Patterns for the eXtensible Access Control Markup Language

Nelly Delessy and Eduardo B. Fernandez
Dept. of Computer Science and Engineering
Florida Atlantic University
Boca Raton, FL 33431
ndelessy@fau.edu, ed@cse.fau.edu

Abstract

Web services are becoming the way for enterprises to interoperate. Many security standards for them have been developed; one of these is XACML (eXtensible Access Control Markup Language). XACML has been defined by OASIS and it includes a policy, an access decision language, and a specialized web services policy language. We present here three architectural patterns for XACML. The XACML Authorization pattern unifies the definition of authorization rules throughout an organization. WSPL is a specialization of XACML Authorization, intended to describe access control rules for web services. The XACML Access Control Evaluation pattern defines a request/response syntax for access control decisions.

1 Introduction

The typical computer system of a large organization is heterogeneous since its applications include off-the-shelf products from different vendors, as well as user-defined applications with different origins. At the same time, driven by business imperatives, these systems are opened to a wide variety of partners, customers or mobile employees, which introduce a new variety of security threats. These organizations must protect their information assets from attacks. Their information assets typically include services, which come in a variety of technologies, components, and data.

To protect these assets, an organization needs to define security policies, which are high level guidelines that specify in what states the system is considered to be secure [Fer06]. These policies need to be enforced by security mechanisms. In large organizations, the policies may be issued by different actors making their management difficult. Moreover, they need to be enforced for a variety of resources.

Furthermore, the ubiquity of the web implies that a subject does not need to be known in advance by the system to request access to a resource. The use of credentials including attributes may be sufficient to trust a subject. Policies should be able to capture this aspect.

XACML (eXtensible Access Control Markup Language) has been defined by OASIS [OAS] and it includes languages for expressing authorization rules and for access decision (enforcement of the rules). The XACML profile for web services, also known as WSPL (Web Services Policy Language), is a language to declare authorization rules for protecting web services endpoints. We describe here patterns for these three aspects of XACML.

XACML is an XML-based language and it is rather complex. Here we present its underlying security model sing UML diagrams. The UML diagrams provide a notation that can be used to better understand the language and to guide the design of systems using this standard. These models can also be used to compare more conveniently this standard with other languages or standards with similar purposes.

The XACML standard proposes an authorization system that consists of five conceptual units (Figure 1). The Policy Enforcement Point (PEP) performs access control by requesting an access decision to the Policy Decision Point (PDP). The PDP uses the policies made available to it by the Policy Administration Point (PAP) and the additional attributes sent by the Policy Information Point (PIP) to render its decision. The PEP communicates with the PDP and PIP through a Context Handler (CH) that is an adapter between the XACML components and the protected application [XAC04].
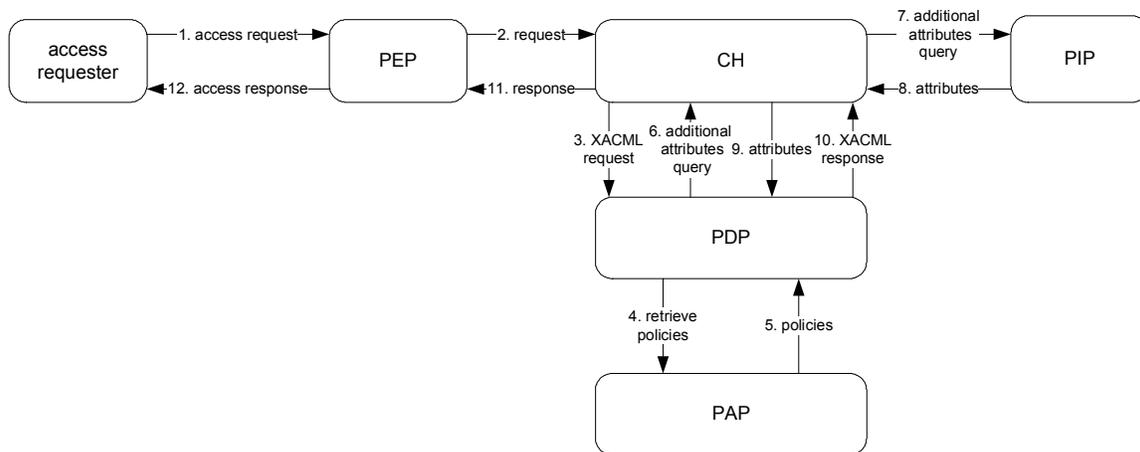

Figure 1: Overview of the data flow in the XACML authorization system (from [XAC04])

2 XACML Authorization

XACML enables an organization to represent authorization rules in a standard manner.

2.1 Example

Consider a financial company that provides financial services to its customers. Their computer systems can be accessed by customers who send orders to the company for buying or selling commodities (stocks, bonds, real estate, art, etc.) by email or through their website. Brokers employed by the company can carry out the orders of the customers by sending requests to the systems of various financial markets or consult information from financial news websites. Also, a government auditor visits periodically to check for application of laws and regulations.

All those activities are regulated by policies with various granularities within the company. For example, the billing department can have the rule «only registered customers whose account status is ok may send orders», the technical department can decide that «emails with attachments bigger than x Mb won't be delivered», the company security policy can state that «only employees with "broker" role can access the financial markets web services» and that «only the broker custodian of a customer can access its transaction information», whereas the legal department issues the rule «auditors can access every transaction information», etc.

