
              <Resources>
                <Resource>
                  <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:
string-equal">
                    <AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#string">InternalGrades</AttributeValue>
                    <ResourceAttributeDesignator AttributeId="resource-class"
DataType="http://www.w3.org/2001/XMLSchema#string"/>
                  </ResourceMatch>
                </Resource>
              </Resources>
              <Actions>
                <Action>
                  <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:
string-equal">
                    <AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#string">Assign</AttributeValue>
                    <ActionAttributeDesignator AttributeId="command"
DataType="http://www.w3.org/2001/XMLSchema#string"/>
                  </ActionMatch>
                </Action>
                <Action>
                  <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:
string-equal">
                    <AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#string">View</AttributeValue>
                    <ActionAttributeDesignator AttributeId="command"
DataType="http://www.w3.org/2001/XMLSchema#string"/>
                  </ActionMatch>
                </Action>
              </Actions>
            </Target>
          </Rule>
        </Policy>
      </PolicySet>
    </PolicySet>
  </PolicySet>
  <PolicySet PolicySetId="RPSlist.1" PolicyCombiningAlgId="urn:oasis:names:
tc:xacml:1.0:policy-combining-algorithm:permit-overrides">
    <Target>
```

```
      <Subjects>
        <Subject>
          <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:
string-equal">
            <AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#string">Student</AttributeValue>
            <SubjectAttributeDesignator SubjectCategory="urn:oasis:names:tc:
xacml:1.0:subject-category:access-subject" AttributeId="role"
DataType="http://www.w3.org/2001/XMLSchema#string"/>
          </SubjectMatch>
        </Subject>
      </Subjects>
    </Target>
    <PolicySet PolicySetId="RPSlist.1.0" PolicyCombiningAlgId="urn:oasis:names:
tc:xacml:1.0:policy-combining-algorithm:permit-overrides">
      <Target/>
      <Policy PolicyId="RPSlist.1.0.0" RuleCombiningAlgId="urn:oasis:names:tc:
xacml:1.0:rule-combining-algorithm:permit-overrides">
        <Target/>
        <Rule RuleId="RPSlist.1.0.0.r.1" Effect="Permit">
          <Target>
            <Resources>
              <Resource>
                <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:
string-equal">
                  <AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#string">ExternalGrades</AttributeValue>
                  <ResourceAttributeDesignator AttributeId="resource-class"
DataType="http://www.w3.org/2001/XMLSchema#string"/>
                </ResourceMatch>
              </Resource>
            </Resources>
            <Actions>
              <Action>
                <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:
string-equal">
                  <AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#string">Receive</AttributeValue>
                  <ActionAttributeDesignator AttributeId="command"
DataType="http://www.w3.org/2001/XMLSchema#string"/>
                </ActionMatch>
              </Action>
            </Actions>
          </Target>
        </Rule>
```

```
      </Policy>
    </PolicySet>
  </PolicySet>
  <PolicySet PolicySetId="RPSlist.2" PolicyCombiningAlgId="urn:oasis:names:tc:
xacml:1.0:policy-combining-algorithm:permit-overrides">
    <Target>
      <Subjects>
        <Subject>
          <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:
string-equal">
            <AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#string">TA</AttributeValue>
            <SubjectAttributeDesignator SubjectCategory="urn:oasis:names:tc:
xacml:1.0:subject-category:access-subject" AttributeId="role"
DataType="http://www.w3.org/2001/XMLSchema#string"/>
          </SubjectMatch>
        </Subject>
      </Subjects>
    </Target>
    <PolicySet PolicySetId="RPSlist.2.0" PolicyCombiningAlgId="urn:oasis:
names:tc:xacml:1.0:policy-combining-algorithm:permit-overrides">
      <Target/>
      <Policy PolicyId="RPSlist.2.0.0" RuleCombiningAlgId="urn:oasis:names:
tc:xacml:1.0:rule-combining-algorithm:permit-overrides">
        <Target/>
        <Rule RuleId="RPSlist.2.0.0.r.1" Effect="Permit">
          <Target>
            <Resources>
              <Resource>
                <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:
string-equal">
                  <AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#string">InternalGrades</AttributeValue>
                  <ResourceAttributeDesignator AttributeId="resource-class"
DataType="http://www.w3.org/2001/XMLSchema#string"/>
                </ResourceMatch>
              </Resource>
            </Resources>
            <Actions>
              <Action>
                <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:
string-equal">
                  <AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#string">Assign</AttributeValue>
                  <ActionAttributeDesignator AttributeId="command"
```

```
DataType="http://www.w3.org/2001/XMLSchema#string"/>
                </ActionMatch>
              </Action>
              <Action>
                <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:
string-equal">
                  <AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#string">View</AttributeValue>
                  <ActionAttributeDesignator AttributeId="command"
DataType="http://www.w3.org/2001/XMLSchema#string"/>
                </ActionMatch>
              </Action>
            </Actions>
          </Target>
        </Rule>
      </Policy>
    </PolicySet>
  </PolicySet>
</PolicySet>
```