UNIVERSIDADE DE AVEIRO

INSTITUTO DE TELECOMUNICAÇÕES

PORTUGAL TELECOM INOVAÇÃO

Policy Administration

OASIS XACML v3

24-03-2011
Projecto de Licenciatura em Tecnologias e Sistemas de Informação
do
Departamento de Electrónica, Telecomunicações e Informática

Autor Francisco Alexandre de Gouveia

Orientador Doutor Diogo Gomes

Colaborador Engenheiro Ricardo Azevedo

Resumo

No âmbito do projecto de licenciatura do curso de Tecnologias e Sistemas de Informação, foi-me dado como objectivo estudar a arquitectura do XACML (Extensible Access Control Markup Language), assim como fazer a implementação de um motor de políticas que o use e que seja compatível com as interfaces já implementadas no gestor de identidades PTIN. Este projecto será enquadrado no âmbito dos esforços desenvolvidos pela Portugal Telecom Inovação no projecto europeu SEMIRAMIS[1] e do grupo de investigação do Instituto de Telecomunicações.

Na primeira fase deste relatório é abordada a linguagem e arquitectura do padrão XACML definido pela OASIS, com alguns exemplos para a sua implementação. No decorrer do segundo semestre proceder-se-á ao desenvolvimento de um motor de políticas, cujo trabalho será documentado na segunda fase deste documento.

[1] http://www.semiramis-cip.eu/

Índice

Índice de Ilustrações

Acrónimos

- **CSS** - Cascade Style Sheet
- **HTML5** - HyperText Markup Language version 5
- **JSON** - JavaScript Object Notation
- **OASIS** - Organization for the Advancement of Structured Information Standards
- **PAP** - Policy Administration Point
- **PDP** - Policy Decision Point
- **PEP** - Policy Enforcement Point
- **PIP** - Policy Information Point
- **XACML** - eXtensible Access Control Markup Language
- **XHTML -** eXtensible HyperText Markup Language
- **XML** - eXtensible Markup Language

1. Introdução

Hoje em dia, com a crescente preocupação em relação à privacidade, surge a necessidade de um método de controlo de acessos que seja dinâmico e eficaz. Dinâmico porque não nos podemos limitar a assumir que um determinado acesso seja constante ao longo do tempo, é preciso que existam requisitos lógicos que induzam à tomada de decisão sobre a permissão de acesso ou à sua rejeição, dependendo do contexto associado. Por outro lado, eficaz porque os recursos a controlar poderão ser sensíveis e a segurança destes não poderá ser posta em causa.

Neste documento será abordada a linguagem e arquitectura da norma XACMLv2 e diferenças na nova norma XACMLv3. Esta arquitectura permite fazer um controlo de acesso com base em combinações de políticas e regras. No capítulo 3 será documentado todo o desenvolvimento da implementação de um sistema de informação para gestão de políticas.

Este projecto é realizado no âmbito do projecto de licenciatura e será enquadrado no âmbito dos esforços desenvolvidos pela Portugal Telecom Inovação no projecto europeu SEMIRAMIS[2] e do grupo de investigação do Instituto de Telecomunicações.

[2] http://www.semiramis-cip.eu/

2. eXtensible Access Control Markup Language (XACML) versão 2

Os controlos de acesso são necessários em cada vez mais sistemas de informação, tais como páginas de administração e gestão de conteúdos, controlos parentais, partilha de recursos, modelos de privacidade em redes sociais, entre muitos outros. Muitos sistemas de informação utilizam a sua própria implementação para fazer o seu controlo de acessos, sendo que estes muitas vezes não são muito eficazes, não abrangem todos os requisitos de controlo e não são facilmente reutilizáveis por outros sistemas, principalmente em outros contextos por serem demasiado específicos para a área à qual foram inicialmente concebidos.

A ideia da OASIS foi de definir uma arquitectura de controlo de acessos que fosse genérica, extensível e de fácil adaptação a qualquer contexto. Daí esta ter criado a linguagem XACML, com uma estrutura genérica facilmente integrável em qualquer sistema. A tomada de decisões desta linguagem baseia-se em avaliação de políticas, cujas regras definem que valores são aceitáveis em determinados atributos, para que seja concedida ou negada a autorização. Nos capítulos seguintes é explicado de uma forma mais clara e detalhada o seu funcionamento.

2.1.A linguagem

A linguagem XACML é uma extensão do XML (eXtensible Markup Language), o que lhe permite ter todas as vantagens do XML, tais como o suporte para uma fácil codificação e descodificação do seu conteúdo, devido aos processadores de XML já existentes, e a extensibilidade do seu conteúdo, o que possibilita definir novos tipos de dados, funções e algoritmos combinatórios para que nenhum requisito tenha de ficar de fora.