All these policies are enforced by different components of the computer system of the company (email server, file system, web service access control component, financial application). This approach has several problems: The policies are described in possibly different syntaxes and it is difficult to have a global view of what policies apply to a specific case. Moreover, two policies can be conflicting and there is no way to combine them in a clear way. In summary, this approach could be error-prone and complex to manage.

2.2 Context

A complex environment such as a large enterprise with many partners, contractors and relations with other enterprises. These various actors are accessing the organization's resources, comprising web services, sensitive documents or system components.

2.3 Problem

An organization's resources are usually from various types (XML documents, web services, web component, CORBA services…). Accesses to these resources are controlled by distributed enforcement mechanisms, according to the security policies of the institution. Since the resources are of different types, the enforcement mechanisms

come in various forms: they can be part of a web server, an application firewall, etc. Therefore, policies have to be implemented in many locations, using different syntaxes. It is important to define precisely the policies about accessing these resources.

Moreover, security policies in an organization are typically issued by different actors from its departments (human resources, legal, marketing departments…), and the policies they write may concern a wide and overlapping set of resources. Defining these policies in a way that the right policies can be applied to each access may be complex, and thus error prone.

How do we unify the definition of access policies throughout the organization, making the whole system simpler and less error-prone? The solution to this problem is affected by the following forces:

- The policies are issued by a variety of actors and may be stored in many locations. This means they may be expressed in different forms.
- The policies are constantly changing and they need to be constantly updated.
- An active entity accessing a resource can be represented in a variety of ways, including certificates.
- Some policies can require a set of actions (or obligations) to be performed in conjunction with policy enforcement (auditing, notification…).
- The environment in which the access is requested can also affect an access decision. For instance, an access may only be permitted at some hours of the day.

2.4 Solution

Write all policies in a common language using a standard format. This format is generic enough to implement some common high level policies or models (open/closed systems, extended access matrix, RBAC, multilevel). In addition, define a way to compose policies so that when several policies apply to one access, it is possible to render one unique decision: the policies have a combining algorithm.

Structure

Figure 2 describes the structure of this pattern.

A **PolicyAdministrationPoint** is a rule repository that centralizes the definition of policies throughout the organization.

The **Subject** intending the access, the **Resource** at which the access is targeted, and the **Environment** of the access are described through their attributes. The **Environment** represents the characteristics of an access that are independent of the **Subject** or **Resource**. It could include the current date, time or other environmental properties.

A **Rule** is a basic unit of policy and it has the usual meaning. In the access matrix model, it defines a set of **Subjects, Resources (**i.e. protection objects), and **Actions** (i.e. access

types). However, in this pattern, a **Rule** associates not only one, but a set of **Subjects**, with a set of **Resources**, and a set of **Actions**. It also includes a set of **Environments** to which the rule is intended to apply, a condition and an effect ("Permit" or "Deny", i.e. positive and negative rules). The condition refines the rule by imposing constraints on the subjects, the resources, or the environment. The **Target** of the rule is made of the sets of **Subjects**, **Resources**, **Actions** and **Environment**s to which the rule is intended to apply. A Target is used for identifying the applicable rules in a given context.
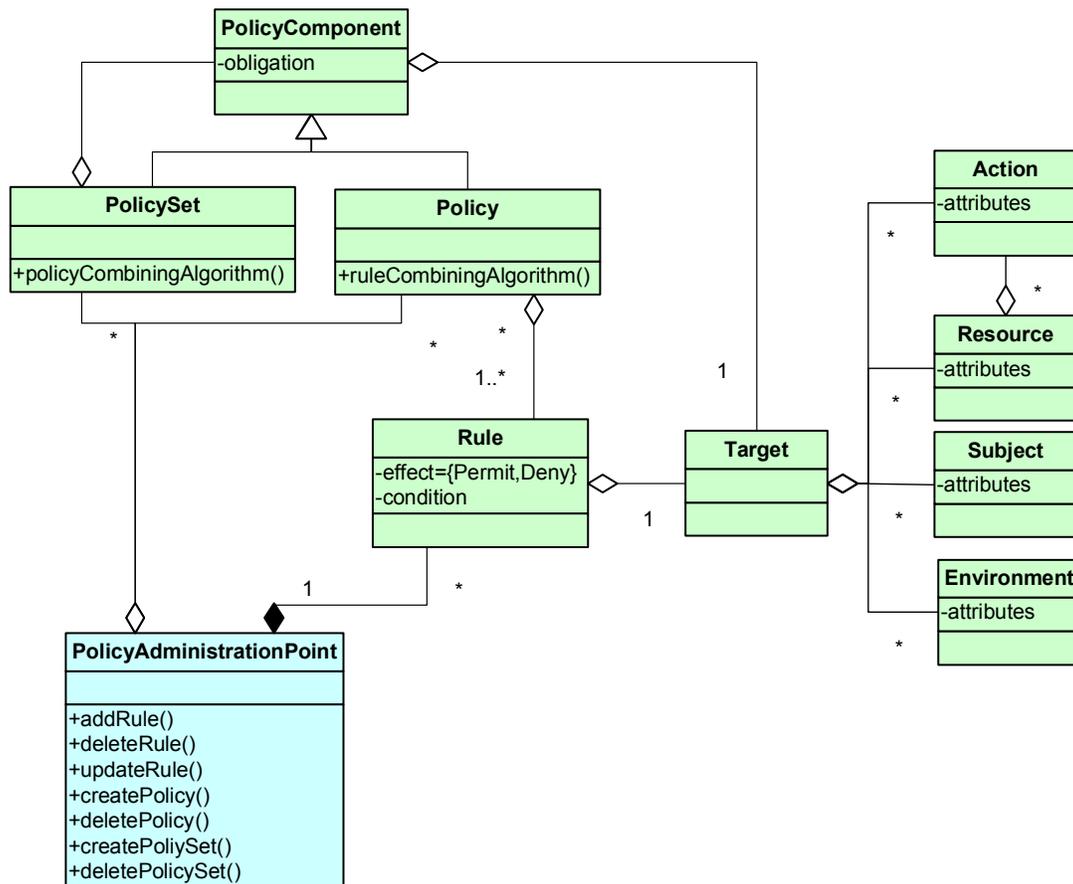

