5-15-2009

Toward Visualizing Potential Policy Conflicts in eXtensible Access Control Markup Language (XACML)

William Domingo Rosa
University of New Orleans

Toward Visualizing Potential Policy Conflicts in eXtensible Access Control Markup Language (XACML)

A Thesis

Submitted to the Graduate Faculty of the
University of New Orleans
in partial fulfillment of the
requirements for the degree of

Master of Science
in
Computer Science

by

William Domingo Rosa

B.S. The University of West Florida, 2002

May 2009

Dedication

This thesis is dedicated to my family. Thank you Christine for the support you have given me throughout my journey. I could have never done this without you. Thank you Willy, Sammy, Dominic, and Eva-Marie for enduring this challenge and giving me some peace and quiet (at times) to finish my research.

Acknowledgment

I would like to thank my advisor Dr. Shengru Tu. The past year has been difficult since I no longer live in Louisiana. Since the day I left Louisiana to embark in a new challenge in Huntsville, Alabama, Dr. Tu has been there to ensure that I could still accomplish my goal of finishing my degree from the University of New Orleans. This is a job he took upon and followed through even when he was no longer the Computer Science graduate advisor.

I would also like to thank Ms. Amanda Athey of the graduate school. She was always readily available to assist me with any issue ensuring that I had everything I needed to apply for graduation.

Last but not least, I would like to thank the faculty in the Computer Science department for the great job they do and for the support they show their students on a daily basis. Thank you for your guidance and for giving me the opportunity to reach my goal.

Table of Contents

List of Figures

Abstract

The eXtensible Access Control Markup Language (XACML) is allowing enterprises to implement a standard way of access control to their resources. Security administrators no longer need to duplicate their efforts in writing multiple policies for different resources since a XACML policy can be applied to multiple resources. Companies such as IBM, Sun Microsystems, Oracle, and Cisco are already providing integration of XACML in their products.

Although the introduction of XACML has provided enterprises with a better approach of commonly realizing access control, there are still inconsistencies that policy administrators need to be aware of. This thesis identifies some of the inconsistencies in XACML today and introduces a new tool that can be used to visualize some of those inconsistencies. This new tool could be used by a policy administrator to visualize possible conflicting data among a set of policies.

Keywords: XACML, OASIS, XML, DOM, XPath, Java, Derby

Chapter 1 Introduction

With the focus in recent years to have data more accessible, it becomes more important to find a common ground for resource access control. A way of enforcing access control is needed at the enforcement level in order to provide a higher level of access control granularity. Provided that a user level policy states that authorized personnel have access to personal information, then the question arises at the enforcement level; who is authorized personnel and what resources are personal information?

The popularity of the Extensible Markup Language (XML) has increased the need for a common language that can protect different portions of an XML document. Since XML documents are basically databases, it is vital to be able to protect the information they contain. For instance, should a receptionist at a medical office be allowed to view the patient's diagnosis or just the patient's contact and appointment information?

These kinds of questions are what the eXtensible Access Control Markup Language (XACML) is able to address. In a world of Service Oriented Architecture (SOA), which supports the exchanging of data among different applications, it is key to be able to control access with a high degree of accuracy and fine degree of granularity.

1.1 Main Goals

As XACML provides a higher degree of access control granularity, there are possible inconsistencies within its policies and rules that can lead to undesirable results. The main goal of this thesis project is to experiment visualization of access policies in XACML and help us identify possible inconsistencies among different policies. In doing so, I have prototyped a simple visualization tool.

1.2 Objectives

In order to move toward providing a potential solution to inconsistencies among policies, a Java Swing application has been developed. The objective of this prototype application is to assist the user to determine if there are inconsistencies among policies or policy sets. Additionally, a visualization of such inconsistencies is to be provided to the user. As a result, the user does not need to carryout error-prone code inspection against multiple XACML documents. Rather, by just a few mouse clicks, the policy comparison process can be achieved visually which is faster and more effective.

Chapter 2 Background

The need for a common and general approach for industries to secure and control access to data has allowed computing platform vendors to develop new products which can be used universally throughout the enterprise. The idea of configuring multiple points of enforcement in an enterprise to protect data is costly and can potentially be unreliable when it comes to modifying the enterprise security policies. Further, it becomes extremely difficult to implement a new security policy when the security measures in place are dispersed throughout the enterprise.

These are some of the reasons why there is a need for a common language to express a security policy. Because of the wide support from all of the main players in the computer industry, XML is the language of choice.

The following are the basic requirements of a policy language according to the Organization for the Advancement of Structure Information Standards (OASIS) access control XACML 2.0 core specification [XACML Core].   Figure 1 was taken from the OASIS XACML 2.0 core specification document.

The basic requirements of a policy language for expressing information system security policy are:

- To provide a method for combining individual ***rules*** and ***policies*** into a single ***policy set*** that applies to a particular ***decision request***.

- To provide a method for flexible definition of the procedure by which ***rules*** and ***policies*** are combined.

- To provide a method for dealing with multiple ***subjects*** acting in different capacities.

- To provide a method for basing an ***authorization decision*** on ***attributes*** of the ***subject*** and ***resource***.

- To provide a method for dealing with multi-valued ***attributes***.

- To provide a method for basing an ***authorization decision*** on the contents of an information ***resource***.

- To provide a set of logical and mathematical operators on ***attributes*** of the ***subject***, ***resource*** and ***environment***.

- To provide a method for handling a distributed set of ***policy*** components, while abstracting the method for locating, retrieving and authenticating the ***policy*** components.

- To provide a method for rapidly identifying the ***policy*** that applies to a given action, based upon the values of ***attributes*** of the ***subjects, resource*** and ***action***.

- To provide an abstraction-layer that insulates the policy-writer from the details of the application environment.

- To provide a method for specifying a set of actions that must be performed in conjunction with policy enforcement.

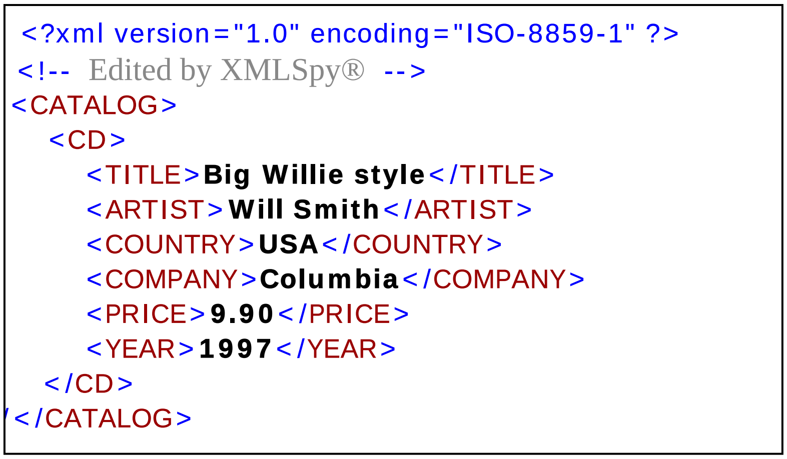
Figure 1: Basic Requirements of a Policy Language

These requirements in conjunction with the common structure in the XML language are what make XACML. Before providing detailed information on XACML, I will briefly discuss some background information which may be beneficial in understanding my thesis.

2.1 Extensible Markup Language (XML)

XML is a standard adopted by the World Wide Web Consortium (W3C). XML in syntax is very similar to the Hyper Text Markup Language (HTML) in that both are products of the Standard Generalized Markup Language (SGML) International Organization for Standardization (ISO) standard. However, XML is a stricter standard which allows users to write their own document structure. Because of the well-formed nature and flexibility of XML, it has become a major player in the development and configuration of computer software.

Essentially, XML is a set of guidelines that allows the user to structure its data. It uses tags and attributes to delimit and describe the data within the document. Tags are the hierarchical words enclosed in angle brackets "< >". I use the term hierarchical because an XML document is a hierarchy of the data it represents. Figure 2 is a basic example of an XML document. This document represents a Compact Disk (CD) catalog. It states that the root element <CATALOG> contains a child element <CD> which contains the description associated with a CD.



```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!--  Edited by XMLSpy®  -->
<CATALOG>
  <CD>
    <TITLE>Big Willie style</TITLE>
    <ARTIST>Will Smith</ARTIST>
    <COUNTRY>USA</COUNTRY>
    <COMPANY>Columbia</COMPANY>
    <PRICE>9.90</PRICE>
    <YEAR>1997</YEAR>
  </CD>
</CATALOG>
```

Figure 2: XML Document Example

2.2 Document Object Model (DOM)

DOM, as defined by the W3C, is a platform and language neutral interface that will allow programs and scripts to dynamically access and update the content, structure, and style of documents [DOM]. Theoretically, since the DOM specification is language neutral, it allows users to easily work with DOM when applying DOM practices with different programming languages.

Most common tasks when using DOM is to traverse, search, and retrieve information from a document. This can be accomplished in Java by first getting a new instance of the document factory method and calling its parse method. Figure 3 is an example of how to retrieve the root element (<CATALOG>) from the displayed XML document in Figure 2.

```
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
DocumentBuilder builder = factory.newDocumentBuilder();

File file = new File(fileName);
Document xacmlDoc = builder.parse(file);
Element root = xacmlDoc.getDocumentElement();
System.out.println(root.getTagName());
```

Figure 3: DOM sample Java code

2.3 XML Path Language (XPath)

XPath, as defined by the W3C, is a language for addressing parts of an XML document [XPath]. XPath uses an expression to locate information in an XML document. An expression is a line of source code that returns a value. The syntax of an XPath expression is to separate the location of each element with a slash "/". To use XPath in Java, the programmer must first obtain an XPath factory instance and then a path from its factory method. Figure 4 is an example of how to retrieve the <TITLE> nodes from the above provided XML document.

```
XPathFactory xpFactory = XPathFactory.newInstance();
XPath path = xpFactory.newXPath();

 NodeList nodeList = (NodeList) path.evaluate("/CATALOG/CD/TITLE", doc,
XPathConstants.NODESET);
```

Figure 4: XPath sample Java code

2.4 XACML

The acronym XACML stands for eXtensible Access Control Markup Language. XACML is a declarative access control language based on XML. The current OASIS approved standard for the XACML specification is Version 2.0. The XACML specification version 3.0 is expected soon.

XACML describes a policy and an access control decision request/response language. The policy language defines general access control policies. The request language is based in the form of a query asking if access to a particular resource is allowed. The response language is an answer to the request granting or denying permission.

A policy may be composed of a single policy and a number of rules or multiple policies and multiple rules. These policies and rules are what determine a request decision. To allow a hierarchical model in which policies can be described, XACML defines the following top-level elements: <Rule>, <Policy>, and <PolicySet>. The <Rule> element is basically a Boolean expression with an “Effect” attribute that it is either “Permit” or “Deny”. This element can exist in isolation but only for the purpose of re-use within multiple policies. The <Policy> element contains either a <Rule> or a set of rule elements in addition to a rule-combining algorithm. The rule-combining algorithm is used by the policy to provide an authorization decision based on its rules. The <PolicySet> element can contain other <PolicySet> elements and/or a <Policy> or a set of policies. The <PolicySet> element also contains an algorithm which combines the results of the policy’s evaluation. Further, the <PolicySet> element is used to enclose multiple policies within its hierarchy.

2.4.1 Combining Algorithms

XACML provides two combining algorithms for determining an authorization decision. The <Policy> element combining algorithm is identified by the attribute RuleCombiningAlgId. The <PolicySet> element combining algorithm is identified by the attribute PolicyCombiningAlgId. The rule-combining algorithm is used to evaluate set of rules within a policy. The policy-combining algorithm is used to evaluate a set of policies. XACML defines the following methods to be used with the combining algorithms:

- Deny-overrides (Ordered and Unordered)
- Permit-overrides (Ordered and Unordered)
- First-applicable
- Only-one-applicable

In the deny-overrides algorithm, the combined result is set to "Deny" if a single <Rule> or <Policy> evaluates to deny within the applicable policy. In the permit-overrides algorithm, the combine result is set to "Permit" if a single <Rule> or <Policy> evaluates to permit within the applicable policy. If deny-overrides or permit-overrides are defined as Ordered, then the rules must be evaluated in the order listed in the policy. The first-applicable algorithm evaluates the result of the first <Rule>, <Policy>, or <PolicySet> within the applicable policy. Lastly, the only-one-applicable algorithm ensures that only one <Policy> or <PolicySet> is applicable. This algorithm only applies to policies. If more than one policy applies to a particular target, the returned result is Indeterminate. A NotApplicable result means that no policies apply. If one policy or policy set applies, then the result is based on the combining algorithm for the single applicable policy or policy set.

2.4.2 Policy Distribution and Indexing

In an effort to allow policies to be easily managed, policies can be written as multiple independent policy components by different policy writers and can be enforced at their particular enforcement points. XACML does not provide a standard way of handling policies that are distributed; however, the Policy decision point (PDP) is still required to examine the <Target> element to determine if a policy is applicable to a decision request. Matching an applicable policy to a request is what is known as policy indexing. XACML supports the following approaches:

- Storing Policies in a database. This approach requires the PDP to query the database and retrieve the policies that are applicable to the request.
- Loading the PDP with all policies. In this approach, the PDP is loaded with all policies and is responsible for evaluating all of the policy <Target> elements to determine which policies are applicable to the request.

2.4.3 XACML Data Flow Diagram

Figure 5 identifies the major actors in XACML. Figure 5 was taken from the OASIS XACML 2.0 core specification document. As the diagram represents, the request for a resource goes through a series of actions between the PDP and the Context handler before a response is sent back to the Policy Enforcement Point (PEP) with a status of “Permit” or “Deny”.