Conforme definido na especificação da OASIS[1], a linguagem XACML serve os propósitos de:

- Elaboração de conjuntos de políticas (**PolicySet**), políticas (**Policy**) e regras (**Rule**);
- Comunicação de pedidos de autorização (**Request**) e respectiva resposta (**Response**).

2.1.1. Elementos PolicySet, Policy e Rule

Para que um pedido de autorização seja concedido, devem de existir requisitos que devem ser preenchidos. Uma regra (**Rule**) é nada mais do que de um conjunto de atributos a serem calculados num predicado, que avalia se esses atributos estão dentro dos parâmetros aceitáveis.

Para uma tomada de decisão, uma regra pode não ser suficiente ou não abranger os casos possíveis e necessários, por isso existem as políticas (**Policy**) que contêm um conjunto de regras. Uma vez que podem existir políticas que se aplicam a mesmos fins (ver capítulo 2.1.2), as políticas são agrupadas em conjunto de políticas (**PolicySet**).

Através do diagrama apresentado na Ilustração 1, podemos perceber como são estruturados os dados.

Os elementos **Obligations**, **Conditions** e **Target** serão explicados mais à frente, assim como os algoritmos combinatórios.

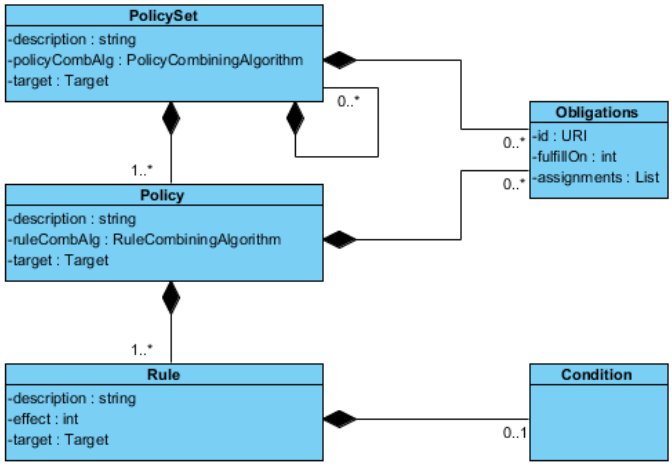

Ilustração 1 - Diagrama dos elementos PolicySet, Policy, Rule, Obligations e Condition

Os elementos **PolicySet**, **Policy** e **Rule** têm um campo não obrigatório com uma descrição, com a finalidade de facilitar a leitura das políticas e regras.

2.1.2. Elemento Target

As politicas e as regras são os elementos fundamentais para as tomadas de decisões, uma vez que estas contêm a informação e algoritmos necessários para determinar a autorização a um sujeito (**Subject**) de executar uma acção (**Action**) a um determinado recurso (**Resource**). Estes três últimos elementos fazem parte do alvo (**Target**), como pode ser visto na Ilustração 2.

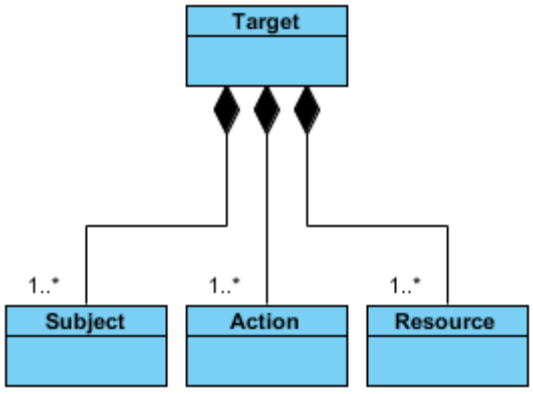

Ilustração 2 - Diagrama do elemento Target

A função do elemento **Target** é identificar a quem, que acção, e a que recurso se aplica a regra, politica ou conjunto de politicas onde este estiver inserido. O agrupamento de regras e políticas pode ser feito a partir de repetições de um dos elementos do **Target**. Isto é, se existir um ou mais elementos do **Target** que sejam iguais em vários elementos **Rule**, então esses **Rule** podem agrupar-se numa **Policy** cujo **Target** contém esses elementos repetidos, sendo que os restantes elementos do **Target** permanecem nas regras. Podemos verificar isso no exemplo ilustrativo (Ilustração 3). O mesmo se aplica para agrupamento de políticas num **PolicySet**, em que outros **PolicySet** podem ser agrupados juntamente com os elementos **Policy**.