Figure 2:   Class diagram for the XACML policy language

Policies are composed of **Rules**. When evaluating a **Policy**, **Rules** are combined according to the **Policy**'s ruleCombiningAlgorithm (Deny-overrides, Permit-overrides, First-applicable, Only-one-applicable, or a user-defined algorithm

Policies are structured according to a Composite Pattern [Gam95], where a **PolicySet** is the composite element.). Similarly, when evaluating a **PolicySet**, **Policie**s are combined according to the **PolicySet**'s policyCombiningAlgorithm. (We could use here a Strategy pattern [Gam95] to have more than one algorithm.). This indicates that policies have a tree structure. Each **PolicyComponent** may include an obligation that defines an operation that should be performed after enforcing the access decision. For example, an obligation could be an audit operation or a notification to an external client.

In addition to its rules' **Target**s, each **PolicyComponent** may be associated with a **Target**. A **Target** at this level is either specified by the **Policy** writer, or calculated as the union or the intersection of the **Target**s of the **Rules** comprising this **Policy**.

Dynamics

We describe the dynamic aspects of the XACML policy language using a sequence diagram for the use case "Create a new policy".

Create a new policy (Figure 3):

Summary: A Policy writer intends to create a new PolicyComponent.
Actors: Policy writer.
Precondition: The Policy writer must have authorization to create Policies.
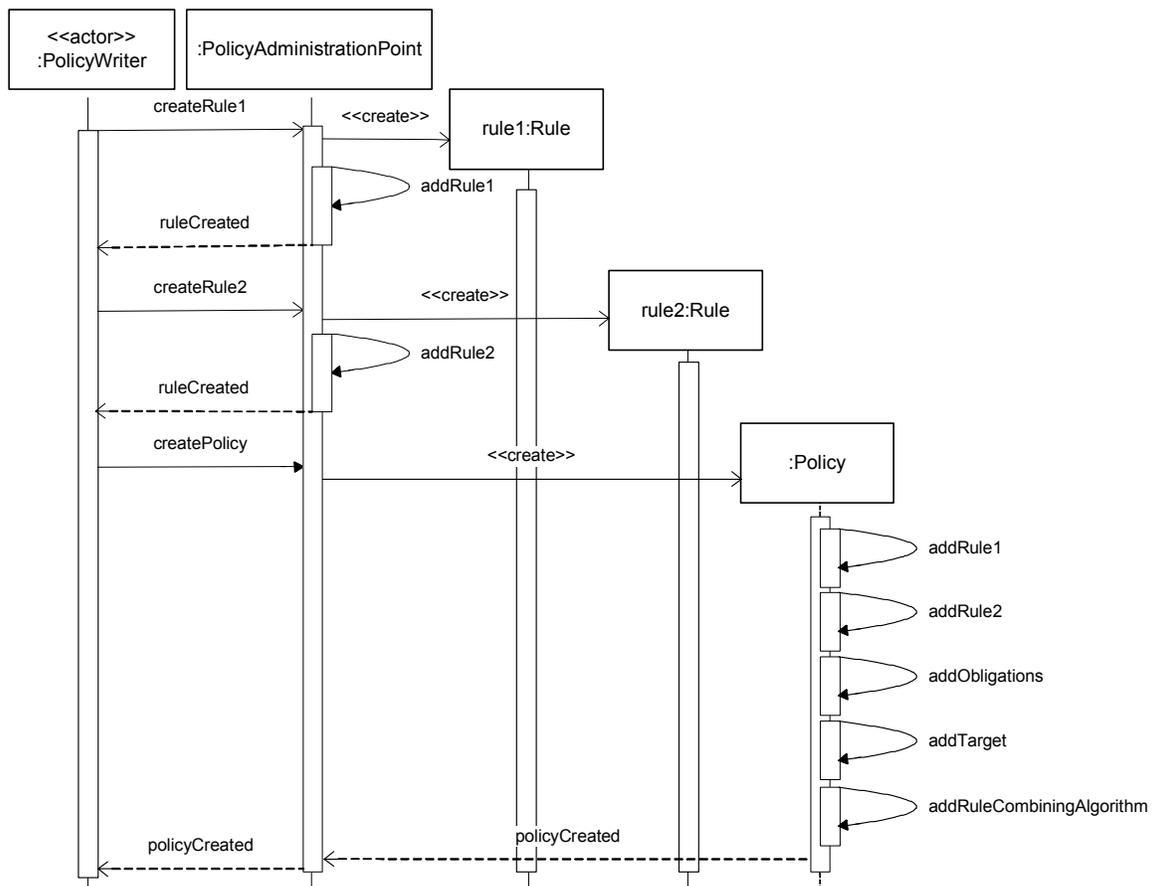

Figure 3: Sequence Diagram for defining a new Policy

Description:
a. The Policy writer creates as many rules as necessary, specifying the target, the effect and possibly a condition for each rule.

b. The rules are added to the set of existing rules.
c. The Policy writer creates a Policy by specifying the rules and optionally some obligations and targets, and the ruleCombiningAlgorithm.
d. The PolicyAdministrationPoint acknowledges the creation of the new Policy.