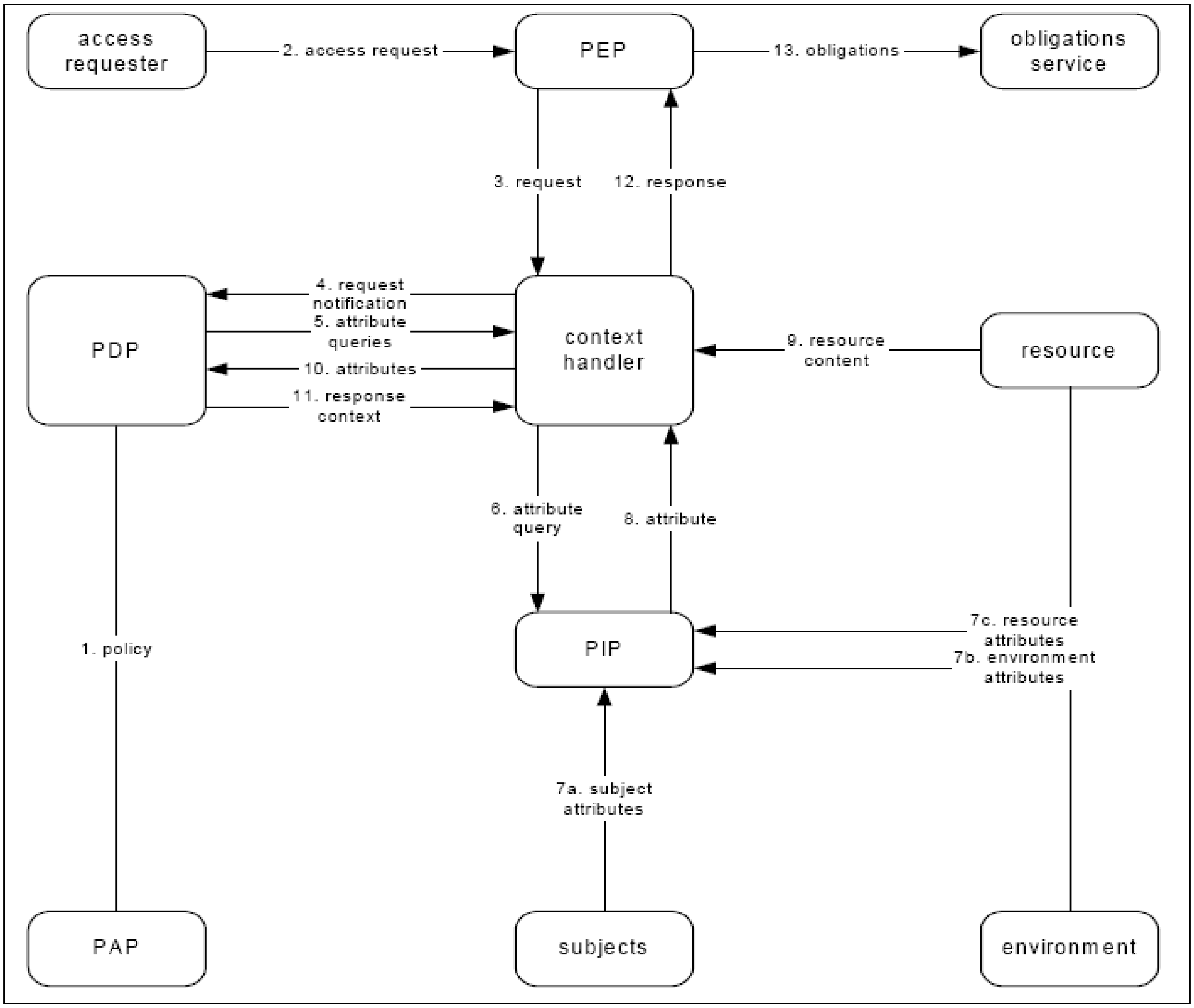


Figure 5: XACML Data Flow Diagram

2.4.4 XACML Context

The XACML context is an abstraction layer that allows the mapping of attributes from different sources to XACML attributes. The context is basically a conversion mechanism between native and XACML format. Figure 6 is a representation of the XACML context. Figure 6 was taken from the OASIS XACML 2.0 core specification document.

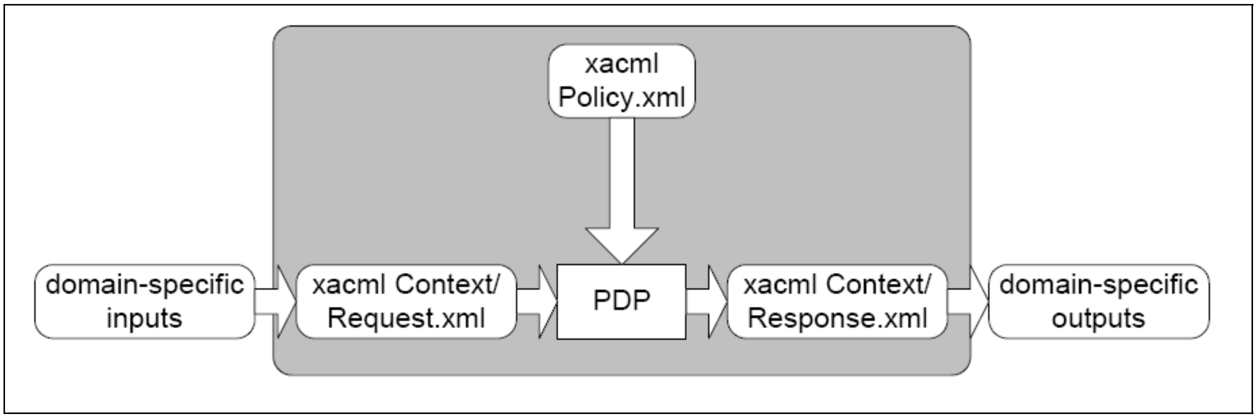


Figure 6: XACML Context

2.4.5 Policy Language Model

Policy set, Policy, and Rule are the main components of the Policy language model. Figure 7 is the representation of the Policy language model.

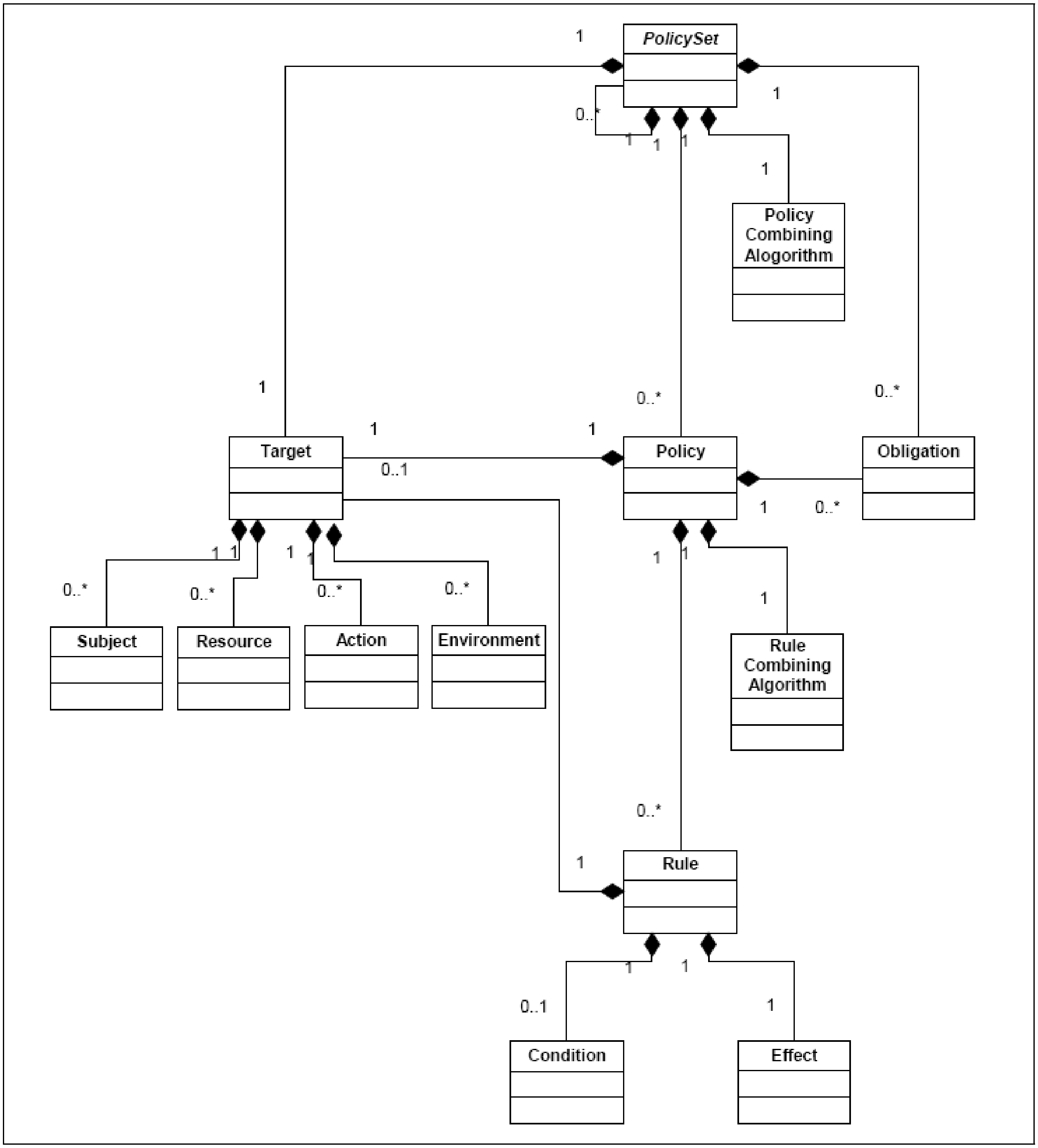


Figure 7: Policy Language Model

2.4.5.1 Rule

A <Rule> is contained within a policy and its main components are a target, an effect, and a condition. The <Target> element contains resources, subjects, actions, and environment which apply to a particular rule. These target attributes are what the PDP uses to match a request context. If the <Target> element of a rule is empty then the target of the rule is the parent <Policy> target. The effect is an attribute value of the <Rule> element which evaluates the rule and can only contain the values of “Permit” or “Deny”. The <Condition> is an optional element which if used must evaluate to true for the rule to be applicable.

2.4.5.2 Policy

The main components of a <Policy> are a target, a rule-combining algorithm-identifier, a set of rules, and obligations. The <Target> element of a policy can be declared by the policy writer or it can be calculated from all of the <Target> elements within a policy. One method of target calculation is using the “outer component”, that is, targets from policy sets or policies as a union of all of their targets. The other method is calculating the “outer components” as the intersection of all of their targets. If any rule has the same <Target> element as the policy, then the target for that rule may be omitted since rules inherit the target of its policy.

The rule-combining algorithm represents the rule by which a decision is made to a particular request. <Obligations> are sets of instructions that must be performed by the PEP concurrently with the authorization request. For example, if the effect of a rule is to allow access to a server, an obligation could be to log the user in the server log as a successful login. Conversely, if access was denied another obligation could be to log that user as a failed login.

2.4.5.3 Policy Set

The main components of a <PolicySet> are a target, a policy-combining algorithm-identifier, a set of policies, and obligations. The policy-combining algorithm is used to determine how policies within a policy set will be combined when making a decision to a request. Figure 8 is an example of a XACML document as found in the OASIS XACML 2.0 core specification document.

```
<?xml version="1.0" encoding="UTF-8"?>
<Policy xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="urn:oasis:names:tc:xacml:2.0:policy:schema:os
        http://docs.oasis-open.org/xacml/access_control-xacml-2.0-policy-schema-os.xsd"
        PolicyId="urn:oasis:names:tc:example:SimplePolicy1"
        RuleCombiningAlgId="identifier:rule-combining-algorithm:deny-overrides">

  <Description>Medi Corp access control policy</Description>
  <Target/>
  <Rule RuleId= "urn:oasis:names:tc:xacml:2.0:example:SimpleRule1" Effect="Permit">
    <Description>
         Any subject with an e-mail name in the med.example.com domain
         can perform any action on any resource.
    </Description>
    <Target>
      <Subjects>
        <Subject>
          <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:rfc822Name-match">
            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
                med.example.com</AttributeValue>
            <SubjectAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
               DataType="urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name"/>
          </SubjectMatch>
        </Subject>
      </Subjects>
    </Target>
  </Rule>
</Policy>
```

Figure 8: XACML Documents Example

Chapter 3 Problem Statement

The inception of XACML has provided agencies with a common language which can be used to safeguard and better manage access to their resources. This approach, however, has its limitations when it comes to managing multiple policies for a single resource or a variety of resources. XACML is based on XML which has been proven to be an industry favored mark-up language.   Figure 9 is a portion of an example of a time-range policy found in the Sun's XACML Implementation [Sun XACML] sample policy folder. The Sun's XACML Implementation is an open source implementation of the OASIS XACML standard written in Java. This implementation provides complete support of the XACML mandatory standards and some support to optional features.

```
<?xml version="1.0" encoding="UTF-8"?>
<Policy xmlns="urn:oasis:names:tc:xacml:1.0:policy"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        PolicyId="TimeRangePolicy"
        RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-
algorithm:ordered-permit-overrides">

  <Description>
    Between 9am and 5pm local time, allow anyone to open the main
    door. All other times, only allow authorized people (ie, those with
    an email address @users.example.com). Deny in all other cases.
  </Description>

  <Target>
    <Subjects>
      <AnySubject/>
    </Subjects>
    <Resources>
      <Resource>
        <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
             main-door</AttributeValue>
          <ResourceAttributeDesignator
             DataType="http://www.w3.org/2001/XMLSchema#string"
             AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"/>
        </ResourceMatch>
      </Resource>
    </Resources>
```

