



Diploma Thesis
HERAS^{AF}: XACML 2.0 Implementation
Developer's Guide

Department of Computer Sciences
University of Applied Sciences Rapperswil

Fall semester 2007

| | |
|-----------------------|--|
| Students: | Sacha Dolski, Florian Huonder, Stefan Oberholzer |
| Responsible lecturer: | Wolfgang Giersche |
| Person in support: | René Eggenschwiler |



| | | |
|--|--|------------------|
| <u>PART I: INTRODUCTION</u> | | <u>5</u> |
| 1 | AUDIENCE..... | 6 |
| 2 | OVERVIEW | 7 |
| 2.1 | DEFINITION | 7 |
| 2.2 | SHORT INTRODUCTION TO XACML [XACMLBI]..... | 8 |
| 2.3 | HERAS ^{AF} | 13 |
| 2.4 | REQUIREMENTS TO A PDP | 14 |
| 2.5 | MODULE DESIGN..... | 15 |
| <u>PART II: CONFIGURATION.....</u> | | <u>16</u> |
| 1 | OVERVIEW | 17 |
| 2 | LOCATOR | 20 |
| 3 | PERSISTENCEMANAGER | 22 |
| 4 | FUNCTIONS..... | 24 |
| 5 | DATATYPEATTRIBUTES..... | 27 |
| 6 | CONTEXTANDPOLICYCONFIGURATION | 29 |
| 7 | COMBININGALGORITHMS..... | 31 |
| 8 | ADVANCED CONFIGURATION OPTIONS | 32 |
| 8.1 | CONFIGURATION FILES..... | 33 |
| 8.2 | PDP | 34 |
| 8.3 | INDEXHANDLER | 35 |
| 8.4 | REFERENCELOADER..... | 36 |
| 8.5 | ATTRIBUTEFINDER..... | 37 |
| 8.6 | TARGETMATCHER..... | 38 |
| <u>PART III: IMPLEMENTING FUNCTIONS, COMBININGALGORITHMS AND DATATYPES.....</u> | | <u>39</u> |
| 1 | OVERVIEW | 40 |
| 2 | IMPLEMENTING FUNCTIONS | 40 |
| 3 | IMPLEMENTING DATATYPES..... | 41 |
| 4 | IMPLEMENTING COMBININGALGORITHMS | 42 |
| <u>PART IV: ARCHITECTURE, EXTENDING AND ADAPTING.....</u> | | <u>44</u> |
| 1 | OVERVIEW | 45 |
| 1.1 | ARCHITECTURE | 45 |
| 1.1.1 | Modules | 45 |
| 1.1.2 | Dataflow..... | 46 |
| 1.1.3 | Evaluation Sequence..... | 48 |
| 1.1.4 | Concurrency..... | 51 |
| 1.1.5 | Design decisions | 52 |
| 2 | CORE | 54 |
| 2.1 | INTERFACES | 54 |
| 2.2 | HERAS ^{AF} XACML 2.0 IMPLEMENTATION | 55 |
| 2.2.1 | Package policy | 57 |
| 2.2.2 | Package attributeFinder | 59 |



| | | |
|---|--|-------------------|
| 2.2.3 | Package context | 60 |
| 2.2.4 | Package converter | 62 |
| 2.2.5 | Package function | 63 |
| 2.2.6 | Package combiningAlgorithm..... | 64 |
| 2.2.7 | Package dataTypeAttribute | 65 |
| 2.2.8 | Package types..... | 66 |
| 2.2.9 | Creating request- and response contexts..... | 66 |
| 2.2.10 | Marshalling and Unmarshalling..... | 69 |
| 2.2.11 | Design Decisions | 70 |
| 3 | PDP..... | 71 |
| 3.1 | INTERFACES | 71 |
| 3.2 | HERAS ^{AF} XACML 2.0 IMPLEMENTATION | 73 |
| 3.3 | SEQUENCE DIAGRAMS..... | 75 |
| 4 | PERSISTENCE | 77 |
| 4.1 | INTERFACES | 77 |
| 4.2 | HERAS ^{AF} XACML 2.0 IMPLEMENTATION | 78 |
| 4.2.1 | Sequence diagram | 78 |
| 4.2.2 | Class diagram..... | 79 |
| 4.2.3 | Design decisions | 79 |
| 5 | LOCATOR | 80 |
| 5.1 | INTERFACES | 80 |
| 5.2 | HERAS ^{AF} XACML 2.0 IMPLEMENTATION | 82 |
| 5.2.1 | Sequence diagrams..... | 82 |
| 5.2.2 | Class diagram..... | 85 |
| 5.2.3 | Indexing algorithm..... | 87 |
| 5.2.4 | Design decisions | 106 |
| 6 | REFERENCELOADER | 112 |
| 6.1 | INTERFACES | 112 |
| 6.2 | HERAS ^{AF} XACML 2.0 IMPLEMENTATION | 112 |
| <u>PART V: DEPLOYMENT</u> | | <u>113</u> |
| 1 | OVERVIEW | 114 |
| 2 | APACHE MAVEN 2..... | 114 |
| 3 | HERAS-PDP-ROOT..... | 118 |
| <u>APPENDIX A: OVERALL</u> | | <u>119</u> |
| 1 | REFERENCES..... | 120 |
| 2 | GLOSSARY | 122 |
| <u>APPENDIX B: EXTENSION POINTS AND REQUIRED REFACTORINGS.....</u> | | <u>123</u> |
| 1 | OVERVIEW | 124 |
| 2 | EXTENSION POINTS | 124 |
| 3 | REFACTORINGS..... | 125 |
| <u>APPENDIX C: KNOWN ISSUES</u> | | <u>127</u> |
| 1 | OVERVIEW | 128 |



| | | |
|----------|------------------------|------------|
| 2 | TIME ZONE | 128 |
| 2.1 | ISSUE | 128 |
| 2.2 | WORKAROUND | 128 |

APPENDIX D: DISCREPANCIES TO THE XACML 2.0 SPECIFICATION..... 129

| | | |
|----------|--|------------|
| 1 | OVERVIEW | 130 |
| 2 | ANYOFALL / ALLOFANY | 130 |
| 3 | DAYTIMEDURATION | 131 |
| 4 | UNIQUE ID OVER POLICY AND POLICYSET | 132 |
| 5 | COMBINERPARAMETERS | 132 |



Part I: Introduction



1 Audience

The parts Introduction and Configuration of this Developer's Guide provide information for developers who want to use the HERAS^{AF} XACML 2.0 implementation. It contains information about the XACML 2.0 basics, the API and the configuration of the HERAS^{AF} XACML 2.0 implementation.

The parts ImplementingFunctions, CombiningAlgorithms and DataTypes as well as the part Architecture, Extending and Adapting and the part Deployment are written for developers who have read the XACML 2.0 specification and want to extend the HERAS^{AF} XACML 2.0 implementation or write modules for it. It contains information about packages, interfaces, classes and flows. It also provides an insight into the developer's train of thoughts of the HERAS^{AF} XACML 2.0 implementation.

In the Appendix there are Extension Points and Required Refactorings, Known Issues and Discrepancies to the XACML 2.0 Specification. These parts are for developers who want to improve the HERAS^{AF} XACML 2.0 implementation.



2 Overview

2.1 Definition

| | |
|---------------------------------------|--|
| <i>What is HERAS^{AF}</i> | <p>HERAS^{AF} is an Open Source project in its formation phase.</p> <p>It assists the entire authorization process. That means all accesses to protected resources are intercepted by a PEP and sent to a PDP, which evaluates the request on the basis of applicable Evaluatables. In case of a positive answer the PEP allows the access to the resource. Additionally the maintenance is guaranteed through a PAP.</p> |
| <i>What is HERAS^{AF} PDP</i> | <p>HERAS^{AF} PDP implements the entire logic for decision making of accesses. This covers fast locating of potential policies, evaluation of a request with the aid of the located policies as well as combining the obtained results.</p> <p>The main focus of the HERAS^{AF} PDP is performance. To achieve this, performance-enhancing mechanisms such as indexing and fast comparison algorithms are developed. Another essential point is accessing the data sink only during the initialization phase of the PDP. Afterwards the policies are kept in the RAM.</p> <p>The HERAS^{AF} PDP will be used as a central unit, which interacts with several PEP's.</p> <p>[HPDPWSEUG] [HPDPWSEDG]</p> |
| <i>What is HERAS^{AF} PAP</i> | <p>The HERAS^{AF} PAP is responsible to build and administrate policies. Additionally it deploys the policies to the PDP's. For the building process of complex policies a graphical user interface is in development. For further information to this component see the paper HERAS^{AF} PAP. [HPAP]</p> |
| <i>What is HERAS^{AF} PEP</i> | <p>The HERAS^{AF} PEP provides mechanisms to implement intercepting agent's which enforces access control in protection worth application. Its main responsibility is building authorization requests and interpreting authorization responses. For further information see the paper HERAS^{AF}: Interzeptoren für Spring AOP und AspectJ.</p> <p>[HPEPUG] [HPEPDG]</p> |
| <i>What is HERAS^{AF} PIP</i> | <p>The HERAS^{AF} PIP is responsible to resolve missing attributes of the request. If a HERAS^{AF} PDP needs further information to evaluate a request, the HERAS^{AF} PIP is called. If possible the HERAS^{AF} PIP returns additional attributes.</p> |
| <i>Evaluatable</i> | <p>Evaluatable is an umbrella term for a Policy or a PolicySet. This term is used in the paper every time a description means a Policy or a PolicySet.</p> |
| <i>Converter</i> | <p>The Converters are connectors between the URN representation of a Function, Datatype or CombiningAlgorithm and their proper implementation classes. Every time a XACML-request, -response or —policy will be marshaled or unmarshaled they become effective.</p> |



2.2 Short introduction to XACML [XACMLBI]

Overview

XACML is an OASIS standard that describes both a policy language and an access control decision request/response language (both written in XML). The policy language is used to describe general access control requirements, and has standard extension points for defining new functions, data types, combining logic, etc. The request/response language lets you form a query to ask whether or not a given action should be allowed, and interpret the result. The response always includes an answer about whether the request should be allowed using one of four values: Permit, Deny, Indeterminate (an error occurred or some required value was missing, so a decision cannot be made) or Not Applicable (the request can't be answered by this service).

The typical setup is that someone wants to take some action on a resource. They will make a request to whatever actually protects that resource (like a file system or a web server), which is called a Policy Enforcement Point (PEP). The PEP will form a request based on the requester's attributes, the resource in question, the action, and other information pertaining to the request. The PEP will then send this request to a Policy Decision Point (PDP), which will look at the request and some policy that applies to the request, and come up with an answer about whether access should be granted. That answer is returned to the PEP, which can then allow or deny access to the requester. Note that the PEP and PDP might both be contained within a single application, or might be distributed across several servers. In addition to providing request/response and policy languages, XACML also provides the other pieces of this relationship, namely finding a policy that applies to a given request and evaluating the request against that policy to come up with a yes or no answer.

There are many existing proprietary and application-specific languages for doing this kind of thing but XACML has several points in its favor:

- It's standard. By using a standard language, you're using something that has been reviewed by a large community of experts and users, you don't need to roll your own system each time, and you don't need to think about all the tricky issues involved in designing a new language. Plus, as XACML becomes more widely deployed, it will be easier to interoperate with other applications using the same standard language.
- It's generic. This means that rather than trying to provide access control for a particular environment or a specific kind of resource, it can be used in any environment. One policy can be written which can then be used by many different kinds of applications, and when one common language is used, policy management becomes much easier.
- It's distributed. This means that a policy can be written which in turn refers to other policies kept in arbitrary locations. The result is that rather than having to manage a single monolithic policy, different people or groups can manage sub-pieces of policies as appropriate, and XACML knows how to correctly combine the results from these different policies into one decision.
- It's powerful. While there are many ways the base language can be extended, many environments will not need to do so. The standard language already supports a wide variety of data types, functions, and rules about combining the results of different policies. In addition to this, there are already standards groups working on extensions and profiles that will hook XACML into other standards like SAML and LDAP, which will increase the number of ways that XACML can be used.

To give you a better idea of how all these aspects fit together, what follows is a discussion of XACML policy, which will demonstrate many of the standard features of the language. Note that XACML is a rich language, so only some of



its features are shown here. You should look at the specification for more information on all of the features.

Policy and PolicySet

At the root of all XACML policies is a Policy or a PolicySet. A PolicySet is a container that can hold other Policies or PolicySets, as well as references to policies found in remote locations. A Policy represents a single access control policy, expressed through a set of Rules. Each XACML policy document contains exactly one Policy or PolicySet root XML tag. Because a Policy or PolicySet may contain multiple policies or Rules, each of which may evaluate to different access control decisions, XACML needs some way of reconciling the decisions each makes. This is done through a collection of Combining Algorithms. Each algorithm represents a different way of combining multiple decisions into a single decision. There are Policy Combining Algorithms (used by PolicySet) and Rule Combining Algorithms (used by Policy). An example of these is the Deny Overrides Algorithm, which says that no matter what, if any evaluation returns Deny, or no evaluation permits, then the final result is also Deny. These Combining Algorithms are used to build up increasingly complex policies, and while there are seven standard algorithms, you can build your own to suit your needs.

Targets and Rules

Part of what an XACML PDP needs to do is find a policy that applies to a given request. To do this, XACML provides another feature called a Target. A Target is basically a set of simplified conditions for the Subject, Resource and Action that must be met for a PolicySet, Policy or Rule to apply to a given request. These use Boolean functions (explained more in the next section) to compare values found in a request with those included in the Target. If all the conditions of a Target are met, then its associated PolicySet, Policy, or Rule applies to the request. In addition to being a way to check applicability, Target information also provides a way to index policies, which is useful if you need to store many policies and then quickly sift through them to find which ones apply. For instance, a Policy might contain a Target that only applies to requests on a specific service. When a request to access that service arrives, the PDP will know where to look for policies that might apply to this request because the policies are indexed based on their Target constraints. Note that a Target may also specify that it applies to any request. Once a Policy has been found and verified to apply to a request, its Rules are evaluated. A policy can have any number of Rules which contain the core logic of an XACML policy. The heart of most Rules is a Condition, which is a Boolean function. If the Condition evaluates to true, then the Rule's Effect (a value of Permit or Deny that is associated with successful evaluation of the Rule) is returned. Evaluation of a Condition can also result in an error (Indeterminate) or discovery that the Condition does not apply to the request (NotApplicable). A Condition can be quite complex, built from an arbitrary nesting of non-Boolean functions and attributes.

Attributes, Attribute Values, and Functions

The currency that XACML deals in is attributes. Attributes are named values of known types that may include an issuer identifier or an issue date and time. Specifically, attributes are characteristics of the Subject, Resource, Action, or Environment in which the access request is made. A user's name, their security clearance, the file they want to access, and the time of day are all attribute values. When a request is sent from a PEP to a PDP, that request is formed almost exclusively of attributes, and they will be compared to attribute values in a policy to make the access decisions. A Policy resolves attribute values from a request or from some other source through two mechanisms: the AttributeDesignator and the AttributeSelector. An AttributeDesignator lets the policy specify an attribute with a given name and



type, and optionally an issuer as well, and then the PDP will look for that value in the request, or elsewhere if no matching values can be found in the request. There are four kinds of designators, one for each of the types of attributes in a request: Subject, Resource, Action, and Environment. Because Subject attributes can be broken into different categories, SubjectAttributeDesignators can also specify a category to look in. AttributeSelectors allow a policy to look for attribute values through an Xpath query. A data type and an Xpath expression are provided, and these can be used to resolve some set of values either in the request document or elsewhere.

Both the AttributeDesignator and the AttributeSelector can return multiple values (since there might be multiple matches in a request or elsewhere), so XACML provides a special attribute type called a Bag. Bags are unordered collections that allow duplicates, and are always what designators and selectors return, even if only one value was matched. In the case that no matches were made, an empty bag is returned, although a designator or selector may set a flag that causes an error instead in this case.

Once some Bag of attribute values has been retrieved, they need to be compared in some way to expected values to make access decisions. This is done through a powerful system of functions. Functions can work on any combination of attribute values, and can return any kind of attribute value supported in the system. Functions can also be nested, so you can have functions that operate on the output of other functions, and this hierarchy can be arbitrarily complex. Custom functions can be written to provide an ever richer language for expressing access conditions.

One thing to note when building these hierarchies of functions is that most functions are defined as working on specific types (like strings or integers) while designators and selectors always return Bags of values. To handle this, XACML defines a collection of standard functions of the form [type]-one-and-only, which accept a bag of values of the specified type and return the single value if there is exactly one item in the bag, or an error if there are zero or multiple values in the bag. This is one of the most common functions that you will see in a Condition. [type]-one-and-only functions are not needed in Targets, however, since the PDP automatically applies the matching function to each element of a bag.

Example Policy

Here is a simple example Policy using those features discussed above. Its Target says that the Policy only applies to requests for the server called "SampleServer". The Policy has a Rule with a Target that requires an action of "login" and a Condition that applies only if the Subject is trying to log in between 9am and 5pm. Note that this example can be extended to include other Rules for different actions. If the first Rule provided here doesn't apply, then a default Rule is used that always returns Deny (Rules are evaluated in order).

```
<Policy PolicyId="SamplePolicy"
  RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-
  combining-algorithm:permit-overrides">
  <!--This Policy only applies to requests on the
  SampleServer →
  <Target>
    <Subjects>
      <AnySubject/>
    </Subjects>
```



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

<!--Rule to see if we should allow the Subject to login
→
<Rule RuleId="LoginRule" Effect="Permit">

  <!--Only use this Rule if the action is login →
<Target>
  <Subjects>
    <AnySubject/>
  </Subjects>
  <Resources>
    <AnyResource/>
  </Resources>
  <Actions>
    <ActionMatch
MatchId="urn:oasis:names:tc:xacml:1.0:function:string-
equal">
      <AttributeValue
DataType= »http ://www.w3.org/2001/XMLSchema#string »>login
</AttributeValue>
      <ActionAttributeDesignator
DataType="http://www.w3.org/2001/XMLSchema#string"
AttributeId="ServerAction"/>
    </ActionMatch>
  </Actions>
</Target>
  <!--Only allow logins from 9am to 5pm →
  <Condition
FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">
    <Apply
FunctionId="urn:oasis:names:tc:xacml:1.0:function:time-
greater-than-or-equal"
      <Apply
FunctionId="urn:oasis:names:tc:xacml:1.0:function:time-one-
and-only">
        <EnvironmentAttributeSelector
DataType= »http ://www.w3.org/2001/XMLSchema#time »
AttributeId="urn:oasis:names:tc:xacml:1.0:environment:curre
nt-time"/>
```



```
</Apply>
  <AttributeValue DataType="http://www.w3.org/
2001/XMLSchema#time">09:00:00</AttributeValue>
</Apply>
<Apply
FunctionId="urn:oasis:names:tc:xacml:1.0:function:time-
less-than-or-equal"
  <Apply
FunctionId="urn:oasis:names:tc:xacml:1.0:function:time-one-
and-only">
  <EnvironmentAttributeSelector
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">17:00:00</AttributeValue>
  </Apply>
</Condition>

</Rule>

<!--We could include other Rules for different actions
here →

<!--A final, "fall-through" Rule that always Denies →
<Rule RuleId="FinalRule" Effect="Deny"/>

</Policy>
```



2.3 HERAS^{AF}

General note This overview is skimped. See [HXI20] for further information.

Overall HERAS^{AF} is an Open Source project in its formation phase. The project was initialized in 2006, after an eight month conception and project planning phase. The brainchild HERAS^{AF} was born in collaboration with Wolfgang Giersche, Yan Graf and René Eggenschwiler. The modules PEP and PAP were developed by Massimo Cerqui and Sandro Strebel. The PDP Web service endpoint as well as the XACML 2.0 implementation was developed by Sacha Dolski, Florian Huonder and Stefan Oberholzer.

Purpose HERAS^{AF} assists the entire authorization process. That means all accesses to protected resources are intercepted by a PEP and sent to a PDP, which evaluates the request on the basis of applicable Evaluatables. In case of a positive answer the PEP allows the access to the resource. Additionally the maintenance is guaranteed through a PAP.

Architecture

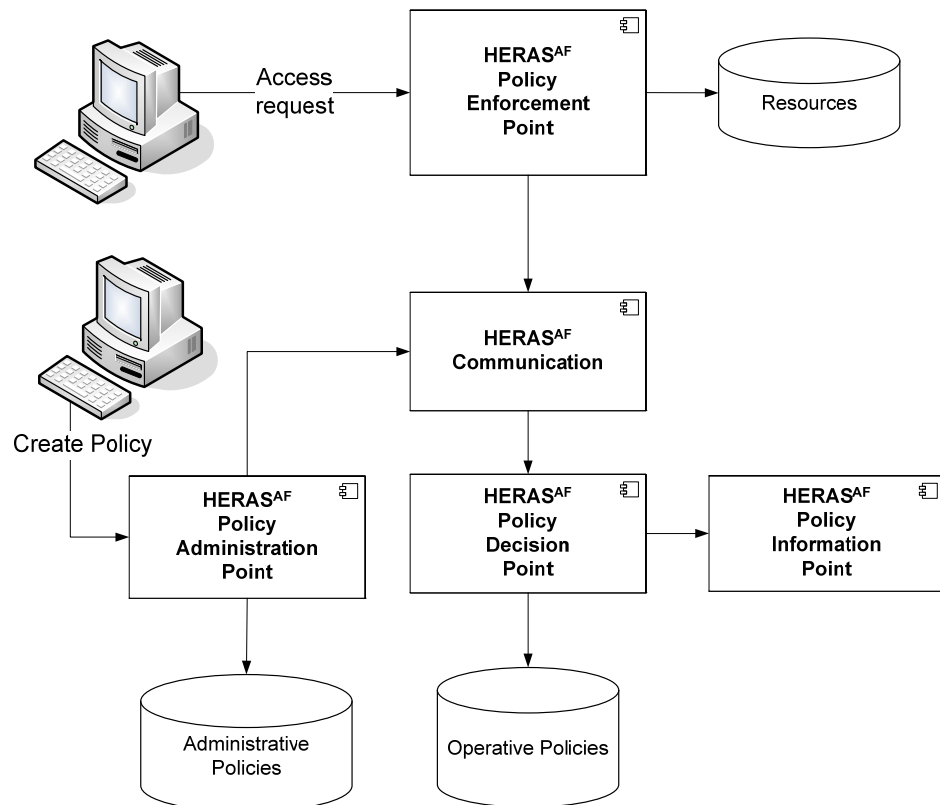


Figure 1: Architecture of HERAS^{AF}

Policy Enforcement Point

The system entity that performs access control, by making decision requests and enforcing authorization decisions.

Policy Administration Point

The system entity that administrates the Evaluatables and deploys them on the PDP.



Policy Decision Point

The system entity that evaluates applicable Evaluatables and renders an authorization decision.

Policy Information Point

The system entity that acts as a source of attribute values.

2.4 Requirements to a PDP

Introduction

There are several requirements to an implementation of a Policy Decision Point. This chapter lists the most important requirements and gives a little explanation on these requirements.

The different requirements are listed below:

Requirements

Requirements

| | |
|--------------------------|---|
| <i>Performance</i> | For a PDP the performance is an essential requirement. A client enforcing a request is blocked until it gets the answer. To reduce the response time, the Evaluatables should be indexed to ensure that the request is only evaluated against potential Evaluatables. |
| <i>Concurrency</i> | A PDP may have to answer hundreds of requests at the same time. To ensure a high performance this should happen concurrently, which means it is essential that an evaluation process does not influence another one. |
| <i>Reliability</i> | Requests containing the same information can be written in different ways. It has to be guaranteed that the answer is always the same, whatever the form of the request might be. |
| <i>Consistency</i> | During the evaluation of a request a Evaluatable with the same identifier must always lead to the same answer. This means, if a local- or remote Evaluatable reference is used several times, it is not allowed that the Evaluatable changes during a request evaluation. |
| <i>Failure tolerance</i> | An incorrect request or Evaluatable must not lead to a crash of the PDP. |



2.5 Module Design

Overview

The HERAS^{AF} implementation is divided up in several modules. Each module is implemented in its own package and contains interfaces. In this chapter the different modules are discussed. The modules separate the framework into logical units with concrete responsibilities. For example the Persistence module is in charge of persisting the Evaluatables.

Module Overview

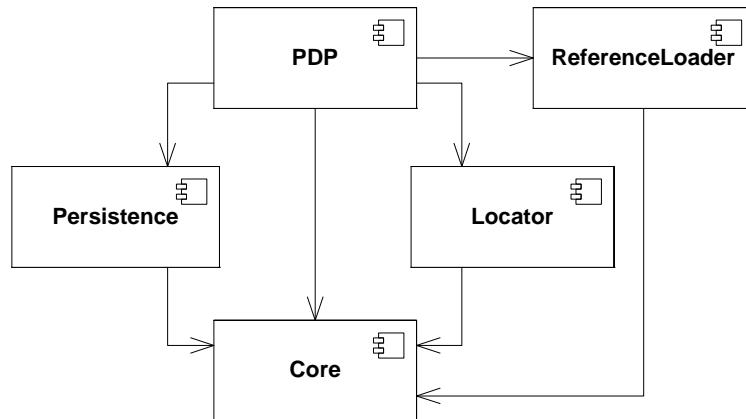


Figure 2: The modules and their dependencies

Module Core

The Core module contains the data classes for Evaluatables, requests and responses. For every specified XML complex type in the XACML specification there is one class.

Module PDP

The PDP module contains the PDP implementation which depends on the other modules.

Module Persistence

The Persistence module contains the logic to persist the Evaluatables to any sort of data store.

Module Locator

The Locator module keeps the loaded Evaluatables in the memory and indexes them.

Module Reference Loader

The ReferenceLoader is responsible for resolving references to local and remote Evaluatables.



Part II: Configuration



1 Overview

Introduction

Most of the modules of the HERAS^{AF} XACML 2.0 implementation are configurable. The HERAS^{AF} XACML 2.0 implementation is configured through the Spring framework application context. Configurability is a very important point in the face of adapting the HERAS^{AF} XACML 2.0 implementation to its destination environment. It opens a wide application area without changing a single line of code. This chapter explains what can be configured and how it is done.

Configuration topics

Following is a listing of all configurable parts subdivided by the various topics of the HERAS^{AF} XACML 2.0 implementation.

Indexing

The indexing is configured in the Locator configuration file. Further information see section “Various configuration files”.

Database access

The database access is configured in the Persistence Manager configuration file. Further information see section “Various configuration files”.

Functions

The functions are configured in the Functions configuration file. Further information see section “Various configuration files”.

Data Type Attributes

The data type attributes are configured in the Data Type Attributes configuration file. Further information see section “Various configuration files”.

Request context marshalling / unmarshalling

The request context marshalling and unmarshalling is configured in the ContextAndPolicyConfiguration configuration file. Further information see section “Various configuration files”.

Response context marshalling / unmarshalling

The response context marshalling and unmarshalling is configured in the ContextAndPolicyConfiguration configuration file. Further information see section “Various configuration files”.

Policy marshalling / unmarshalling

The policy marshalling and unmarshalling is configured in the ContextAndPolicyConfiguration configuration file. Further information see section “Various configuration files”.



Various configuration files

In the following all configurable parts subdivided by the various configuration files are listed.

Locator

Configuration possibility for the index in the locator.
It is possible to configure the elements' subject, resource, action and environment itself. This means for every element the indexed attribute id, data type and issuer are selectable. Therefore the maximum of performance while obtaining Evaluatables can be reached because the index is adaptable to the specific requirements.

Persistence Manager

Configuration possibility for the persistence layer.
The database is fully exchangeable what means that the driver, the schema path and the user information can be configured. Further the possibility for configuring the SQL statements with placeholders exists that enables the persistence layer to adapt completely to the specific requirements.

Functions

Adding additional or overridden functions to the PDP.
The PDP loads an implementation of all functions defined by the XACML specification. To override a function simply add a function with the same id to the configuration file. Additionally add own functions to the PDP.

Data Type Attributes

Adding additional or overridden data type attributes to the PDP.
The PDP loads an implementation of all data type attributes defined by the XACML specification. To override a data type attribute simply add a data type attribute with the same id to the configuration file. Additionally add own data type attributes to the PDP.

Context And Policy Configuration

Configuration possibility for validating and formatting of input and output XML.
This configuration affects request and response context and policy marshalling or unmarshalling.

Combining Algorithms

Adding various combining algorithms to the PDP. The XACML 2.0 combining algorithms are included by default in the application context.

Advanced configuration possibilities

There are further configuration possibilities. These configurations need advanced knowledge and should only be done with a cogent reason.
It is possible to configure the PDP itself, the embedding of the configuration files, the index handler, the attribute finder and the target matcher.



*Extensible XML
Authoring*

The configuration of the XACML 2.0 implementation is using the possibility for Extensible XML Authoring of the Spring framework [SEXMLA] since version 2.0. It provides the capability of writing an own schema for configuring beans. It is an extension of the basic Spring XML format

Schema

A XML schema must be written that describes the custom beans.

Coding

A custom NamespaceHandler must be written. It is responsible for mapping the bean definition with the appropriate parser.

A BeanDefinitionParser must be written which does the real work of parsing the custom bean definition.

Registering

The above artifacts must be registered with the Spring framework.

*Configuration
files*

The configuration of the HERAS^{AF} XACML 2.0 implementation is divided into several significant configuration files. Each configurable part of the implementation is described in detail in the next chapters.



2 Locator

Overview

This chapter explains the configuration of the indexing in the Locator module. Because of this the configuration of the indexing is done in the Locator.xml. It contains information about the indexable attribute id, data type and issuer as well as the index type. This information is configurable for each one these elements: subject, resource, action and environment.

The index is configurable by reason of its application area. Depending on the domain of the HERAS^{AF} XACML 2.0 implementation application, different attributes preponderate.

Configuration file

The configuration file is an XML file that fulfils the locator.xsd schema file. This schema file is located in the resources of the PDP module.