Postcondition: The new Policy is added to the Policy set of the PolicyAdministrationPoint.

2.5 Implementation

The enterprise must have decided to use XACML to provide security for its documents and services. This decision is based on the fact that XACML is a standard and several products support its use. Once this decision is made, we need:
1. Define semantics for the subject, the resource and the environment's attributes for each intended authorization. These attributes can be from existing standards (LDAP attributes, SAML, …), and are extensible.
2. Translate existing rules into the XACML format.
3. Define new rules and implement them as XACML rules and policies.
4. Add/Remove policies when needed.

For example, we can have rules describing authorization for individual users, roles or any relevant active entity.

2.6 Consequences

The XACML Policy Language pattern presents the following advantages:
- The organization's policies to control access are easily defined using he constructs of the language. This makes the whole system less complex, and thus more secure.
- A variety of policy types can be described, as the policy language includes the resource, the subject and the environment' attributes.
- Similarly, a variety of subject types can be described.
- Policies and rules can be easily combined.
- A policy writer can specify complex conditions.
- This pattern enables logging or other actions through the obligation concept.

The pattern also has some (possible) liabilities:
- The structure of a policy is complex. It is verbose for even simple rules and may require a longer processing time to evaluate a request.

2.7 Known Uses

This pattern is used in several commercial products, such as Xtradyne's WS-DBC (an XML Firewall) [Xtr05], DataPower's XS40 XML Security Gateway [Dat05]. Parthenon Computing has produced a suite of Policy products based on XACML (Policy Tester,

Policy Engine, Policy Server) [Par05]. In addition, Sun provides an open source implementation written in Java [Sun04].

2.8 Example Resolved

The use of XACML authorization rules makes it possible for the company to centralize a wide range of policies and rules. Those can be easily managed, and the conflicts can be resolved by using rights combining algorithms when evaluating an access request.

2.9 Related Patterns

The policies are structured according the Composite Pattern [Gam95]. Rules correspond to a specialization of the Authorization pattern [Fer01].

3 *XACML Access Control Evaluation*

This pattern decides if a request is authorized to access a resource according to policies defined by the XACML Authorization pattern.

3.1 Example

We consider the same financial company. Its policies and rules are enforced by different components of the computer system of the company (some by the email server, file system, web service access control component, financial application). It requires much time and money to administer access control on those different systems.

3.2 Context

A complex environment such as a large enterprise with many partners, contractors and relations with other enterprises. These various actors are accessing the organization's resources, comprising web services, sensitive documents or system components. These accesses are controlled at several enforcement points, according to security policies.

3.3 Problem

An organization's resources are usually of various types. Accesses to these resources are controlled by distributed enforcement mechanisms, according to its security policies. Since the resources are from different types, the enforcement mechanisms come in various forms: they can be a part of a web server, an application firewall, etc. Therefore, the organization has to set up and maintain numerous authorization systems for its networks.

How do we enforce the rules defined in the institution policies? The solution to this problem is affected by the following forces:

- Enforcement points could be implemented in a variety of systems (part of a Web Server, in a WAN, …).
- Any type of security policy should be enforced.
- Enforcement may require reading system or environment variables.

3.4   Solution

Protect resources by **PolicyEnforcementPoints.** All access requests to this **PolicyEnforcementPoint** are evaluated by submitting them to a unique **PolicyDecisionPoint** in a common format. This **PolicyDecisionPoint** returns the access decision, based on the **ApplicablePolicy** corresponding to the access context. The **PolicyInformationPoint** provides attributes from the subject.

Structure

Figure 4 illustrates the XACML access control evaluation pattern. A **Subject** can access a **Resource** in the current **Environment** only if an **XACMLAccessResponse** authorizes it to do so. The **Subject**, **Resource** and **Environment** are described through their attributes. Its specificity is that an access is realized through three entities, the **Subject**, the **Resource** and the **Environment**, instead of just the Subject and the Resource.   This enables to fully describe the characteristics of an access to be evaluated.

The **PolicyEnforcementPoint** requests an access decision to the **PolicyDecisionPoint** through a **ContextHandler**, which is an adapter between any specific enforcement mechanism and the XACML **PolicyDecisionPoint**. The **PolicyDecisionPoint** is responsible for deciding whether or not an access should be permitted, by locating the **ApplicablePolicySet**, that is the set of policies that is applicable to the particular access attempt applying it to the **XACMLAccessRequest**, and issuing a corresponding **XACMLAccessResponse**.