Figure 9: Sun's XACML Implementation TimeRange Policy

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

  <Rule RuleId="EveryoneDuringBusinessHours" Effect="Permit">
    <Condition
FunctionId="http://research.sun.com/projects/xacml/names/function#time-in-range">
      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:time-one-and-only">
        <EnvironmentAttributeDesignator
          DataType="http://www.w3.org/2001/XMLSchema#time"
          AttributeId="urn:oasis:names:tc:xacml:1.0:environment:current-time"/>
      </Apply>
      <AttributeValue
        DataType="http://www.w3.org/2001/XMLSchema#time">09:00:00</AttributeValue>
      <AttributeValue
        DataType="http://www.w3.org/2001/XMLSchema#time">17:00:00</AttributeValue>
    </Condition>
  </Rule>

  <Rule RuleId="EmployeesAlways" Effect="Permit">
    <Target>
      <Subjects>
        <Subject>
          <SubjectMatch
              MatchId="urn:oasis:names:tc:xacml:1.0:function:rfc822Name-match">
            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
              users.example.com</AttributeValue>
            <SubjectAttributeDesignator
             DataType="urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name"
             AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"/>
          </SubjectMatch>
        </Subject>
      </Subjects>
      <Resources>
        <AnyResource/>
      </Resources>
      <Actions>
        <AnyAction/>
      </Actions>
    </Target>
  </Rule>

  <Rule RuleId="DenyAllOthers" Effect="Deny"/>

</Policy>
```

Figure 10: Sun's XACML Implementation TimeRange Policy (Continue)

This policy states that anyone has access to open the main door between the hours of 9:00am and 5:00pm. Only authorized personnel have access at other times, and in any other circumstance, access to open the main door is denied. As Figures 9 and 10 illustrate, it is not straight forward for a human to carryout code inspection for this document element tags by element tags especially across multiple documents. Furthermore, this is just a very basic and simple policy document. A policy document has multiple pages, it can also be contained within a policy set or multiple policy sets. Doing code inspection for multiple policies can make it extremely difficult to catch possible conflicts between multiple policies. Figures 11 and 12 are modified versions of the Sun’s XACML Implementation time-range policy. This policy states that anyone has access to open the main door between the hours of 7:00am and 3:00pm.

```
<?xml version="1.0" encoding="UTF-8"?>
<Policy xmlns="urn:oasis:names:tc:xacml:1.0:policy"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        PolicyId="TimeRangePolicy2"
        PolicyCombiningAlgId="urn:oasis:names:tc:xacml:1.0:policy-combining-
        algorithm:deny-overrides">

  <Description>
    Between 9am and 5pm local time, allow anyone to open the main
    door. All other times, only allow authorized people (ie, those with
    an email address @users.example.com). Deny in all other cases.
  </Description>

  <Target>
    <Subjects>
      <AnySubject/>
    </Subjects>
    <Resources>
      <Resource>
        <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
             main-door</AttributeValue>
          <ResourceAttributeDesignator
            DataType="http://www.w3.org/2001/XMLSchema#string"
            AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"/>
```

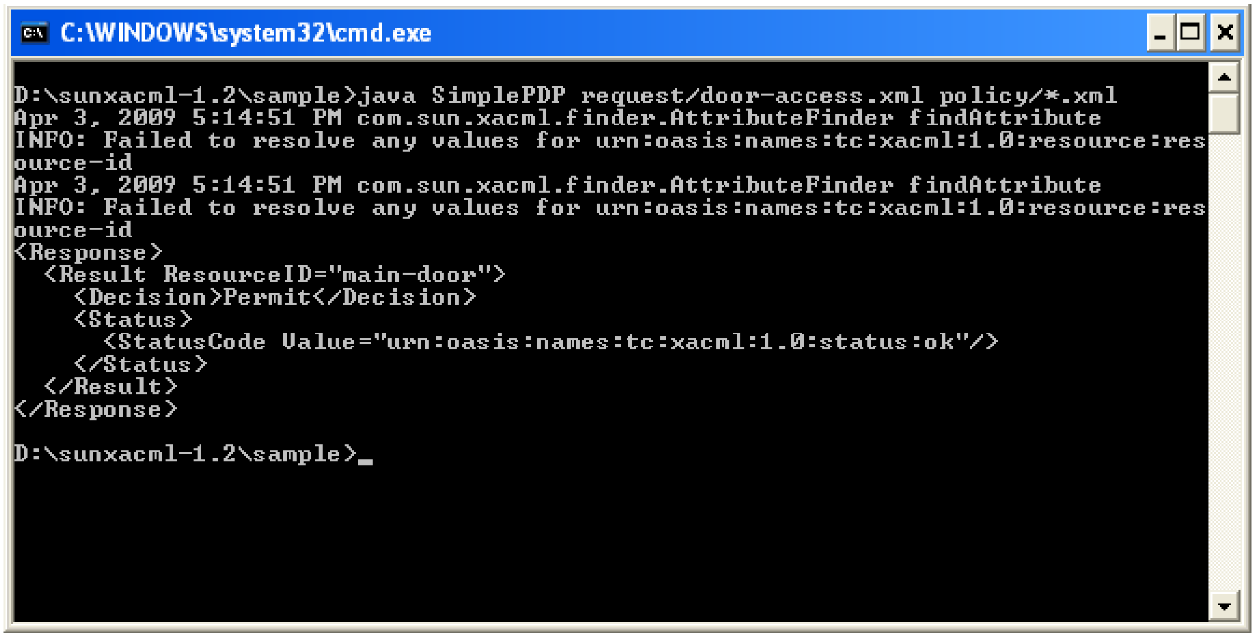
Figure 11: Sun's XACML Implementation TimeRange Policy2

```
        </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">open</AttributeValue>
          <ActionAttributeDesignator
              DataType="http://www.w3.org/2001/XMLSchema#string"
              AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"/>
        </ActionMatch>
      </Action>
    </Actions>
  </Target>

  <Rule RuleId="EveryoneDuringBusinessHours" Effect="Permit">
    <Condition
       FunctionId="http://research.sun.com/projects/xacml/names/function#time-in-range">
      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:time-one-and-only">
        <EnvironmentAttributeDesignator
           DataType="http://www.w3.org/2001/XMLSchema#time"
           AttributeId="urn:oasis:names:tc:xacml:1.0:environment:current-time"/>
      </Apply>
      <AttributeValue
           DataType="http://www.w3.org/2001/XMLSchema#time">07:00:00</AttributeValue>
      <AttributeValue
           DataType="http://www.w3.org/2001/XMLSchema#time">15:00:00</AttributeValue>
    </Condition>
  </Rule>

  <Rule RuleId="EmployeesAlways" Effect="Permit">
    <Target>
      <Subjects>
        <Subject>
          <SubjectMatch
              MatchId="urn:oasis:names:tc:xacml:1.0:function:rfc822Name-match">
            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
              users.example.com</AttributeValue>
            <SubjectAttributeDesignator
              DataType="urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name"
              AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"/>
          </SubjectMatch>
        </Subject>
      </Subjects>
      <Resources>
        <AnyResource/>
      </Resources>
      <Actions>
        <AnyAction/>
      </Actions>
    </Target>
  </Rule>

  <Rule RuleId="DenyAllOthers" Effect="Deny"/>

</Policy>
```

Figure 12: Sun's XACML Implementation TimeRange Policy2 (Continue)

If policies, as depicted in Figure 9 and Figure 11, existed in a repository for a particular resource; it will conflict and potentially can be an issue when determining if and when the subject has access to the main door. Furthermore, maintaining policy integrity is a very crucial aspect of being able to recognize if a duplicated policy is introduced with modified rules to gain access to resources. For a policy administrator, the task of manually scanning through each document for potential conflicts can be overwhelming if not impossible.

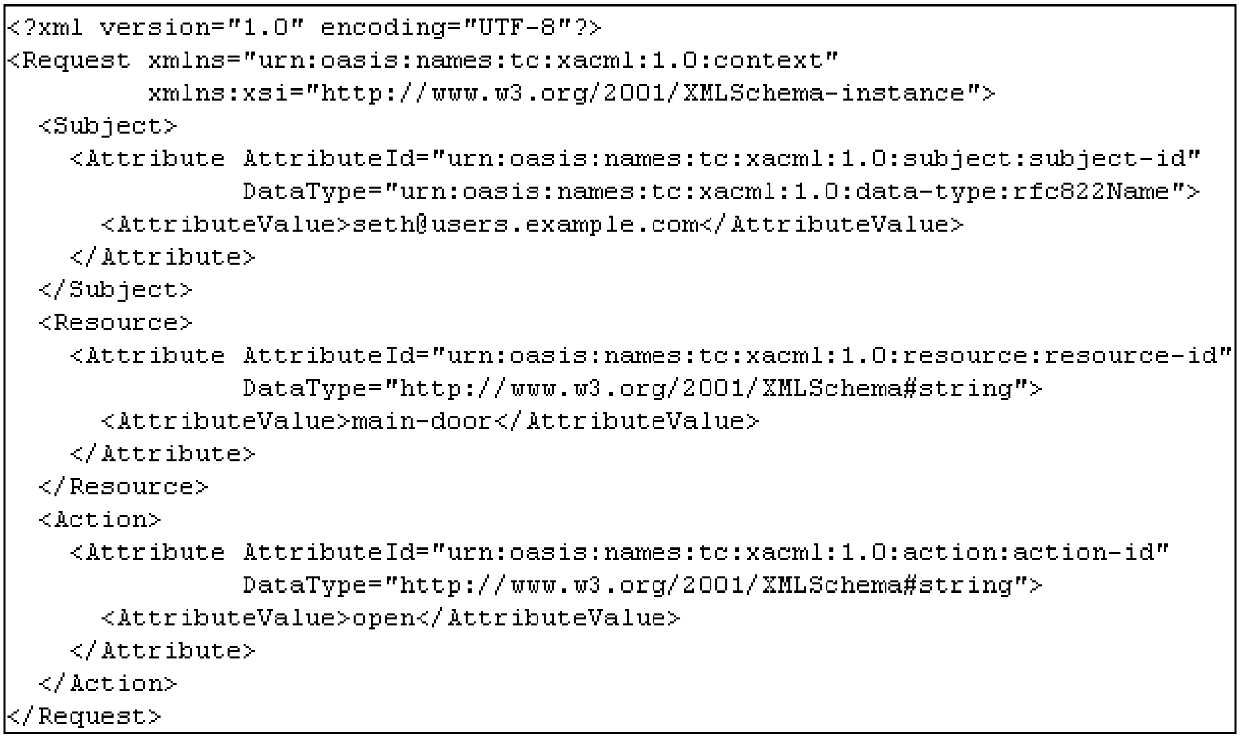
Figure 13 displays a sample demo application called SimplePDP from the Sun's XACML implementation. The request for this particular example is the door-access.xml and the SimplePDP was instructed to scan through all XML documents located in the policy directory.



```
D:\sunxacml-1.2\sample>java SimplePDP request/door-access.xml policy/*.xml
Apr 3, 2009 5:14:51 PM com.sun.xacml.finder.AttributeFinder findAttribute
INFO: Failed to resolve any values for urn:oasis:names:tc:xacml:1.0:resource:res
ource-id
Apr 3, 2009 5:14:51 PM com.sun.xacml.finder.AttributeFinder findAttribute
INFO: Failed to resolve any values for urn:oasis:names:tc:xacml:1.0:resource:res
ource-id
<Response>
  <Result ResourceID="main-door">
    <Decision>Permit</Decision>
    <Status>
      <StatusCode Value="urn:oasis:names:tc:xacml:1.0:status:ok"/>
    </Status>
  </Result>
</Response>

D:\sunxacml-1.2\sample>_
```

Figure 13: Sun's XACML Implementation SimplePDP

This command based application is included as part of the Sun's XACML implementation so the user can run policies, evaluate his/her request, and gain a better understanding of XACML. This particular example is for the above mentioned policy to access the main door. The response for this example is also displayed in Figure 13. The "INFO:" label displayed in the SimplePDP (Figure 13) command prompt window, is simply telling the user that some policies did not match the request. The "INFO:" label messages are part of a login mechanism and can be removed at the user's discretion. The response provided back to the user in this example states that access is allowed to the main door. Figure 14 shows the request that was sent to the SimplePDP application.

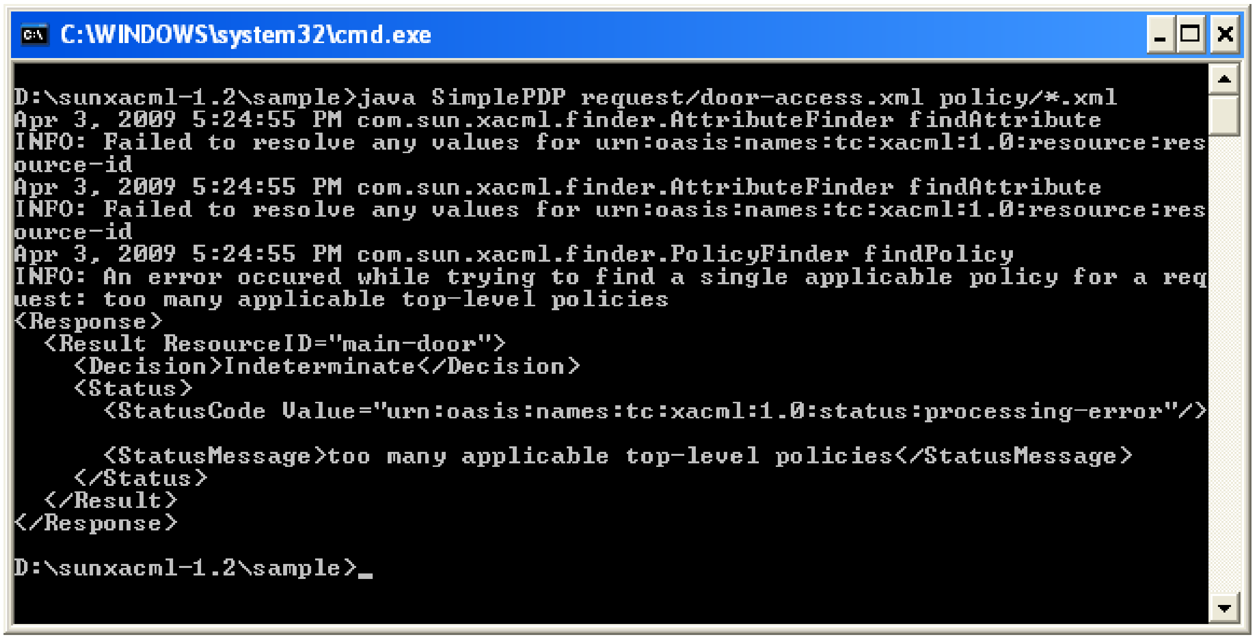


```
<?xml version="1.0" encoding="UTF-8"?>
<Request xmlns="urn:oasis:names:tc:xacml:1.0:context"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <Subject>
    <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
               DataType="urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name">
      <AttributeValue>seth@users.example.com</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>main-door</AttributeValue>
    </Attribute>
  </Resource>
  <Action>
    <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
               DataType="http://www.w3.org/2001/XMLSchema#string">
      <AttributeValue>open</AttributeValue>
    </Attribute>
  </Action>
</Request>
```

Figure 14: XACML Request Example

Notice that the requested subject matches the subject of RuleId=”EmployeesAlways” in the time-range policy document. This is a correct interpretation of the request where the matching policy was found and the requested level of access was granted.

Figure 15 displays the same request; however, this time a second policy that matches the same target was introduced. As stated, this is a copy of the same policy with a different policy id and a slightly different time-range rule. The response for this request is coded as Indeterminate because more than one policy applies.