```

<?xml version="1.0" encoding="UTF-8"?>

<sb:beans xmlns="http://heras-af.org/pdp/namespace/locator"

xmlns:sb="http://www.springframework.org/schema/beans"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"

xsi:schemaLocation="http://www.springframework.org/
schema/beans http://www.springframework.org/schema/
beans/spring-beans-2.0.xsd
http://heras-af.org/pdp/namespace/locator http://heras-
af.org/pdp/namespace/locator/Locator.xsd">

    <locator id="locator">
        <subject>
            <attributeId>...</attributeId>
            <dataType>... </dataType>
            <issuer>...</issuer>
            <indexType>...</indexType>
        </subject>
        <resource>
            <attributeId>...</attributeId>
            <dataType>...</dataType>
            <issuer>...</issuer>
            <indexType>...</indexType>
        </resource>
        <action>
            <attributeId>... </attributeId>
            <dataType>...</dataType>
            <issuer>...</issuer>
            <indexType>...</indexType>
        </action>
        <environment>
            <attributeId>...</attributeId>
            <dataType>...</dataType>
            <issuer>...</issuer>
            <indexType>...</indexType>
        </environment>
    </locator>

</sb:beans>

```

Figure 3: Example of the Locator configuration.



Element description

| | |
|--------------------|---|
| <i>sb:beans</i> | <p>The root element containing the whole configuration of the Locator module.</p> <p>The namespaces declarations in this element must not be changed.</p> |
| <i>locator</i> | <p>Indicates the beginning of the locator-configuration.</p> <p>The <i>id</i> is the Spring-Id which is used to refer to the bean described here. This id ought not to be changed.</p> |
| <i>subject</i> | <p>Contains the configuration for the subject element.</p> |
| <i>resource</i> | <p>Contains the configuration for the resource element.</p> |
| <i>action</i> | <p>Contains the configuration for the action element.</p> |
| <i>environment</i> | <p>Contains the configuration for the environment element.</p> |
| <i>attributeID</i> | <p>Contains the attributeID which tells that the AttributeValue of this element is a candidate to index.</p> |
| <i>dataType</i> | <p>Contains the data type which tells that the AttributeValue of this element is a candidate to index.</p> <p>The default data types are defined by XACML 2.0.</p> |
| <i>issuer</i> | <p>Contains the issuer which tells that the AttributeValue of this element is a candidate to index.</p> <p>This element is optional.</p> |
| <i>indexType</i> | <p>Contains the index type which is used to index if attributeID, data type and issuer match.</p> <p>This implementation of the XACML 2.0 specification supports the following index types:</p> <ul style="list-style-type: none">• String_Forward, the regular expression <code>.*</code> can only appear at the end of a string.• String_Backward, the regular expression <code>.*</code> can only appear at the beginning of a string.• Comparable, only data types that implements the Comparable interface can be indexed. |



3 PersistenceManager

Overview

The PersistenceManager.xml describes the database parameters needed to connect to a database. Further on the SQL statements are in the file as well to allow a maximum of customization.

Configuring the Persistence layer allows the user of the HERAS^{AF} XACML 2.0 implementation to integrate the storing of the Evaluatables fully into his existing environment without a directive that says how the database must look like.

Configuration file

The configuration file is an XML file that fulfils the persistenceManager.xsd schema file. This schema is located in the resources of the PDP module.

structure

```
<?xml version="1.0" encoding="UTF-8"?>
<sb:beans xmlns="http://heras-af.org/pdp/namespace/persistence"
  xmlns:sb="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
http://heras-af.org/pdp/namespace/persistence http://heras-af.org/pdp/namespace/persistence/PersistenceManager.xsd">

  <persistenceManager id="persistenceManager">
    <database>
      <driver>...</driver>
      <schemaURL>...</schemaURL>
      <username>...</username>
      <password>...</password>
    </database>
    <evaluatables>
      <selectAll>...</selectAll>
      <select>...</select>
      <insert>...</insert>
      <deleteAll>...</deleteAll>
      <delete>...</delete>
    </evaluatables>
  </persistenceManager>
</sb:beans>
```

Figure 4: Example of the PersistenceManager configuration.

Element description

| | |
|---------------------------|---|
| <i>sb:beans</i> | The root element containing the whole configuration of the persistence module. The namespaces declarations in this element must not be changed. |
| <i>persistenceManager</i> | Indicates the beginning of the persistence-manager configuration. The <i>id</i> is the Spring-Id which is used to refer to the bean described here. This id ought not to be changed. |



| | |
|---------------------|---|
| <i>database</i> | Contains the configuration for accessing the database. |
| <i>evaluatables</i> | Contains the various statements used to access the database in the right way. |
| <i>driver</i> | Contains the location of the JDBC driver to use (Important: The JAR of the JDBC driver must be available in the classpath). |
| <i>schemaURL</i> | The URL to the database-schema. |
| <i>username</i> | The username of the user who has the access rights to the database. |
| <i>password</i> | The password of the user defined in the username section. |
| <i>selectAll</i> | Statement to get all policies or policy sets respectively. example: <pre>SELECT evaluatable FROM evaluatables;</pre> |
| <i>select</i> | Statement to get a policy or policy set respectively. Use the placeholder :id where the id of the evaluatable should be set while executing the command. example: <pre>SELECT evaluatable FROM evaluatables WHERE evaluatableId = :id;</pre> |
| <i>insert</i> | Statement to insert a policy or policy set respectively. Use the placeholder :id where the new id of the evaluatable should be placed in the statement and :eval where the evaluatable-BLOB should be placed. example: <pre>INSERT INTO evaluatables (evaluatableId, evaluatable) VALUES (:id, :eval);</pre> |
| <i>deleteAll</i> | Statement to delete all policies or policy sets respectively. example: <pre>DELETE FROM evaluatables;</pre> |



delete

Statement to delete a policy or policy set respectively.

Use the placeholder **:id** where the id of the evaluatable should be set while executing the command.

examples:

```
DELETE FROM evaluatables WHERE  
evaluatableId = :id;
```

4 Functions

Overview

The Functions.xml describes the various methods to include Functions into the implementation.

The HERAS^{AF} XACML 2.0 implementation provides the complete set of functions defined by [XACML20]. The user of the HERAS^{AF} XACML 2.0 implementation is free to exchange a subset or all of these functions with own ones. Furthermore complete own functions can be added. The only restriction is that they implement the Function interface.

Important hint

Additionally added packages or classes must be visible to the class loader of the XACML 2.0 implementation. Otherwise it is impossible to load the given functions.

If functions are defined multiple times, the last occurrence in the configuration file overrides the forerunner.

Configuration file

The configuration file is an XML file that fulfils the Functions.xsd schema file. This schema is found in the resources of the PDP module.



structure

```
<?xml version="1.0" encoding="UTF-8"?>

<sb:beans xmlns="http://heras-
af.org/pdp/namespace/functions"
  xmlns:sb="http://www.springframework.org/schema/
beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/
schema/beans
http://www.springframework.org/schema/beans/spring-beans-
2.0.xsd
http://heras-af.org/pdp/namespace/functions http://heras-
af.org/pdp/namespace/functions/Functions.xsd">

  <functions id="URNTToFunctionConverter">
    <packages>
      <package jar="...">...</package>
    </packages>
    <classes>
      <class>...</class>
    </classes>
  </functions>

</sb:beans>
```

Figure 5: Example of the Functions configuration.

Structure if only the XACML 2.0 default functions should be used:

```
<?xml version="1.0" encoding="UTF-8"?>

<sb:beans xmlns="http://heras-
af.org/pdp/namespace/functions"
  xmlns:sb="http://www.springframework.org/schema/
beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/
schema/beans
http://www.springframework.org/schema/beans/spring-beans-
2.0.xsd
http://heras-af.org/pdp/namespace/functions http://heras-
af.org/pdp/namespace/functions/Functions.xsd">

  <functions id="URNTToFunctionConverter" />

</sb:beans>
```

Figure 6: Example of the Functions configuration (default functions)

Element description

| | |
|-----------------|--|
| <i>sb:beans</i> | The root element containing the whole configuration for the functions. The namespaces declarations in this element must not be changed. |
|-----------------|--|



| | |
|------------------|---|
| <i>functions</i> | Indicates the beginning of the functions configuration. The <i>id</i> is the Spring-Id which is used to refer to the bean described here. This id ought not to be changed. |
| <i>packages</i> | Contains the packages which contain functions. This element is optional. |
| <i>classes</i> | Contains the Function-classes. This element is optional. |
| <i>package</i> | Contains a single package that contains functions. This package must be in the given jar. |
| <i>class</i> | Contains a single function-class. |



5 DataModelAttribute

Overview The DataModelAttribute.xml describes the various data type attributes to include into the implementation.

The HERAS^{AF} XACML 2.0 implementation provides the complete set of data type attributes defined by [XACML20]. The user of the HERAS^{AF} XACML 2.0 implementation is free to exchange a subset or all of these data type attributes with own ones. Furthermore complete own data type attributes can be added. The only restriction is that they implement the DataModelAttribute<?> interface ("?" stands for the containing data type).

Important hint Additionally added packages or classes must be visible to the class loader of the XACML implementation. Otherwise it is impossible to load the given functions.

If data type attributes are defined multiple times, the last occurrence in the configuration file overrides the forerunner.

Configuration file The configuration file is an XML file that fulfils the DataModelAttribute.xsd schema file. This schema is located in the resources of the PDP module.

structure

```
<?xml version="1.0" encoding="UTF-8"?>

<sb:beans xmlns="http://heras-
af.org/pdp/namespace/dataModelAttribute"
  xmlns:sb="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/
schema/beans
http://www.springframework.org/schema/beans/spring-beans-
2.0.xsd
http://heras-af.org/pdp/namespace/dataModelAttribute
http://heras-af.org/pdp/namespace/dataModelAttribute/
DataModelAttribute.xsd">

  <dataModelAttribute id="URNToDataModelAttribute">
    <packages>
      <package jar="...">...</package>
    </packages>
    <classes>
      <class>...</class>
    </classes>
  </dataModelAttribute>

</sb:beans>
```

Figure 7: Example of the DataModelAttribute configuration.



Structure if only the XACML 2.0 default data type attributes should be used:

```
<?xml version="1.0" encoding="UTF-8"?>

<sb:beans xmlns="http://heras-af.org/pdp/namespace/dataTypeAttributes"
          xmlns:sb="http://www.springframework.org/schema/beans"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://www.springframework.org/
schema/beans
http://www.springframework.org/schema/beans/spring-beans-
2.0.xsd
http://heras-af.org/pdp/namespace/dataTypeAttributes
http://heras-af.org/pdp/namespace/dataTypeAttributes/
DataTypeAttributes.xsd">

    <dataTypeAttributes id="URNToDataTypeConverter" />

</sb:beans>
```

Figure 8: Example of the `DataTypeAttributes` configuration (default data type attributes)

Element description

| | |
|---------------------------|---|
| <i>sb:beans</i> | The root element containing the whole configuration for the data type attributes. The namespaces declarations in this element must not be changed. |
| <i>dataTypeAttributes</i> | Indicates the beginning of the data type attributes configuration. The <code>id</code> is the Spring-Id which is used to refer to the bean described here. This id must not to be changed. |
| <i>packages</i> | Contains the packages which contain data type attributes. This element is optional. |
| <i>classes</i> | Contains the <code>DataTypeAttribute</code> -classes. This element is optional. |
| <i>package</i> | Contains a single package that contains data type attributes. This package must be in the given jar. |
| <i>class</i> | Contains a single <code>DataTypeAttribute</code> -class. |



6 ContextAndPolicyConfiguration

Overview

The ContextAndPolicyConfiguration.xml describes the various configuration options of the context and the policy files.

The user of the HERAS^{AF} XACML 2.0 implementation has the possibility to configure the formatting and the validating of requests, responses and policies. A distinction is drawn between input and output validation.

Configuration file

The configuration file is an XML file that fulfils the ContextAndPolicyConfiguration.xsd schema file. This schema is located in the resources of the PDP module.

structure

```
<?xml version="1.0" encoding="UTF-8"?>

<sb:beans
  xmlns="http://heras-
af.org/pdp/namespace/contextandpolicyconfiguration"
  xmlns:sb="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/
schema/beans
http://www.springframework.org/schema/beans/spring-beans-
2.0.xsd
http://heras-
af.org/pdp/namespace/contextandpolicyconfiguration
http://heras-
af.org/pdp/namespace/contextandpolicyconfiguration/
ContextAndPolicyConfiguration.xsd">

  <contextAndPolicyConfiguration id="jaxbUtils">
    <requestCtx>
      <formatted_output>...</formatted_output>
      <writeSchemaLocation>...</writeSchemaLocation>
      <schemaLocation>...</schemaLocation>
      <validateWriting>...</validateWriting>
      <validateParsing>...</validateParsing>
      <schemaByPath>...</schemaByPath>
      <fragment>...</fragment>
      <package>...</package>
    </requestCtx>
    <responseCtx>
      <formatted_output>...</formatted_output>
      <writeSchemaLocation>...</writeSchemaLocation>
      <schemaLocation>...</schemaLocation>
      <validateWriting>...</validateWriting>
      <validateParsing>...</validateParsing>
      <schemaByPath>...</schemaByPath>
      <fragment>...</fragment>
      <package>...</package>
    </responseCtx>
  </contextAndPolicyConfiguration>
</sb:beans>
```



```

<policy>
  <formatted_output>...</formatted_output>
<writeSchemaLocation>...</writeSchemaLocation>
  <schemaLocation>...</schemaLocation>
  <validateWriting>...</validateWriting>
  <validateParsing>...</validateParsing>
  <schemaByPath>...</schemaByPath>
  <fragment>...</fragment>
  <package>...</package>
</policy>
</contextAndPolicyConfiguration>

```

```
</sb:beans>
```

Figure 9: Example ContextAndPolicyConfiguration configuration

Element description

| | |
|---------------------------------------|--|
| <i>sb:beans</i> | The root element containing the whole configuration of the context and policy. The namespaces declarations in this element must not be changed. |
| <i>contextAndPolicy Configuration</i> | Indicates the beginning of the context and policy configuration. The <i>id</i> is the Spring-Id which is used to refer to the bean described here. This id ought not to be changed. |
| <i>requestCtx</i> | Contains the configuration of the request context. |
| <i>responseCtx</i> | Contains the configuration of the response context. |
| <i>policy</i> | Contains the configuration of the policy. |
| <i>formatted_output</i> | True if the XML output should be formatted. |
| <i>writeSchemaLocation</i> | True if the schema location should be added to the output XML. If this value is true, then the <i>schemaLocation</i> must be set. |
| <i>schemaLocation</i> | The location of the schema. |
| <i>validateWriting</i> | True if the output XML should be validated. If true the <i>schemaByPath</i> should be set. |
| <i>validateParsing</i> | True if the input XML should be validated. If true the <i>schemaByPath</i> should be set. |
| <i>schemaByPath</i> | The path to the schema file for validating the input / output XML. |
| <i>fragment</i> | True if the output XML can be a fragment of an other XML. |
| <i>package</i> | The package of the object factory of the context or policy. |



7 Combining Algorithms

Overview

The combining algorithms cannot be configured with a custom XML schema and must therefore be configured in the normal Spring framework application context.

The reason is that the configuration file loads a converter which contains a map of combining algorithms. These combining algorithms take the TargetMatcher (responsible for finding Evaluatables that match a request) as constructor argument. It is impossible to create such an object because while creating the converter no reference to the TargetMatcher is available.

The HERAS^{AF} XACML 2.0 implementation provides the complete set of combining algorithms defined by [XACML20]. The user of the HERAS^{AF} XACML 2.0 implementation is free to exchange a subset or all of these combining algorithms with own ones. Furthermore complete own combining algorithms can be added. The only restriction is that they implement the either the RuleCombiningAlgorithm interface or the PolicyCombiningAlgorithm interface.

structure

```
<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/
schema/beans
http://www.springframework.org/schema/beans/spring-
beans.xsd">

...

<bean id="URNToZZZCombiningAlgorithmConverter"
      class="URNToZZZCombiningAlgorithmConverter">
  <property name="combiningAlgorithms">
    <map>
      <entry key="deny-overrides"
            value-ref="ruleDenyOverridesAlgorithm" />
    </map>
  </property>
</bean>

...

<bean id="zzzDenyOverridesAlgorithm"
      class="ZZZDenyOverridesAlgorithm">
  <constructor-arg ref="targetMatcher" />
</bean>

...

</beans>
```

Figure 10: Example of theCombiningAlgorithm configuration.



Element description

| | |
|---|---|
| <i>beans</i> | <p>The root element containing the whole configuration of the Spring framework application context.</p> <p>The namespaces declarations in this element must not be changed.</p> |
| <i>bean with id</i> <i>URNToZZZCombining</i> <i>Algorithm</i> <i>Converter</i> | <p>ZZZ is either Rule or Policy.</p> <p>This bean is the converter which contains the various combining algorithms.</p> <p>The <i>id</i> is the Spring-Id which is used to refer to the bean described here. This id ought not to be changed.</p> |
| <i>bean with id</i> <i>zzzDenyOverrides</i> <i>Algorithm</i> | <p>zzz is either rule or policy.</p> <p>This bean is the combining algorithm which takes the TargetMatcher as constructor argument.</p> |
| <i>map property</i> | <p>The map in the bean with the id <i>URNToZZZCombiningAlgorithmConverter</i> injects the combining algorithms into the converter. The key of the map is the id of the combining algorithm and the value is the combining algorithm itself.</p> |

8 Advanced configuration options

| | |
|------------------|---|
| <i>Overview</i> | <p>This XACML 2.0 implementation uses the Spring framework for wiring and configuring the modules.</p> <p>Where possible, user-friendly configuration files are defined and embedded using the Spring framework feature of Extensible XML Authoring.</p> <p>In cases where this was impossible the configuration must take place in the application context configuration file.</p> |
| <i>structure</i> | <p>The first subchapter is about configuring the Extensible XML Authoring of the Spring framework.</p> <p>Afterwards it is explained how to inject the different modules into the PDP.</p> <p>The rest of the subchapters deal with configuring the index handler, the reference loader, the attribute finder and the target matcher.</p> |



8.1 Configuration files

Overview

This part shows how to embed the locator, persistence manager, functions, data type attributes and the context and policy configuration files into the application context of the Spring framework.

The user of the HERAS^{AF} XACML 2.0 implementation is free to change these imports to exchange a specific configuration with an own one.

structure

```
<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
       xsi:schemaLocation="http://www.springframework.org/
schema/beans
http://www.springframework.org/schema/beans/spring-
beans.xsd">

  <import resource="PersistenceManager.xml" />
  <import resource="Locator.xml" />
  <import resource="ContextAndPolicyConfiguration.xml" />
  <import resource="Functions.xml" />
  <import resource="DataTypeAttributes.xml" />

  <bean class="PDPPPostProcessor" /> <!-- PostProcessor -->

  ...

</beans>
```

Figure 11: Example of the configuration files imports.

Important hint

The order of the imports does not matter but the imports must occur at the beginning of the beans part. No bean-definition may appear before.

Element description

| | |
|--|--|
| import | The import statements to import the configuration files. The resource attribute contains the path to the configuration file. |
| bean with id <i>PDPPPostProcessor</i> | This bean is the postprocessor of all beans that are created and executes the initialize method of the PersistenceManagerImpl. |



8.2 PDP

Overview

This part explains how to configure the PDP with the intended implementations of the needed modules.

The user of the HERAS^{AF} XACML 2.0 implementation is free to use an own implementation of the PDP. The only restriction is that this PDP must implement the PDP interface.

structure

```
<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
       xsi:schemaLocation="http://www.springframework.org/
schema/beans
http://www.springframework.org/schema/beans/spring-
beans.xsd">

    ...

    <bean id="pdp" class="PDPImpl">
        <constructor-arg ref="locator" />
        <constructor-arg ref="persistenceManager" />
        <constructor-arg ref="remoteLoader" />
        <constructor-arg ref="attributeFinder" />
        <constructor-arg ref="onlyOneApplicableAlgorithm" />
    </bean>

    ...

</beans>
```

Figure 12: Example of the PDP configuration.

Element description

| | |
|-------------------------|--|
| bean with id <i>pdp</i> | This element contains the wiring of the PDP, and it shows what the implementation class is. |
| constructor-arg | These elements are references to the various modules that are plugged-in into the PDP. These references must reference a bean within this application context. |



8.3 IndexHandler

Overview

This part explains how to configure the index handler with the intended implementation needed for the locator module.

The user of the HERAS^{AF} XACML 2.0 implementation is free to use an own implementation of the IndexHandler. The only restriction is that this index handler must implement the IndexHandler interface.

structure

```
<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
       xsi:schemaLocation="http://www.springframework.org/
schema/beans
http://www.springframework.org/schema/beans/spring-
beans.xsd">

    ...

<bean id="indexHandler" class="IndexHandlerImpl" />

    ...

</beans>
```

Figure 13: Example of the IndexHandler configuration.

Element description

| | |
|---|---|
| bean with id <code>indexHandler</code> | This element configures which implementation of the index handler should be used. |
|---|---|



8.4 ReferenceLoader

Overview

This part explains how to configure the remote loader with the intended implementation needed for the PDP module.

The user of the HERAS^{AF} XACML 2.0 implementation is free to use an own implementation of the ReferenceLoader. The only restriction is that this reference loader must implement the ReferenceLoader interface.

structure

```
<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
        xsi:schemaLocation="http://www.springframework.org/
schema/beans
http://www.springframework.org/schema/beans/spring-
beans.xsd">

    ...

<bean id="referenceLoader" class="ReferenceLoaderImpl" />

    ...

</beans>
```

Figure 14: Example of the ReferenceLoader configuration.

Element description

| | |
|--|--|
| bean with id <code>referenceLoader</code> | This element configures which implementation of the reference loader should be used. |
|--|--|



8.5 AttributeFinder

Overview

This part explains how to configure the AttributeFinder with the intended implementation needed for the PDP module.

The user of the HERAS^{AF} XACML 2.0 implementation is free to use an own implementation of the AttributeFinder. The only restriction is that this attribute finder must implement the AttributeFinder interface.

structure

```
<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
       xsi:schemaLocation="http://www.springframework.org/
schema/beans
http://www.springframework.org/schema/beans/spring-
beans.xsd">

    ...

    <bean id="attributeFinder" class="AttributeFinderImpl" />

    ...

</beans>
```

Figure 15: Example of the AttributeFinder configuration.

Element description

| | |
|--|--|
| bean with id <i>attributeFinder</i> | This element configures which implementation of the attribute finder should be used. |
|--|--|



8.6 TargetMatcher

Overview

This part explains how to configure the target matcher with the intended implementation needed for the locator module.

The user of the HERAS^{AF} XACML 2.0 implementation is free to use an own implementation of the TargetMatcher. The only restriction is that this target matcher must implement the TargetMatcher interface.

structure

```
<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
        xsi:schemaLocation="http://www.springframework.org/
schema/beans
http://www.springframework.org/schema/beans/spring-
beans.xsd">

    ...

    <bean id="targetMatcher" class="TargetMatcherImpl" />

    ...