The **ContextHandler** can also get additional attributes from a **PolicyInformationPoint**, which is responsible for obtaining attributes from the subject.
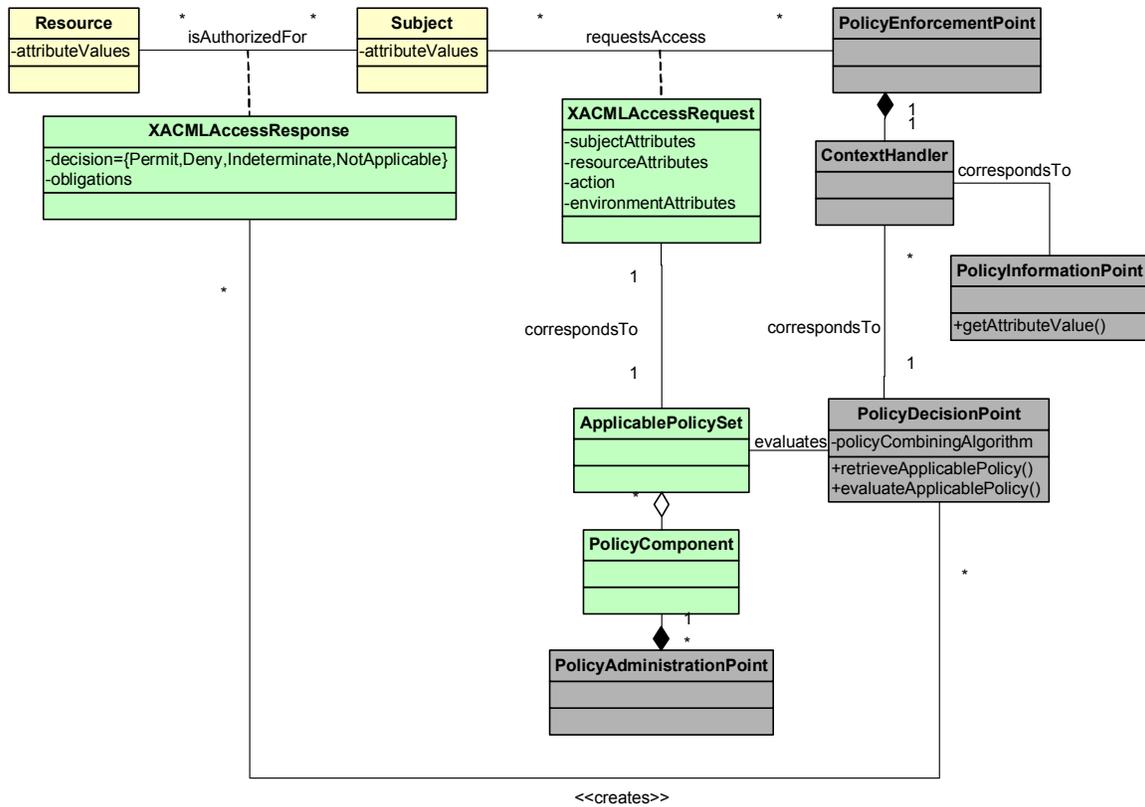
Figure 4:   Class diagram for the XACML access control evaluation

Dynamics

We describe the dynamic aspects of the XACML access control model using a sequence diagram for the use case "Control an access request to a resource".

Control an access request for a resource (Figure 5):

Summary: A Subject requests access to a resource. The access request is made through its PolicyEnforcementPoint, which in turn accesses the PolicyDecisionPoint through its ContextHanlder, in order to determine whether to accept or deny the request.

Actors: A Subject

Precondition: An existing PolicyAdministrationPoint must be accessible by the PolicyDecisionPoint. It contains policies defined by the organization.

Description:

a. A Subject sends a request for access to a Resource to its PolicyEnforcementPoint.

b. The PolicyEnforcementPoint sends the request to the ContextHandler in its native format.

c. The ContextHandler sends a corresponding XACML request to the PolicyDecisionPoint.
d. The PolicyDecisionPoint retrieves the ApplicablePolicy for this XACMLRequest from the PolicyAdministrationPoint.
e. The PolicyDecisionPoint may request additional attributes from the ContextHandler.
f. The ContextHandler obtains the attributes from a PolicyInformationPoint and returns them to the PolicyDecisionPoint.
g. The PolicyDecisionPoint evaluates the ApplicablePolicy corresponding to the XACMLRequest and returns an XACMLResponse to the ContextHandler or sends a request to the PolicyInforcementPoint if the attributes are not enough to.make a decision.
h. The ContextHandler translates the response to the native response format of the PolicyEnforcementPoint.
i. The PolicyEnforcementPoint fulfills the Obligations contained in the response.
j. If the access is permitted, the PolicyEnforcementPoint allows the requester to access the resource.

Alternate Flows:

If the XACMLAccessResponse's decision is 'Deny', the PolicyEnforcementPoint denies access to the resource.

If the XACMLAccessResponse's decision is 'Indeterminate' or 'NotApplicable', the decision has to be made by the PolicyEnforcementPoint.

Postcondition: Access control to a resource has been realized, based on the Subject's attributes, the Resource's attributes, the Environment's attributes, and an applicable policy.

The Appendix includes pseudo-code for the functions retrieveApplicablePolicy() and evaluateApplicablePolicy().

3.5 Implementation

To implement the XACML access control evaluation, the following tasks need to be performed:
1. Implement a ContextHandler for applications that already have a PolicyEnforcementPoint that use another access decision language
2. Implement an XACML PolicyEnforcementPoint for those applications that do not implement access control
3. Add the translated existing authorization rules to the PolicyAdministrationPoint
4. Add the new authorization rules to the PolicyAdministrationPoint

3.6 Consequences

The XACML access control pattern presents the following advantages:
• Since the access decisions are requested in a standard format, an access decision becomes independent from its enforcement. A broad variety of enforcement

mechanisms could be supported and can evolve separately from the PolicyDecisionPoint.