```
D:\sunxacml-1.2\sample>java SimplePDP request/door-access.xml policy/*.xml
Apr 3, 2009 5:24:55 PM com.sun.xacml.finder.AttributeFinder findAttribute
INFO: Failed to resolve any values for urn:oasis:names:tc:xacml:1.0:resource:res
ource-id
Apr 3, 2009 5:24:55 PM com.sun.xacml.finder.AttributeFinder findAttribute
INFO: Failed to resolve any values for urn:oasis:names:tc:xacml:1.0:resource:res
ource-id
Apr 3, 2009 5:24:55 PM com.sun.xacml.finder.PolicyFinder findPolicy
INFO: An error occured while trying to find a single applicable policy for a req
uest: too many applicable top-level policies
<Response>
  <Result ResourceID="main-door">
    <Decision>Indeterminate</Decision>
    <Status>
      <StatusCode Value="urn:oasis:names:tc:xacml:1.0:status:processing-error"/>

      <StatusMessage>too many applicable top-level policies</StatusMessage>
    </Status>
  </Result>
</Response>

D:\sunxacml-1.2\sample>_
```

Figure 15: Sun’s XACML SimplePDP Indeterminate Response

Because of the afore mentioned issues, it is important that the policy administrator is able to visualize potentially conflicting targets. The basis of my research is to provide a way for the policy administrator to quickly visualize a set of policies for potential conflicts. A visualization program will not only make it easier for the administrator to scan policy documents, it will also reduce errors that potentially lead to security leaks and inconsistencies with providing required access to legitimate users.

Another issue worth mentioning is the possibility of inconsistencies within the same policy. This portion is not implemented in the prototype program; however, it should be an area of focus for further research. Furthermore, this should be an area where policy administrators pay close attention to when writing policies. In the following examples, the generated.xml files from the Sun’s XACML Implementation sample and request folders were edited to introduce potential inconsistencies within the policy. Figure 16 illustrates the policy id and the combining algorithm for this policy. Figures 17, 18 and 19 depict the rules for policy GeneratedPolicy4. Figure 20 is the request against this policy. The policy applies to any accounts at users.example.com accessing server.example.com. Accounts that belong to users.example.com can read from server.example.com. The account will@users.example.com can read and write to server.example.com. As depicted in Figure 16, this policy implements the permit-overrides rule-combining-algorithm.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Policy xmlns="urn:oasis:names:tc:xacml:1.0:policy"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 PolicyId="GeneratedPolicy4"
 RuleCombiningAlgId=
 "urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:permit-overrides">

 <Description>
  This policy applies to any accounts at users.example.com accessing
  server.example.com. Accounts that belong to users.example.com can read
  server.example.com. Account will@users.example.com can read or write
  to server.example.com.
 </Description>
 <Target>
    <Subjects>
      <AnySubject/>
    </Subjects>
    <Resources>
      <AnyResource/>
    </Resources>
    <Actions>
      <AnyAction/>
    </Actions>
 </Target>
```

Figure 16: rule-combining-algorithm:permit-overrides

```
<Rule RuleId="UsersRule" Effect="Permit">
  <Target>
    <Subjects>
      <Subject>
        <SubjectMatch
         MatchId="urn:oasis:names:tc:xacml:1.0:function:rfc822Name-match">
          <AttributeValue
           DataType="http://www.w3.org/2001/XMLSchema#string">
           users.example.com</AttributeValue>
          <SubjectAttributeDesignator
           DataType="urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name"
           AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"/>
        </SubjectMatch>
      </Subject>
    </Subjects>
    <Resources>
      <Resource>
        <ResourceMatch
         MatchId="urn:oasis:names:tc:xacml:1.0:function:anyURI-equal">
          <AttributeValue
           DataType="http://www.w3.org/2001/XMLSchema#anyURI">
           http://server.example.com/</AttributeValue>
           <ResourceAttributeDesignator
            DataType="http://www.w3.org/2001/XMLSchema#anyURI"
            AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"/>
        </ResourceMatch>
      </Resource>
    </Resources>
    <Actions>
      <Action>
        <ActionMatch
          MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
        <AttributeValue
          DataType="http://www.w3.org/2001/XMLSchema#string">
          read</AttributeValue>
        <ActionAttributeDesignator
          DataType="http://www.w3.org/2001/XMLSchema#string"
          AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"/>
        </ActionMatch>
      </Action>
    </Actions>
  </Target>
</Rule>
```

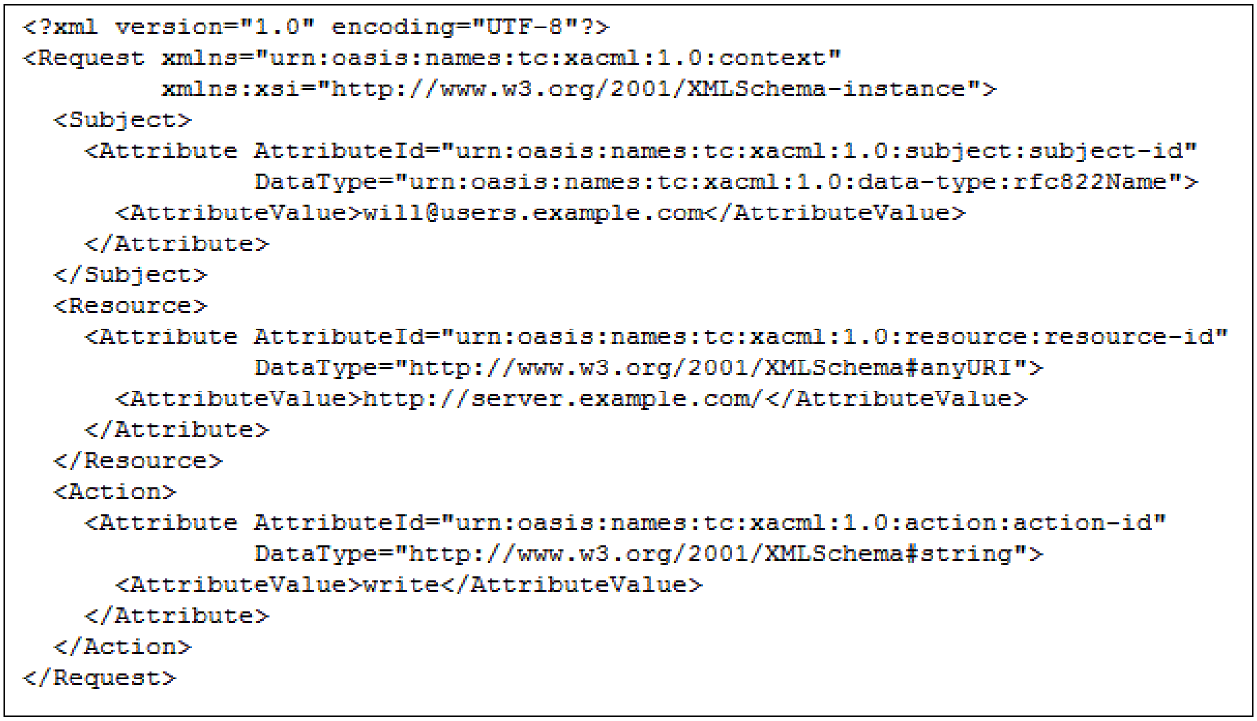
Figure 17: GeneratedPolicy4 RuleId = “UsersRule”

```
<Rule RuleId="WritersRule" Effect="Permit">
  <Target>
    <Subjects>
      <Subject>
        <SubjectMatch
         MatchId="urn:oasis:names:tc:xacml:1.0:function:rfc822Name-match">
          <AttributeValue
           DataType="http://www.w3.org/2001/XMLSchema#string">
           will@users.example.com</AttributeValue>
          <SubjectAttributeDesignator
           DataType="urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name"
           AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"/>
        </SubjectMatch>
      </Subject>
    </Subjects>
    <Resources>
      <Resource>
        <ResourceMatch
         MatchId="urn:oasis:names:tc:xacml:1.0:function:anyURI-equal">
          <AttributeValue
           DataType="http://www.w3.org/2001/XMLSchema#anyURI">
           http://server.example.com/</AttributeValue>
           <ResourceAttributeDesignator
            DataType="http://www.w3.org/2001/XMLSchema#anyURI"
            AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"/>
        </ResourceMatch>
      </Resource>
    </Resources>
    <Actions>
      <Action>
        <ActionMatch
          MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
        <AttributeValue
          DataType="http://www.w3.org/2001/XMLSchema#string">
          write</AttributeValue>
        <ActionAttributeDesignator
          DataType="http://www.w3.org/2001/XMLSchema#string"
          AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"/>
        </ActionMatch>
      </Action>
    </Actions>
  </Target>
</Rule>
```

Figure 18: GeneratedPolicy4 RuleId = "WritersRule"

```
<Rule RuleId="UsersWriterRule" Effect="Deny">
  <Target>
    <Subjects>
      <Subject>
        <SubjectMatch
         MatchId="urn:oasis:names:tc:xacml:1.0:function:rfc822Name-match">
          <AttributeValue
           DataType="http://www.w3.org/2001/XMLSchema#string">
           users.example.com</AttributeValue>
          <SubjectAttributeDesignator
           DataType="urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name"
           AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"/>
        </SubjectMatch>
      </Subject>
    </Subjects>
    <Resources>
      <Resource>
        <ResourceMatch
         MatchId="urn:oasis:names:tc:xacml:1.0:function:anyURI-equal">
          <AttributeValue
           DataType="http://www.w3.org/2001/XMLSchema#anyURI">
           http://server.example.com/</AttributeValue>
           <ResourceAttributeDesignator
            DataType="http://www.w3.org/2001/XMLSchema#anyURI"
            AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"/>
        </ResourceMatch>
      </Resource>
    </Resources>
    <Actions>
      <Action>
        <ActionMatch
          MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
        <AttributeValue
          DataType="http://www.w3.org/2001/XMLSchema#string">
          write</AttributeValue>
        <ActionAttributeDesignator
          DataType="http://www.w3.org/2001/XMLSchema#string"
          AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"/>
        </ActionMatch>
      </Action>
    </Actions>
  </Target>
</Rule>
```

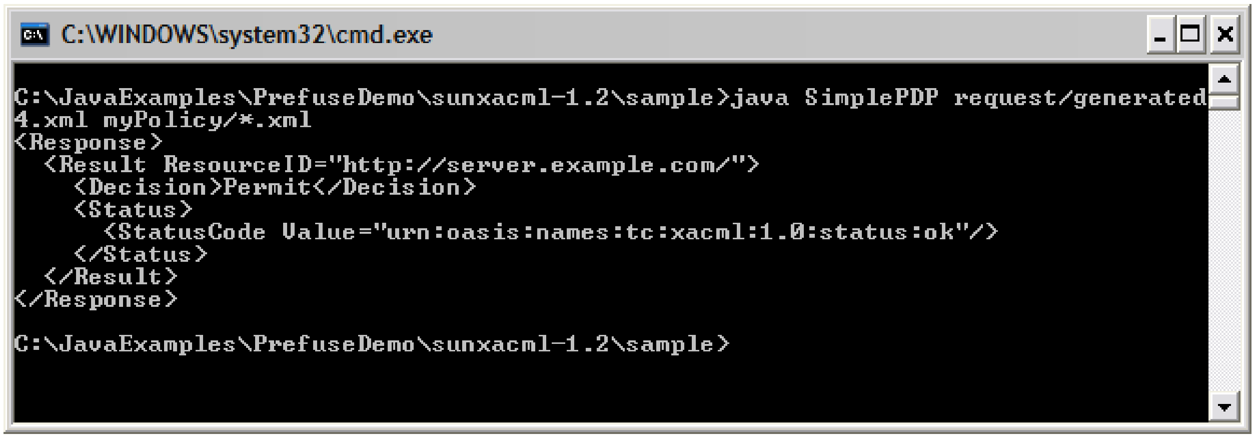
Figure 19: GeneratedPolicy4 RuleId = "UsersWriteRule"



```
<?xml version="1.0" encoding="UTF-8"?>
<Request xmlns="urn:oasis:names:tc:xacml:1.0:context"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <Subject>
    <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
               DataType="urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name">
      <AttributeValue>will@users.example.com</AttributeValue>
    </Attribute>
  </Subject>
  <Resource>
    <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
               DataType="http://www.w3.org/2001/XMLSchema#anyURI">
      <AttributeValue>http://server.example.com/</AttributeValue>
    </Attribute>
  </Resource>
  <Action>
    <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
               DataType="http://www.w3.org/2001/XMLSchema#string">
      <AttributeValue>write</AttributeValue>
    </Attribute>
  </Action>
</Request>
```

Figure 20: will@users.example.com Write Request

The request is asking to allow write access to the server http://server.example.com/ for will@users.example.com. Figure 21 displays the response from the SimplePDP command line application.