</beans>
```

Figure 16: Example of the TargetMatcher configuration.

Element description

| | |
|--------------------------------------|--|
| bean with id <i>targetMatcher</i> | This element configures which implementation of the target matcher should be used. |
|--------------------------------------|--|



Part III: Implementing Functions, Combining Algorithms and DataTypes



1 Overview

Introduction

This part of the Developer's Guide explains how new Functions, DataTypes and CombiningAlgorithms can be implemented and included into the existing HERAS^{AF} XACML 2.0 implementation. In the first chapter, the Functions are explained. Afterwards the DataTypes and at last the CombiningAlgorithms are explained.

2 Implementing Functions

Overview

This XACML 2.0 implementation offers the possibility to include own functions into the implementation or override existing ones.

This chapter is about how to proceed in writing and embedding own or overridden Functions into the HERAS^{AF} XACML 2.0 implementation.

Writing

The class representing the function must implement the `org.herasaf.core.function.Function` interface and the `toString`-method must return the Function id.

In case of an error while processing a Function an `org.herasaf.core.function.exception.`

`FunctionProcessingException` must be thrown.

Grouping

Groups of functions can be packaged into a java archive (jar). There is no limitation to the package path of a Function.

Embedding

Functions can be embedded into the implementation according to the configuration guide in section II of this Developer's Guide.



Example function

As example the StringEqualFunction:

```
package org.herasaf.core.function.impl.equalityPredicates;

import org.herasaf.core.function.Function;

/**
 * The implementation of the urn:oasis:names:tc:xacml:1.0:function:string-equal function.
 * See: Appendix A.3 of the <a href="http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml#XACML20">
 * OASIS eXtensible Access Control Markup Language (XACML) 2.0, Errata 29 June 2006</a> page 105, for further information.
 *
 * @author Sacha Dolski (sdolski@solnet.ch)
 * @version 1.0
 */
public class StringEqualFunction implements Function {
    private static final long serialVersionUID = -3491926450245801282L;
    private static final String ID = "urn:oasis:names:tc:xacml:1.0:function:string-equal";

    /**
     * {@inheritDoc}
     *
     * Takes two {@link String} objects as parameters and returns whether they are
     * equal or not as {@link Boolean} value.
     */
    public Object handle(Object... args) throws FunctionProcessingException {
        try {
            if (args.length != 2) {
                throw new FunctionProcessingException("Invalid number of parameters");
            }
            String string = (String)args[1];
            return new Boolean(((String)args[0]).equals(string));
        } catch (ClassCastException e) {
            throw new FunctionProcessingException(e);
        } catch (Exception e) {
            throw new FunctionProcessingException(e);
        }
    }

    /**
     * (non-Javadoc)
     * @see org.herasaf.core.function.Function#toString()
     */
    @Override
    public String toString() {
        return ID;
    }
}
```

Figure 17: Example implementation of the StringEqualFunction.

3 Implementing DataTypes

Overview

This XACML 2.0 implementation offers the possibility to include own data types into the implementation or override existing ones.

This chapter is about how to proceed in writing and embedding own or overridden data types into the HERAS^{AF} XACML 2.0 implementation.

To use own or overridden data types a `DataTypeAttribute` must be implemented which is responsible for converting the `String` representation into the according data type.

Hint

It is not always necessary to write a data type from scratch. In many cases it is possible to use the java data type, maybe with some boundaries.

Writing

The class representing the data type attribute must implement the `org.herasaf.core.function.dataTypeAttribute`.

`DataTypeAttribute<E>` interface and the `toString`-method must return the data type id.

In case of an error while converting `String` representation an `org.herasaf.exception.SyntaxException` must be thrown.

Grouping

Groups of data type attributes can be packaged into a java archive (jar). There is no limitation to the package path of a data type attribute.

Embedding

The data type attributes can be embedded into the implementation according to the configuration guide in section II of this Developer's Guide.



Example data type attribute

As example the TimeDataTypeAttribute:

```
package org.herasaf.core.dataTypeAttribute.impl;

import org.herasaf.core.dataTypeAttribute.DataTypeAttribute;

/**
 * The Name of this data type is http://www.w3.org/2001/XMLSchema#time.<br>
 * See: <A HREF="http://www.w3.org/TR/xmlschema-2/#time"
 * target=" _blank">http://www.w3.org/TR/xmlschema-2/#time</A> for further
 * information.
 *
 * @author Stefan Oberholzer (soberholzer@gmx.ch)
 * @version 1.0
 */
public class TimeDataTypeAttribute implements DataTypeAttribute<Time> {
    private static final long serialVersionUID = -2419136782434407365L;
    private static final String ID = "http://www.w3.org/2001/XMLSchema#time";

    /*
     * (non-Javadoc)
     * @see org.herasaf.core.dataTypeAttribute.DataTypeAttribute#convertTo(java.lang.String)
     */
    public Time convertTo(String jaxbRepresentation) throws SyntaxException {
        try {
            return new Time(jaxbRepresentation.trim());
        } catch (IllegalArgumentException e) {
            throw new SyntaxException(e);
        }
    }

    /*
     * (non-Javadoc)
     * @see java.lang.Object#toString()
     */
    @Override
    public String toString() {
        return ID;
    }
}
```

Figure 18: Example implementation of the TimeDataTypeAttribute.

4 Implementing Combining Algorithms

Overview

This XACML 2.0 implementation offers the possibility to include own rule and policy combining algorithms into the implementation or override existing ones. This chapter is about how to proceed in writing and embedding own or overridden combining algorithms into the HERAS^{AF} XACML 2.0 implementation.

Combining Algorithm superclasses

All combining algorithms must implement the `org.herasaf.core.combiningAlgorithm.CombiningAlgorithm` interface.

The interface contains the evaluate method, taking a policy which has to be evaluated using the algorithm.

The `org.herasaf.core.combiningAlgorithm`.

`AbstractCombiningAlgorithm` abstract class is an implementation of this interface, implementing a method to match a target.

Writing a Policy Combining Algorithm

The class representing the policy combining algorithm must implement `org.herasaf.core.combiningAlgorithm.policy`.

`PolicyCombiningAlgorithm` interface. The interface contains the `evaluateEvaluatableList` method, which is used to get the combined decision of the `Evaluatables`.



The abstract classes `org.herasaf.core.combiningAlgorithm.policy.PolicyOrderedCombiningAlgorithm` and `org.herasaf.core.combiningAlgorithm.policy.PolicyUnorderedCombiningAlgorithm` implements the `evaluate` method of the interface, calling the `evaluateEvaluatableList` method with the `Evaluatables` contained in the `PolicySet` as an ordered `List` of `Evaluatables` or an unordered `List` of `Evaluatables`.

*Writing a Rule
Combining
Algorithm*

The class representing the rule combining algorithm must implement `org.herasaf.core.combiningAlgorithm.rule.RuleCombiningAlgorithm` interface. The interface contains the `evaluateRuleList` method, which is used to get the combined decision of the `Rules`.

The abstract classes `org.herasaf.core.combiningAlgorithm.rule.RuleOrderedCombiningAlgorithm` and `org.herasaf.core.combiningAlgorithm.rule.RuleUnorderedCombiningAlgorithm` implements the `evaluate` method of the interface, calling the `evaluateRuleList` method with the `Rules` contained in the `Policy` as an ordered `List` of `Rules` or an unordered `List` of `Rules`.

Embedding

The combining algorithm can be embedded into the implementation according to the configuration guide in section II of this Developer's Guide.

Limitations

The current implementation of the HERAS^{AF} XACML 2.0 implementation does only support `AttributeValues` containing one value. See Appendix A.2 (Single `AttributeValue`) for further information.



Part IV: Architecture, Extending and Adapting



1 Overview

Introduction

This part of the Developer's Guide explains the architecture of the HERAS^{AF} XACML 2.0 implementation, how it can be extended and how a new implementation of a module can be integrated.

1.1 Architecture

Overview

This chapter explains the architecture of the HERAS^{AF} XACML 2.0 implementation.

The HERAS^{AF} XACML 2.0 implementation has a modular architecture. This ensures that different parts of the logic can be changed without affecting the rest of the PDP. The only module that affects all other module is the core module. In this chapter the responsibilities of the different modules are explained, how they interact and how the HERAS^{AF} realized the implementations.

1.1.1 Modules

Overview

This chapter explains how the modules are wired and gives a short description of every module.

Package diagram

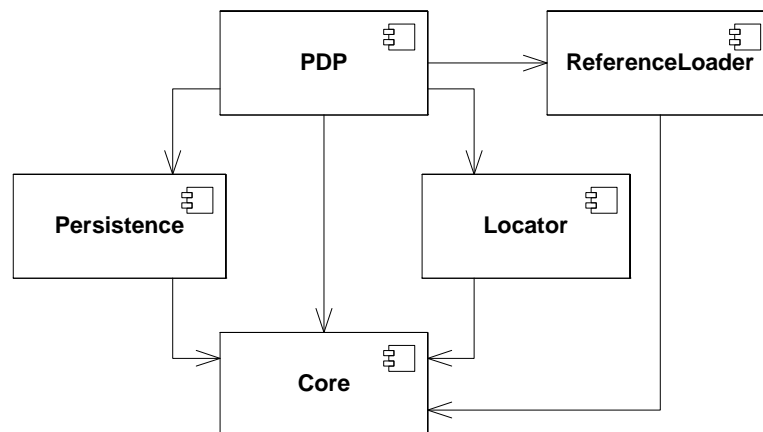


Figure 19: Modules and their dependencies.

core

The core module contains the data classes for policies, requests and responses. For every specified XML complex type in the XACML 2.0 specification there is one class. Additionally the data classes, converter classes and classes used by the data classes are also in this module. Also the target matcher and the attribute finder are in the core.

pdp

The pdp module is the main module that contains all other modules. It includes the PDP interface and the classes needed for the Spring framework Extensible XML Authoring [SEXMLA].



persistence

The persistence module contains the logic to persist the Evaluatables.

referenceLoader

The referenceLoader is able to resolve references to Evaluatables. It provides methods to return an Evaluatable requested by another PDP or referenced in another Evaluatable.

locator

The locator module is responsible to return the Evaluatables that possibly match the request. It includes indexing algorithms over the Evaluatables.

1.1.2 Dataflow

Overview

This chapter explains the main data flows of the HERAS^{AF} XACML 2.0 implementation. These data flows have the PDP as their main interaction part because the PDP is the heart of the HERAS^{AF} XACML 2.0 implementation.

Instantiation and initialization of the PDP

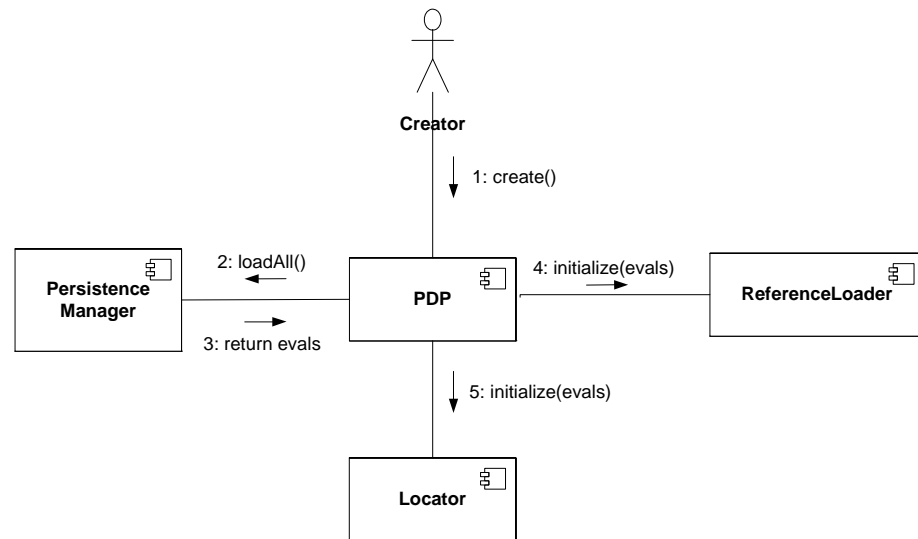
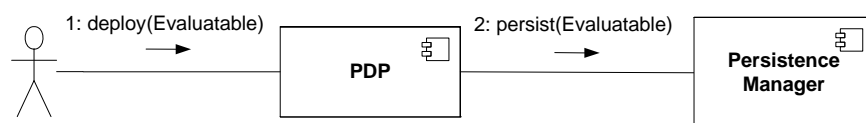


Figure 20: Dataflow of the PDP initialization.

Description

1. Instantiates a new PDP.
2. The PDP loads all Evaluatables from the underlying persistence layer.
3. These Evaluatables are returned to the PDP.
4. The loaded Evaluatables (evals) are passed to the ReferenceLoader which resolves the local references with the given Evaluatables.
5. The loaded Evaluatables (evals) are passed to the Locator which initializes the index with the given Evaluatables.

Policy Deployment



Operator

Figure 21: Dataflow of the deployment of an Evaluatable.



Description

1. The operator calls the `deploy` method of the PDP with the `Evaluatables` as arguments.
2. The PDP calls the `persist` method of the Persistence Manager with the `Evaluatables` as argument.

Because no hot deployment is supported, the `Evaluatables` are not added to the index and therefore do not take any influence on the decisions of the PDP until the PDP is newly instantiated.

Evaluating a request

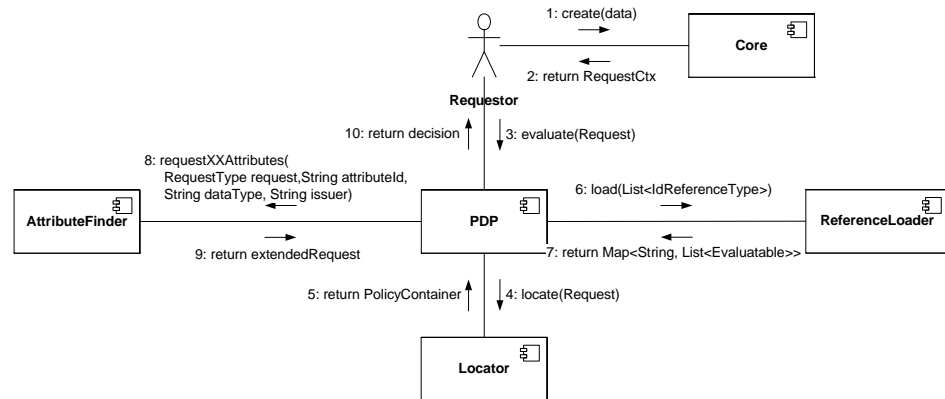


Figure 22: Dataflow of the evaluation of an Evaluatable. (The “Requestor” is an application with the intention to evaluate)

Description

1. The Requestor calls the `create` method on the `RequestCtxFactory` in the core module with the request specific data.
2. The factory returns a `RequestCtx`.
3. The `RequestCtx` is passed to the PDP’s `evaluate`-method.
4. The PDP calls the `locate` method on the `Locator`.
5. The `Locator` searches the `Evaluatables` and references in the index and returns them.
6. The PDP calls the `load` method on the `ReferenceLoader`.
7. The `ReferenceLoader` resolves all references and returns a `Map` containing the `Evaluatables`.
8. If there is a missing attribute, the PDP asks the `AttributeFinder` to resolve it.
9. The resolved missing attributes are returned and the evaluation continues.
10. The decision is returned.



1.1.3 Evaluation Sequence

Overview

This chapter explains how the evaluation sequence looks like. It gives an overview about the evaluation process.

PDP sequence

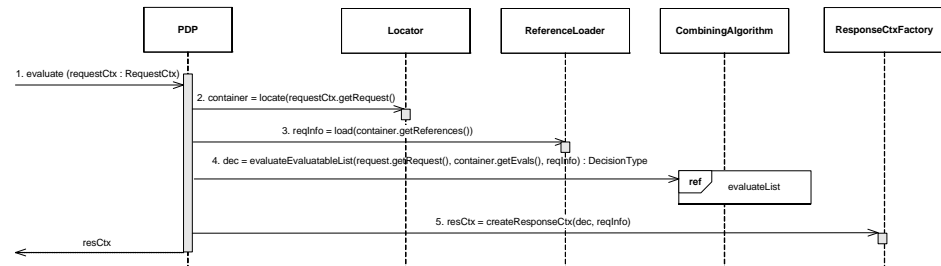


Figure 23: Evaluation sequence of the PDP.

Description

1. The evaluate method of the PDP is called.
2. The PDP calls the Locator to obtain a PolicyContainer object, containing possibly matching Evaluatables and the references to Evaluatables.
3. The PDP calls the ReferenceLoader to resolve the Evaluatables. The referenced Evaluatables are added to the RequestInformation.
4. The PDP calls the evaluateEvaluatableList-method on the default combining algorithm.
5. The PDP creates a ResponseCtx using the ResponseCtxFactory.



Target match sequence of the combining algorithms

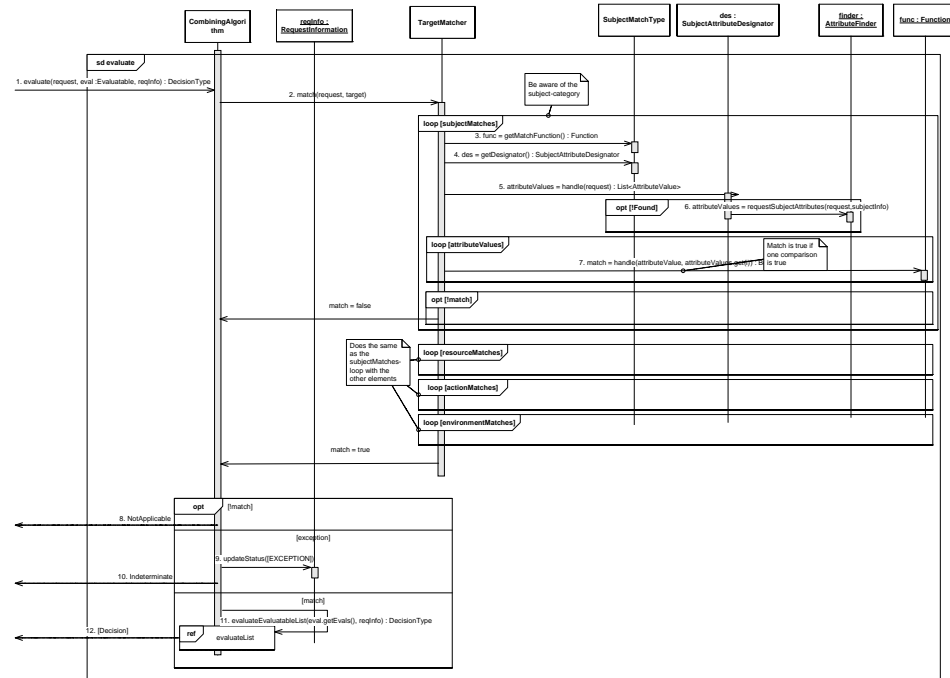


Figure 24: Target matching sequence.

Description

1. The evaluate method of a combining algorithm is called.
2. The combining algorithm calls the match method of the TargetMatcher to match the target of an Evaluatable.
3. The TargetMatcher calls the method to get the Function included in the SubjectMatchType.
4. The TargetMatcher calls the method to get the Designator included in the SubjectMatchType.
5. The TargetMatcher calls the handle Method of the Designer.
6. If the handle method did not find an appropriate subject, the requestSubjectAttribute method of the AttributeFinder is called.
7. The TargetMatcher calls the handle Method of the Function, using the returned values of the Designer as argument.
8. If the target did not match, NotApplicable is returned.
9. If the TargetMatcher threw an Exception, the status in the RequestInformation is updated to the status of the exception.
10. Indeterminate is returned.
11. If the Target has matched the evaluateEvaluatableList-method is called if the combining algorithm is a PolicyCombiningAlgorithm. If the combining algorithm is a RuleCombiningAlgorithm evaluateRuleList would be called instead.
12. The result is returned.



*Evaluate
EvaluatableList
Sequence*

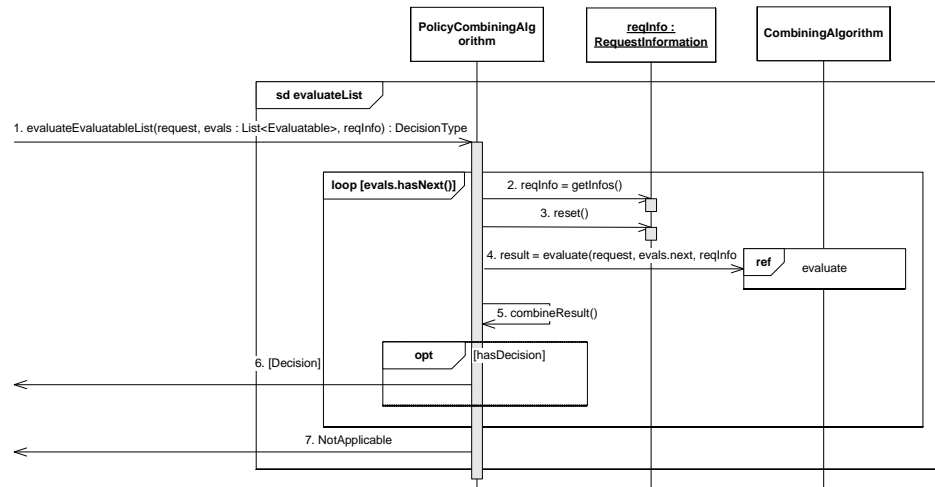


Figure 25: Evaluation of a list of Evaluatables.

Description

1. The evaluateEvaluatableList is called on the PolicyCombiningAlgorithm.
2. The status and missing attributes in the RequestInformation are read.
3. The RequestInformation is set to its default values.
4. The evaluate method of the actual Evaluatable is called.
5. The results are combined.
6. If a decision can be made it is returned. The RequestInformation is set to the actual status.
7. If no decision could be made NotApplicable is returned.

*RuleCombining
sequence*

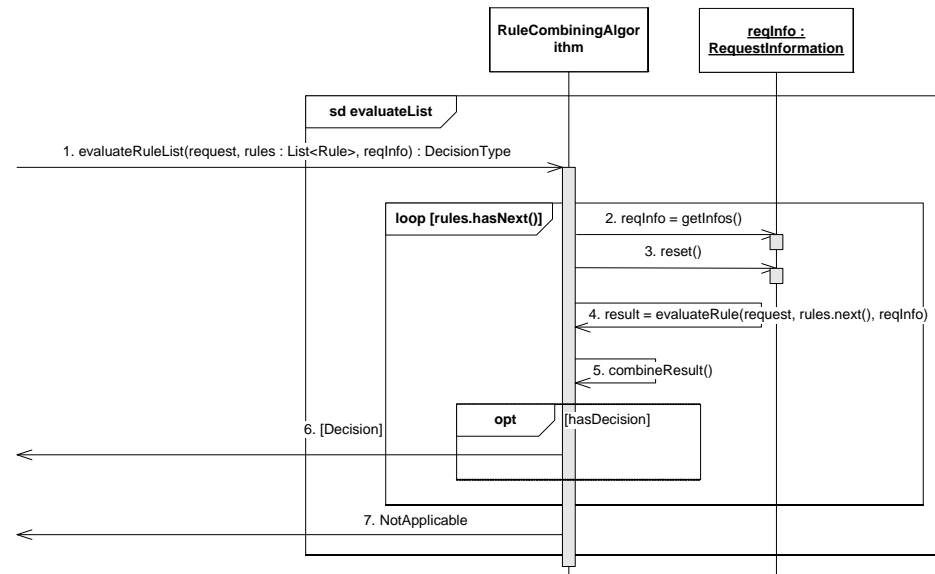


Figure 26: Combining of rule evaluation results.

Description

1. The evaluateRuleList is called on the RuleCombiningAlgorithm.
2. The status and missing attributes in the RequestInformation are read.
3. The RequestInformation is set to its default values.
4. The rules are evaluated.
5. The result are combined.
6. If a decision can be made it is returned. The RequestInformation is set to the actual status.
7. If no decision could be made NotApplicable is returned.



Evaluating a Condition

A Condition has the following structure:

- Each Condition contains an expression returning a Boolean value
- Each Expression may contain other Expressions
- Each class extending the `ExpressionType` contains a `handle` method to handle the logic

Because of this the Condition just calls the handle method of the contained Expression and casts the result to a Boolean value. If the cast is successful, the value is returned. Otherwise an indeterminate exception is thrown. If in one of the Designator classes a `MissingAttribute` occurs the designator specific method of the `AttributeFinder` class is called.

1.1.4 Concurrency

Overview

This chapter deals with the challenge of concurrent access to the PDP.

Importance

Concurrent access is very important because it is unimaginable that only one request at once could be handled. A PDP in a business environment for example has to process hundreds of requests per second at peak times. All the parallel accesses must not interfere with each other, otherwise a false decisions might result.

Realization

The HERAS^{AF} PDP deals in a very simple but effective way with the concurrency issue. All classes involved into the decision making do not write to or iterate over any class variables, expect of the requests and responses. But this has no influence because these objects are per accessing thread. The whole processing happens with method-local variables which causes that all these variables are on the stack of the thread of the accessing request. Therefore it is impossible that interferences occur.



1.1.5 Design decisions

Overview This chapter explains the most important design decision made in the HERAS^{AF} XACML 2.0 implementation. Each design decision consists of a description of the issue, the possible solutions and an explanation which solution was chosen.

1.1.5.1 ReferenceLoader as independent module

Issue The implementation to load referenced Evaluatables should be exchangeable to ensure that different caching strategies are possible and the PDP can communicate with other PDPs. To be sure that the evaluation of an Evaluatable is consistent all referenced Evaluatables must be loaded at the same time. Where should the logic, for loading referenced Evaluatables be placed?

Alternatives **Alternative 1: As module integrated into the Locator module**

The ReferenceLoader is a module integrated into the Locator module.

Advantages

- The PDP does not care how the referenced Evaluatables are resolved
- Logic encapsulated in its own module
- Easy exchangeability

Disadvantages

- Changes to the Locator module affect the ReferenceLoader module

Alternative 2: As module integrated into the PDP module

By calling the locate-method on the PDP a PolicyContainer, including the local Evaluatables and references, is returned. The references are passed to the ReferenceLoader module which is an independent module.

Advantages

- Only the PDP depends on the ReferenceLoader
- Logic encapsulated in its own module
- Easy exchangeability

Disadvantages

- The PDP needs logic how Evaluatables are loaded

Alternative 3 The PolicySet knows the ReferenceLoader and resolves its references on its own.

The PolicySet contains the logic how the referenced policies are resolved. If one referenced Evaluatable is needed, all referenced Evaluatables are resolved.

Advantages

Disadvantages



-
- The referenced Evaluatables are only loaded if they are needed
 - The core has a dependency to the ReferenceLoader
 - Different caching strategies are not possible

Decision

The decision fell on solution 2. This solution encapsulates the logic to load referenced policies into an independent module. In solution 1, a change of the Locator module requires that it still uses the ReferenceLoader module. Using solution 3 would mean that the logic for loading remote Evaluatables is integrated into the core module.



2 Core

Overview

This chapter explains the core module. The core module provides all the required data classes to build XACML-requests, -responses and -policies. It also contains the `AttributeFinder` which provides the functionality to get attributes from a PIP. The package also contains the `Functions`, `CombiningAlgorithms` and also the data types. These artifacts are all defined by the XACML 2.0 specification [XACML20].

The responsible classes to build a policy, request or response are the `PolicyConverter`, `RequestCtxFactory` and `RequestCtx`, as well as the `ResponseCtxFactory` and `ResponseCtx`.

The `AttributeFinder` implements the `AttributeFinder` interface.

In addition the core contains all classes for successful marshalling and unmarshalling. The core module is, as its name suggests, the core of the implementation. Changes in this module have an impact on every other module. The context and policy classes are the data classes representing the XACML 2.0 structure. It does not make any sense to abstract them with an interface because a change to the data structure would cause a change to the interfaces.

2.1 Interfaces

Overview

This chapter explains the `AttributeFinder` interface. Implementing this interface is the only place where the functionality of the core module can be changed without affecting the other modules.

AttributeFinder

An `AttributeFinder` is used to find attributes which are not in a request. It should enquire the attribute values from a PIP. The obtained attribute values are added to the request. The request is extended and the resolved attributes are returned.

AttributeFinder interface

| <<interface>> AttributeFinder |
|--|
| requestSubjectAttributes(request : RequestType, attributeld : String, dataType : String, issuer : String, subjectCategory : String) : List<AttributeValueType> |
| requestResourceAttributes(request : RequestType, attributeld : String, dataType : String, issuer : String) : List<AttributeValueType> |
| requestActionAttributes(request : RequestType, attributeld : String, dataType : String, issuer : String) : List<AttributeValueType> |
| requestEnvironmentAttributes(request : RequestType, attributeld : String, dataType : String, issuer : String) : List<AttributeValueType> |

Figure 27: The `AttributeFinder` interface.



*requestSubject
Attributes
method*

This method is called to get the attribute values of a subject which are missing. In this method the values should be requested, added to the request and be returned. This method is different to the others because it contains a category attribute.

*rest of the
request
methods*

This method is called to get the attribute values which are missing. In this method the values should be requested, added to the request and be returned.

2.2 HERAS^{AF} XACML 2.0 implementation

Overview

This chapter explains the packages and classes of the core module.

*Package-
diagram*

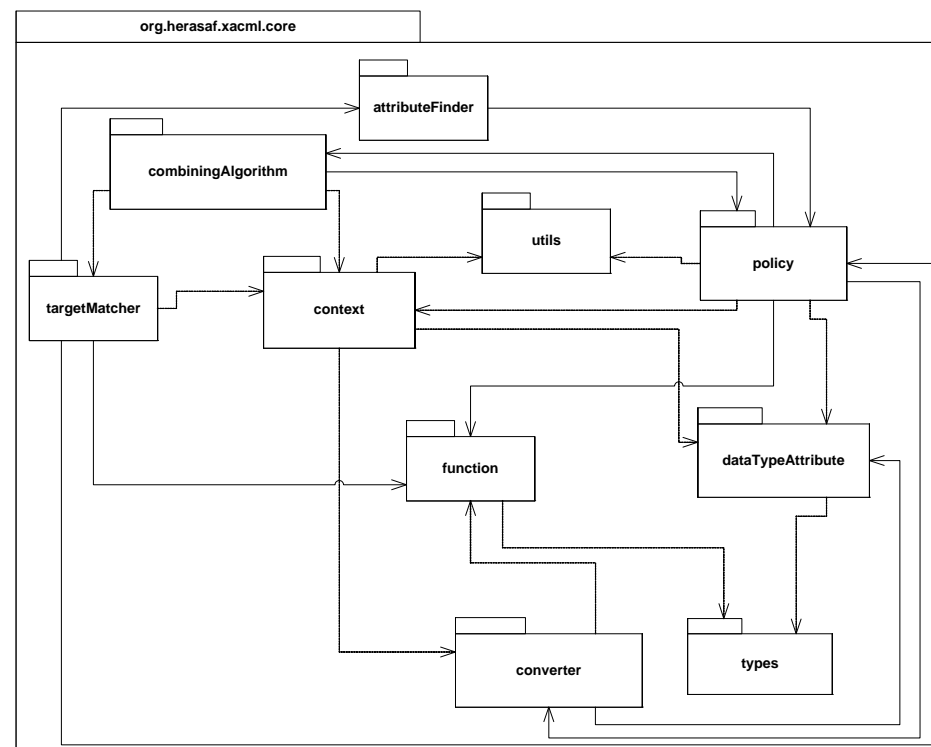


Figure 28: Core packages and their dependencies.

*Package
description*

policy

The package policy contains all the data classes to build XACML Evaluatables. It is important for marshalling and unmarshalling XACML Evaluatables.

attributeFinder

This package contains the AttributeFinder interface and its implementations. The AttributeFinder is used to find attributes on a PIP.

context

The package context contains all the data classes to build XACML-requests or -responses. It is important for marshalling and unmarshalling XACML-requests



and -responses.

converter

The package converter contains all the necessary converter classes, which are needed during the marshalling or unmarshalling process to convert into the proper data objects or XML representation.

combiningAlgorithm

The package combiningAlgorithm contains the rule- and policyCombiningAlgorithms which are used to decide how evaluation results are combined.

function

The package function contains all the functions which are defined by the XACML 2.0 specification [XACML20].

dataTypeAttribute

The package dataTypeAttribute contains all the dataTypeAttributes which converts string representation of a data type into the proper data type.

types

The package types contains all the data types defined by the XACML 2.0 specification [XACML2.0].

targetMatcher

The package targetMatcher contains the classes responsible to match an Evaluatable-target against a request.

utils

The utils package contains various utilities used by the core.



2.2.1 Package policy

Overview

The package policy contains all the data classes to build XACML Evaluatables as well as a PolicyConverter which is responsible to marshal and unmarshal the java objects to XML and backwards.

class diagram

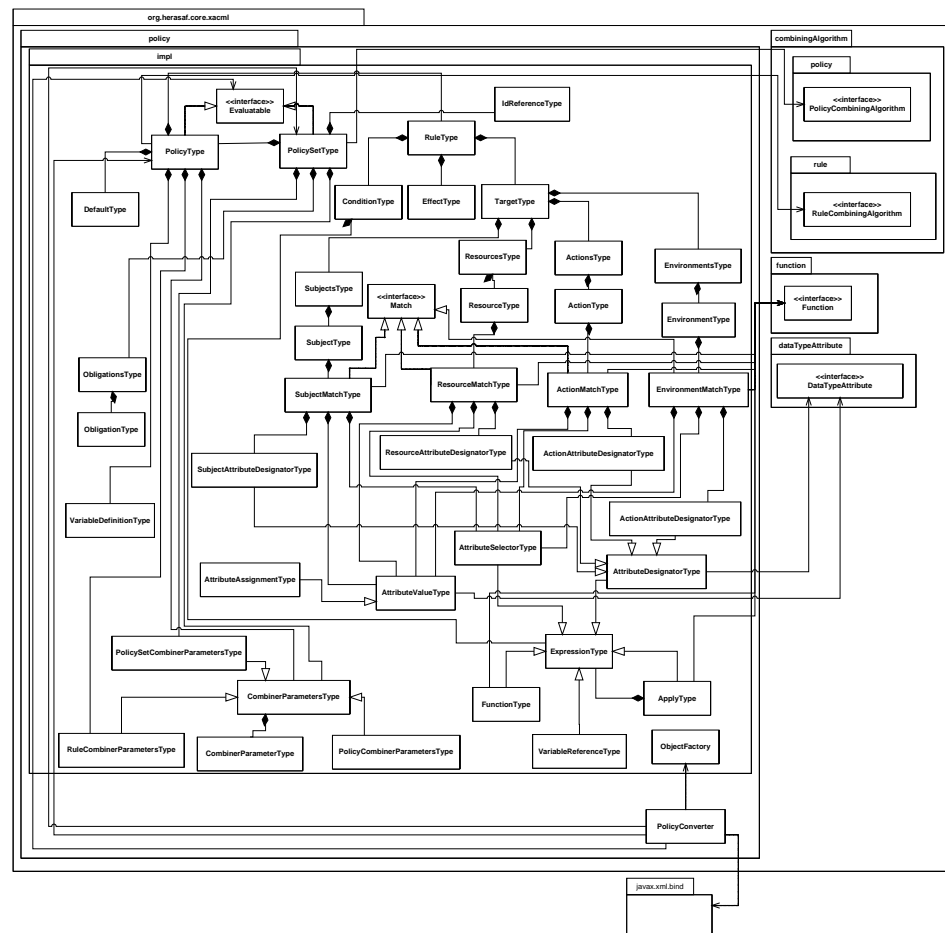


Figure 29: Core class diagram.

relations

Relation to the interface Function

The reference from a data object to the `Function` interface is needed to convert during marshalling and unmarshalling.

Relation to the interface DataTypeAttribute

The reference from a data object to the `DataTypeAttribute` interface is needed to convert during marshalling and unmarshalling.

Relation to the interface PolicyCombiningAlgorithm

The reference from a data object to the `PolicyCombiningAlgorithm` interface is needed to convert during marshalling and unmarshalling.



Relation to the interface RuleCombiningAlgorithm

The reference from a data object to the `RuleCombiningAlgorithm` interface is needed to convert during marshalling and unmarshalling.

Relation to the package javax.xml.bind

The reference to the package `javax.xml.bind` is needed to use the JAXB classes for unmarshalling and marshalling.

2.2.1.1 Important classes and interfaces

Overview

This chapter explains the most important classes and interfaces of the policy package.

The policy package contains the data classes needed to represent policies and policy sets, respectively as objects.

PolicyConverter

| PolicyConverter |
|---|
| CONTEXT : String objectFactory : ObjectFactory context : JAXBContext |
| +marshal(Evaluatable, ContentHandler) +marshal(Evaluatable, File) +marshal(Evaluatable, Result) +marshal(Evaluatable, OutputStream) +marshal(Evaluatable, Writer) +marshal(Evaluatable, Node) +marshal(Evaluatable, XMLStreamWriter) +marshal(Evaluatable, XMLEventWriter) +unmarshal(File) : Evaluatable +unmarshal(InputStream) : Evaluatable +unmarshal(Reader) : Evaluatable +unmarshal(URL) : Evaluatable +unmarshal(InputSource) : Evaluatable +unmarshal(Node) : Evaluatable +unmarshal(Source) : Evaluatable +unmarshal(XMLStreamReader) : Evaluatable +unmarshal(XMLEventReader) : Evaluatable |

Figure 30: The PolicyConverter class.

The class `PolicyConverter` provides static methods to unmarshal from a given `Evaluatable-XML` to java data objects or marshal from java data objects to `Evaluatable-XML`. The `PolicyConverter` has a reference to the `ObjectFactory` in the `impl`-package to create the appropriate data objects.

Marshal methods

The marshal methods marshal a given concrete `Evaluatable` object into a selectable data sink.

Unmarshal methods

The unmarshal methods unmarshal from a selectable data source and returns a concrete `Evaluatable` object.



ObjectFactory The class `ObjectFactory` is responsible to instantiate and return the various policy data objects.

Evaluatable

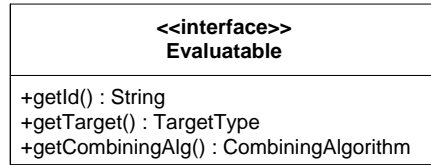


Figure 31: The `Evaluatable` interface.

The interface `Evaluatable` abstracts the classes `PolicyType` and `PolicySetType`.

2.2.2 Package `attributeFinder`

overview The package `attributeFinder` contains the `AttributeFinder` interface and its implementations.

This class is responsible to find attributes not contained in the request.

class-diagram

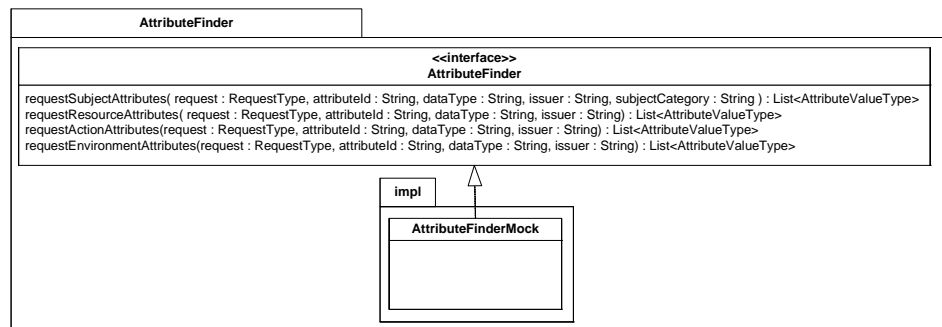


Figure 32: The `AttributeFinder` class diagram.

description

AttributeFinder

This interface is described in chapter 2.1.

AttributeFinderMock

The `AttributeFinderMock` is an `AttributeFinder` implementation that always acts as if no attribute was found. It always returns an empty list.



2.2.3 Package context

overview

The package context contains all the data classes to build XACML-requests and responses, as well as factories to create a `RequestCtx` or `ResponseCtx`. The `RequestCtx` and `ResponseCtx` and their factories are responsible to marshal or unmarshal the java objects to XML and backwards.

class-diagram

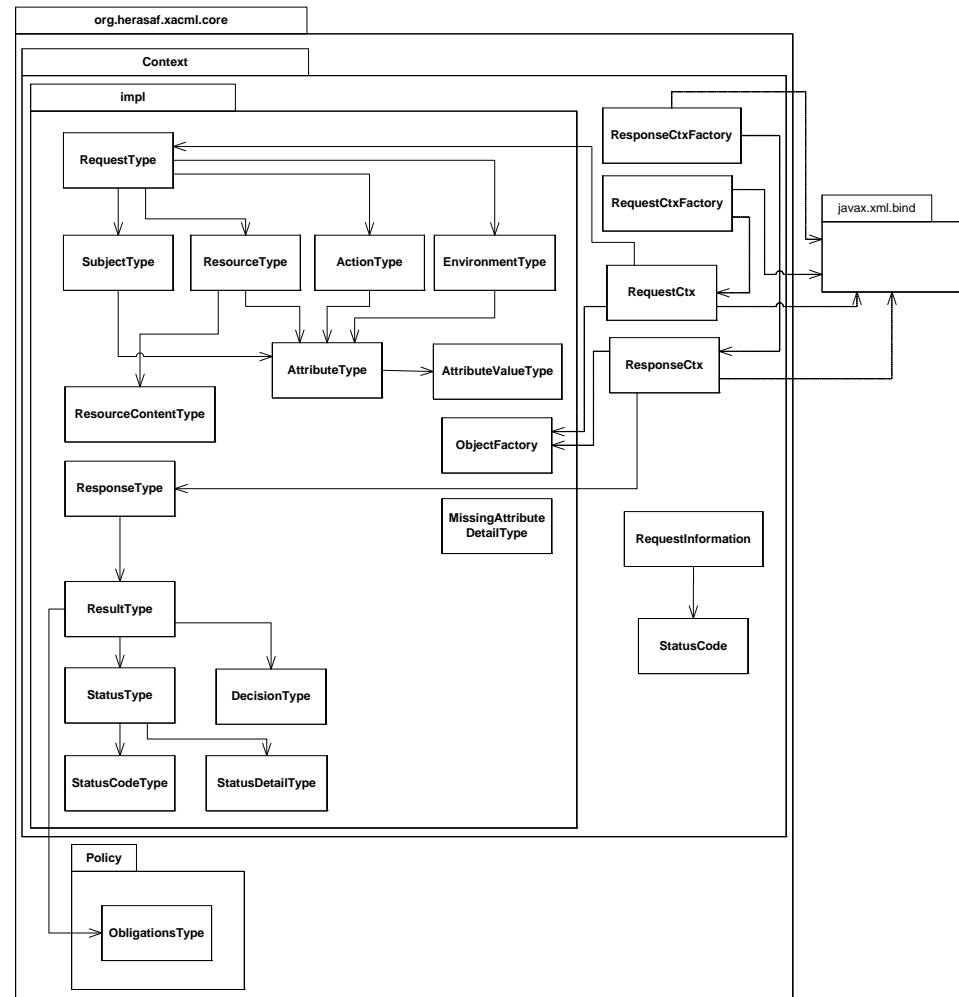


Figure 33: The AttributeFinder class diagram.

relations

Relation to the Obligations class

The reference from the `ResultType` to the package `Policy` is needed to include `Obligations` in a XACML response.

Relation to the package `javax.xml.bind`

The reference to the package `java.xml.bind` is needed for marshalling and unmarshalling.



2.2.3.1 Important classes

Overview

This chapter explains the most important classes and interfaces of the context package.

The context package contains the data classes needed to represent requests and responses, respectively as objects.

Factory classes

| ZZZCtxFactory |
|---|
| <u>-factory : ObjectFactory</u> <u>-ZZZCtx : JAXBProfile</u> |
| +unmarshal(File) : ZZZCtx +unmarshal(InputStream) : ZZZCtx +unmarshal(Reader) : ZZZCtx +unmarshal(URL) : ZZZCtx +unmarshal(InputSource) : ZZZCtx +unmarshal(Node) : ZZZCtx +unmarshal(Source) : ZZZCtx +unmarshal(XMLStreamReader) : ZZZCtx +unmarshal(XMLEventReader) : ZZZCtx |

Figure 34: The CtxFactory (ZZZ stands for Response or Request, respectively).

The ResponseCtxFactory additionally provides two methods for creating a ResponseCtx directly:

```
+create(RequestType, DecisionType, RequestInformation)
+create(RequestType, DecisionType, StatusCode)
```

The RequestCtxFactory has a create method for creating a RequestCtx:

```
+create(SubjectTransformable, ResourceTransformable,
ActionTransformable, EnvironmentTransformable)
```

These methods create a response to the given request with the given decision with the given status code.

Unmarshal methods

The unmarshal methods unmarshal from a selectable data source and return either a ResponseCtx or RequestCtx object, respectively.

Context classes

| ZZZCtx |
|--|
| <u>-factory : ObjectFactory</u> <u>-ZZZCtx : JAXBProfile</u> |
| +getZZZ() +marshall(ContentHandler) +marshall(File) +marshall(Result) +marshall(OutputStream) +marshall(Writer) +marshall(Node) +marshall(XMLStreamWriter) +marshall(XMLEventWriter) |

Figure 35: The Ctx (ZZZ stands for Response or Request, respectively).



Marshal methods

The marshal methods marshal the `ZZZType` object into a selectable data sink.

getZZZ()

Returns either the `ResponseType` or the `RequestType`.

ObjectFactory

The class `ObjectFactory` is responsible to instantiate and return the various context data objects.

2.2.4 Package converter

overview

The package converter contains all the necessary converter classes, which are needed during the marshalling or unmarshalling process. They convert the string representation of a function into the proper data object or backwards.

class-diagram

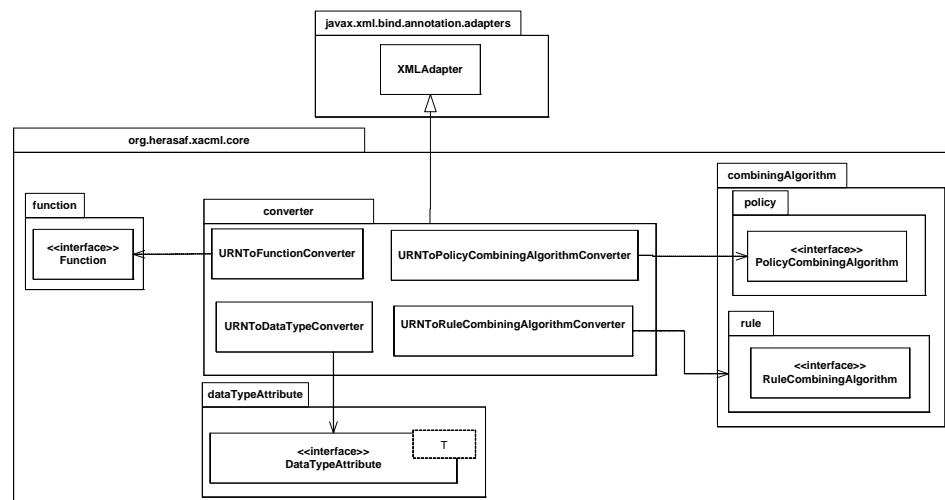


Figure 36: The converter package and its dependencies.

relations

Relation to package javax.xml.bind.annotation.adapters

The reference to this package is needed to extend the `XMLAdapter` class and use the classes as converters for JAXB.

Relation to the package combiningAlgorithm

The reference to this package is that the `Evaluatables` offer the appropriate combining algorithm interface.

Relation to the package function

The reference to this package is that a `Function` can be injected into an `Evaluatable`.



Relation to the package dataTypeAttribute

The reference to this package is that a `DataTypeAttribute` can be injected into an `Evaluatable`.

2.2.5 Package function

overview

The package `function` contains the interface `Function` as well as different concrete function implementations in sub packages. All the functions are defined by the XACML 2.0 specification [XACML20].

class diagram

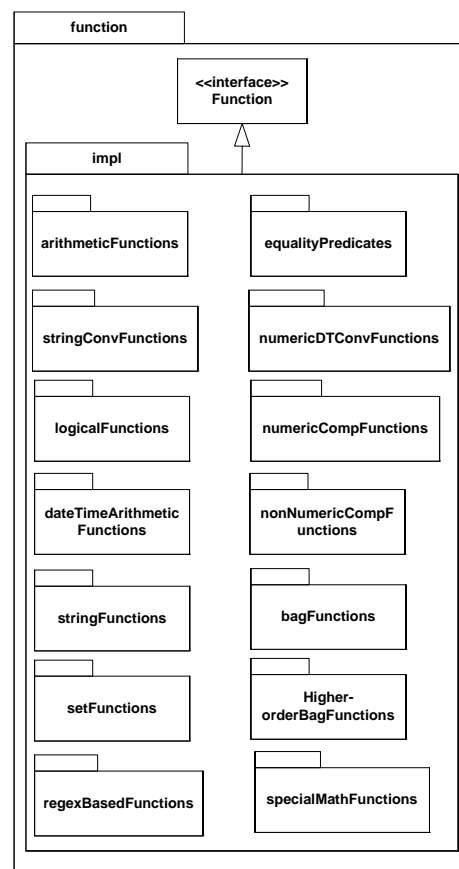


Figure 37: The function package.



2.2.6 Package combiningAlgorithm

overview

The package `combiningAlgorithm` contains two sub packages for policy combining- and rule combining algorithms. Each of them has an interface which has to be implemented by a concrete combining algorithm.

class diagram

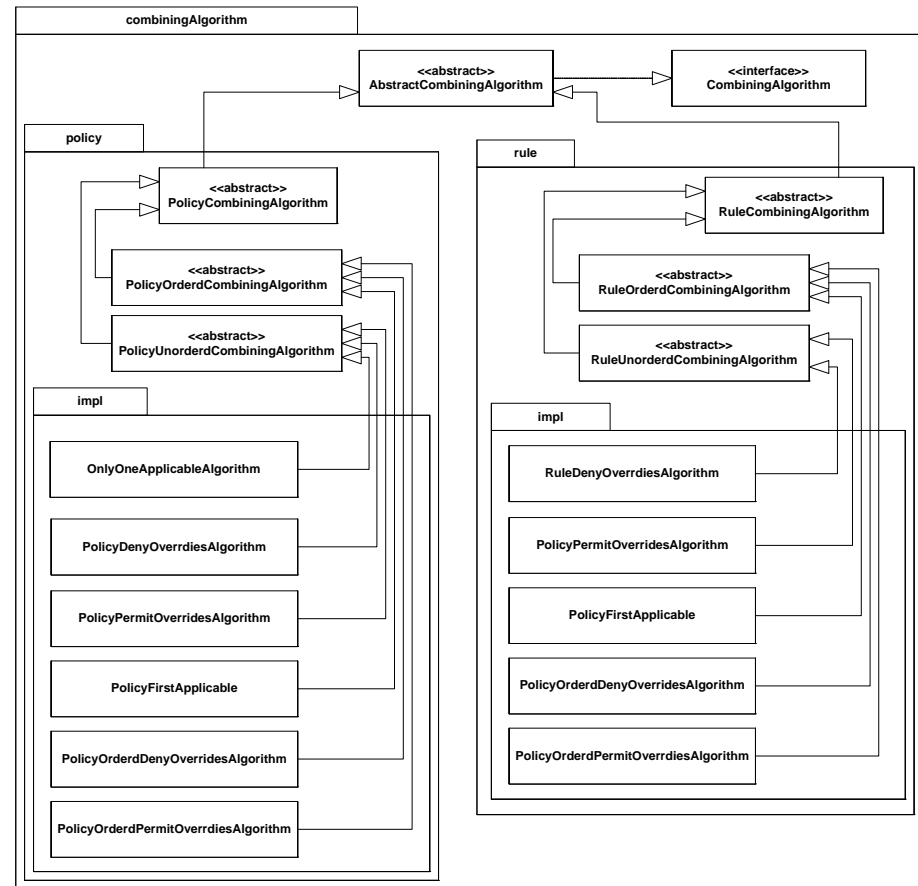


Figure 38: The CombiningAlgorithms class diagram and dependencies.

description

policy package

The `policy` package contains the combining algorithms injected into the policy sets for combining multiple policy evaluation results. The implementations are described in the XACML 2.0 specification [XACML20].

rule package

The `rule` package contains the combining algorithms injected into the policy sets for combining multiple rule evaluation results. The implementations are described in the XACML 2.0 specification [XACML20].

CombiningAlgorithm interface

This interface is the super type of all combining algorithms and provides the `evaluate`-method.

AbstractCombiningAlgorithm

This class is an abstract implementation of the `CombiningAlgorithm`



interface and adds the functionality for matching the target of an `Evaluatable` or a `Rule`.

OrderedandUnorderedCombiningAlgorithms

There are ordered and unordered combining algorithms. Implementations of an algorithm should extend the appropriate classes.

2.2.7 Package `dataTypeAttribute`

overview

The package `dataTypeAttribute` contains a generic interface `DataTypeAttribute`, as well as the concrete implementations of the data type attributes. These classes are responsible for converting the string representation of a data type into an object of the specific type.

class-diagram

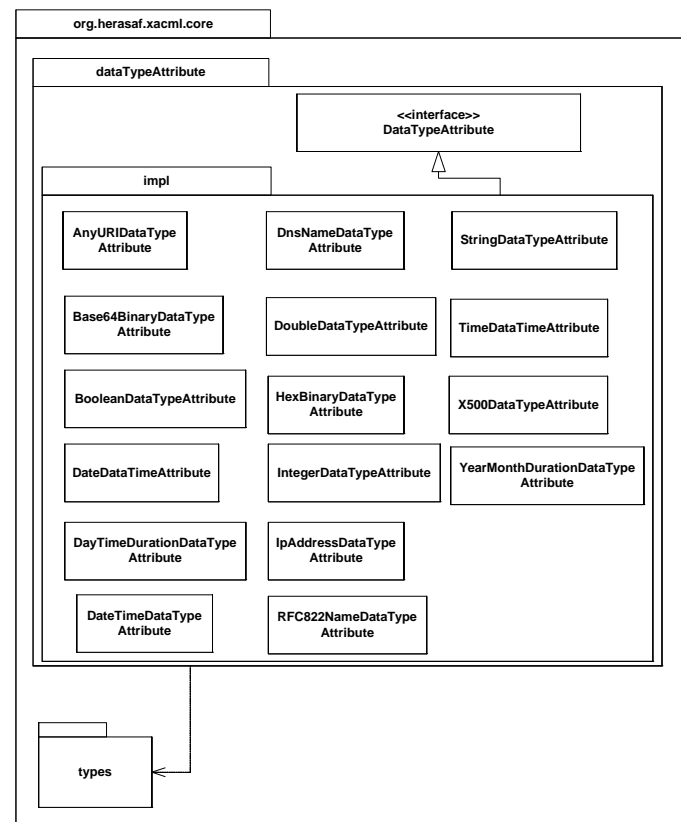


Figure 39: The `DataTypeAttribute` class diagram and dependencies.

relations

Relation to package `types`

The reference to the package `types` is needed to use the own defined types in some of the `DataTypeAttribute` implementations where no equivalent Java type is available.



2.2.8 Package types

overview

The package `types` contains the implementation of the types specified by the XACML 2.0 specification [XACML20].

These types are only needed in case where no Java equivalent is available otherwise it is recommended to use the Java data type.

class-diagram

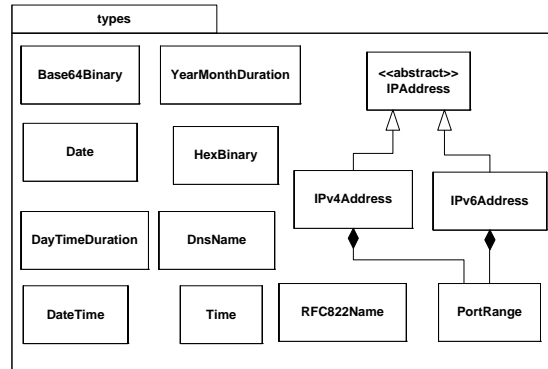


Figure 40: The data types class diagram.

2.2.9 Creating request- and response contexts

Overview

The core provides the possibility to create responses and requests without having a context XML.

Creating a response

The class `ResponseCtxFactory` is used to create a `ResponseCtx`. This class provides two methods to create a `ResponseCtx`.