- This pattern can support the access matrix, RBAC or multilevel models for access control.

The pattern also has some (possible) liabilities:
- It is intrusive for existing applications that already have security, since they require the implementation of a ContextHandler.
- It could affect the performance of the protected system since XML is a verbose language.

3.7 Known Uses

This pattern is used in the commercial products mentioned in the previous pattern.

3.8 Example Resolved

The use of XACML Access Control allows the company to centralize the decisions of accesses to resources in the company. Consequently, applications do not need to care about access control decisions anymore. Every access request or response is in the XACML format.

3.9 Related Patterns

The Reference Monitor [Fer01] defines the security model for this pattern. It includes the Metadata-based Access control Model [Pri04]. The Application Firewall pattern [Del04] could be implemented according to the XACML patterns. This pattern uses the MBAC model [Pri04] as a component.
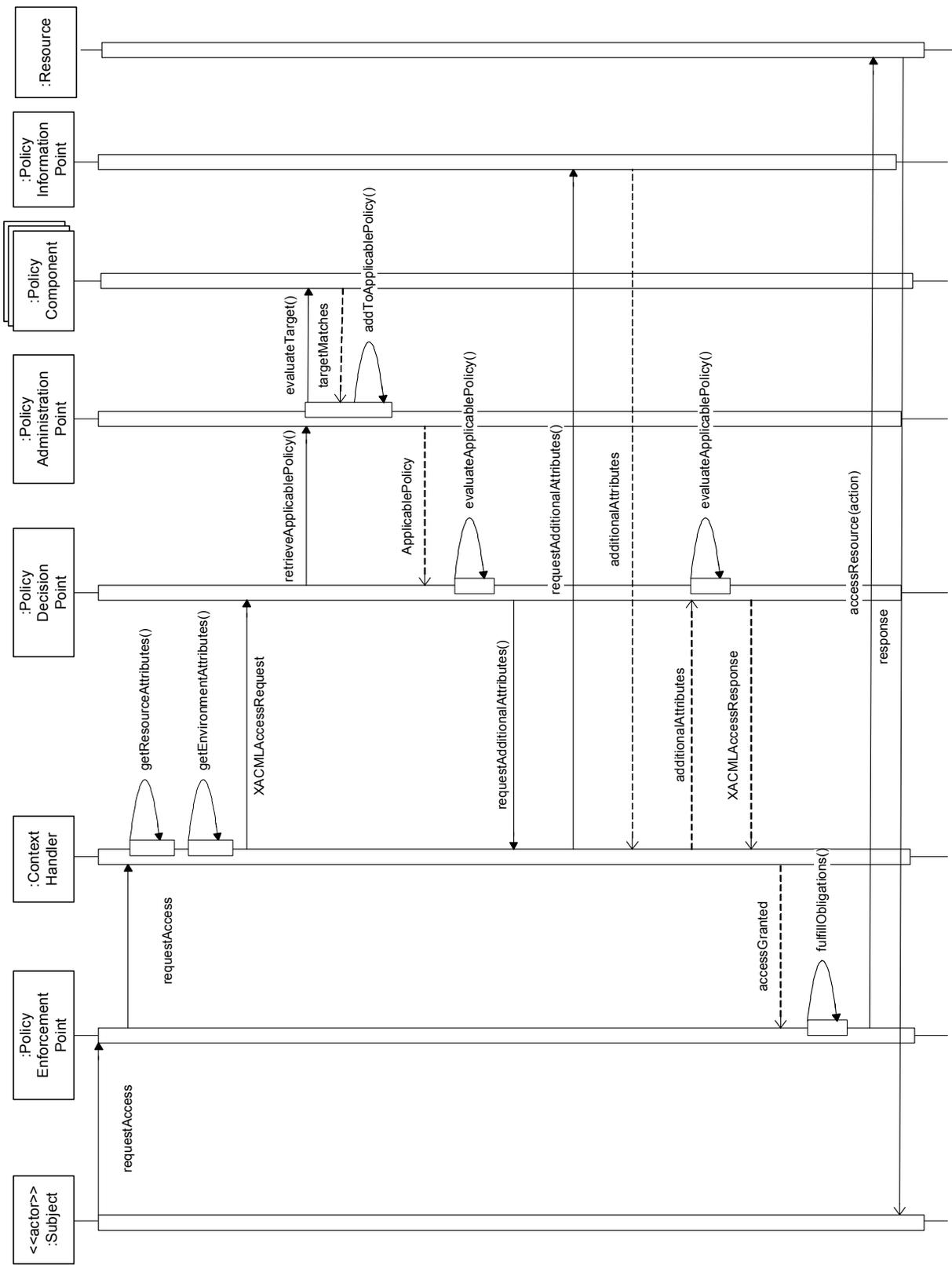
Figure 5: Sequence Diagram for controlling an access request for a resource

13

4 WSPL

WSPL enables an organization to represent access control policies for its web services in a standard manner. It also enables a web services consumer to express its requirements in a standard manner.

4.1 Example

Our company has a variety of web services for different purposes. Applications incorporate them as part of their structure. Application users pay for the use of these web services. If we want to make any money, we need to control access to them.

4.2 Context

Applications that use web services. Providers have security policies to control access to their web services, consumers have requirements for a web service invocation.