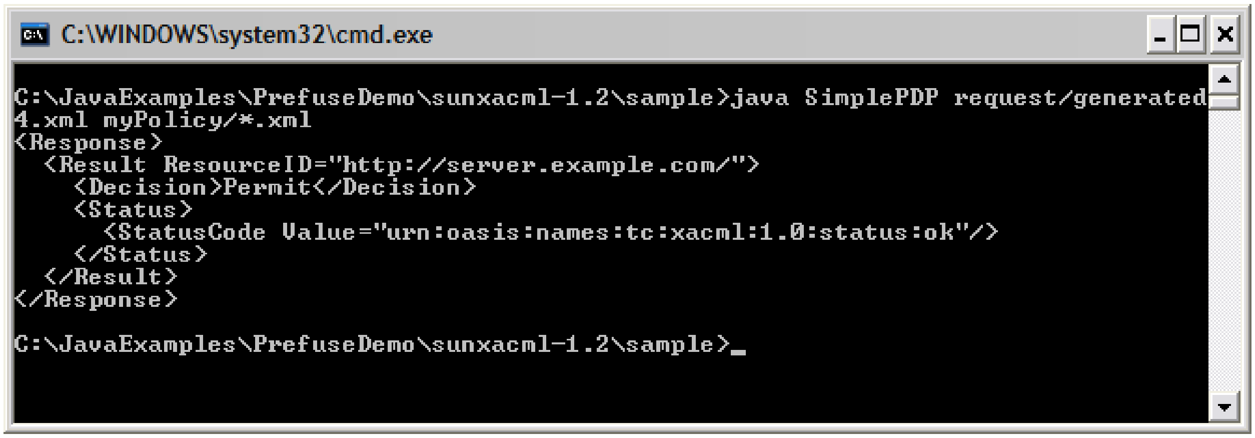
```
C:\JavaExamples\PrefuseDemo\sunxacml-1.2\sample>java SimplePDP request/generated
4.xml myPolicy/*.xml
<Response>
  <Result ResourceID="http://server.example.com/">
    <Decision>Permit</Decision>
    <Status>
      <StatusCode Value="urn:oasis:names:tc:xacml:1.0:status:ok"/>
    </Status>
  </Result>
</Response>

C:\JavaExamples\PrefuseDemo\sunxacml-1.2\sample>
```

Figure 21: Generated Request for permit-overrides

The PDP arrived at the “Permit” decision because of the three rules identified in the GeneratedPolicy4 XACML document. One of the rules applied, since the rule-combining algorithm used for the policy was permit-overrides. The rule-combining algorithm was changed to deny-override for the next test.   Figure 22 displays the new change. The request for this example was also changed to request read access to the resource. Figure 23 is the result returned from the SimplePDP command line program for the read request. Figure 24 is the result from the write request.

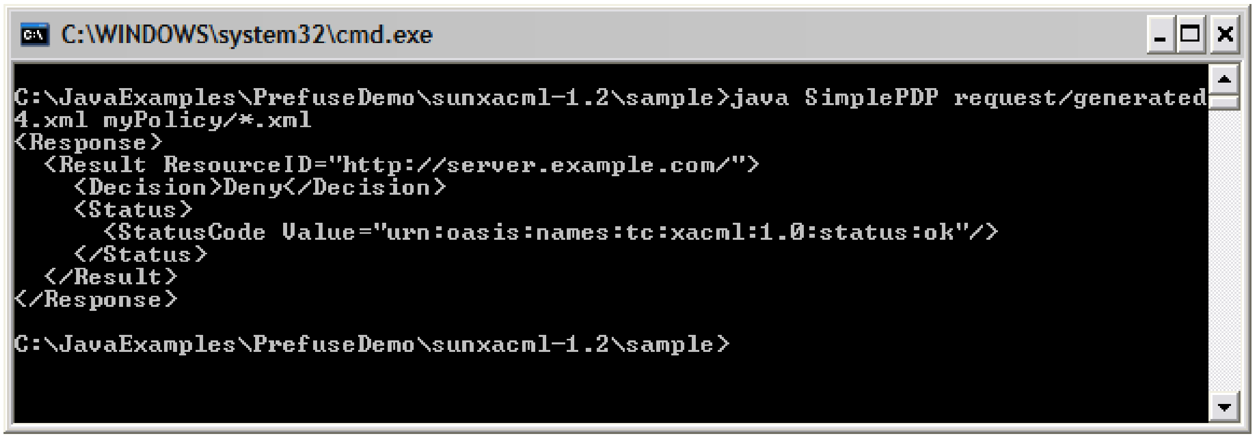
```
<?xml version="1.0" encoding="UTF-8"?>
<Policy xmlns="urn:oasis:names:tc:xacml:1.0:policy"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 PolicyId="GeneratedPolicy4"
 RuleCombiningAlgId=
 "urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-overrides">

 <Description>
  This policy applies to any accounts at users.example.com accessing
  server.example.com. Accounts that belong to users.example.com can read
  server.example.com. Account will@users.example.com can read or write
  to server.example.com.
 </Description>
 <Target>
    <Subjects>
      <AnySubject/>
    </Subjects>
    <Resources>
      <AnyResource/>
    </Resources>
    <Actions>
      <AnyAction/>
    </Actions>
 </Target>
```

Figure 22: rule-combining-algorithm:deny-overrides



```
C:\JavaExamples\PrefuseDemo\sunxacml-1.2\sample>java SimplePDP request/generated
4.xml myPolicy/*.xml
<Response>
  <Result ResourceID="http://server.example.com/">
    <Decision>Permit</Decision>
    <Status>
      <StatusCode Value="urn:oasis:names:tc:xacml:1.0:status:ok"/>
    </Status>
  </Result>
</Response>

C:\JavaExamples\PrefuseDemo\sunxacml-1.2\sample>_
```

Figure 23: Generated Response for deny-override Read Request



```
C:\JavaExamples\PrefuseDemo\sunxacml-1.2\sample>java SimplePDP request/generated
4.xml myPolicy/*.xml
<Response>
  <Result ResourceID="http://server.example.com/">
    <Decision>Deny</Decision>
    <Status>
      <StatusCode Value="urn:oasis:names:tc:xacml:1.0:status:ok"/>
    </Status>
  </Result>
</Response>

C:\JavaExamples\PrefuseDemo\sunxacml-1.2\sample>
```

Figure 24: Generated Response for deny-override Write Request

The response displayed in Figure 23 is as expected “Permit” since the subject belongs to the users group and such group is allowed read access. The response displayed in Figure 24 may not be as expected since there is a specific rule for will@users.example.com to allow write access to the resource. However, since will@users.example.com belongs to users.example.com the last rule applies which states that users who belong to users.example.com do not have write access to the resource. This is one of the possibly conflicting issues that can be found in XACML. In this example, there are three different rules within the policy for a subject in the users.example.com domain. Rule “UsersRule” should apply because will@users.example.com is part of the users group. Rule “WritersRule” should also apply because the rule is specific to will@users.example.com. Rule “UsersWriteRule” should also apply since will@users.example.com belongs to the users group.

As previously stated, inconsistencies within the same policy are not being addressed as part of this thesis; this will be one of my future works.

Chapter 4 Design and Prototype Implementation

In order to find a possible solution to some of the issues stated in Chapter 3, a prototype implementation named XACMLViz was created. This prototype implementation is based on the Java language. It uses Java Swing components for the Graphical User Interface (GUI) and the Document Object Model (DOM) XML parser and XPath to locate and extract significant data.

4.1 Design

The design is based on a simple concept of extracting data from the policy files, populating the data in class objects, and saving the data in a database table where it can be queried for potential conflicts. The database used for the prototype is the bundled Java DB based on Apache Derby. Figure 25 is the class diagram of the XACMLViz prototype program. Figures 26, 27, and 28 are the zoomed version of the classes in Figure 25.