```

+create(RequestType, DecisionType, RequestInformation)
+create(RequestType, DecisionType, StatusCode)
  
```

RequestType

The `RequestType` is the corresponding request to the response to create.

DecisionType

The `DecisionType` is the decision to include into this response.

RequestInformation

Provides the `StatusCode` to include into this response.

StatusCode

The `StatusCode` to include into this response.



Creating a request

The class `RequestCtxFactory` is used to create a `RequestCtx`. This class provides a `create` method to create a `RequestCtx` from scratch. This `create` method takes four arguments (`SubjectTransformable`, `ResourceTransformable`, `ActionTransformable`, `EnvironmentTransformable`). Each of these arguments is called a transformable and provides a `transform` method. These Transformables are needed because the source-format of the elements, such as `Subject`, is unknown. So the developer who wants to use this factory method must additionally implement these four Transformables which fit to his data. The `transform` method returns the created element or elements.

SubjectTransformable

Is responsible for transforming the subjects in any representation into a collection of `SubjectType`.

Example:

```
public class MySubjectTransformable implements SubjectTransformable {  
  
    private List<MySubjectType> mySubjects;  
  
    public MySubjectTransformable(List<MySubjectType> subjects) {  
        mySubjects = subjects;  
    }  
  
    public Collection<? extends SubjectType> transform() {  
        List<SubjectType> transformedSubjects = new ArrayList<SubjectType>();  
  
        for(MySubjectType myS : mySubjects){  
            transformedSubjects = doSomething(myS);  
        }  
        return transformedSubjects;  
    }  
  
    private List<SubjectType> doSomething(MySubjectType myS) {  
        // Some work to do  
    }  
}
```

Figure 41: Example of a Subject-Transformable.



ResourceTransformable

Is responsible for transforming the resources in any representation into a collection of `ResourceType`.

Example:

```
public class MyResourceTransformable implements ResourceTransformable {  
  
    private List<MyResourceType> myResources;  
  
    public MyResourceTransformable(List<MyResourceType> resources) {  
        myResources = resources;  
    }  
  
    public Collection<? extends ResourceType> transform() {  
        List<ResourceType> transformedResources = new ArrayList<ResourceType>();  
  
        for(MyResourceType myR : myResources){  
            transformedResources = doSomething(myR);  
        }  
        return transformedResources;  
    }  
  
    private List<ResourceType> doSomething(MyResourceType myR) {  
        // Some work to do  
    }  
}
```

Figure 42: Example of a Resource-Transformable.

ActionTransformable

Is responsible for transforming an action in any representation into an `ActionType`.

Example:

```
public class MyActionTransformable implements ActionTransformable {  
  
    private MyActionType myAction;  
  
    public MyActionTransformable(MyActionType action) {  
        myAction = action;  
    }  
  
    public ActionType transform() {  
        ActionType transformedAction = doSomething(myAction);  
        return transformedAction;  
    }  
  
    private ActionType doSomething(MyActionType myA) {  
        // Some work to do  
    }  
}
```

Figure 43: Example of an Action-Transformable.



EnvironmentTransformable

Is responsible for transforming an environment in any representation into an EnvironmentType.

Example:

```
public class MyEnvironmentTransformable implements EnvironmentTransformable {  
  
    private MyEnvironmentType myEnvironment;  
  
    public MyEnvironmentTransformable(MyEnvironmentType environment) {  
        myEnvironment = environment;  
    }  
  
    public EnvironmentType transform() {  
        EnvironmentType transformedEnvironment = doSomething(myEnvironment);  
        return transformedEnvironment;  
    }  
  