4.3 Problem

Web services are services that are accessible by means of messages sent using standard web protocols, notations and naming conventions [W3C]. In addition, they are self-describing through WSDL and can be discovered (maybe automatically discovered) using UDDI. Therefore, using different syntaxes for their policy descriptions would reduce these two properties.

Moreover, security policies are typically issued by different actors in different enterprise departments and the policies they write may concern a wide and overlapping set of web services. Applying the right policies to each access to a web service may also be complex, and thus error prone.

How do we describe policies to control web services invocations? The solution to this problem is constrained by the following forces:
- The policies are issued by a variety of actors of an organization and may be stored in many locations
- Web services consumers can also issue policies (requirements). For instance, a consumer could require a service to have a certificate from a well-known certification authority.
- Any type of security policy should be enforced.
- The policies are constantly changing and they need to be constantly updated
- We have a variety of subjects, e.g. roles.
- The environment in which the access is requested can also affect an access decision

- Some policies can require a set of obligations to be performed in conjunction with policy enforcement (auditing, …)

4.4 Solution

WSPL bind each WSDL web service component to an XACML component. In addition, define combination rules for such policies.

Structure

Figure 6 describes the structure of this pattern.

Each WSDL's web service component, **Endpoint** (port), **Message**, and **Operation**, involves several **Aspect**s, such as reliable messaging, privacy, authorization, trust, authentication, or cryptographic security. Each of the web service components respectively corresponds to an **EndpointPolicy**, **MessagePolicy**, and **OperationPolicy** and are described by XACML **PolicySet**s.

An **EndpointPolicy**, **MessagePolicy**, or **OperationPolicy** consists of **Objective**s that govern an aspect of the web service components. All **Objective**s must be achieved by the service invocation. An **Objective** is defined by an XACML **Policy**.

Each **Objective** consists of a set of ordered **Strategi**es. At least one **Strategy** must be achieved for its **Objective** to be achieved. This ordering may enable to perform functions such as policy or trust negotiation. **Strategi**es are represented by XACML **Rule**s.

An XACML **Attribute** concept is refined as an **UnconstrainedAttribute** can have its value assigned by the policy-user, whereas a **ConstrainedAttribute** cannot. An **AuthorizedAttribute** must have its value assigned by an authority.
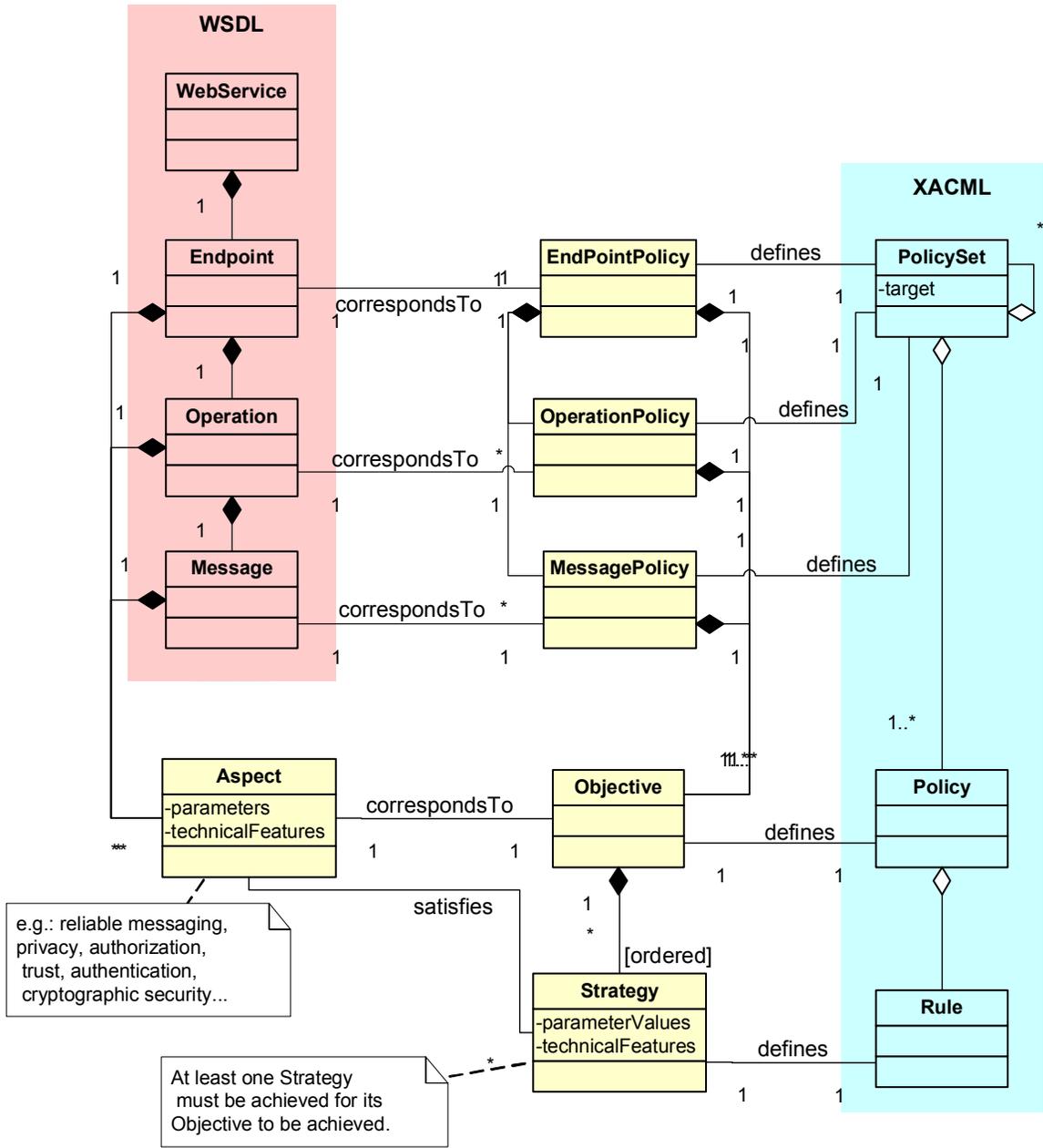
Figure 6: Class diagram for the WSPL pattern