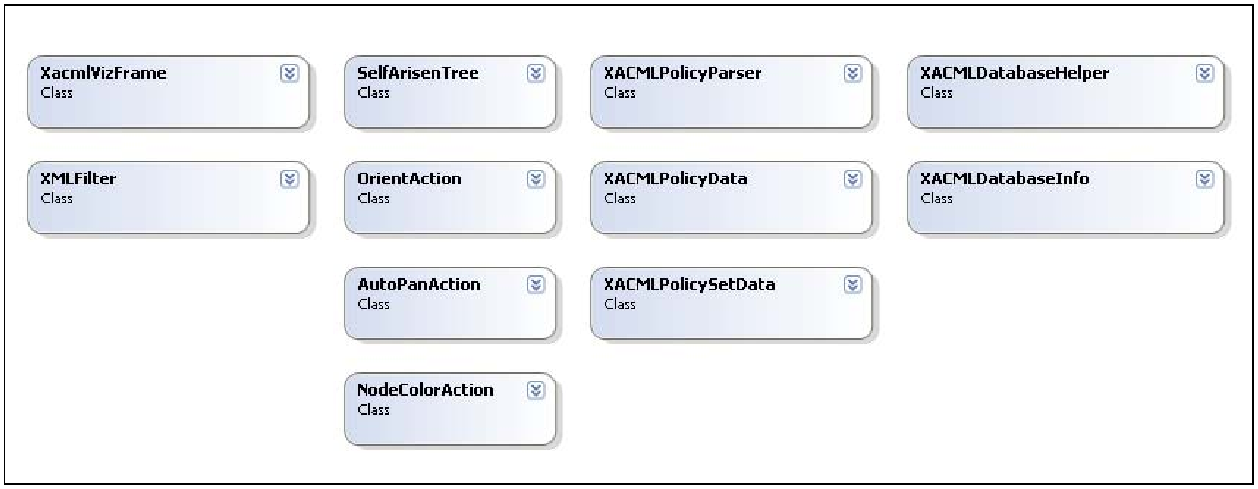


Figure 25: XACMLVizClass Diagram

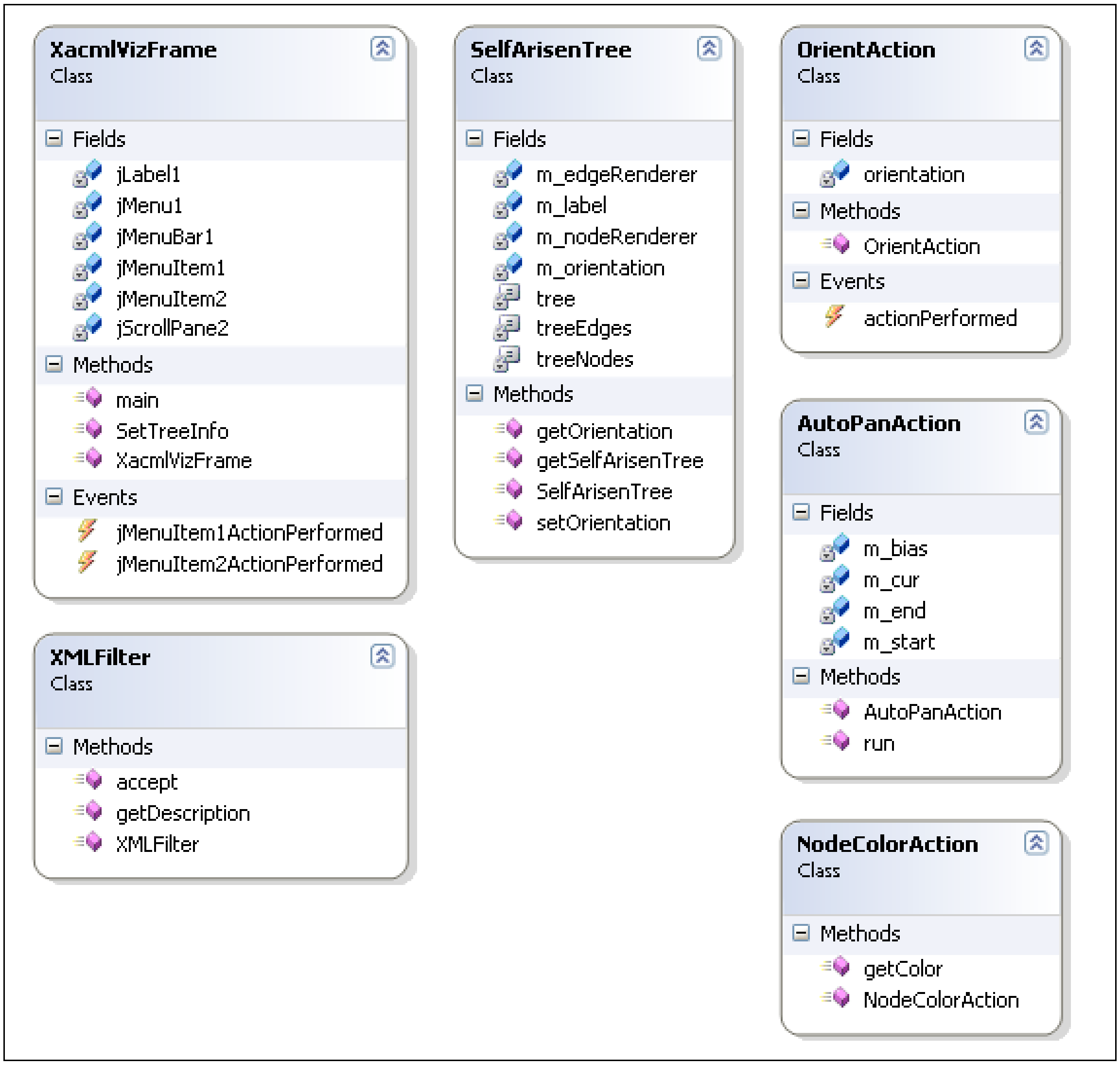
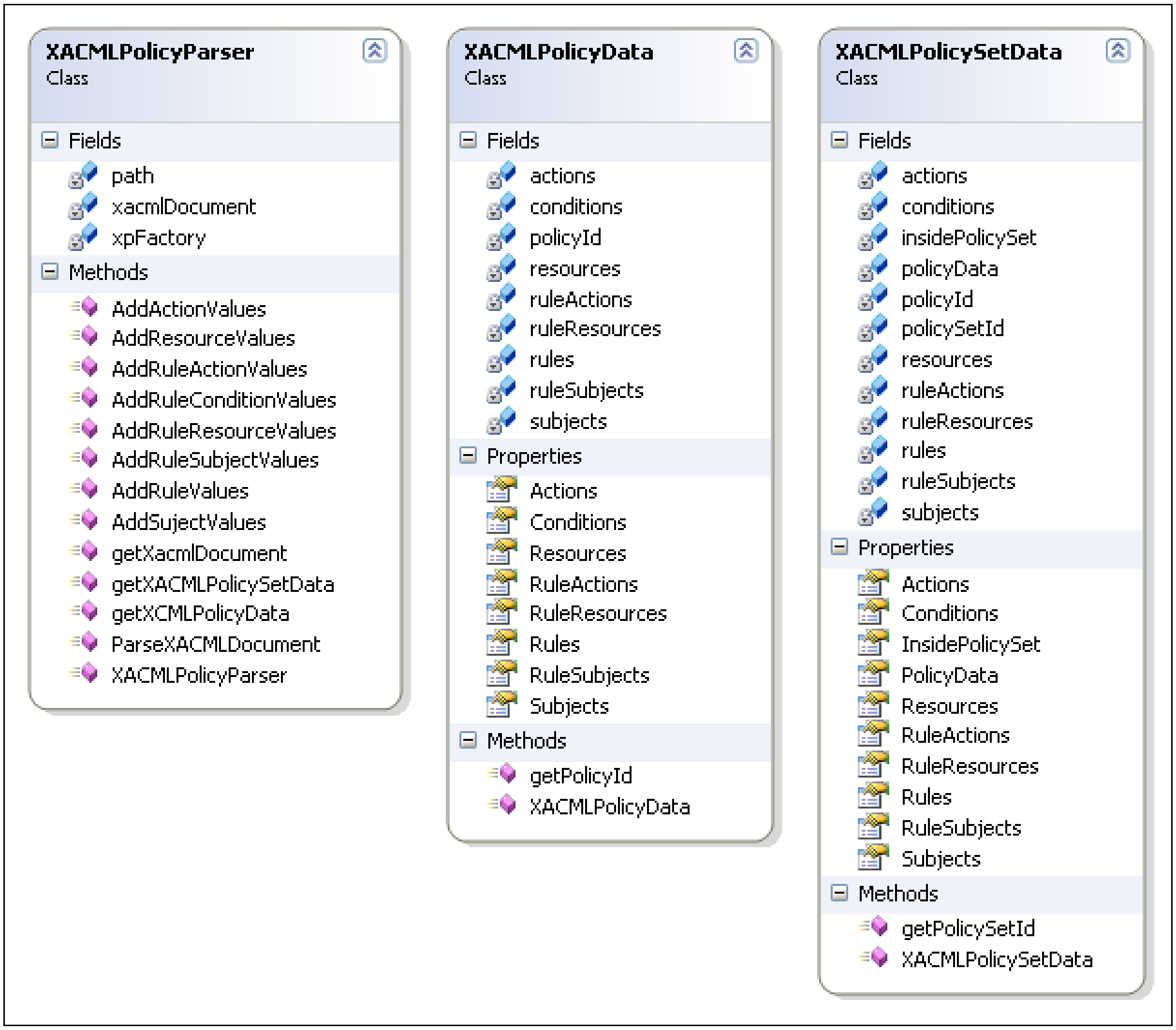


Figure 26: XACMLVizClass Diagram Zoom-1



XACMLPolicyParser
Class

Fields
- path
- xacmlDocument
- xpFactory

Methods
- AddActionValues
- AddResourceValues
- AddRuleActionValues
- AddRuleConditionValues
- AddRuleResourceValues
- AddRuleSubjectValues
- AddRuleValues
- AddSujectValues
- getXacmlDocument
- getXACMLPolicySetData
- getXCMLPolicyData
- ParseXACMLDocument
- XACMLPolicyParser

XACMLPolicyData
Class

Fields
- actions
- conditions
- policyId
- resources
- ruleActions
- ruleResources
- rules
- ruleSubjects
- subjects

Properties
- Actions
- Conditions
- Resources
- RuleActions
- RuleResources
- Rules
- RuleSubjects
- Subjects

Methods
- getPolicyId
- XACMLPolicyData

XACMLPolicySetData
Class

Fields
- actions
- conditions
- insidePolicySet
- policyData
- policyId
- policySetId
- resources
- ruleActions
- ruleResources
- rules
- ruleSubjects
- subjects

Properties
- Actions
- Conditions
- InsidePolicySet
- PolicyData
- Resources
- RuleActions
- RuleResources
- Rules
- RuleSubjects
- Subjects

Methods
- getPolicySetId
- XACMLPolicySetData

Figure 27: XACMLVizClass Diagram Zoom-2

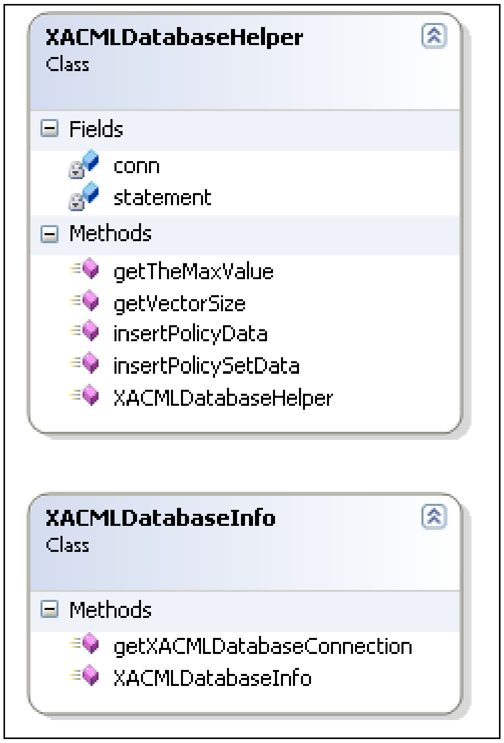


Figure 28: XACMLVizClass Diagram Zoom-3

4.2 Prototype Implementation

The XACMLViz prototype is a Java Swing GUI application where the user, via a menu, can select which policy files need to be scanned for potential policy conflicts. If no potential conflicts are found, a dialog message is presented to the user with a "No potential conflicts found." message.

When the user selects the files to be scanned and clicks the "Open" button, the files are scanned one-by-one by instantiating the XACMLPolicyParser class with the filename and calling its ParseXACMLDocument method. Figure 29 displays the portion of the code inside the XACMLVizFrame.java which iterates through the collection of the file names selected.

```
private void jMenuItem1ActionPerformed(java.awt.event.ActionEvent evt)
 {
        JFileChooser chooser = new JFileChooser();
        chooser.setCurrentDirectory(new File("."));
        chooser.setMultiSelectionEnabled(true);
        chooser.setFileFilter(new XMLFilter());
        chooser.setAcceptAllFileFilterUsed(false);
        int result = chooser.showOpenDialog(this);

        if (result == JFileChooser.APPROVE_OPTION)
        {
            File[] names = chooser.getSelectedFiles();

            for (File f : names)
            {
                try
                {
                    String s = f.getPath();
                    XACMLPolicyParser par = new XACMLPolicyParser(s);
                    par.ParseXACMLDocument();
                }
                catch (IOException ex)
                {
                    Logger.getLogger(XacmlVizFrame.class.getName()).log(Level.SEVERE, null, ex);
                }
                catch (ParserConfigurationException ex)
                {
                    Logger.getLogger(XacmlVizFrame.class.getName()).log(Level.SEVERE, null, ex);
                }
                catch (SAXException ex)
                {
                    Logger.getLogger(XacmlVizFrame.class.getName()).log(Level.SEVERE, null, ex);
                }
                catch (XPathExpressionException ex)
                {
                    Logger.getLogger(XacmlVizFrame.class.getName()).log(Level.SEVERE, null, ex);
                }
            }
```

Figure 29: ParseXACMLDocument Method Call

Inside the XACMLPolicyParser, an instance of the DocumentBuilderFactory and XPathFactory are created to allow for the parsing of the XACML document. Since the top level root element of a XACML document is either a <PolicySet> or <Policy> element, the XACMLPolicyParser retrieves the document root to identify if the particular document is a policy or a policy set. If the root node is a policy set, the method getElementsByTagName of the Document object is called to get all policy sets within the document. All policy set elements are subsequently passed to the XACMLPolicyParser method getXACMLPolicySetData. Here all significant data is extracted and placed in the XACMLPolicySetData object. The information inside the XACMLPolicySetData object will be used later in the process to insert the data into a database table. Figure 30 displays the portion of the described code.

```
public void ParseXACMLDocument() throws IOException, ParserConfigurationException, SAXException,
XPathExpressionException
{
    DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
    DocumentBuilder builder = factory.newDocumentBuilder();

    File file = new File(this.xacmlDocument);
    Document xacmlDoc = builder.parse(file);

    XACMLDatabaseHelper helper = new XACMLDatabaseHelper();

    Element root = xacmlDoc.getDocumentElement();
    String rootName = root.getTagName();
    if (rootName.equalsIgnoreCase("PolicySet"))
    {
      NodeList polSetNodes = xacmlDoc.getElementsByTagName("PolicySet");
      for (int i = 0; i < polSetNodes.getLength(); i++)
      {
        Node pSetNode = polSetNodes.item(i);
        if (pSetNode instanceof Element)
        {
          if (((Element) pSetNode).equals(xacmlDoc.getDocumentElement()))
          {
            Vector<XACMLPolicySetData> data = this.getXACMLPolicySetData(pSetNode);
            helper.insertPolicySetData(data, null);
            continue;
          }

          Node pParent = pSetNode.getParentNode();
          Element parentElement = (Element) pParent;
          String parentId = parentElement.getAttribute("PolicySetId");

          Vector<XACMLPolicySetData> data = this.getXACMLPolicySetData(pSetNode);
          helper.insertPolicySetData(data, parentId);
        }
      }
```

Figure 30: PolicySet Element Iteration

A similar process is followed if the root element is a policy and the extracted data is placed inside the XACMLPolicyData object. The major distinction when parsing between the policy set and the policy is that a policy has a <Rule> element that must be parsed all the time. With the policy set, only if a policy set has a non empty <Target> element then the policy (or policies) is parsed and the <Rule> element is parsed as well. The reason for parsing policies and policy sets differently is because a root policy set can be used as a method of enclosing multiple policies or policy sets. Figure 31 illustrates a portion of the code for extracting the policy set information. Figure 32 illustrates a portion of the code for extracting the policy information.

```
public Vector<XACMLPolicySetData> getXACMLPolicySetData(Node policySet)
{
        Vector<XACMLPolicySetData> rtnPolicySetVector = new Vector<XACMLPolicySetData>();
        String policySetId = ((Element) policySet).getAttribute("PolicySetId");
        XACMLPolicySetData policySetData = new XACMLPolicySetData(policySetId);

        try
        {
            this.AddSujectValues(policySet, policySetData);
            this.AddResourceValues(policySet, policySetData);
            this.AddActionValues(policySet, policySetData);

            if (policySetData.getSubjects().size() != 0
                || policySetData.getResources().size() != 0
                || policySetData.getActions().size() != 0)
            {
                NodeList policyNodes = (NodeList) path.evaluate("./Policy", policySet, XPathConstants.NODESET);

                //add the policy id information for the policy data set so it can be retrieved later
                //during the database selection process.
                Element pElement = (Element) policyNodes.item(0);
                if(pElement != null)
                {
                    String id = pElement.getAttribute("PolicyId");
                    XACMLPolicyData pData = new XACMLPolicyData(id);
                    Vector<XACMLPolicyData> vecPData = new Stack<XACMLPolicyData>();
                    vecPData.add(pData);
                    policySetData.setPolicyData(vecPData);
                }

                for (int i = 0; i < policyNodes.getLength(); i++)
                {
                    Node child = policyNodes.item(i);
                    if (child instanceof Element)
                    {
                        this.AddRuleValues(child, policySetData);
                        this.AddRuleSubjectValues(child, policySetData);
                        this.AddRuleResourceValues(child, policySetData);
                        this.AddRuleActionValues(child, policySetData);
                        this.AddRuleConditionValues(child, policySetData);
                    }
```

Figure 31: getXACMLPolicySetData Method

```
public Vector<XACMLPolicyData> getXCMLPolicyData(Node policy) throws
ParserConfigurationException, SAXException, IOException
{
      Vector<XACMLPolicyData> rtnPolicyVector = new Vector<XACMLPolicyData>();

      String policyId = ((Element) policy).getAttribute("PolicyId");
      XACMLPolicyData policyData = new XACMLPolicyData(policyId);
      try
      {
         this.AddSujectValues(policy, policyData);
         this.AddResourceValues(policy, policyData);
         this.AddActionValues(policy, policyData);
         this.AddRuleValues(policy, policyData);
         this.AddRuleSubjectValues(policy, policyData);
         this.AddRuleResourceValues(policy, policyData);
         this.AddRuleActionValues(policy, policyData);
         this.AddRuleConditionValues(policy, policyData);

      }
      catch (XPathExpressionException ex)
      {
         Logger.getLogger(XACMLPolicyParser.class.getName()).log(Level.SEVERE, null, ex);
      }

      rtnPolicyVector.add(policyData);

      return rtnPolicyVector;
   }
```

Figure 32: getPolicyData Method

It is important to note that the approach taken in identifying potential conflicts in policies is based on the union of all <Target> elements within the policy sets. In addition, only the information contained within the <Target> element is compared for equality in determining if there is a potential conflict. Furthermore, this prototype is built to parse policy data that is contained within the selected files. Files that are referenced outside the containing policy file are not taken into consideration at this time.