    private EnvironmentType doSomething(MyEnvironmentType myE) {  
        // Some work to do  
    }  
}
```

Figure 44: Example of an Environment-Transformable.

2.2.10 Marshalling and Unmarshalling

| | |
|----------------------|--|
| <i>Overview</i> | HERAS ^{AF} provides various possibilities for marshalling and unmarshalling requests and responses. These possibilities are explicitly not listed up. Refer to the API documentation for further information. |
| <i>Marshalling</i> | The marshalling methods are in the RequestCtx and ResponseCtx themselves. The marshalling methods for an Evaluatable are in the PolicyConverter. |
| <i>Unmarshalling</i> | The unmarshalling methods are in the RequestCtxFactory and ResponseCtxFactory. The unmarshalling methods for an Evaluatable are in the PolicyConverter. |



2.2.11 Design Decisions

Overview This chapter explains the most important design decisions, made for the core.

2.2.11.1 Transformables

Issue How to create a `RequestCtx` without having the information provided through an XML request.

Alternatives

Alternative 1: Transformer that cannot transform itself

A **Transformer** interface with a `transform` method that takes for arguments is implemented. These arguments are **Transformables** for the elements' subject, resource, action and environment.

This **Transformer** is passed to the `create` method of the **RequestCtxFactory**.

It is up to the developer how to implement the concrete classes. They are fully adaptable to the requirements of the domain.

Advantages

Disadvantages

- | | |
|--|---|
| <ul style="list-style-type: none"> • More than one <code>Transformer</code> for the factory is possible | <ul style="list-style-type: none"> • The <code>Transformer</code> object itself is overhead because it could contain the data itself |
|--|---|

Alternative 2: The RequestCtxFactory takes the values directly

The `RequestCtxFactory` has a method that takes the domain specific values for creating the subject, resource, action and environment elements.

Advantages

Disadvantages

- | | |
|---|--|
| <ul style="list-style-type: none"> • The factory interface is clear and understandable | <ul style="list-style-type: none"> • The factory is bound to a specific implementation which is not exchangeable in an easy way |
|---|--|

Alternative 3: Transformables that can transform themselves

The `create` method of the `RequestCtxFactory` takes four `Transformables`, one for each element (subject, resource, environment and action). These `Transformables` have a `transform` method and can transform themselves.

Advantages

Disadvantages

- | | |
|---|--|
| <ul style="list-style-type: none"> • More than one <code>Transformable</code> for each element is possible | <ul style="list-style-type: none"> • none |
|---|--|



Alternative 4: Information objects

For each element (subject, resource, action and environment) exists a corresponding information object that must be passed to the `create` method of the `RequestCtxFactory`.

Advantages

- The factory interface is clear and understandable

Disadvantages

- The factory is bound to a specific implementation which is not exchangeable in an easy way

Decision

The choice fell on solution 3 because with these **Transformable** objects the exchangeability and adaptability to a specific domain is fully provided. The advantage to solution 2 is that the needless **Transformer** interface is eliminated.

3 PDP

Overview

This chapter contains the description of the PDP interfaces and the implementation provided by HERAS^{AF}.

Structure

This chapter contains the description of the Policy Decision Point (PDP). First of all the interfaces are described. Afterwards the implementation in HERAS^{AF}.

3.1 Interfaces

Overview

This chapter describes the interfaces of the PDP.

PDP interface

The `PDP` interface is the façade of the PDP module. It provides the functionality of a Policy Decision Point.

The PDP interface

| PDP |
|---|
| <pre>+evaluate(RequestCtx) : ResponseCtx +getEvaluatable(EvaluatableID) : Evaluatable +deploy(Collection<Evaluatable>) : void +deploy(Evaluatable) : void +undeploy(EvaluatableID) : void +undeploy(Collection<EvaluatableID>) : void +undeployAll() : void</pre> |

Figure 45: The PDP interface.

evaluate method

Evaluates the given `RequestCtx` and returns an appropriate `ResponseCtx`. This method is used by a PEP for instance to evaluate a request.



*getEvaluatable
method*

Returns an `Evaluatable` with the given `EvaluatableID`.
This method is used by another PDP for instance to retrieve a referenced `Evaluatable`.

deploy method

Deploys the single `Evaluatable` or the collection of `Evaluatables` to the PDP.
This method is used by a PAP for instance to deploy a new `Evaluatable`.

*undeploy
method*

Undeploys the single `Evaluatable` or the collection of `Evaluatables` from the PDP.
This method is used by the PAP for instance to undeploy `Evaluatables`.

*undeployAll
method*

Undeploys all `Evaluatables` from the PDP.
This method is used by the PAP for instance to undeploy all `Evaluatables`.

EvaluatableID interface

This interface encapsulates the id of an `Evaluatable` for accessing the PDP.

*The
EvaluatableID
interface*

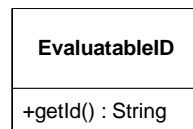


Figure 46: The EvaluatableID interface.

getId method

Returns the ID in its string representation.



3.2 HERAS^{AF} XACML 2.0 implementation

Overview

This chapter shows the realization of the PDP in the HERAS^{AF} XACML 2.0 implementation.

Important hint

The HERAS^{AF} does not support hot deployment of Evaluatables. The consequences are that after deploying or undeploying Evaluatables, the PDP must be restarted to be aware of the new state. This originates from the fact that the used index is not thread-safe for changing its content, only for reading. This means that evaluating would not be possible while recreating the index.

class diagram

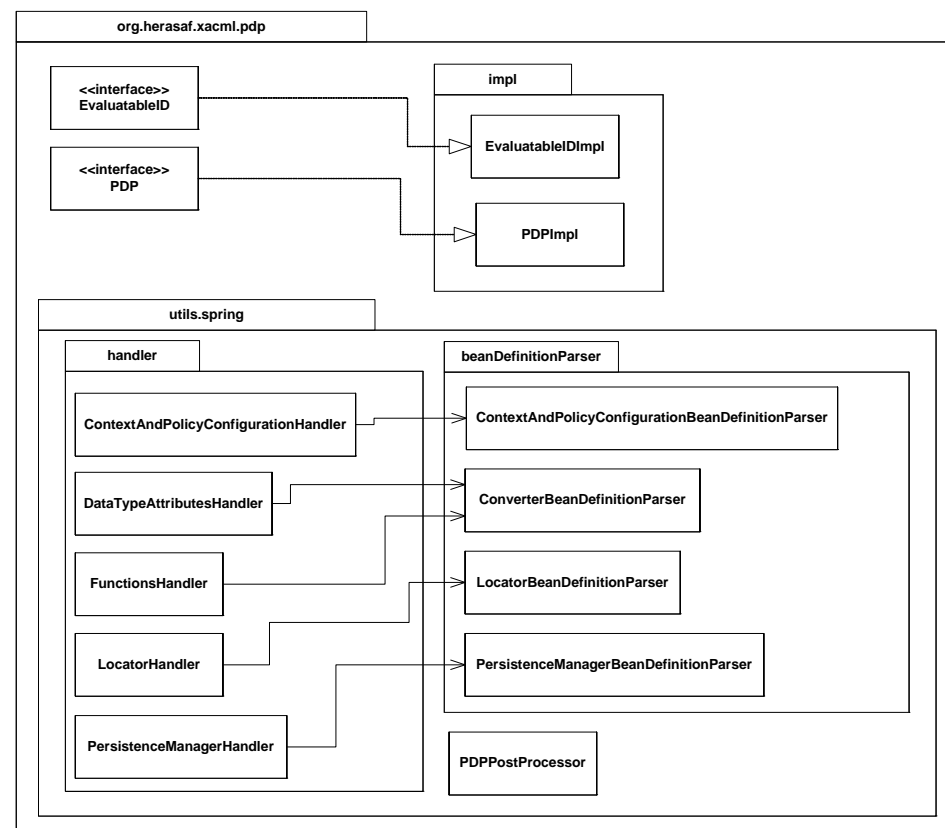


Figure 47: The PDP class diagram.

Package descriptions

org.herasaf.xacml.pdp

This is the main package containing the Policy Decision Point interfaces and its implementation.

org.herasaf.xacml.pdp.impl

This package contains the concrete implementation of the PDP.

org.herasaf.xacml.pdp.util.spring.handler

This package contains the handlers used by the Spring framework to register the BeanDefinitionParsers.



org.herasaf.xacml.pdp.utils.spring.beanDefinitionParser

This package contains the parsers used by the Spring framework to parse the own bean definition namespaces (Spring's Extensible XML Authoring).

Interfaces The interfaces are described 3.1.

Classes

PDPImpl

The implementation of the PDP interface.

EvaluatableIDImpl

The implementation of the `EvaluatableID` interface.

ContextAndPolicyConfigurationHandler

Registers the XML element `contextAndPolicy` to the `ContextAndPolicyConfigurationBeanDefinitionParser`.
The `contextAndPolicy` element contains the configuration for the `RequestCtx`, `ResponseCtx` and their factories.

DataTypesAttributesHandler

Registers the XML element `dataTypeAttributes` to the `DataTypeAttributesBeanDefinitionParser`.
The `dataTypeAttributes` element contains the data type attributes to load while PDP initialization.

FunctionsHandler

Registers the XML element `functions` to the `FunctionsBeanDefinitionParser`.
The `functions` element contains the functions to load while PDP initialization.

LocatorHandler

Registers the XML element `locator` to the `LocatorBeanDefinitionParser`.
The `locator` element contains the configuration for the `Locator`.

PersistenceManagerHandler

Registers the XML element `persistenceManager` to the `PersistenceManagerBeanDefinitionParser`.
The `persistenceManager` element contains the configuration for the `PersistenceManager`.

ContextAndPolicyConfigurationBeanDefinitionParser

Parses the `contextAndPolicy` element and its subelements and extracts the configuration parameters which will be passed to the `ContextAndPolicyConfiguration` bean.

DataTypesAttributesBeanDefinitionParser

Parses the `dataTypeAttributes` element and its subelements and extracts the location of `DataTypeAttributes` available in the classpath. This `DataTypeAttributes` will be loaded, instantiated and passed to the `URNToDataTypeAttributesConverter`.



FunctionsBeanDefinitionParser

Parses the `functions` element and its subelements and extracts the location of Functions available in the classpath.,This Functions will be loaded, instantiated and passed to the `URNToFunctionConverter`.

LocatorBeanDefinitionParser

Parses the `locator` element and its subelements and extracts the configuration parameters which will be passed to the `Locator` bean.

PersistenceManagerBeanDefinitionParser

Parses the `persistenceManager` element and its subelements and extracts the configuration parameters which will be passed to the `PersistenceManager` bean.

3.3 Sequence diagrams

Overview

This chapter explains the most important sequences in the PDP module.

Creation of the PDP

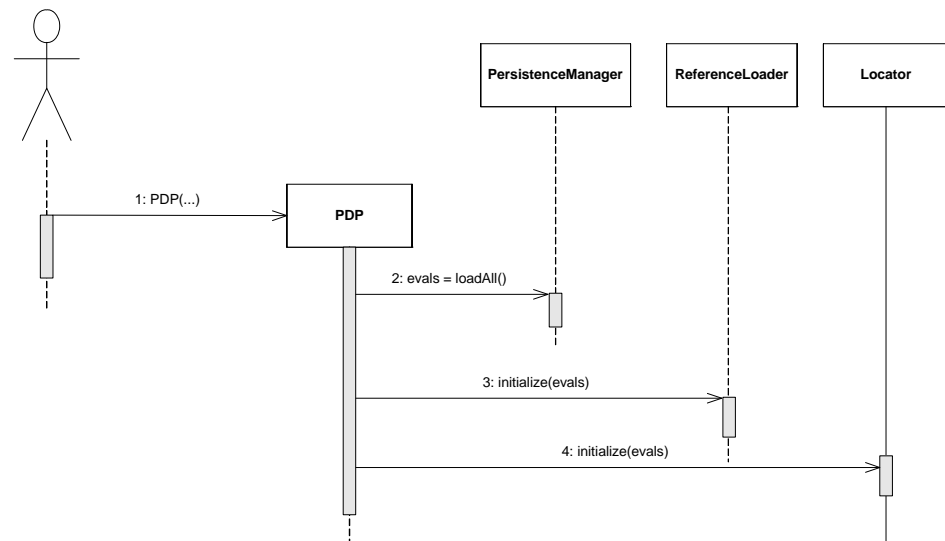


Figure 48: The initialization of the PDP.

Description

1. The PDP is created.
2. The PDP retrieves all `Evaluatables` from the persistence layer.
3. The PDP provides these `Evaluatables` to the `ReferenceLoader` which resolves the local references.
4. The PDP provides these `Evaluatables` to the `Locator` which initializes its index with them.



Request evaluation

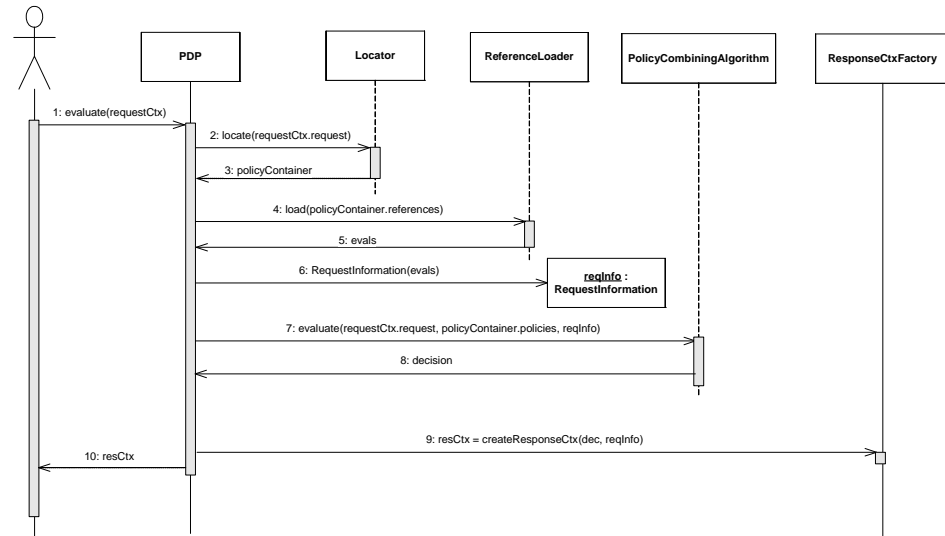


Figure 49: The evaluation of a request.

Description

1. The `evaluate` method of the PDP is called.
2. The PDP calls the `locate` method on the `Locator` with a `RequestType` as parameter.
3. The `Locator` returns a `PolicyContainer` which includes potential policies and policy references.
4. The PDP calls the `load` method of the `ReferenceLoader` to resolve the policy references in the `PolicyContainer`.
5. The `ReferenceLoader` returns a `List` with the resolved `Evaluatables`.
6. The PDP creates a new `RequestInformation` object with a list of `Evaluatables` as parameter.
7. The PDP calls the `evaluate` method on the `PolicyCombiningAlgorithm` with the request, the policies contained in the `PolicyContainer` and the `RequestInformation` object as parameter.
8. After evaluating, the `PolicyCombiningAlgorithm` returns a `decision`.
9. With the `ResponseCtxFactory`, a new `ResponseCtx` will be created.
10. The created `ResponseCtx` will be returned.



4 Persistence

Overview The Persistence module is responsible to persist and load the `Evaluatables` which are deployed onto the PDP. This is necessary that after a shutdown or reboot of the PDP the `Evaluatables` are available without a redeploying from the PAP. While starting up, the `Evaluatables` are automatically loaded into the memory of the PDP and after deploying they will be persisted onto the database.

structure This chapter contains the interface and the HERAS^{AF} description.

4.1 Interfaces

Overview The Persistence contains one interface, the `PersistenceManager`. It abstracts the implementation of the persistence mechanism. So the developer is free to exchange the whole persistence layer without affecting the rest of the application.

PersistenceManager Interface

The `PersistenceManager` interface is the façade of the Persistence module. The class implementing this interface can use one of various possibilities to persist and load `Evaluatables`.

The interface

| |
|--|
| <<interface>> PersistenceManager |
| + loadAll() : List<Evaluatable> + load(id : String) : Evaluatable + persist(evaluatable : Evaluatable) : void + persistAll(evaluatables : Collection<Evaluatable>) : void + delete(id : String) : void + deleteAll() : void |

Figure 50: The `PersistenceManager` interface.

loadAll method Loads all persisted `Evaluatables` from the underlying persistence layer.

load method Loads the `Evaluatable` with the given id from the underlying persistence layer.

persist method Persists the given `Evaluatable` to the underlying persistence layer.

persistAll method Persists all given `Evaluatables` to the underlying persistence layer.

delete method Deletes the `Evaluatable` with the given id from the underlying persistence layer.

deleteAll method Deletes all `Evaluatables` from the underlying persistence layer.



4.2 HERAS^{AF} XACML 2.0 implementation

Overview In this chapter the HERAS^{AF} XACML 2.0 implementation will be discussed. First, the sequences are explained. Second the design decisions made, are listed up.

Database access Because of performance and simplicity reasons the database is accessed with Spring JDBC Templates and the Evaluatables are stored as BLOBs. See the design decisions for further information.

4.2.1 Sequence diagram

Overview This chapter explains the important sequence in the Persistence module. It is a generalization of many sequences which all have the same flow.

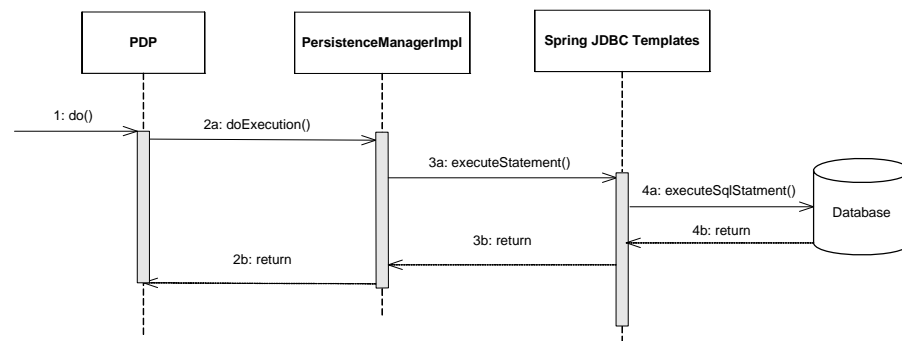


Figure 51: Interaction with the PersistenceManager and the attached database.

- Description*
1. An object calls a method on the PDP telling to persist or load Evaluatables.
 2. The PDP calls the right method (listed above in the interface description) on the PersistenceManager (2.a.). The PersistenceManager returns an Evaluatable or a list of Evaluatables if there is a return value (2.b.).
 3. The appropriate method on the Spring JDBC Template is called (3.a.). If there is a result it is returned (3.b.).
 4. The call will be executed on the underlying database (4.a.) and if there is a result it will be returned (4.b.).



4.2.2 Class diagram

Overview This chapter explains the class diagram of the persistence module.

Class diagram

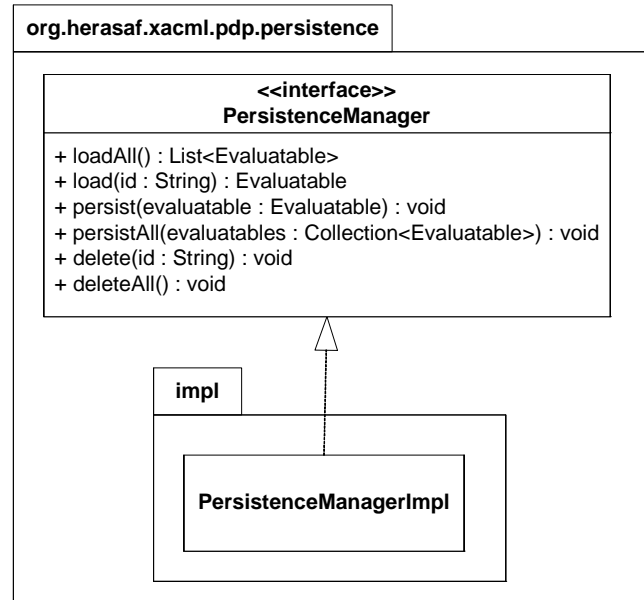


Figure 52: The PersistenceManager class diagram.

Description

org.herasaf.pdp.persistence.PersistenceManager

The `PersistenceManager` interface is described in chapter 4.1

org.herasaf.pdp.persistence.impl.PersistenceManagerImpl

The implementation of the `PersistenceManger` interface persists and loads the `Evaluatables` from a database. It prepares the SQL statements for the database. The database access is done by the Spring JDBC Templates.

4.2.3 Design decisions

Overview This chapter explains the most important design decisions, made in the Persistence module.

4.2.3.1 Plain Java object serialization

Overview The analyzing of the various possibilities to persist `Evaluatables` is handled in [HPDPWSE].

Decision The possibility “Plain Java object serialization” was chosen because of the following reasons:

Persisting and loading is very fast because no abstraction-layer is in between.

No additional libraries must be used.



`Evaluatables` are persisted for a specific implementation of a PDP therefore no interoperability must be assured.

`Evaluatables` are persisted, if an update is necessary the affected `Evaluatable` is replaced, so no possibility for updating single values in an `Evaluatable` must be provided.

5 Locator

Overview

To minimize the response time of a Policy Decision Point, indexing the `Evaluatables` is an important point. The `Locator` module implements the indexing of `Evaluatables`.

If the Policy Decision Point has to evaluate a request, it calls the `Locator` to return the essential `Evaluatables` for this request.

It also contains an index over the `Evaluatable`-identifier to respond to a request for a policy specified by its identifier. Such a request will normally be performed by a remote PDP.

5.1 Interfaces

Overview

The `Locator` Module contains two interfaces which abstract the concrete implementation.

The `Locator` interface is the façade interface of the module.

The `IndexHandler` interface is an interface of the HERAS^{AF} XACML 2.0 implementation. It provides the possibility to use another indexing algorithm with the same restrictions made for the HERAS^{AF} XACML 2.0 implementation.

Locator Interface

The `Locator` interface is the façade of the `Locator` module. A class implementing this interface can be used as alternative implementation of the locator. *Part I* explains how the PDP Module can be configured to use an alternative implementation.

The interface

| |
|---|
| <<interface>> Locator |
| + locate(request : RequestType) : PolicyContainer + initialize(evals : List<Evaluatable>) : void |

Figure 53: The `Locator` interface.

locate method

When the PDP has to evaluate an `Evaluatable`, it calls this method to get the `Evaluatables` and references to `Evaluatables` essential for this request.

The argument is the request to which the `Evaluatables` should be returned. The `Evaluatables` are returned in a `PolicyContainer` which contains the located `Evaluatables` and the references.

It has to be made sure that all `Evaluatables` are returned, that might take an effect on the evaluation result. `Evaluatables` that do not affect the result just slow down the decision process, but are no serious threat to the PDP.

initialize method

This method is called to initialize the locator with `Evaluatables`. The argument is a list of `Evaluatables` that should be indexed.



IndexHandler Interface

The `IndexHandler` interface is used in the HERAS^{AF} XACML 2.0 implementation and is a façade for the implementation of the index algorithm. An `IndexHandler` indexes the policies with the attributes specified in the `IndexAttributes`. It may consist of different index implementations for the attributes.

The HERAS^{AF} XACML 2.0 implementation contains two index algorithms, one for Comparable data types and one for policies with a target match function containing a regular expression or an equal function.

| <<interface>> IndexHandler |
|---|
| + <code>initializeIndex(List<Evaluatable>) : void</code> + <code>locate(subject : String, resource : String, action : String, environment : String) : PolicyContainer</code> + <code>setSubjectIndexAttribute(subjectAttribute : IndexAttribute) : void</code> + <code>setResourceIndexAttribute(resourceAttribute : IndexAttribute) : void</code> + <code>setActionIndexAttribute(actionAttribute : IndexAttribute) : void</code> + <code>setEnvironmentIndexAttribute(environmentAttribute : IndexAttribute) : void</code> |

Figure 54: The `IndexHandler` interface.

initializeIndex method

The `initializeIndex` method is called after the `IndexAttributes` have been set. The list in the parameter contains all `Evaluatables` that should be indexed. This method is only called once.

locate method

This method is called when the possibly matching `Evaluatables` for a request have to be found. It is called for every combination of subject, resource, action and environment in the request. The returned `PolicyContainer` should contain the found `Evaluatables` and the references to `Evaluatables` contained in these.

setXXIndex- Attribute method

The methods `setSubjectIndexAttribute`, `setResourceIndexAttribute`, `setActionIndexAttribute`, `setEnvironmentIndexAttribute` are used to set the attributes to the `IndexHandler`. They must be called before the `initializeIndex` method is called.



5.2 HERAS^{AF} XACML 2.0 implementation

Overview

In this chapter the HERAS^{AF} XACML 2.0 implementation will be discussed. First the sequences are explained. Afterwards follows an explanation of the class diagram and the index algorithms. At last, a discussion of the design decision made in this module.

5.2.1 Sequence diagrams

Overview

This chapter explains the most important sequences in the Locator module.

Initialization overview

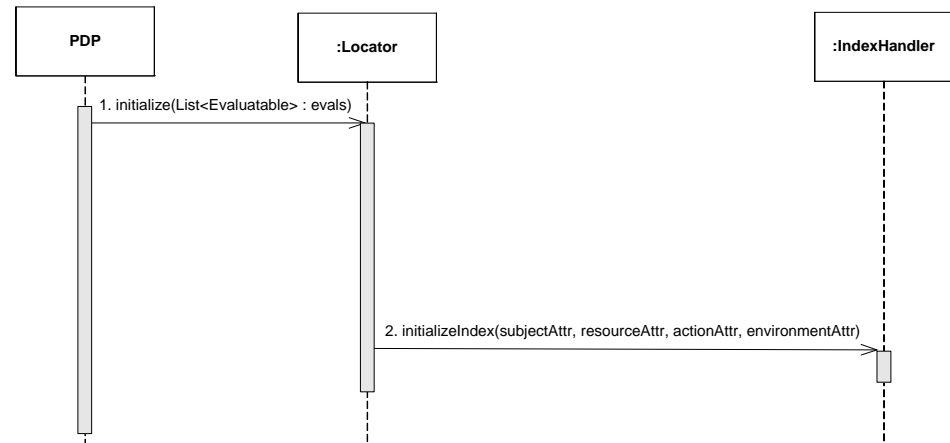


Figure 55: The initialization of the Locator.

Description

1. The PDP runs the initialization of the locator.
2. The Locator runs the initialization on the index handler.



Initializing the IndexHandler

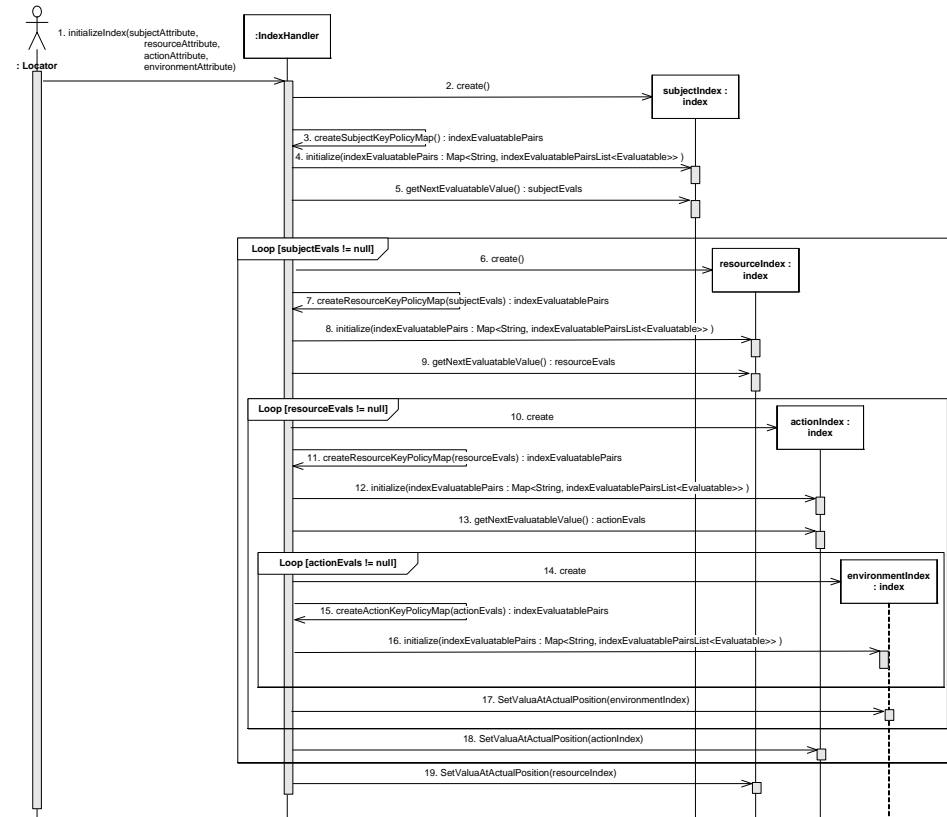


Figure 56: The initialization of the IndexHandler.

Description

- 1) The Locator starts the initialization of the IndexHandler.
- 2, 6, 10, and 14) A new index is created. The index implementation is specified in the Attribute Subject.
- 3, 7, 11, and 15) The IndexHandler generates a map.
- 4, 8, 12 and 16) The Index is initialized with the map. The key of the map is the value that should be indexed and the value contains the Evaluatables that match to this indexed value.
- 5, 9 and 13) The Indexhandler loads the next PolicyContainer of the Index. Then the containing Evaluatables are indexed.
- 17-19) The created index is set into the index at the position the PolicyContainer was.



Locating an Evaluatable

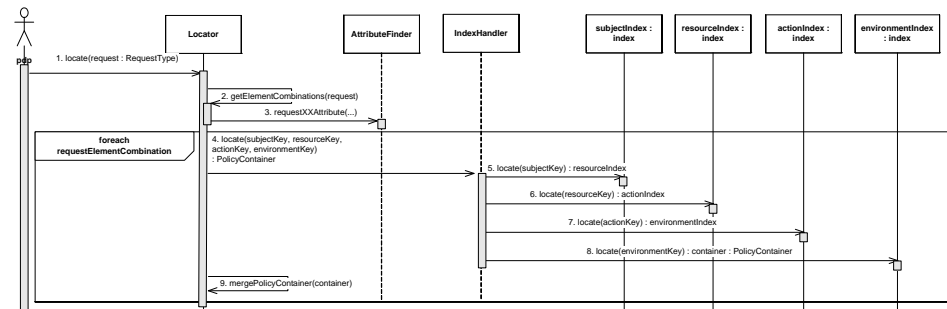


Figure 57: The locating of an Evaluatable.

Description

1. The PDP asks for the indexed *Evaluatable* for the request.
2. The *Locator* searches all possible combinations of the indexed attributes from the request.
3. If the request does not contain an attribute which is indexed, the *AttributeFinder* is called to find the attribute at the *Policy Information Point*.
4. The *locate* method of the *IndexHandler* is invoked.
5. The *IndexHandler* calls the *locate* method on the *SubjectIndex*. The *SubjectIndex* returns a *ResourceIndex* indexed found by the *subjectKey*.
6. The *IndexHandler* calls the *locate* method on the *ResourceIndex*. The *ResourceIndex* returns the *ActionIndex* found by the *resourceKey*.
7. The *IndexHandler* calls the *locate* method on the *ActionIndex*. The *ActionIndex* returns the *EnvironmentIndex* found by the *actionKey*.
8. The *IndexHandler* calls the *locate* method on the *EnvironmentIndex*. The *EnvironmentIndex* returns the *PolicyContainer* found by the *environmentKey*.
9. The *Locator* merges the returned *PolicyContainers* with all other *PolicyContainers* returned for the other *locate* calls of the same request.



5.2.2 Class diagram

Overview

This chapter explains the classes of the Locator module.

Class Diagram

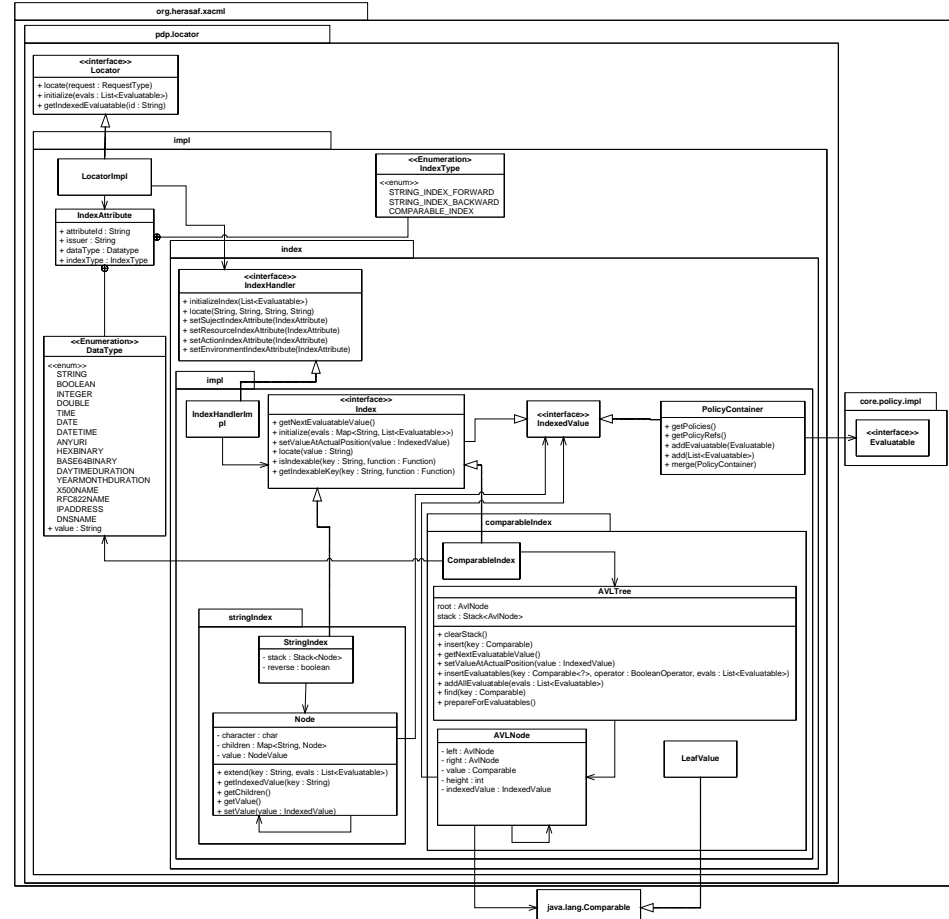


Figure 58: The Locator class diagram and dependencies.

Description

org.herasaf.xacml.pdp.locator

Contains the interface of the Locator module. This interface is described in chapter 5.1

IndexAttribute

The IndexAttribute class contains the indexing information of the configuration file for a subject, resource, action or environment element.

IndexType

The IndexType is an enumeration for the type of index that should be created for the IndexAttribute.

DataType

The DataType is an enumeration for the data type that should be indexed for the IndexAttribute.



org.herasaf.xacml.pdp.locator.impl.index

Contains the `IndexHandler` interface. This is described in chapter 5.1.

org.herasaf.xacml.pdp.locator.impl.index.impl

Contains the `IndexedValue` interface. This encapsulates an `Index` or a `PolicyContainer`. It is used to make it possible that an index can have another `Index` or a `PolicyContainer` as value.

Index

Interface for an `Index` implementation.

PolicyContainer

A `PolicyContainer` contains `Evaluatables` and the references contained in them to other `Evaluatables`.

org.herasaf.xacml.pdp.locator.impl.index.impl.stringIndex

The `StringIndex` implements an `Index` which can index equal functions and regular expression functions. Regular expression functions have the restriction that only `.*xx` and `xx.*` can be indexed. A detailed description of the `StringIndex` algorithm can be found in chapter 5.2.3.1.

Node

A `Node` represents a node in the `StringIndex` tree. The `Nodes` build themselves recursively. A detailed description of the `StringIndex` algorithm can be found in chapter 5.2.3.1.

org.herasaf.xacml.pdp.locator.impl.index.impl.comparableIndex

The `ComparableIndex` implements an index for data types implementing the `Comparable` Interface. A detailed description of the `ComparableIndex` algorithm can be found in chapter 5.2.3.2.

AVLTree

The AVL tree used by the `ComparableIndex`.

AVLNode

A node in the AVL tree.

LeafValue

A `LeafValue` represents the values in the leaf nodes of the `AVLTree`. They are special because they match every value. This is important because values that are not indexable have to be found in the index, too. These values are attached to the leaves.



5.2.3 Indexing algorithm

Overview The HERAS^{AF} XACML 2.0 implementation implements two Evaluatable indexing algorithms. This chapter explains how they work, the possibilities of the algorithms and how they are implemented.

The Challenge The challenge in finding an indexing algorithm is that the Evaluatable has to be indexed and not the request. This means that the search pattern is indexed and searched by a value matching the pattern. Hence “normal” indexing algorithms are useless in indexing policies. The XACML 2.0 specification specifies the target match of an Evaluatable in a way that it can match in nearly every possible way. OR-relationships between attributes are possible, as well as AND-relationships. Another point is that the data type of each attribute can differ to the other attributes. This makes it really difficult, if not impossible, to find a fast indexing algorithm.

The solution In the HERAS^{AF} XACML 2.0 implementation this problem is solved by reducing the target match possibilities in a way that the Evaluatables can be indexed by the implemented algorithms. In respect of the fact that different companies define their subjects, resources, actions or environment in different ways, it is possible to configure the index. For each element of the Target, the applied indexing algorithm can be configured. This is solved by chaining the indexes for the different parts (see figure 59). For every indexed value in the subject index exists a resource index, and so on.

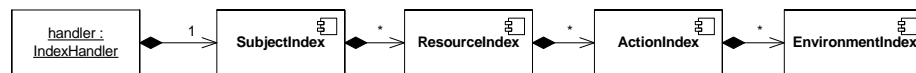


Figure 59: The Index chain.

To ensure that Evaluatables which cannot be indexed are still returned by the index, they are included in every search result on the index.

To get a faster index, two different algorithms are defined.

One for Evaluatables that match values with an equal, greater or less value and another one for Evaluatables matching the string representation with equal- or regular expression functions.

5.2.3.1 String Index

Overview The `StringIndex` is used for Evaluatables with a target matching with equal- or regular expression functions.

That means that all data types are indexed in their string representation. This leads to the fact that every data type first is converted into its string representation.

Regular expressions are difficult to index. Finding an indexing algorithm to index as much regular expressions as possible would have been too much work for this diploma thesis. This is the reason why the indexing algorithm has hard restrictions on indexing regular expressions.

Indexable regular expressions The chosen indexing algorithm for regular expressions is able to index any string with a defined amount of characters in the middle of a string or an undefined amount of undefined characters at the beginning or at the end of the regular



expression.

Because of lack of time and the fact that undefined characters at the beginning or the end of a match-string are the most used expressions the implementation is only able to handle them.

*forward-
reverse mode*

The `StringIndex` can be configured to handle regular expressions with wild cards at the beginning (backward index) or at the end (forward index) of the match-string.

Algorithm

The `StringIndex` is a tree with multiple child nodes per node. Every node in the tree represents a character or wildcard.

Starting with the root node the nodes are linked in a way that they build the sought string.

The indexing algorithm goes through the tree. If it is not possible to go ahead because the tree does not contain the sought value, the wildcard-node is returned from that point where the matching ends.

Example:

The tree contains the path `b -> u -> g -> *` (Hint: `b` and `u` also have a wildcard-node attached).

Now if the search-string is "buggy" the algorithm walks through the tree until it reaches the "g". Then the wildcard-node is returned because this node contains all `Evaluatables` that match to "bug" and an appendix. That means that in the wildcard-node an `Evaluatable` matching to "buggy" is contained (and all not indexable `Evaluatables`).

*Searching a
value*

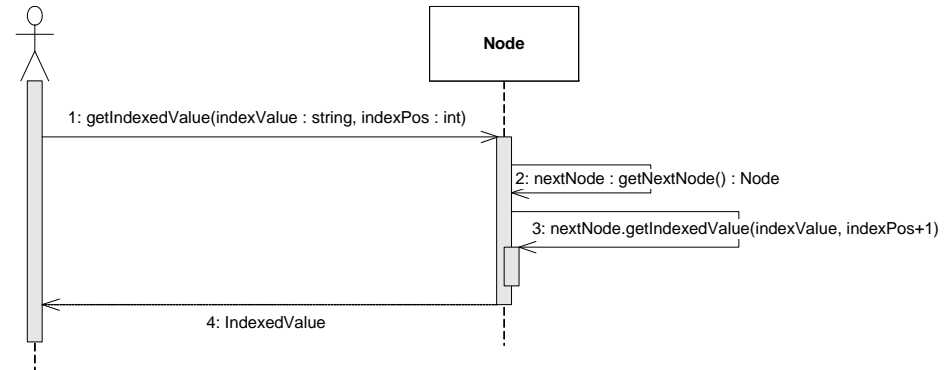


Figure 60: Searching a value in the `StringIndex`.

1. The `getIndexedValue` method is called. The `indexValue` is the sought value and the `indexPos` is a pointer to the actual search position in the searchValue.
2. The node searches the node representing the following character in the search string. If no such node is present, the wildcard node is the next node.
3. The node calls the `getIndexedValueNode` on the `nextNode`.
4. The indexed value is returned.