4.5 Example Resolved

Using WSPL we can define precise rules about who can access which resources and in what way. We can then provide security to our users and prevent users who do not pay for using our services.

4.6 Consequences

In addition to the advantages of the XACML pattern, the WSPL pattern presents the following advantage:

- Consumers and providers 's policies can be combined to decide how a service invocation should occur.

The pattern also has some (possible) liabilities:

- It is intrusive for existing web services that already implement security, since they require to implement a ContextHandler.
- It could affect the performance of the protected system as XML since a verbose language.

4.7 Known Uses

OpenWSPL is an open source Java implementation of the Web-Service Policy language [WSPL].

4.8 Related Patterns

WSPL defines a type of Adaptor [Gam94] between WSDL and XACML.

The architecture defined by the XML Firewall pattern [Del04] could be implemented using this pattern.

Acknowledgements

References

[Dat05]   DataPower, http:///www.datapower.com

[Del04]   N. Delessy-Gassant, E.B.Fernandez, S. Rajput, and M.M.Larrondo-Petrie, "Patterns for application firewalls", Procs. of PLoP 2004.

[Fer01a]   E. B. Fernandez and R. Pan, "A Pattern Language for security models", *Proc. of PLoP 2001,* http://jerry.cs.uiuc.edu/~plop/plop2001/accepted_submissions

[Fer06]   E.B.Fernandez, E. Gudes, and M. Olivier, *The design of secure systems*, under contract with Addison-Wesley.

[Fer05]   E.B.Fernandez, T. Sorgente, M. M. Larrondo-Petrie, and N. Delessy, "Web services security: Standards, industrial practice, and research issues", submitted for publication.

[Gam94]   E. Gamma, R. Helm, R. Johnson, J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software,* Addison-Wesley, Boston, Mass., 1994.

[OAS]   http://www.oasis-open.org/home/index.php

[Par05]   http://www.parthcomp.com/

[Pri04]   T. Priebe, E. B. Fernandez, J. I. Mehlau, and G. Pernul, "A pattern system for access control ", in *Research Directions in Data and Applications Security XVIII*, C. Farkas and P. Samarati (Eds.), Proc. of the 18th. Annual IFIP WG 11.3 Working Conference on Data and Applications Security, Sitges, Spain, July 25-28, 2004.

[Sun04]   http://sunxacml.sourceforge.net/

[W3C]   http://www.w3.org/2003/glossary/subglossary/xkms2-req

[WSPL]   http://sourceforge.net/projects/openwspl/

[XAC04]   XACML – http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml

[Xtr05]   Xtradyne technologies, http://www.xtradyne.com/

5 Appendix

5.1 *Pseudocode for retrieveApplicablePolicy()*

```
retrieveApplicablePolicy(XACMLAccessRequest){
     FOR EACH PolicyComponent Є PolicyAdministrationPoint
          evaluateTarget(XACMLAccessRequest, PolicyComponent)
          IF targetMatches
          THEN add PolicyComponent to ApplicablePolicy
}


evaluateTarget(XACMLAccessRequest, PolicyComponent){
          IF SubjectsMatch() AND
```

```
                ResourcesMatch() AND
                ActionsMatch() AND
                EnvironmentMatch()
          THEN
                targetMatches
}

SubjectsMatch(XACMLAccessRequest, PolicyComponent){//at least one
                                            //subject matches
          FOR EACH SubjectDescriptor ∈
          PolicyComponent.Target.SubjectDescriptors
                IF SubjectMatches() RETURN true
          RETURN false
}

SubjectMatches(XACMLAccessRequest, PolicyComponent){//all qualifiers
                                            //match
          FOR EACH SubjectAttributeQualifier ∈ SubjectDescriptor
                IF ! SubjectAttributeQualifier.operator(
          SubjectAttributeQualifier .value,
          XACMLAccessRequest.SubjectAttributeValue)
                      RETURN false
          RETURN true
}
```

5.2   *Pseudocode for evaluateApplicablePolicy:*

```
evaluateApplicablePolicy(ApplicablePolicy, XACMLAccessRequest){
          FOR EACH PolicyComponent p ∈ ApplicablePolicy
                DepthFirstSearch(p)
          RETURN PolicyDecisionPoint.policyCombiningAlgorithm()
}


depthFirstSearch(PolicyComponent p){
          FOR EACH PolicyComponent or Rule x ∈ p
                IF x is a Rule
                      evaluateRule(x)
                ELSE
                      depthFirstSearch(x)
                      p.result = p.combiningAlgorithm()
}


evaluateRule(Rule x){
          IF evaluate(Rule.condition)
                RETURN x.result = x.effect
          ELSE RETURN x.result = NotDeterminate
}
```