After extracting all of the desired information, the data is then inserted into a database table. This table contains combined data, that is, the union of all of the targets and rules found during the parsing of the XACML document. This table is then queried to determine if there are any potential conflicts among policies. If there are potential conflicts, they are displayed in the XACMLViz main frame. Figure 33 illustrates the table column information.

```
create table POLICYINFO
(
   ID INTEGER NOT NULL GENERATED ALWAYS AS IDENTITY,
   POLICY_SET_PARENT_ID VARCHAR(50),
   POLICY_SET_ID VARCHAR(50),
   POLICY_ID VARCHAR(50),
   POLICY_SUB VARCHAR(50),
   POLICY_RES VARCHAR(50),
   POLICY_ACT VARCHAR(50),
   POLICY_ENV VARCHAR(50),
   POLICY_OBL VARCHAR(50),
   RULE_ID VARCHAR(50),
   RULE_COND VARCHAR(50),
   RULE_EFFECT VARCHAR(25)
);
```

Figure 33: Derby Table Information

The visualization obtained by running the XACMLViz prototype is based on the prefuse visualization toolkit [Prefuse]. Prefuse is written in Java and can be used freely. The type of visualization chosen for this prototype is based on a prefuse Tree. The tree is populated by iterating through the result set returned from the select query statement. After the tree is populated, the database table rows are deleted. Next, there is a check to ensure that there is more than one tree populated. If there is more than one tree, the trees are then passed to the SelfArisenTree [PAP] class to make the view for each tree. The SelfArisenTree is a sample class which shows the user how to create visualizations with prefuse by using a prefuse tree. This class was slightly modified to serve the purposes of the XACMLViz prototype. If there are less than two trees, a message is displayed to the user as previously mention. When conflicting policies are found, each tree is placed in a TableLayout [Barbalace] inside a JScrollPane arranged by rows and columns. Once the trees are displayed, they can be moved around their visualization area by clicking the mouse and dragging. In addition, the user can click on a tree leaf node and if there are any children the tree expands. The visualization area of the tree can also be zoomed in and out by using the mouse wheel. The tree can also be made to fit its visualization area by clicking the right mouse button.

Chapter 5 Experimental Results

In order to test and verify the XACMLViz prototype a set of policy files was needed. The Sun's XAMCL Implementation contains a few policy examples which were used during my research.

5.1 Multiple Policies Experiment

The generated.xml policy contained in the sample folder of the Sun's XACML Implementation was used. Figures 34 and 35 are the generated.xml policy. This policy states that a developer who belongs to users.example.com can commit to the server server.example.com. Two copies of this policy were made with what can be determined as potentially conflicting information. The first copied policy was named generated2.xml. The PolicyId attribute of the <Policy> element was changed to GeneratedPolicy2. The rule for this policy was changed to be interpreted as allowing read access to server.example.com if the subject belonged to users.example.com. The second copied policy was named generated3.xml. The PolicyId attribute of the <Policy> element was also changed to GeneratedPolicy3. The rule for this policy was changed as well to allow write access to server.example.com if the subject belonged to users.example.com.

```
<?xml version="1.0" encoding="UTF-8"?>
<Policy xmlns="urn:oasis:names:tc:xacml:1.0:policy"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    PolicyId="GeneratedPolicy"
    RuleCombiningAlgId=”urn:oasis:names:tc:xacml:1.0:
    rule-combining-algorithm:ordered-permit-overrides">

  <Description>
    This policy applies to any accounts at users.example.com accessing
    server.example.com. The one Rule applies to the specific action of
    doing a CVS commit, but other Rules could be defined that handled
    other actions. In this case, only certain groups of people are
    allowed to commit. There is a final fall-through rule that always
    returns Deny.
  </Description>

  <Target>
    <Subjects>
      <Subject>
        <SubjectMatch
           MatchId="urn:oasis:names:tc:xacml:1.0:function:rfc822Name-match">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
           users.example.com</AttributeValue>
          <SubjectAttributeDesignator
           DataType="urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name"
           AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"/>
        </SubjectMatch>
      </Subject>
    </Subjects>
    <Resources>
      <Resource>
        <ResourceMatch
           MatchId="urn:oasis:names:tc:xacml:1.0:function:anyURI-equal">
          <AttributeValue
           DataType="http://www.w3.org/2001/XMLSchema#anyURI">
           http://server.example.com/</AttributeValue>
          <ResourceAttributeDesignator
           DataType="http://www.w3.org/2001/XMLSchema#anyURI"
           AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"/>
        </ResourceMatch>
      </Resource>
    </Resources>
    <Actions>
      <AnyAction/>
    </Actions>
  </Target>
```

Figure 34: Sun's XACML Generated Policy

```
<Rule RuleId="CommitRule" Effect="Permit">
    <Target>
      <Subjects>
        <AnySubject/>
      </Subjects>
      <Resources>
        <AnyResource/>
      </Resources>
      <Actions>
        <Action>
          <ActionMatch
            MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
             <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
              commit</AttributeValue>
             <ActionAttributeDesignator
              DataType="http://www.w3.org/2001/XMLSchema#string"
              AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"/>
          </ActionMatch>
        </Action>
      </Actions>
    </Target>
    <Condition FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
      <Apply
        FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
          <SubjectAttributeDesignator
              DataType="http://www.w3.org/2001/XMLSchema#string"
              AttributeId="group"
              Issuer="admin@users.example.com"/>
      </Apply>
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
          developers</AttributeValue>
    </Condition>
  </Rule>

  <Rule RuleId="FinalRule" Effect="Deny"/>
</Policy>
```

Figure 35: Sun's XACML Generated Policy (Continue)

The expected result for this particular test is for the XACMLViz prototype to point out that these three policies can be potentially conflicting policies. The expected results are displayed in Figure 36. As expected, the XACMLViz prototype tool identified all three policies as having potentially conflicting data. The highlighted trees in the XACMLViz visualization area depict the possibly conflicting policies. Figures 37, 38, and 39 are a zoomed version of the trees in Figure 36.