building the index

inserting the Evaluatables

To build a `StringIndex`, the different key-Evaluatable pairs are inserted one after another. The `StringIndex` calls the `append` method on the root node. Beginning with the root node, the nodes call themselves recursively. A node is implemented like this:

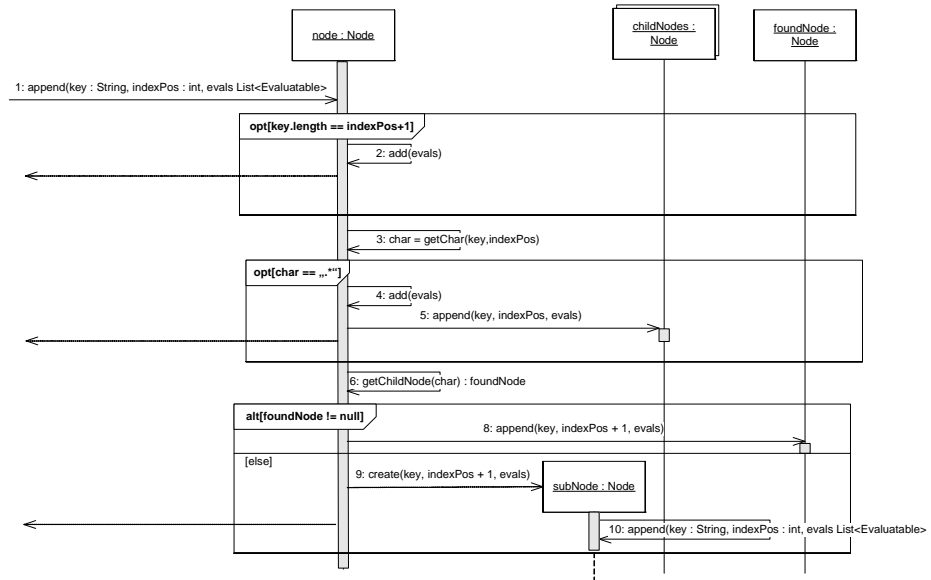


Figure 61: Building the StringIndex.

1. The `append` method is called on a node.
2. If the `indexPos` points to the end of the key, the `Evaluatables` are added to the `Node`.
3. The actual character in the index is read and held in the `char` variable.
4. If the character is "." the `Evaluatables` are added to the node.
5. The `Evaluatables` are added to all child nodes.
6. The child node representing the `char` is sought.
7. If a child node was found, call the `append` command on this child node and increment the `indexPos` integer by 1.
8. If the child node was not found, create a new child node.
9. The child node calls the `append` command to itself.



locating an indexed Value

To locate an IndexedValue, the StringIndex calls the `getIndexedValue` method of the root node. The nodes call themselves recursively to get the IndexedValue.

The logic of a node looks like this:

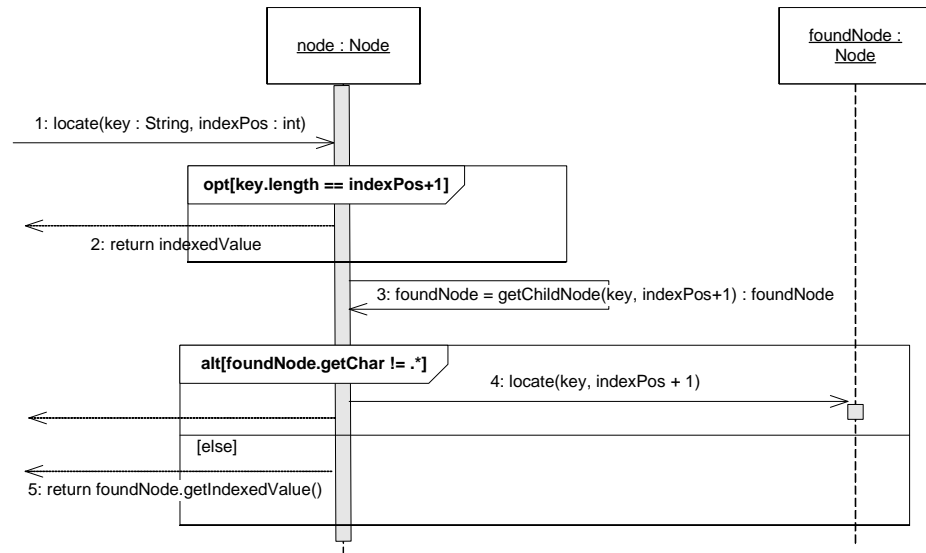


Figure 62: Locating an indexed value.

1. The `locate` method on the node is called.
2. If the `indexPos` points to the end of the key, return the `indexedValue` of this node.
3. Search the node representing the next key. If no node exists, the node for every value is returned.
4. If the returned node is not the node for every value, call the `locate` method on this node.
5. If the returned node is the node for every value, return it is `IndexedValue`.



5.2.3.1.1 Examples

Overview This chapter gives two examples about how a `StringIndex` might look like.

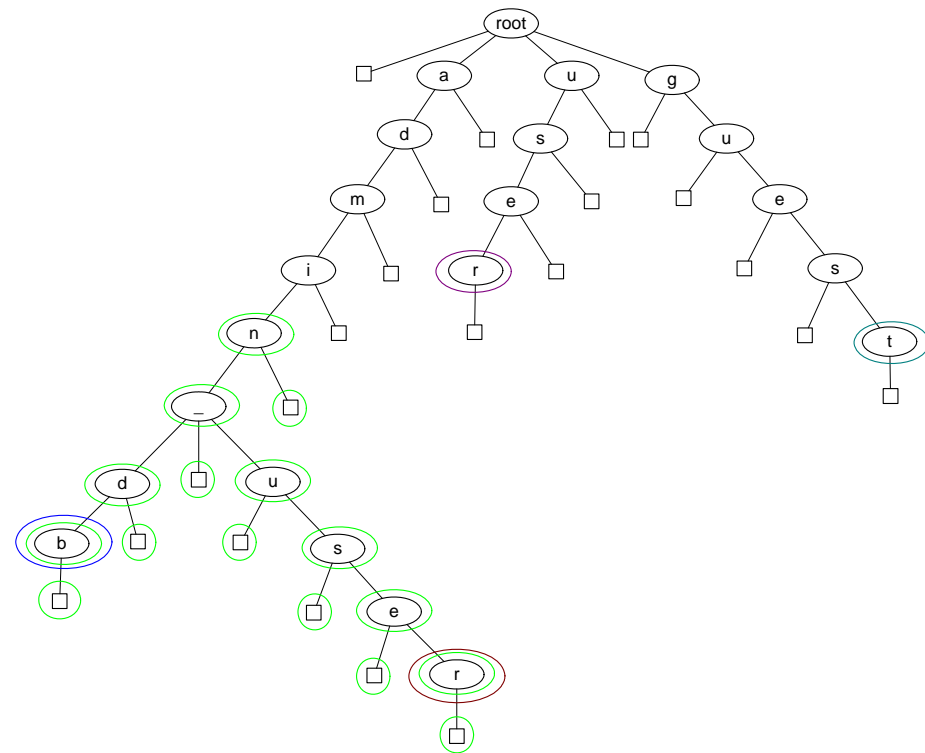
Forward Tree The example forward tree indexes `Evaluatables` matching the following words:

- admin.*
- admin_db
- admin_user
- user
- guest

Because it is a forward tree, no conversion is done on the words.

The boxes are the nodes applying to any character.

The nodes where the `Evaluatables` can be found are shown with the colored rings. Sought words are inserted as written in the request.



- = admin.*
- = admin_user
- = admin_db
- = user
- = guest

Figure 63: Example of a `StringIndex` forward tree.



Reverse Tree

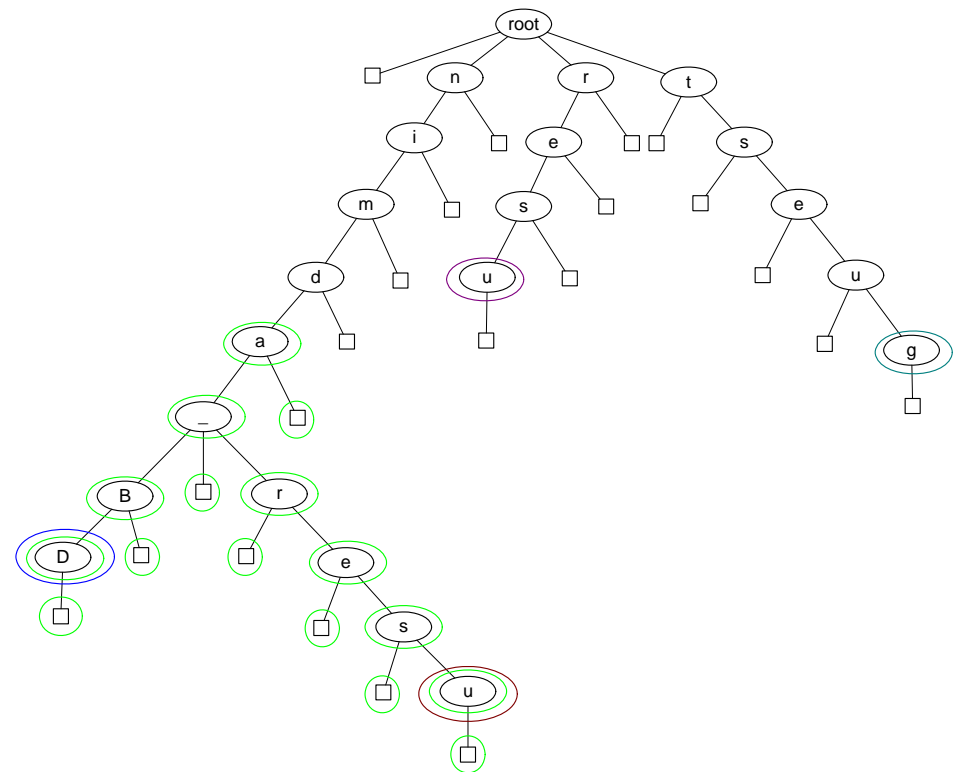
The example forward tree indexes Evaluatables matching the following words:

- .*admin
- DB_admin
- user_admin
- user
- guest

In a forward tree, every word has to be switched in its reverse form. “.” is handled as one character. So .*admin becomes nimda.*

The boxes are the nodes applying to any character.

The nodes where the Evaluatables can be found are shown with the colored rings. To search a word, its reverse form has to be built as a first step.



- = .*admin
- = user_admin
- = DB_admin_db
- = user
- = guest

Figure 64: Example of a StringIndex reverse tree.



5.2.3.2 ComparableIndex

- Overview* The `ComparableIndex` is needed for subjects, resources, actions or environments, in a `Policy` or `PolicySet` target, whose data type implements the `java.lang.Comparable` interface. If this applies a `ComparableIndex` can be built.
- AVLTree* The core of such a comparable index is the `AVLTree`, taking the tree in balance. This is important to accelerate the process of finding the indexed policies once the PDP is operating.
- Algorithm* The `ComparableIndex` is an `AVLTree` with several child elements per node. Every `AttributeValue` of the Subject, Resource, Environment and Action element, in the Target element will be a node in the Tree. While inserting new values the tree balances its self. Subject-, Resource-, Action- and EnvironmentMatch elements with a matchfunction containing "greater-than", goes to the assigned value and inserts the policy to every node with a value greater than the specified. If the matchfunction contains "less-than", the policy will be inserted to every node less than the specified value. Not indexable Policies will be inserted to every node in the tree.



Initialize the index

Figure 65 describes how an `IndexHandler` initializes a `ComparableIndex`. For a better survey the diagram references to other sequence diagrams, see below in this document.

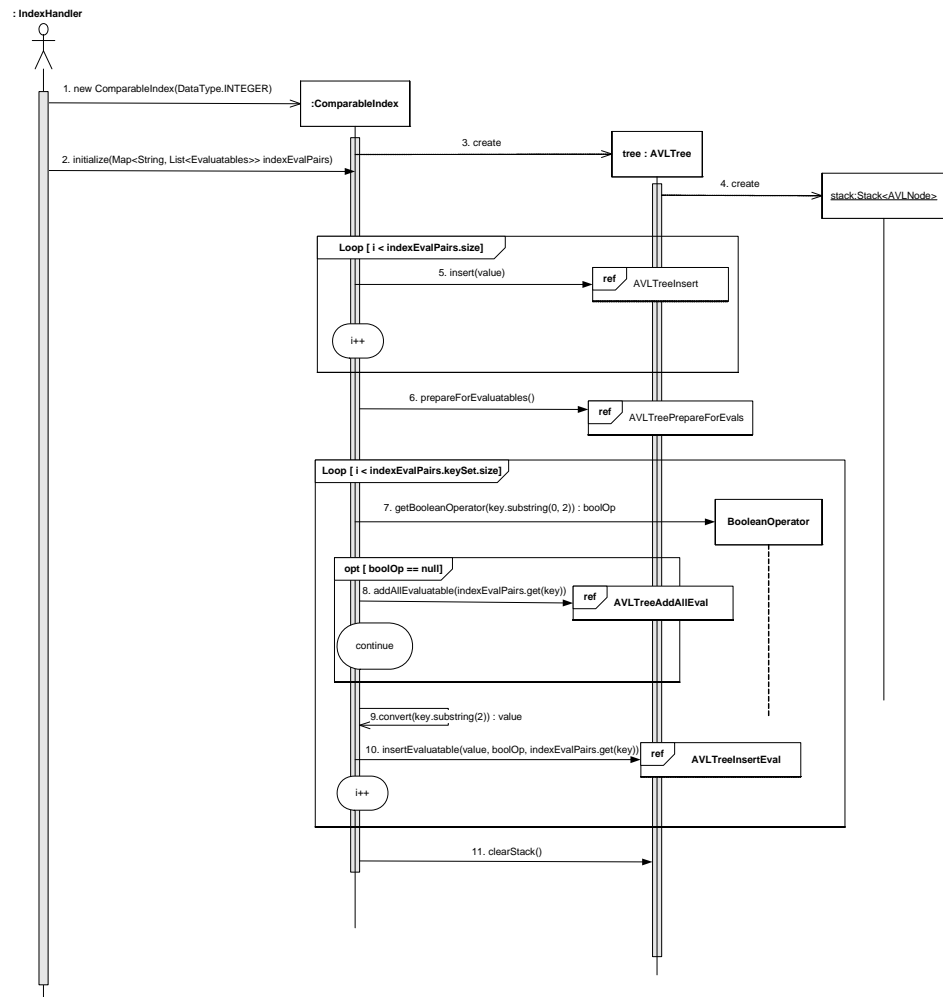


Figure 65: Initialization of the ComparableIndex.



Description

1. The `IndexHandler` creates a new `ComparableIndex` with a `dataType` as parameter. This parameter specifies which concrete comparable type should be used for the index (e.g. `Integer`, `Double`, `String` etc.).
2. The `IndexHandler` calls the `initialize` method of the `ComparableIndex`. As parameter is a map specified, in which the `String` value the key to insert into the index and the `List<Evaluatables>` the value for the key is.
3. The `ComparableIndex` creates a new `AVLTree`.
4. The `AVLTree` creates a new `Stack<AVLNode>`.
5. Inserts the value in the tree. See figure 66 (`AVLTree Insert`) to find out more about this method.
6. Prepares the Tree to insert `Evaluatable` as value of the tree nodes. See figure 67 (`AVLTree prepareForEvals`) to find out more about this method.
7. Gets the `boolValue` for a substring of the key (lt, gt, eq, ge, le).
8. Adds the `Evaluatables` in the `List` to all nodes in the tree. This method is used for the `Evaluatables` which can't be indexed. See figure 68 (`AVLTree AddAllEvals`) to find out more about this method.
9. Converts the substring of the key in its real representation (e.g. `Integer`, `Double`, etc.).
10. Inserts the `Evaluatables` to the index at the position specified by the `boolOperator` and the value. See figure 69 and 70 (`AVLTree InsertEvals`) to find out more about this method.
11. Clears the stack.



AVLTreeInsert

Figure 66 describes how the insert method of the AVLTree works.

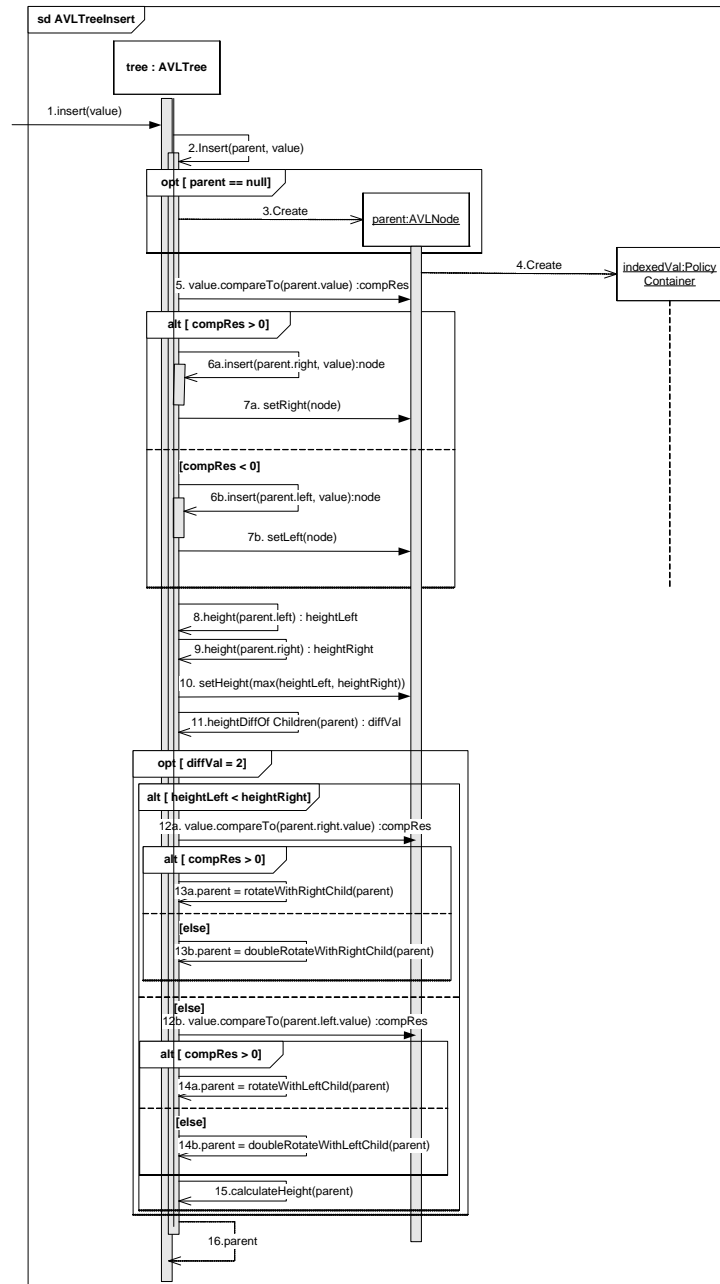


Figure 66: Inserting into the AVL tree.



Description

1. Insert call with an insertable value.
2. Calls the private method `insert`, with the parent `AVLNode` and the value to insert as parameters.
3. If the parent object is null, a new `AVLNode` will be created.
4. The `AVLNode` creates a new `PolicyContainer`.
5. The received value will be compared with the value of the parent node.
- 6a. If the `compareResult` from 5 is greater than 0, the `insert` method will be recursive called. The right child and the value will be submitted as parameters of the method call.
- 7a. The right child of the parent will be set.
- 6b. If the `compareResult` from 5 is less than 0, the `insert` method will be recursive called. The left child and the value will be submitted as parameters of the method call.
- 7b. The left child of the parent will be set.
8. The height of the left child from the parent will be calculated.
9. The height of the right child from the parent will be calculated.
10. The height of the parent node will be set to the max of the `leftchildheight` and the `rightchildheight`.
11. The height difference of the children will be calculated.
- 12a. If the height difference of the children equals 2 and the height of the left child is less than the height of the right child, the received value will be compared to the value of the right child. And a `compRes` will be returned.
- 13a. If the `compRes` is greater than 0, the method `rotateWithRightChild` will be called. The parent `AVLNode` is submitted as parameter. The method returns an `AVLNode` which will be assigned to its parent.
- 13b. If the `compRes` is less than 0, the method `DoubleRotateWithRightChild` will be called. The parent `AVLNode` is submitted as parameter. The method returns an `AVLNode` which will be assigned to its parent.
- 12b. If the height difference of the children equals 2 and the height of the left child is not less than the height of the right child, the received value will be compared to the value of the left child. A `compRes` will be returned.
- 14a. If the `compRes` is greater than 0, the method `rotateWithLeftChild` will be called. The parent `AVLNode` is submitted as parameter. The method returns an `AVLNode` which will be assigned to its parent.
- 14b. If the `compRes` is less than 0, the method `DoubleRotateWithLeftChild` will be called. The parent `AVLNode` is submitted as parameter. The method returns an `AVLNode` which will be assigned to its parent.
15. The height of the parent will be recalculated.
16. Returns the parent.



AVLTree
PrepareFor
Evals

Figure 67 describes how the `prepareForEvaluatables` method of the AVLTree works.

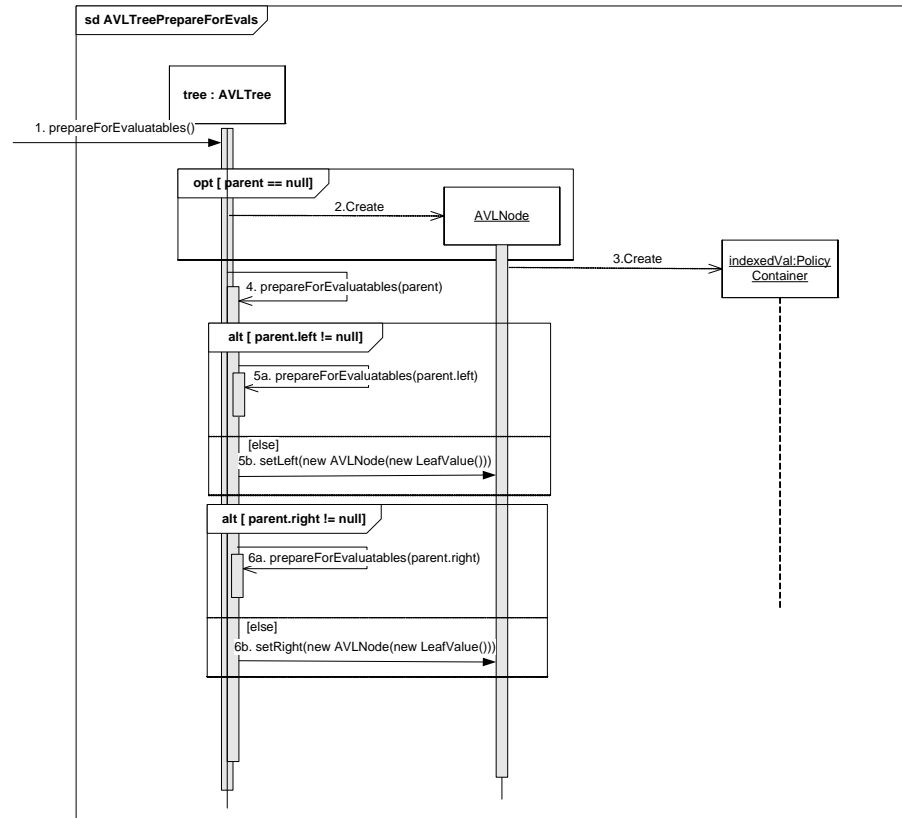


Figure 67: Prepares the tree for inserting Evaluatables.

Description

1. Call of the method `prepareForEvaluatables`.
2. Creates a new `AVLNode`.
3. Creates a new `PolicyContainer`.
4. Calls the private method `prepareForEvaluatables` and submits an `AVLNode` parent as parameter.
- 5a. If the left child of the parent unequals null, the method `prepareForEvaluatables` will be called recursive, with the left child of the parent as parameter.
- 5b. If the left child of the parent equals null, a new `AVLNode` containing a new `LeafValue` will be set as left child.
- 6a. If the right child of the parent unequals null, the method `prepareForEvaluatables` will be called recursive, with the right child of the parent as parameter.
- 6b. if the right child of the parent equals null, a new `AVLNode` containing a new `LeafValue` will be set as right child.



AVLTree
AddAllEval

Figure 68 describes how the addAllEvaluatable method of the AVLTree works.

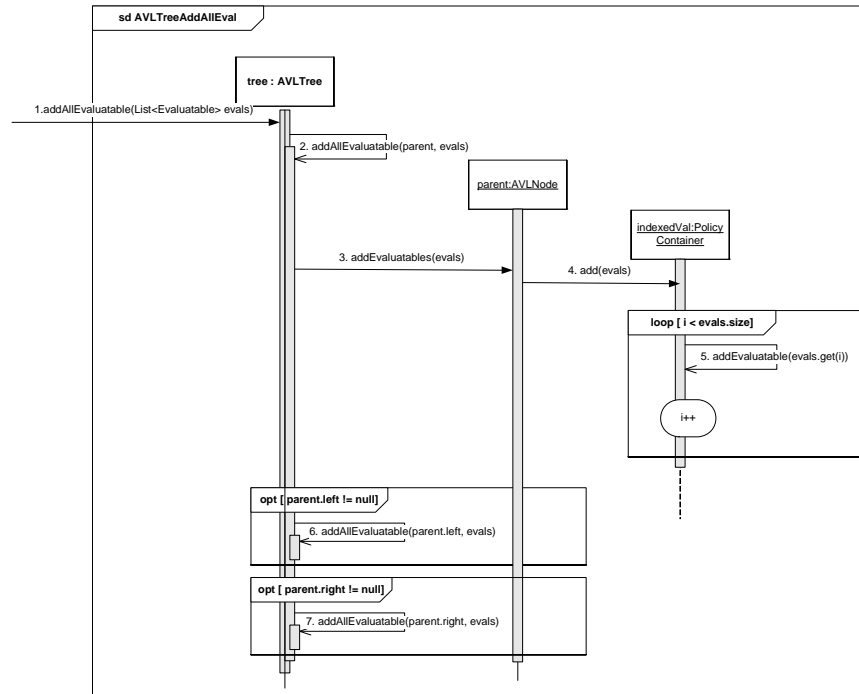


Figure 68: Add Evaluatables to the tree.

Description

1. Call of the method `addAllEvaluatable`.
2. Calls the private method `addAllEvaluatable` and submits an `AVLNode` `parent` and a `List` of `Evaluatables` (`evals`) as parameters.
3. Adds a `List` of `Evaluatables` to the `parent`.
4. Adds a `List` of `Evaluatables` to the `PolicyContainer` (`indexedVal`).
5. Iterates over the `List` of `Evaluatables` and adds the `Evaluatable` to a `List`.
6. If the left child of the `parent` unequal null, the method `addAllEvaluatables` will be called recursive, with the left child of the `parent` and the received `List` of `Evaluatables` as parameters.
7. If the right child of the `parent` unequal null, the method `addAllEvaluatables` will be called recursive, with the right child of the `parent` and the received `List` of `Evaluatables` as parameters.



AVLTree
InsertEval
part 1

Figure 69 and 70 describes how the insertEvaluatables method of the AVLTree works.

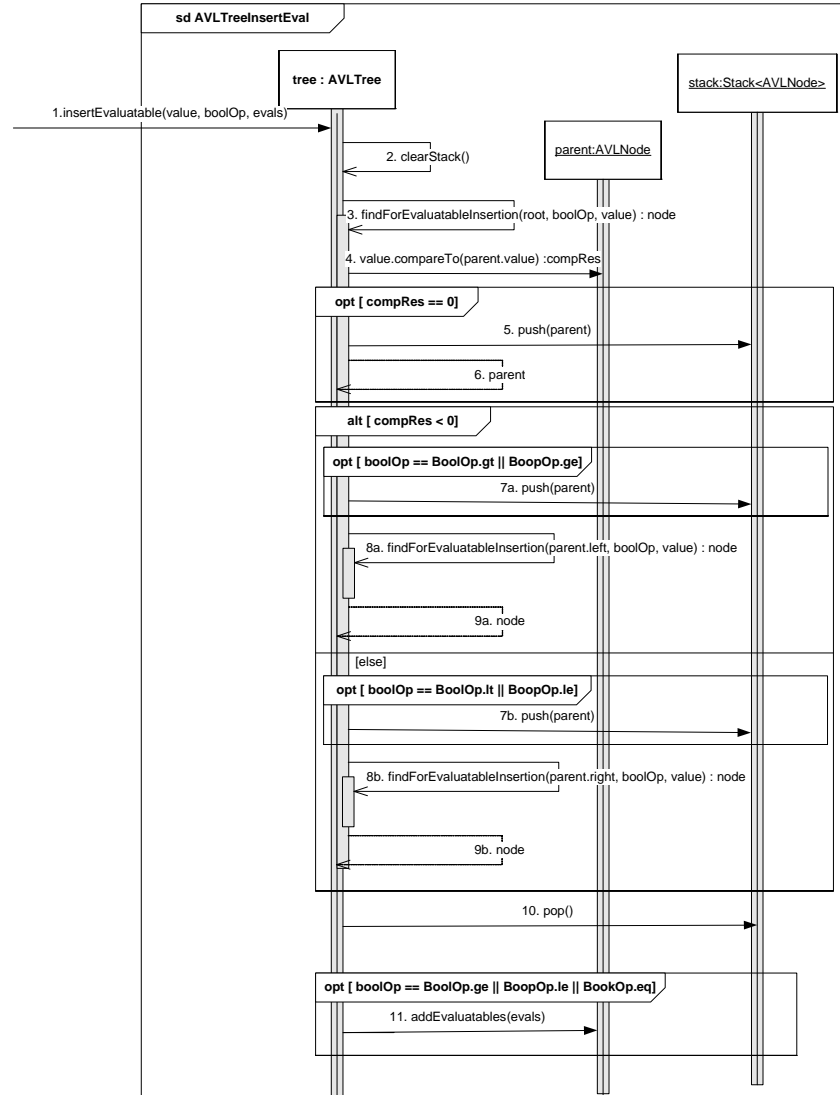


Figure 69: Insert an Evaluatable into the tree. (part 1)



Description part 1

1. Call of the method `insertEvaluatables` with a `Comparable` value, a `BooleanOperator` `boolOp` and a `List` of `Evaluatables` evals as parameters.
2. Clears the stack.
3. Calls the private method `findForEvaluatableInsertion` to find the node to which the `Evaluatable` should be inserted. The method takes the parameters `AVLNode` `parent` `BooleanOperator` `boolOp` and a `Comparable` value.
4. Compares the received value with the value of the parent and returns an `Integer` `compRes`.

If [`compRes == 0`]

5. The parent will be pushed to the stack.
6. The method returns the parent.

End If

If [`compRes < 0`]

- 7a. If the `boolOp` equals `BooleanOperator.gt` OR `boolOp` equals `BooleanOperator.ge`, then the parent will be pushed to the stack.
- 8a. The method `findForEvaluatableInsertion` will be called recursive, with the left child of the parent, the `boolOp` and the value as parameters and returns a node in which an `Evaluatable` can be inserted.
- 9a. The method returns the node returned in point 8a.

End If

If [`compRes > 0`]

- 7b. If the `boolOp` equals `BooleanOperator.lt` OR `boolOp` equals `BooleanOperator.le`, then the parent will be pushed to the stack.
- 8b. The method `findForEvaluatableInsertion` will be called recursive, with the right child of the parent, the `boolOp` and the value as parameters and returns a node in which an `Evaluatable` can be inserted.
- 9b. The method returns the node returned in point 8b.

End If

10. Pops an `AVLNode` from the stack.
11. If the `boolOp` equals `BooleanOperator.ge` OR `boolOp` equals `BooleanOperator.le` OR `boolOp` equals `BooleanOperator.eq`, then the method `addEvaluatables` with a `List` of `Evaluatable` as parameter is called.



AVLTree
InsertEval
part 2

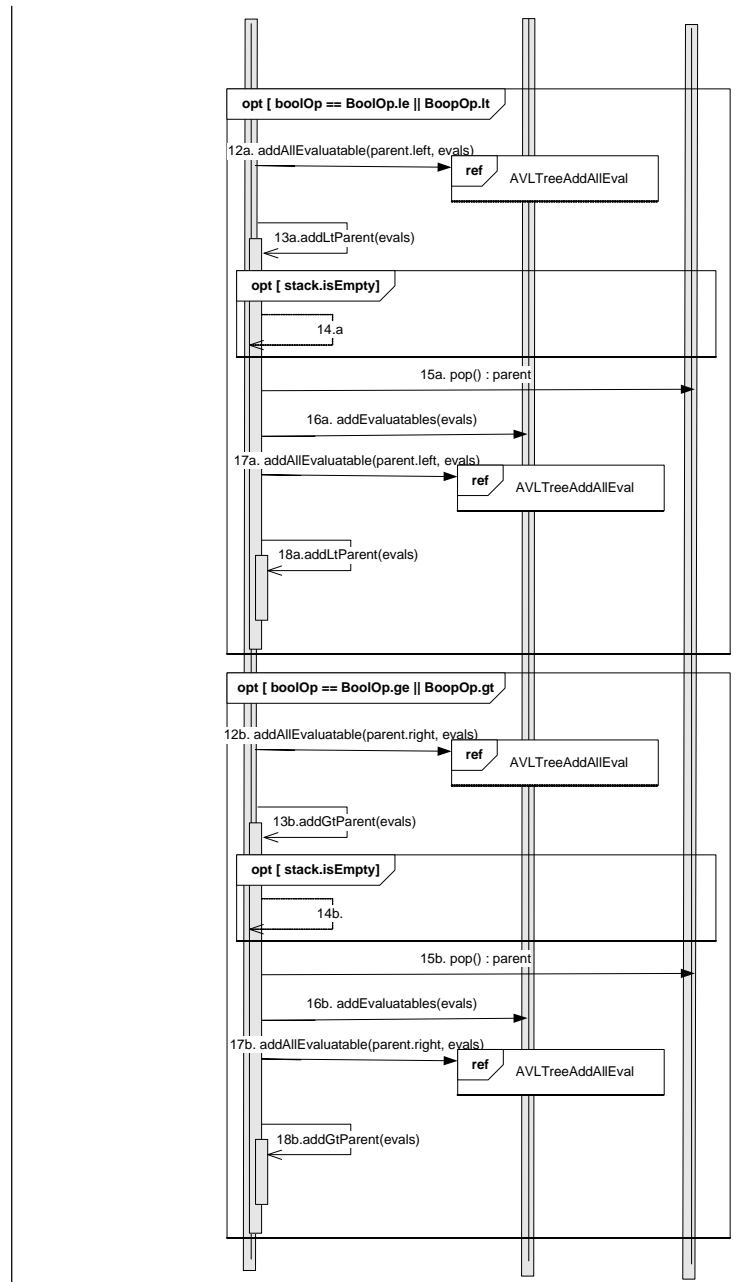


Figure 70: Insert an Evaluatable into the tree. (part 2)



Description part 2

If[boolOp == BooleanOperator.le || BooleanOperator.lt]

- 12a. The method `addAllEvaluatable` (see figure 68) with the left child of the parent and a `List of Evaluatable` as parameter is called.
- 13a. The method `addLtParent` with a `List of Evaluatables` as parameter is called.
- 14a. If the stack is empty the method terminates.
- 15a. Pops an `AVLNode` parent from the stack and returns it.
- 16a. Calls the method `addEvaluatables` with a `List of Evaluatables` as parameter.
- 17a. The method `addAllEvaluatable` (see figure 68) with the left child of the parent and a `List of Evaluatable` as parameter is called.
- 18a. The method `addLtParent` with `evals` as parameter will be called recursive.

End If

If[boolOp == BooleanOperator.ge || BooleanOperator.gt]

- 12b. The method `addAllEvaluatable` (see figure 68) with the right child of the parent and a `List of Evaluatable` as parameter is called.
- 13b. The method `addGtParent` with a `List of Evaluatables` as parameter is called.
- 14b. If the stack is empty the method terminates.
- 15b. Pops an `AVLNode` parent from the Stack and returns it.
- 16b. Calls the method `addEvaluatables` with a `List of Evaluatables` as parameter.
- 17b. The method `addAllEvaluatable` (see figure 68) with the right child of the parent and a `List of Evaluatable` as parameter is called.
- 18b. The method `addGtParent` with `evals` as parameter will be called recursive.

End If



Locate
Evaluable

Figure 71 describes how an IndexHandler locates Evaluatables in a ComparableIndex.

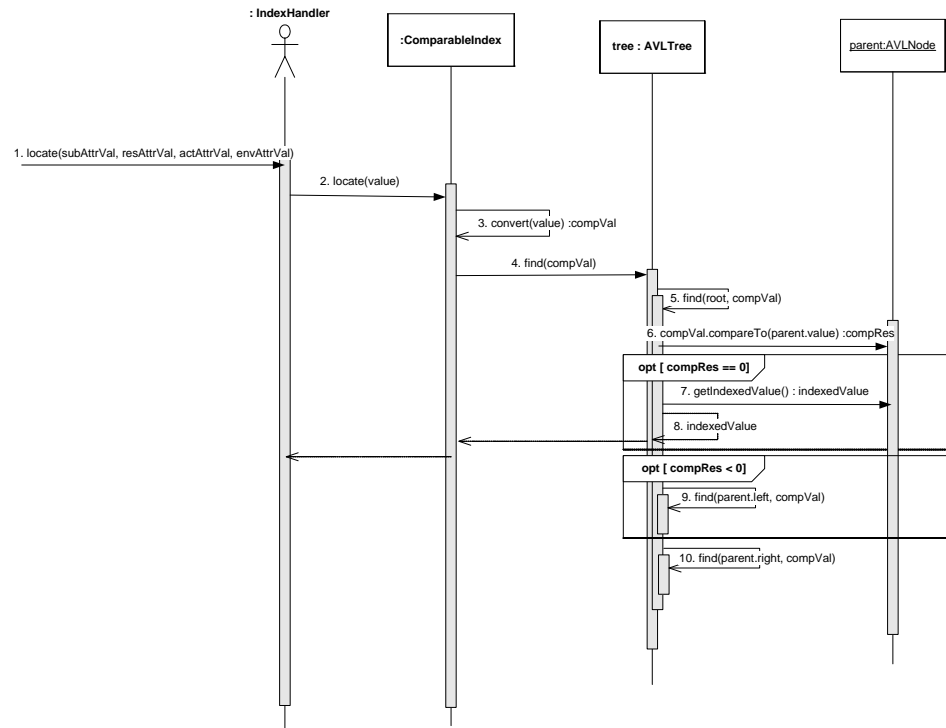


Figure 71: Locating an Evaluable.

Description

1. Calls the method `locate` on the `IndexHandler`.
2. Calls the method `locate` on the `ComparableIndex` with a value as parameter.
3. Converts the received value to a specific `Comparable` value (e.g. `Integer`, `Double`, etc.) and returns it.
4. Calls the `find` method on the `AVLTree` with the specific `Comparable compVal` as parameter.
5. Calls the private method `find` with an `AVLNode` and a `Comparable` as parameters.
6. Compares the received `compVal` with the value of the parent and returns an `Integer compRes`.
7. If `compRes == 0` the `getIndexedValue` method on the parent will be called and returns an `IndexedValue`.
8. Terminates the method and returns the `IndexedValue`.
9. If `compRes < 0` the `find` method is called recursive with the left child of the parent and the `compVal` as parameters.
10. The `find` method is called recursive with the right child of the parent and the `compVal` as parameters.



5.2.3.2.1 Example

Overview

This chapter gives an example about how a ComparableIndex might look like.

Comparable-Index

Policies with the following values will be indexed:

- >100
- <=75
- 102
- <50
- >=160

Figure 72 shows on which values the indexed policies are findable.

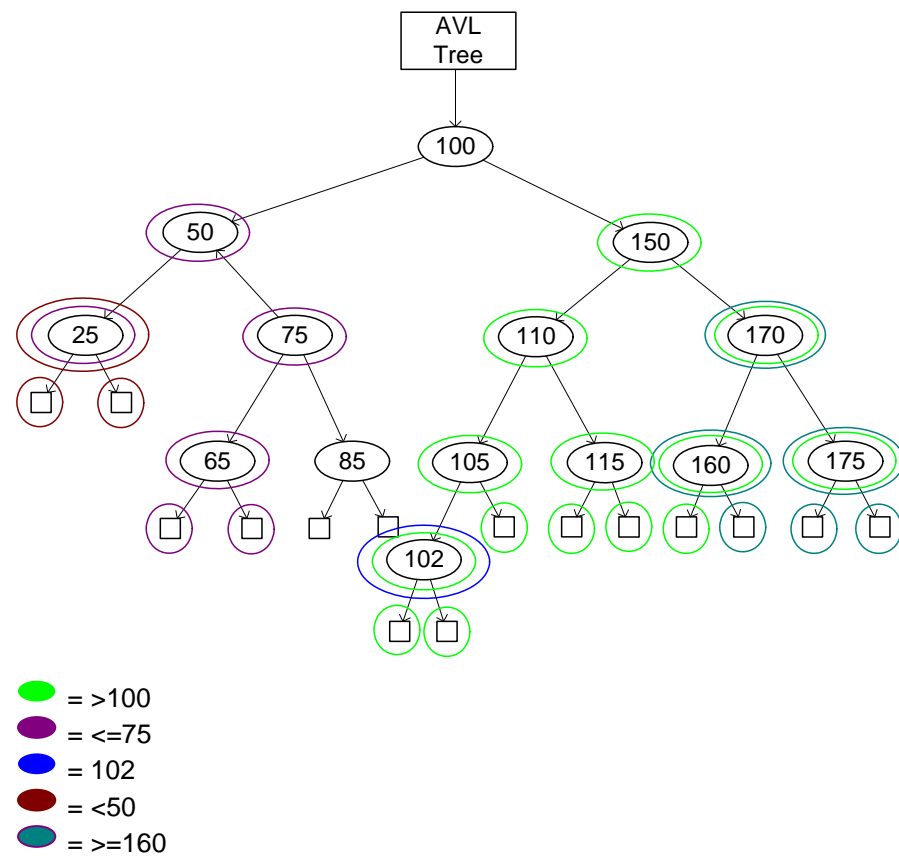


Figure 72: Example of a ComparableIndex.



5.2.4 Design decisions

Overview This chapter explains the most important design decisions, made in the Locator module.

5.2.4.1 Index chain instead of one index

Issue Building an index over the policies is very complex because there are different elements of a request that have to be indexed as one. Every element of a target match might need a different way to index it. How should this complex index be implemented?

Alternatives

Alternative 1: One big index

One big index over all elements is built. It is possible that different parts of this index are implementing a different indexing strategy than the others.

Advantages

- No additional method calls are required
- Very fast

Disadvantages

- Very complex to implement

Alternative 2: Four chained indexes, one per element of the target match

A separate Index for each element of a target match is built. To ensure that no values have to be merged a separate index is built for each index result of the parent index.

As example for each search result that might occur on the subject index, a separate resource index is built.

Advantages

- Reduces the complexity of the index
- Fast
- Every indexing strategy can be implemented for its own and does not need to know the other strategies

Disadvantages

- A controlling instance is required which calls the next index indexed by its parent



Alternative 3: Four indexes are built and the result is the intersection of their results

For each element of the target match exists a standalone index. The result of the index search is the intersection of the results of every index.

Advantages

- Simple to implement
- Every indexing strategy can be implemented for its own and does not need to know the other strategies

Disadvantages

- The intersection reduces the performance

Decision

The decision fell on solution 2. This implementation is very easy to implement and even so fast. The error-proneness of solution 1 is very high. Solution 3 does not gain a lot performance.

5.2.4.2 Configurable Indexes

Issue

We decided to index only one element-id per element of a subject-, resource-, action- and environmentmatch of a target. The consequence of this decision is that it is important to index the correct element-id to have a good index. How should it be ensured that the values of the correct element-id are indexed?

Alternatives

Alternative 1: indexed element-id's are fix

We define element-id values that have to be used as element-id if an element should be indexed. For example only the elements with the id subject-role are indexed.

Advantages

- Fast index policies are optimized on the index algorithm
- Easy to implement

Disadvantages

- The users of the PDP have restrictions how their ids should be named. If they do not follow the restrictions the policies aren't indexed.



Alternative 2: Indexed element id's can be configured

For every element of a subject-, resource-, action- and environmentmatch of a target, it is configurable which element-id of the sub element should be indexed. Using this, an policy creator can configure how his element-id's are named

| Advantages | Disadvantages |
|--|---------------|
| <ul style="list-style-type: none">The policy administrator can define which element-id should be indexedfast index if policies are optimized on the index algorithm | none |

Alternative 3: The most used element-id is indexed

An algorithm checks all Evaluatables that should be indexed and searches for the most used element-id. The values using the element-id are indexed.

| Advantages | Disadvantages |
|--|--|
| <ul style="list-style-type: none">If every policy matches the same amount of requests, this leads to the fastest possible index. But even then it is slow because only a part of the policies are indexed. | <ul style="list-style-type: none">Not all policies are indexedThe writer of the policies doesn't exactly know which policies are indexed. Maybe the most requests will match only one policy. In this case the wrong policy is indexed. |

Decision

The decision fell on solution 2. Solution 3 has too many disadvantages. Solution 1 is nearly the same as solution 2, with the one difference that solution 2 is configurable. The algorithm for indexing on a configurable element-id is not slower than the one to index on a fix element-id that means that the only extra work to implement solution 2 is that a configuration file has to be read. We decided that this is not worth to take the disadvantages of solution 1.

5.2.4.3 Only forward or reverse tree

Issue

Regular expressions with an undefined amount of any character signs can only be indexed if the wildcard-characters are in front or back of the expression. Indexing both, any character signs in front and back of the expression, in one index is not possible. How should this be handled?

Alternatives

Alternative 1: Both a forward and reverse tree is held and the result is the intersection of both

A `StringIndex` contains two indexes, one for the forward tree and one for the reverse tree. Afterwards the results of both trees are merged.



Advantages

- The tree can handle both, forward and reverse regular expressions

Disadvantages

- If the overall index contains only `StringIndex` indexes in the chain, 16 intersection actions are needed to get the overall value

Alternative 2: Only a forward or a reverse tree is possible

Only a forward or reverse index is implemented and supported. The other possibility is not supported.

Advantages

- Easy to implement

Disadvantages

- The amount of indexable policies is reduced. The writer of policies has more restrictions on how he can write policies.

The definitions of the value in the requests have to be in such a way that they can be indexed with a regular expression using this restrictions

Alternative 3: It is configurable if a reverse or forward tree should be used

For every element in a target match, using a `StringIndex` as indexing algorithm, it is configurable if the `StringIndex` should be indexing in forward or reverse mode.

Advantages

- Allows the policy writer to define which mode is more usable for the element of the target match

Disadvantages

- Indexable `Evalutables` have more restrictions

Decision

The decision fell on solution 3, because it is a good trade-off between flexibility in the policies and performance. Solution 1 results in too many merge operations, which affects the performance of the index algorithm. Solution 2 has the same performance as solution 3, but with more restrictions to the indexable `Evalutables`. The amount of work to implement the configuration does not excuse the restrictions that would be made in not implementing the configuration.



5.2.4.4 IndexHandler interface

Issue While writing this diploma thesis, there was not enough time to optimize the index algorithms. Beside this there is a good chance that a better implementation of the algorithm will be found. How can it be solved, that the implementation and initialization of the indexes might get changed?

Alternatives **Alternative 1: The changes are done directly in the code of the diploma thesis**

To change the implementation, the code of the diploma thesis is replaced by a better implementation. That means that a new class with the same name is placed in the same package and replaces the implementation of the diploma thesis.

Advantages

- No interfaces are needed
- Easy to solve

Disadvantages

- It is not possible to step back and take the code of the diploma thesis or allow that a user might decide which implementation should be used.

Alternative 2: An interface is made which encapsulates the IndexHandler

An interface for the IndexHandler is created. A new implementation implements this interface and is set to the Locator using the Spring framework.

Advantages

- Multiple implementations are possible
- Easy to change the used implementation

Disadvantages

- An additional interface is needed

Decision The decision fell to solution 2. Using a complete different implementation for the same class is poor design. The interface does not cost too much performance and using spring it is easy to use another implementation.



5.2.4.5 IndexedValue interface

Issue Due to the decision to chain the indexes, it is recommended that an entry in the index can contain another index or a `PolicyContainer`. How should it be solved, that a node in the index can contain both?

Alternatives

Alternative 1: Two variables per entry

Every entry in the `Index` contains two variables. One for the next interface and one a `PolicyContainer`. Only one of them is used at a time.

Advantages

- No additional interfaces are required

Disadvantages

- Different methods for returning an `Index` or returning a `PolicyContainer` are needed
- Additional variables are used, where one of them is always null

Alternative 2: A interface IndexedValue

An interface for indexed values is created. `Index` and `PolicyContainer` implement this interface. So the `Index` does not care if it contains an index or a `PolicyContainer`. The `IndexHandler` has to know if an `Index` contains an `Index` or a `PolicyContainer`.

Advantages

- `Index` implementations do not need to care what they contain
- Only one variable is needed

Disadvantages

- An additional level of abstractness is needed

Decision

The decision fell on solution 2. This solution requires less lines of code and has no code that does nearly the same. The `Index` must not care if it contains another `Index` or a `PolicyContainer`.



6 ReferenceLoader

Overview The ReferenceLoader module is responsible for resolving `Evaluatables` that are referenced from another. These references are resolved local or remote.

structure This chapter contains the description of the ReferenceLoader interfaces and the HERAS^{AF} implementation.

6.1 Interfaces

Overview This chapter contains the description of the `ReferenceLoader` interface.

The interface

| |
|--|
| <<interface>> ReferenceLoader |
| <pre>+initialize(List<Evaluatable>) : void +load(List<IdReferenceType> : Map<String, Evaluatable> +get(String) : Evaluatable</pre> |

Figure 73: The ReferenceLoader interface

initialize method Initializes the reference loader with the given `Evaluatables`. These `Evaluatables` are used for resolving local references.

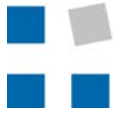
load method This method takes a list of references and returns a `Map` containing the loaded `Evaluatables` with their id as key. The references are resolved on a remote PDP.

get method This method takes a `String` representing the id of an `Evaluatable` that should be resolved local.

6.2 HERAS^{AF} XACML 2.0 implementation

Overview This chapter describes how the `ReferenceLoader` is realized in the HERAS^{AF} XACML 2.0 implementation.

Hint The implementation of the `ReferenceLoader` is beyond the scope of the diploma thesis. Therefore a mock is provided which always returns an empty `Map`. There is no exception thrown because it is possible that this `Evaluatable` is `NotApplicable` and will be ignored.



Part V: Deployment



1 Overview

| | |
|------------------|--|
| <i>Content</i> | To build the whole HERAS ^{AF} XACML 2.0 Implementation the needs of automated software build is obvious. This part describes how the library dependencies, the automated running of tests, and the deployment of HERAS ^{AF} are managed. Maven 2 is that detailed in this guide because the information for configuration of Maven 2 and its plugins is spread over several web pages and books. It is much easier if this information is collected at one point and provided to you. |
| <i>Structure</i> | In the first paragraph there is an overview over Apache Maven 2. Then the responsibilities of the heras-pdp-root projects are explained. |

2 Apache Maven 2

| | |
|-----------------------------|--|
| <i>Introduction</i> | To use Maven 2 as automated software build, it is required to install and configure Maven 2 properly. |
| <i>settings.xml</i> | In the settings.xml from Maven configuration-settings for maven are specified, for example where the Local-Maven-Repository is located or define profiles. |
| <i>POM</i> | A Project Object Model is the fundamental unit of work in Maven. It is an XML file that contains information about the project and configuration details used by Maven to build the project. |
| <i>External libraries</i> | To use external libraries in a project, you have to declare a dependency in the pom.xml which points to the library in the repository. |
| <i>Project dependencies</i> | Project dependencies are also declared in the pom.xml. Like external libraries, you have to declare a dependency to each project. |
| <i>Tests</i> | The directory src/test/java is the Maven default source directory for test classes. Every project could contain tests. |

maven-surefire-plugin

To run automated tests you declare in the pom.xml the maven-surefire-plugin and specify which tests should be excluded, included or if all the tests should be skipped.

example:

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-surefire-plugin</artifactId>
  <configuration>
    <skip>>false</skip>
  </configuration>
</plugin>
```