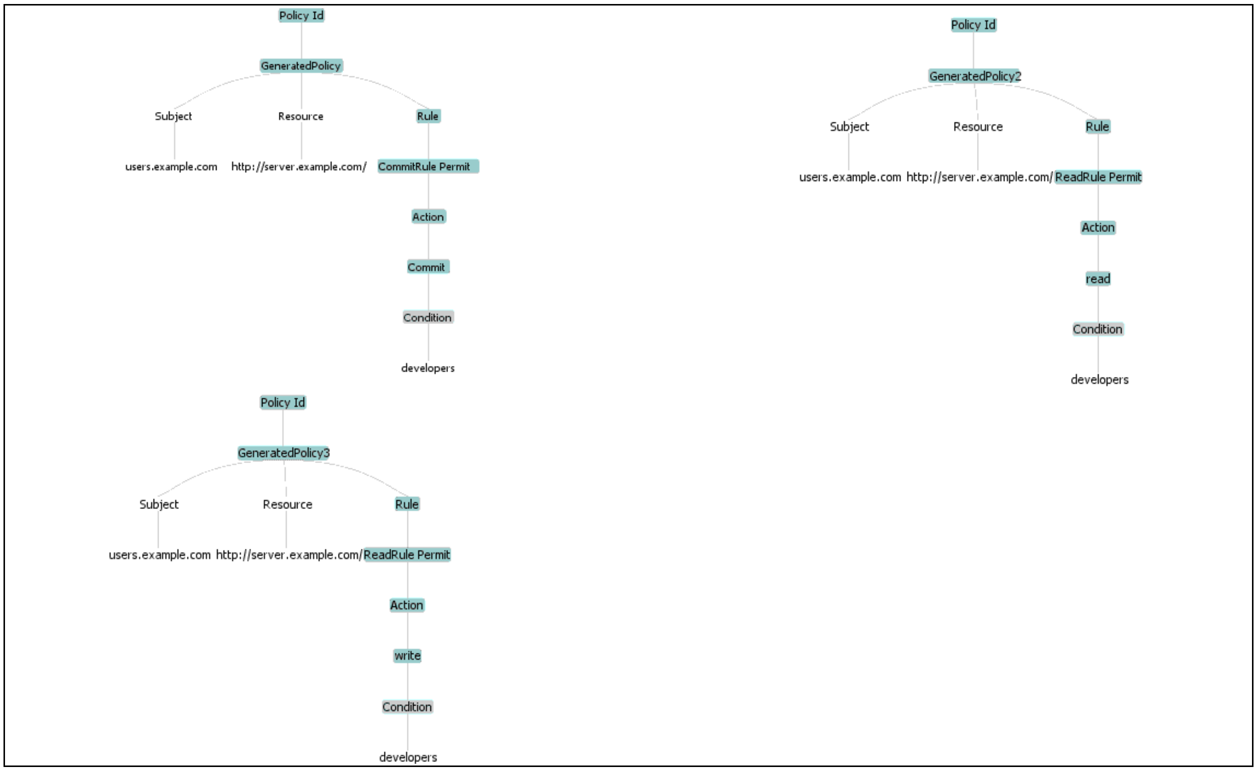


Figure 36: XACMLViz Potential Conflict Visualization Generated Policies

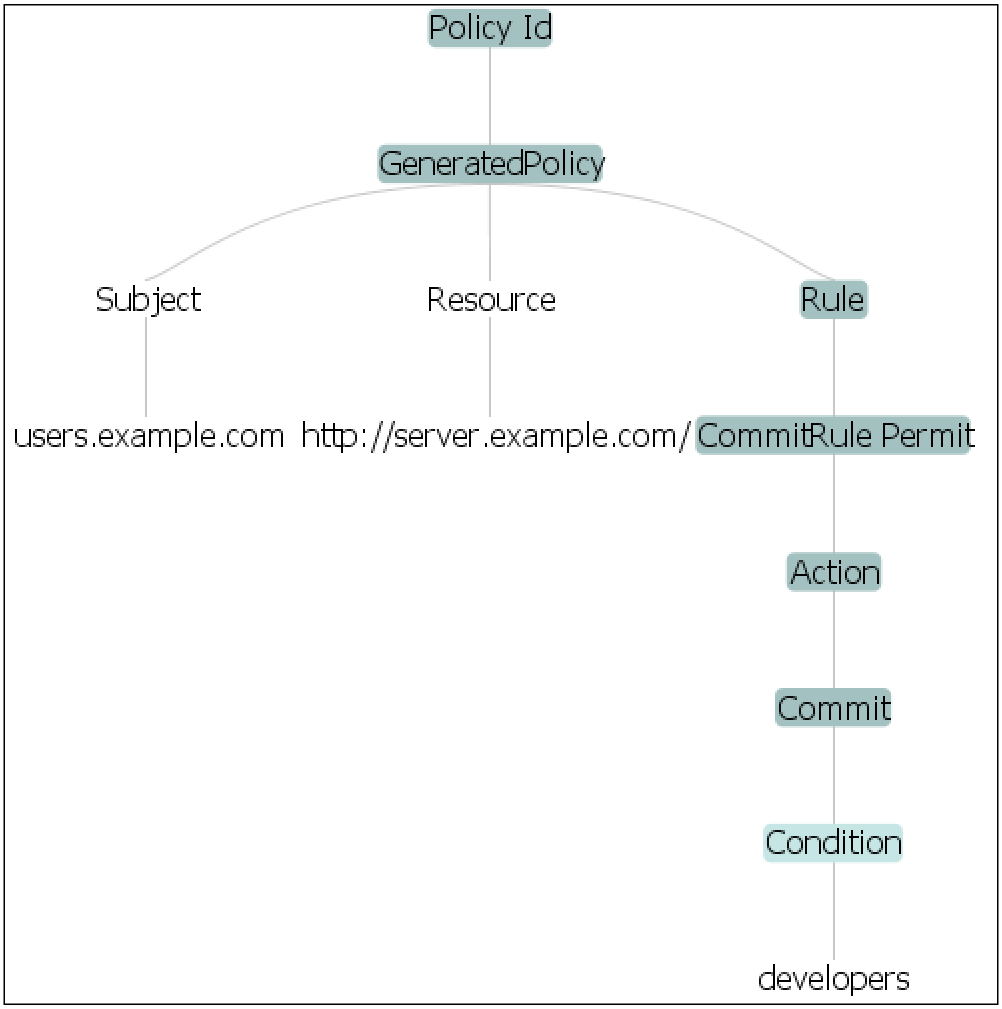


Figure 37: XACMLViz (Figure 36) GeneratedPolicy Tree Zoom

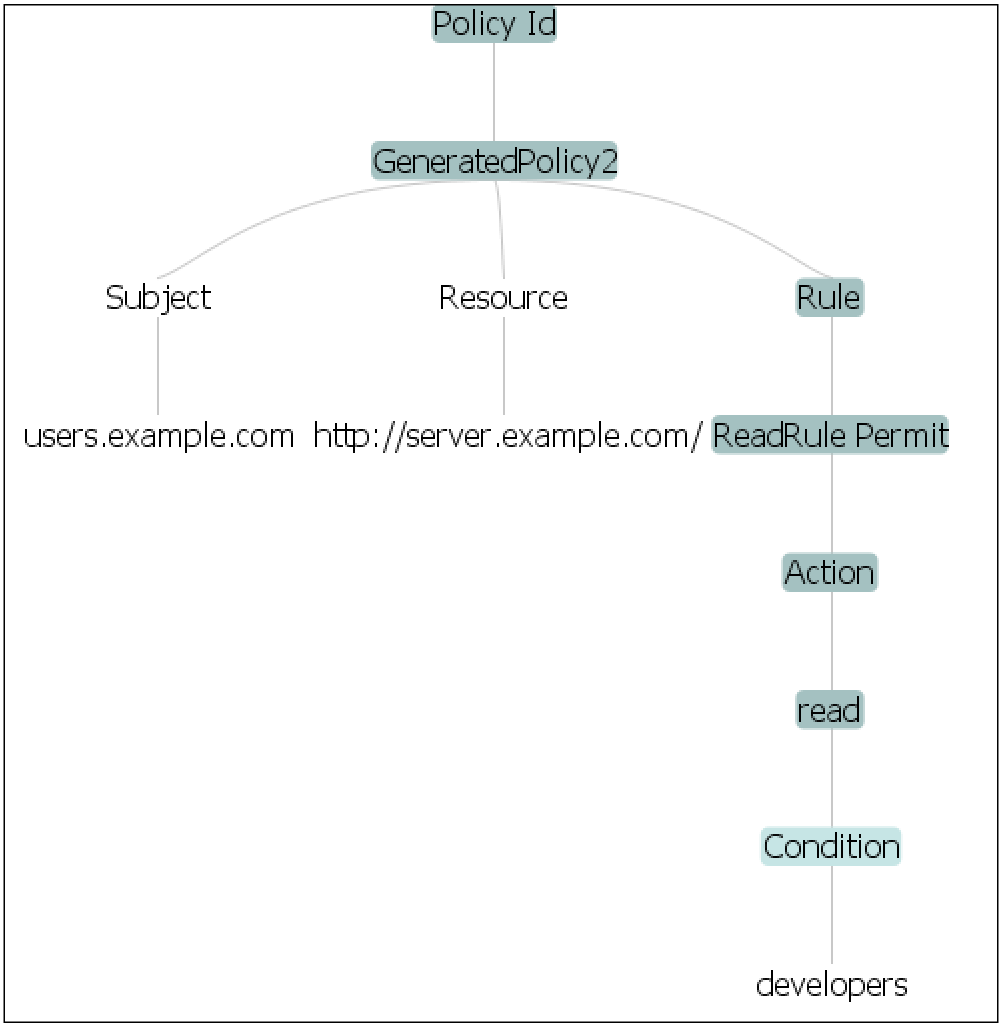


Figure 38: XACMLViz (Figure 36) GeneratedPolicy2 Tree Zoom

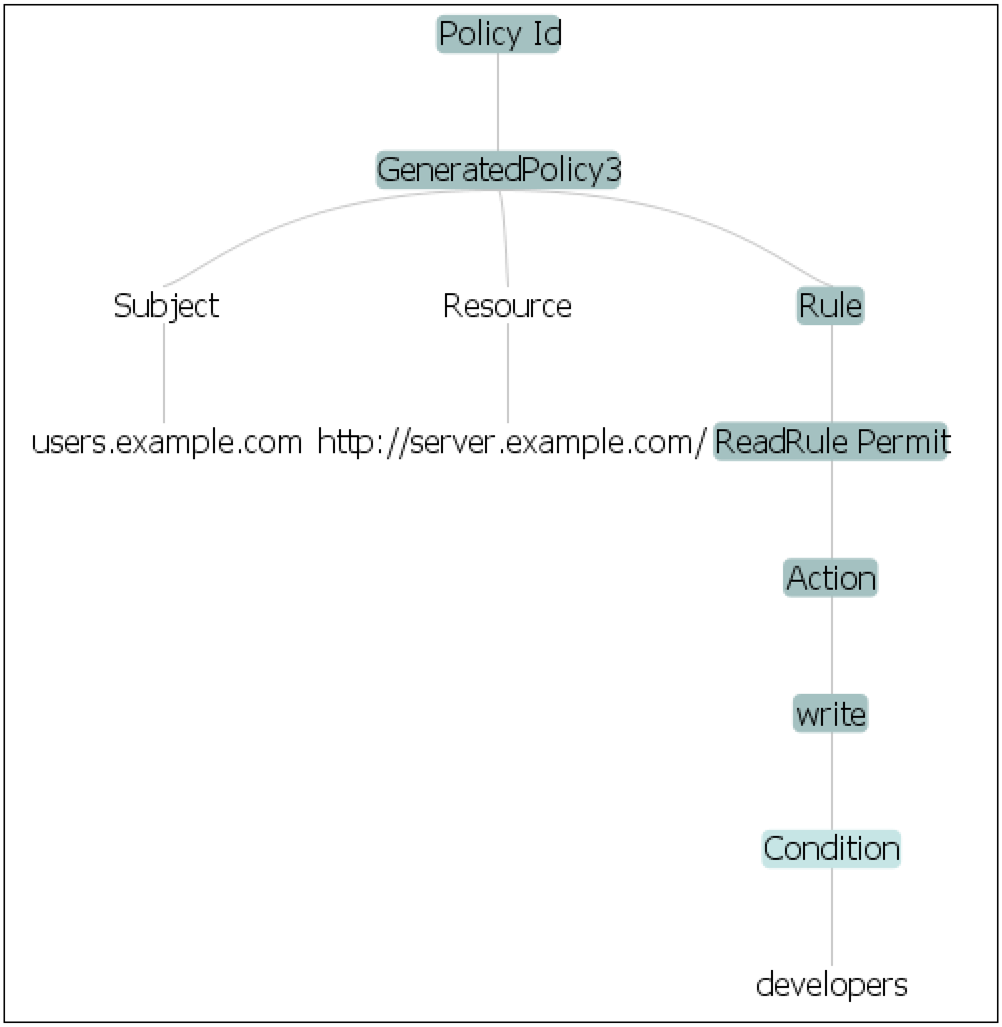


Figure 39: XACMLViz (Figure 36) GeneratedPolicy3 Tree Zoom

5.2 PolicySet Experiment

Another test was done with the time-range policy (see Figure 9). This policy was modified by including the original policy and a copied policy as part of a policy set. The copied policy was changed to reflect a different time range in the <Condition> element. The expected result of this test was to show that there are potential conflicts when non-authorized personnel are allowed to open the main door. As expected, the XACMLViz tool identified both policies within the policy set as containing potentially conflicting data. The expected results are displayed in Figure 40.

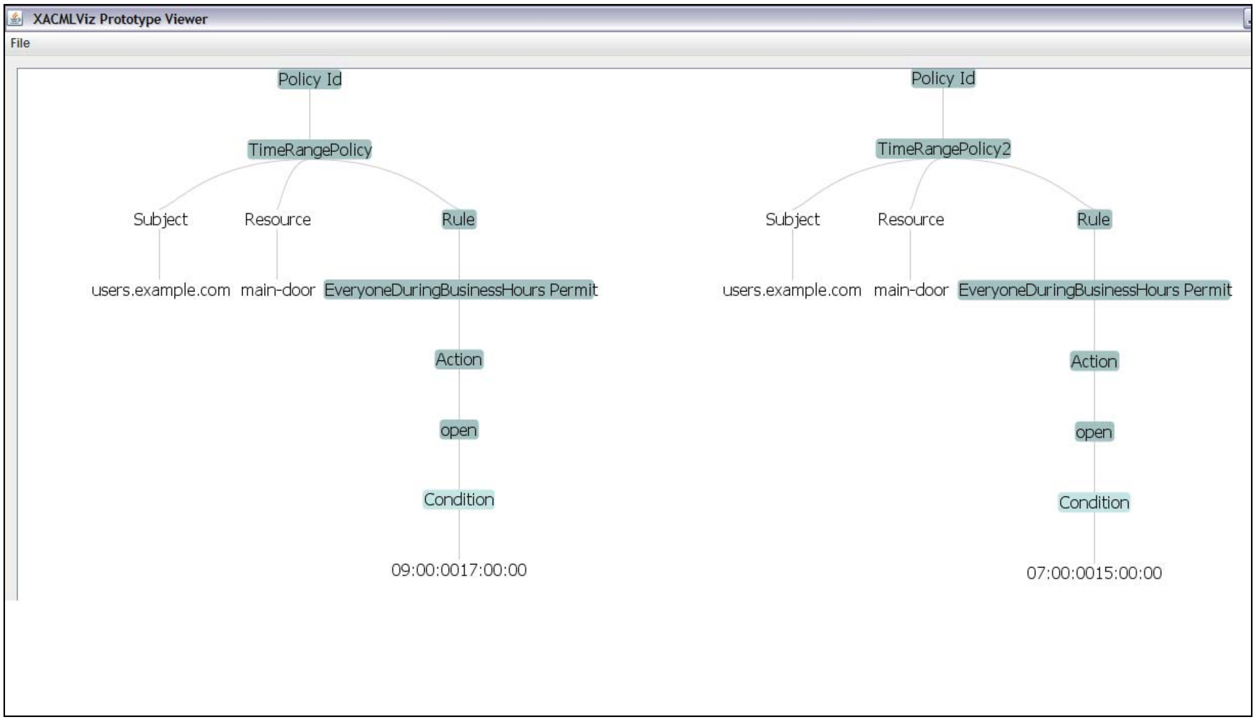


Figure 40: XACMLViz Potential Conflict Visualization TimeRange PolicySet

The potentially conflicting policies are displayed where the conditions from PolicyId TimeRangePolicy states that access is permitted to open the door between the hours of 0900 and 1700; whereas in PolicyId TimeRangePolicy2, it allows access to the main door between the hours of 0700 and 1500.

5.3 Further Development

This prototype can be further enhanced by allowing the scanning of single policies for rule inconsistencies as demonstrated in Chapter 3. One can potentially take advantage of the Sun’s XACML Implementation classes and use some of the matching functions to match subjects to particular rules and resources.

Chapter 6 Discussion and Conclusion

In conclusion, the issue of potentially conflicting policies was addressed in this thesis. Possible solutions to these conflicts were discussed as part of a new prototype or as part of a further enhancement to this prototype. Additionally, the testing command line program from the Sun's XACML Implementation project was discussed and executed to test possible issues that can be encountered by a PDP when parsing through conflicting policies.

During my research, I achieved a wealth of knowledge about XACML and how to programmatically work with XML by using the Document Object Model (DOM) and XPath. This was a valuable experience since the skills I learned during this process are a welcome addition to any developer's tool box.

In addition, the ease and practicality of the XACMLViz prototype program could be a great resource to anyone wishing to scan through a set of policies for potential inconsistencies within its targets.

References

[XACML Core] OASIS XACML 2.0 Core: eXtensible Access Control Markup Language (XACML) Version 2.0
http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-rbac-profile1-spec-os.pdf

[Sun XACML] Sun's XACML Implementation
http://sunxacml.sourceforge.net/

[Prefuse] The Prefuse Visualization Toolkit
http://prefuse.org/

[SDN] Sun Developer Network (SDN), Article
XACML: A new Standard Protects Content in Enterprise Data Exchange
http://java.sun.com/developer/technicalArticles/Security/xacml/xacml.html

[IBM] Effective XML processing with DOM and XPath in Java
http://www.ibm.com/developerworks/xml/library/x-domjava/

[W3C] World Wide Web Consortium http://www.w3.org/

[XPath] W3C, XML Path Language (XPath)
http://www.w3.org/TR/xpath

[DOM] W3C, Document Object Model (DOM)
http://www.w3.org/DOM/

[PAP] Perfuse Assistance Pool (PAP)
http://goosebumps4all.net/34all/bb/showthread.php?tid=46

[Barbalace] TableLayout Tutorial, Part 1
https://tablelayout.dev.java.net/articles/TableLayoutTutorialPart1/TableLayoutTutorialPart1.html

[Anderson] OASIS eXtensible Access Control Markup Language (XACML), XML Community of Practice, 21 June 2006
Anne Anderson, Senior Staff Engineer
Sun Microsystems Laboratories

[Logrippo] Access Control Policies: Modeling and Validation
Luigi Logrippo & Mahdi Mankai

Université du Québec en Outaouais

[Hwang] XACML Access Control Policy Testing
JeeHyun Hwang

[Montemayor] Information Visualization for Rule-based Resource Access Control
Jaime Montemayor, Andrew Freeman, John Gersh, Thomas Llanso, Dennis Patrone
The Johns Hopkins University Applied Physics Laboratory

Vita

William Domingo Rosa was born in Bronx, New York on February 27, 1970. He was raised by his paternal grandparents on the island of Puerto Rico in the western town of Aguada. He enlisted in the United States Navy in March of 1990 and served honorably until July of 2001. He received an Associate in Arts Degree in Computer Information Systems from Pensacola Junior College, Pensacola, Florida in May of 2000. He received a Bachelor of Science Degree in Computer Science/Information Systems from The University of West Florida, Pensacola, Florida in August of 2002.

His software development professional experiences include working for the Defense Finance and Accounting Services (DFAS) Technology Services Organization Pensacola (TSOPE), Pensacola, Florida as an Information Technology Specialist; the Space and Naval Warfare Systems Command (SPAWAR) Systems Center New Orleans, New Orleans, Louisiana as a Technical Specialist; and currently Intergraph Corporation, Process, Power & Marine, Huntsville, Alabama as a Senior Software Analyst.