Figure 74: Example maven-surfire-plugin plugin configuration.



integration tests

Integration tests are tests running in a special phase of the Maven deployment lifecycle. They test the functionality over more than one unit.

maven-surefire-plugin

Integration tests run like normal unit tests with the surefire-plugin in a special phase of the Maven build lifecycle.

example:

```
<plugin>
  <artifactId>maven-surefire-plugin</artifactId>
  <executions>
    <execution>
      <phase>integration-test</phase>
      <goals>
        <goal>test</goal>
      </goals>
      <configuration>
        <skip>>false</skip>
      </configuration>
    </execution>
  </executions>
</plugin>
```

Figure 75: Example maven-sure-fire plugin plugin configuration for integration tests.

conformance test

Conformance tests are a special kind of integration tests. These tests are used to guarantee the conformance of an implementation of a specification.



SQL execution

There are some tests like the integration tests that depend on a database. Before and after execution of the tests this database must be created and dropped. For this purpose the sql-maven-plugin exists. With this plugin the developer is capable of executing SQL statements on a database.

sql-maven-plugin

```
<plugin>
  <groupId>org.codehaus.mojo</groupId>
  <artifactId>sql-maven-plugin</artifactId>
  <executions>
    <execution>
      <id>Create database</id>
      <phase>pre-integration-test</phase>
      <goals>
        <goal>execute</goal>
      </goals>
      <configuration>
        <autocommit>>true</autocommit>
        <srcFiles>
          <srcFile>Create.sql</srcFile>
        </srcFiles>
      </configuration>
    </execution>
    <execution>
      <id>Drop database</id>
      <phase>post-integration-test</phase>
      <goals>
        <goal>execute</goal>
      </goals>
      <configuration>
        <autocommit>>true</autocommit>
        <srcFiles>
          <srcFile>Drop.sql</srcFile>
        </srcFiles>
      </configuration>
    </execution>
  </executions>
  <dependencies>
    <dependency>
      <groupId>mysql</groupId>
      <artifactId>mysql-connector-java</artifactId>
      <version>5.0.7</version>
      <classifier>bin</classifier>
    </dependency>
  </dependencies>
  <configuration>
    <driver>com.mysql.jdbc.Driver</driver>
    <url>jdbc:mysql://localhost</url>
    <username>root</username>
    <password>root</password>
  </configuration>
</plugin>
```

Figure 76: Example sql-mvn-plugin plugin configuration.



Code coverage

For proving how good the test coverage of the code is there are many plugins. The maven-clover-plugin is one of them.

maven-clover-plugin

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-clover-plugin</artifactId>
  <configuration>
    <generateXml>true</generateXml>
    <generateHtml>true</generateHtml>
    <targetPercentage>50%</targetPercentage>
    <jdk>1.5</jdk>
    <excludes>
      <exclude>org/herasaf/**/*Exception.java</exclude>
    </excludes>
  </configuration>
  <executions>
    <execution>
      <phase>verify</phase>
      <goals>
        <goal>instrument</goal>
        <goal>clover</goal>
        <goal>log</goal>
        <goal>aggregate</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

Figure 77: Example maven-clover-plugin plugin configuration.

Javadoc generation

To automated generate javadoc there is also a plugin for Maven provided.

maven-javadoc-plugin

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-javadoc-plugin</artifactId>
  <configuration>
    <aggregate>true</aggregate>
  </configuration>
  <executions>
    <execution>
      <phase>deploy</phase>
      <goals>
        <goal>javadoc</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

Figure 78: Example maven-javadoc-plugin plugin configuration.



Profiles

Maven also addresses the circumstances of different requirements to the configuration. For example while testing the database should not point to the productive one and so on.

It is possible as well to make profiles for different combinations of modules. This fact can be used to make a profile that excludes the integration tests for example for a faster build while developing.

example profile

This profile only includes the core module.

```
<profile>
  <id>disableIT</id>
  <activation>
    <property>
      <name>enableProfile</name>
      <value>true</value>
    </property>
  </activation>
  <modules>
    <module>../heras-core</module>
  </modules>
</profile>
```

Figure 79: Example Maven profile configuration.

3 heras-pdp-root

Introduction

The heras-pdp-root is the main project of the HERAS^{AF} XACML 2.0 Implementation. It is used to link to the other projects and performs a maven build.

Standard build

To perform a maven build with all integration- and conformance tests go to the heras-root directory in the command console and execute:

```
Mvn clean install
```

Build without integration tests

To perform a maven build with all conformance- but no integration tests go to the heras-root directory in the command console and execute:

```
Mvn clean install -P disableIT
```

Build without conformance tests

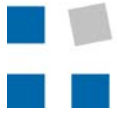
To perform a maven build with all integration - but no conformance tests go to the heras-root directory in the command console and execute:

```
mvn clean install -P disableCT
```

Build without integration- and conformance tests

To perform a maven build without conformance- and integration tests go to the heras-root directory in the command console and execute:

```
mvn clean install -P disableALL
```



Appendix A: Overall



1 References

- [XACML20] Organization for the Advancement of Structured Information Standards (OASIS)
XACML 2.0 specification
http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml#XACML20
- [XACMLBI] Organization for the Advancement of Structured Information Standards (OASIS)
XACML 2.0 specification brief introduction
http://www.oasis-open.org/committees/download.php/2713/Brief_Introduction_to_XACML.html
- [SEXMLA] Interface 21
Spring framework Extensible XML Authoring
<http://static.springframework.org/spring/docs/2.0.x/reference/extensible-xml.html>
- [HPDPWSE] Sacha Dolski, Florian Huonder, Stefan Oberholzer
HERAS^{AF} – Holistic Enterprise-Ready Application Security Architecture Framework
Students Research Project – HERAS^{AF} XACML PDP Web Service Endpoint (German), July 2007
University of Applied Sciences Rapperswil (Switzerland)
- [HPDPWSEUG] Sacha Dolski, Florian Huonder, Stefan Oberholzer
HERAS^{AF} – Holistic Enterprise-Ready Application Security Architecture Framework
Students Research Project – HERAS^{AF} XACML PDP Web Service Endpoint User's Guide, July 2007
University of Applied Sciences Rapperswil (Switzerland)
- [HPDPWSEDG] Sacha Dolski, Florian Huonder, Stefan Oberholzer
HERAS^{AF} – Holistic Enterprise-Ready Application Security Architecture Framework
Students Research Project – HERAS^{AF} XACML PDP Web Service Endpoint Developer's Guide, July 2007
University of Applied Sciences Rapperswil (Switzerland)
- [HPAP] Massimo Cerqui, Sandro Strebel
HERAS^{AF} – Holistic Enterprise-Ready Application Security Architecture Framework
Diploma Thesis – HERAS^{AF} PAP, November 2007
University of Applied Sciences Rapperswil (Switzerland)
- [HPEPUG] Massimo Cerqui, Sandro Strebel
HERAS^{AF} – Holistic Enterprise-Ready Application Security Architecture Framework
Students Research Project – HERAS^{AF} PEP User's Guide, July 2007
University of Applied Sciences Rapperswil (Switzerland)
- [HPEPDG] Massimo Cerqui, Sandro Strebel



HERAS^{AF} – Holistic Enterprise-Ready Application Security Architecture Framework
Students Research Project – HERAS^{AF} PEP Developer's Guide, July 2007
University of Applied Sciences Rapperswil (Switzerland)

[HXI20]

Sacha Dolski, Florian Huonder, Stefan Oberholzer
HERAS^{AF} – Holistic Enterprise-Ready Application Security Architecture Framework
Diploma Thesis – HERAS^{AF} XACML 2.0 Implementierung (German),
November 2007
University of Applied Sciences Rapperswil (Switzerland)



2 Glossary

| | |
|---------------------------|--|
| <i>API</i> | Application Programming Interface |
| <i>AVL Tree</i> | Adel'son-Vel'skii and Landis Tree |
| <i>HERAS^{AF}</i> | Holistic Enterprise Ready Application Security <small>Architecture Framework</small> |
| <i>JAR</i> | Java Archive |
| <i>JAXB</i> | Java API for XML Binding |
| <i>JDBC</i> | Java Database Connectivity |
| <i>LDAP</i> | Lightweight Directory Access Protocol |
| <i>OASIS</i> | Organization for the Advancement of Structured Information Standards |
| <i>PAP</i> | Policy Administration Point |
| <i>PDP</i> | Policy Decision Point |
| <i>PEP</i> | Policy Enforcement Point |
| <i>PIP</i> | Policy Information Point |
| <i>POM</i> | Project Object Model |
| <i>SQL</i> | Structured Query Language |
| <i>XACML</i> | eXtensible Access Control Markup Language |
| <i>XML</i> | eXtensible Markup Language |
| <i>XSD</i> | XML Schema Definition |



Appendix B: Extension Points and required refactorings



1 Overview

- Introduction* This appendix contains extension points of the HERAS^{AF} XACML 2.0 implementation and refactorings that should be made for better performance or design of the implementation.
- Structures* The first part discusses the possible extensions to this implementation and the second one shows the refactorings that should be done.

2 Extension Points

- Overview* This chapter contains the possible extension points to this implementation. Listed are optional capabilities that XACML 2.0 offers and others are capabilities making the work with the application easier.
- Index order* At the moment the index order is fix and not adaptable. The order is subject -> resource -> action -> environment.
This is an assumption made by HERAS^{AF} that the subject narrows the choice down most and then resource, action and environment.
To allow an administrator to maximize the performance of his configuration of the XACML 2.0 implementation this order should made be configurable.
- Hot deployment* The current implementation has a minor disadvantage. The issue is that after deploying an Evaluatable the PDP must be restarted in order that these new Evaluatables take effect. The same is true for the undeploying.
Also new or overridden functions or data type attributes can only be made available to the PDP after a reinitialization.
The deployment processes should be changed in such a way that a hot deployment or undeployment, respectively is possible to minimize down time.
Another point is that it is quite unattractive that for each change in the Evaluatables a reinitialization is needed.
It also would be very nice if it were possible that an administrator can hotdeploy new functions or data type attributes that a reinitialization of the PDP is not needed at all.
- Obligations* The optional possibility of `Obligations` should be implemented to achieve more coverage of the XACML 2.0 specification [XACML20]. `Obligations` would allow the PDP to set conditions on an evaluation result to the PEP.
- Combining Parameters* `CombinerParameters` allow an `Evaluatable` or `Rule` to pass arguments to the assigned `CombiningAlgorithm`. The optional possibility of `CombinerParameters` should be implemented to achieve more coverage of the XACML 2.0 specification [XACML20].
- Attribute Selector* To expand the possibilities for accessing the request context while evaluation next to the `Designators` the `AttributeSelectors` should be implemented. The `AttributeSelectors` allow accessing the request context with XPath expressions which opens a wider application area.



| | |
|--|--|
| <i>AttributeFinder</i> | <p>The AttributeFinder is a mock implementation and is not able to resolve attributes. It always returns an empty list which means no attributes could be resolved.</p> <p>For accessing a PIP to resolve missing attributes the attribute finder module must be implemented.</p> |
| <i>Reference Loader</i> | <p>The ReferenceLoader is a mock implementation and is neither able to resolve references to remote nor to local Evaluatables.</p> <p>The ReferenceLoader simply returns an empty list what is equal to that no references could be resolved.</p> <p>To add the capability to resolve referenced Evaluatables the ReferenceLoader module must be implemented.</p> |
| <i>Single AttributeValue</i> | <p>The current implementation only supports AttributeValues with one value. To add the capability of handling multiple values the convert method of the DataTypeAttributes must take the AttributeValue itself and not a string anymore.</p> |
| <i>DeleteAll method in Persistence-Manager</i> | <p>The current implementation of the PersistenceManager provides a deleteAll method. By the reason that a HERAS^{AF} PAP knows which policies are deployed to the PDP, it can undeploy the policies by passing a list of these policies. In case that several PAP can deploy policies to a PDP or multiple users with different qualifications can define and deploy policies, a deleteAll method does not make sense.</p> |
| <i>JNDI Data-source in a J2EE container</i> | <p>In the current implementation of the PersistenceManager all the parameter to build a Datasource are configured in the spring application context.</p> <p>To provide the possibility to use a JNDI Datasource, the PersistenceManager has to be modified to set directly the usable Datasource, instead of the parameters. Further, the spring application context must provide a feasibility to enable a selectable Datasource.</p> |

3 Refactorings

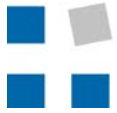
| | |
|---|---|
| <i>Overview</i> | <p>This chapter contains refactorings that should be done to improve performance or design of the HERAS^{AF} XACML 2.0 implementation.</p> |
| <i>Locator</i> | <p>The index handler implementation of the HERAS^{AF} XACML 2.0 Implementation contains a lot of duplicated code (The initialize method of the indexes).</p> <p>With the adoption of a super class for SubjectType, ResourceType, ActionType and EnvironmentType these initialize methods could be united.</p> |
| <i>Ordered / Unordered Combining Algorithms</i> | <p>After unmarshalling an Evaluatable it is possible to obtain the CombiningAlgorithm for the containing policies or rules, respectively. With this information it is possible, in case that an order CombiningAlgorithm is set, to order the containing policies or rules. So it is unnecessary to differentiate between ordered and unordered because the policies or rules, respectively are already in the right order after unmarshalling.</p> |
| <i>Unmarshalling</i> | <p>The JAXB generated an element additionalInformation in the PolicyType and the PolicySetType. This element has its origin in the fact that JAXB generates from a choice of such an element.</p> <p>Now to access for example the variable definitions or rules. The list must be accessed then the content must be tested of what instance it is and if it is for</p> |



example a variable definition a cast must be made to use this object. This work could be done in the `after unmarshal` method of the `Policy` or `PolicySet`, respectively, so this overhead only occurs while deploying when it does not matter at all.

Spring
Framework
Extensible XML
Authoring

The classes to handle the Spring Framework Extensible XML Authoring are in the PDP module. Some of the modules may be used on their own. This means that the classes to handle the Spring Framework Extensible XML Authoring should be moved to the module they configure. This allows configuration of module even if it is used on its own.



Appendix C: Known issues



1 Overview

- Introduction* This appendix contains known issues with the HERAS^{AF} XACML 2.0 implementation and if existing a workaround.
- Structures* The section contains all known issues listed up.

2 Time zone

2.1 Issue

The time, date and datetime functions all depend on time zone awareness of the implementation. The HERAS^{AF} XACML 2.0 implementation does not explicitly handle time zones. This means that the default Java time zone handling of the `GregorianCalendar` is used.

These functions do not handle the time zone correctly. There are functions where for example one argument has a time zone and another does not. In such a case the one without a time zone must use the one from the argument containing a time zone. All these scenarios are not handled.

2.2 Workaround

For this issue, no workaround is available. It will be fixed in the next release.



Appendix D: Discrepancies to the XACML 2.0 specification



1 Overview

- Introduction* This appendix contains discrepancies to the XACML 2.0 specification [XACML20].
- Structures* The section contains all discrepancies listed up.

2 AnyOfAll / AllOfAny

- Introduction* A dependency between several functions leads to a false evaluation of the PDP. The problem occurs if several Higher-Order Bag functions are nested and call a regular expression function at the end. On the basis of three pseudo code examples the problem is described in detail:

functions

AnyOf function

The pseudo code complies to the XACML 2.0 specification:

```
anyOf.handle(p1, p2, p3){
    for( e : p3 ){
        p1.handle(p2, e)
    }
}
```

Figure 80: Pseudo code of the AnyOf function

AllOfAny function

The pseudo code complies to the AllOfAny function of the XACML 2.0 specification:

```
AllOfAnyFunction.handle( string-regex-match, p2 , p3){
    for(e1 : p2){
        AllOfFunction.handle( string-regex-match, e , p3){
            for( e2 : p3 ){
                string-regex-match.handle(e1, e2)
            }
        }
    }
}
```

Figure 81: Pseudo code of the AllOfAny function.



AnyOfAll function

The pseudo code complies to the AnyOfAll function of the XACML 2.0 specification:

```
AnyOfAllFunction.handle( string-regex-match, p2 , p3){  
    for(e1 : p3){  
        AllOfFunction.handle( string-regex-match, e1, p2){  
            for( e2 : p2){  
                string-regex-match.handle(e2, e1)  
            }  
        }  
    }  
}
```

Figure 82: Pseudo code of the AnyOfAll function.

Origin of the discrepancy

The above written pseudo code 1 describes according to the XACML 2.0 specification how the AnyOf function should look like. This function is used by the functions AllOfAny and AnyOfAll.

The green part of pseudo code 2 and the blue part of pseudo code 3 are part of the AnyOf Function. Exactly at this position the problem occurs because the regular expression function accurately describes which of the parameters has to be the regular expression.

How is it possible that once the element from the list is at second position (green pseudo code part) and the other time at first position (blue pseudo code part)? In the pseudo code examples above the parameter e1 is one time at first and the other time at second position.

3 DayTimeDuration

Introduction

The DayTimeDuration implementation differs from the specification in several points.

Value in seconds

According to the specification the calculation of the value in seconds has the following algorithm:

```
('value of the day component' * 24) +  
( 'value of the hour component' * 60) +  
( 'value of the minute component' * 60) +  
( 'value of the second component' )
```

Figure 833: Calculation of the seconds according to the XACML 2.0 specification.

This algorithm doesn't properly calculate the seconds, because there are missing brackets. This is why this algorithm was changed to go sure it matches the intention of the author.

Form

The regular expression, which describes the form of a DayTimeDuration seems to have some failures. This has the effect, that the DayTimeDuration isn't a specialization of the xs:duration anymore.

The form was changed to ensure it is only a specialization of the xs:duration.



4 Unique ID over Policy and PolicySet

- Introduction* In the HERAS^{AF} XACML 2.0 implementation, the identifier of `Policy` and `PolicySets` has to be unique over both of them.
- Specification* According to the XACML 2.0 Specification, the identifier must be unique for the `Policy` and `PolicySet`, but not over both of them.
- Origin of the Discrepancy* Having the same identifier for a `Policy` and a `PolicySet` is a bad naming-style. This is why the HERAS^{AF} XACML 2.0 implementation does not support it.

5 CombinerParameters

- Introduction* The HERAS^{AF} XACML 2.0 implementation core does not support all the possibilities for the use of `CombinerParameters`.
- Discrepancy* In a `Policy` element, `CombinerParameters` can only be used in the choice part and not in the sequence between `PolicyDefaults` and the `Target`.
- Origin of the Discrepancy* The marshalling and unmarshalling of the data classes is implemented with JAXB 2.1. JAXB 2.1 does not support to differentiate the position of an element if it might appear in two different parts of the same element. Because of this, the HERAS^{AF} XACML 2.0 implementation does not support both possibilities.