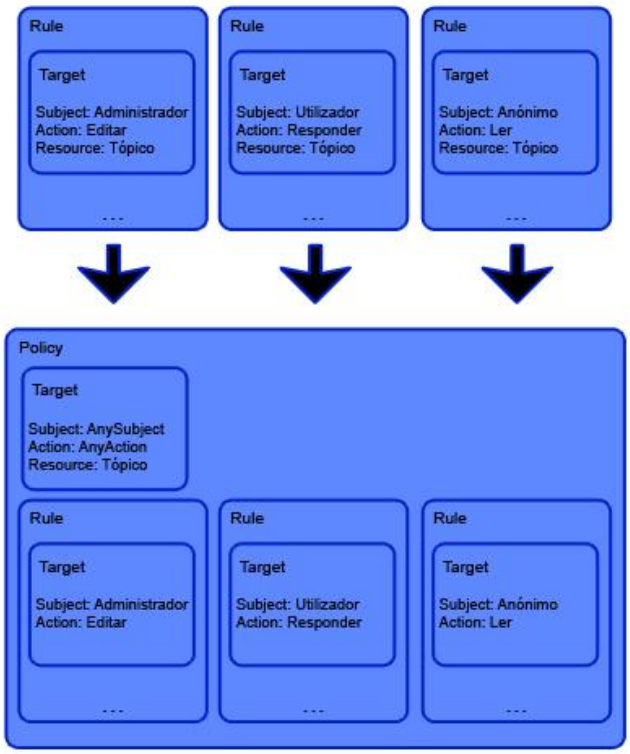

Ilustração 3 - Exemplo de agrupamento de regras numa política

Desta forma, temos uma árvore de políticas, em que a raiz é o **PolicySet** e as folhas são os elementos **Rule**, com os elementos **Policy** e outros **PolicySet** como ramos. Quando é feito um pedido de autorização, a procura de políticas começa sempre pelo topo, ou seja, pelo **PolicySet** cujo **Target** esteja de acordo com o pedido. Há que ter em atenção que este facto conduz a que o elemento **Target** seja obrigatoriamente definido no **PolicySet**. Após encontrado o **PolicySet** apropriado, são avaliados todos os **Policy** e **PolicySet** nele contido, ou seja, é novamente verificado o **Target**, mas de todos esses elementos. Dentro dos **Policy** aceites, são então verificados os **Target** dos elementos **Rule** neles contidos. Um elemento é aplicável numa tomada de decisões de um pedido de autorização se e só se os elementos do **Target** estiverem de acordo com o pedido. Caso contrário, o elemento não é aplicável e o seu resultado ignorado.

Caso algum elemento não tenha especificações no **Target** (por estas serem irrelevantes e se aplicarem a qualquer caso), estas especificações são naturalmente

herdadas do elemento superior. O facto deste elemento poder ter dados não especificados não significa que estes podem estar simplesmente omissos. Para esses casos, ou é utilizada uma lista vazia ou é colocada uma tag **<Any**<*Nome_elemento*> **/>** no caso do xml, como por exemplo **<AnySubject />**. Caso nenhum dos elementos do **Target** seja relevante, o target pode ser representado apenas por **<Target />**. Exemplos em xml são mostrados no capítulo 2.2.A avaliação dos elementos do **Target** é feito com recurso a predicados e os dados do pedido como argumentos.

2.1.3. Avaliação das regras

Uma das coisas mais importante numa regra, é saber qual o efeito que ela produz. Por outras palavras, qual será a autorização que a regra dará caso todos os valores estejam nos parâmetros aceitáveis. Para isso, existe um campo **Effect** no elemento **Rule**, conforme pode ser verificado na Ilustração 1. Com este campo, podemos definir se queremos que a autorização seja permitida colocando "Permit", ou negada colocando "Deny". O elemento **Target** do elemento **Rule** tem a função de avaliar se o sujeito pode executar determinada tarefa a um determinado recurso, e é este elemento que serve como base de avaliação dessa regra. Tomemos como exemplo a Ilustração 3, em que temos a seguinte informação:

- Subject: Anónimo
- Action: Ler
- Resource: Tópico

Imaginemos que o efeito dessa regra é de negação, ou seja, **Effect="Deny"**. Caso um utilizador anónimo tente ler o tópico, está a obedecer ao **Target** desta regra. Como o efeito desta regra é de negação, então a decisão final será que o anónimo não pode ler o tópico. Por outro lado, caso o efeito fosse de permitir (**Effect="Permit"**), então o anónimo poderia ler o tópico.

Algo importante de referir, é que caso as condições não se verifiquem, o resultado do elemento será "Not Applicable". Usando o exemplo anterior, com o atributo **Effect="Deny"**, se existir pelo menos uma condição que não seja satisfeita, então o resultado é "Not Applicable".

A avaliação do elemento **Target** não é a única forma de avaliar uma regra. A avaliação de uma regra pode ser feita através de condições definidas, como pode ser visto no capítulo seguinte, e com atributos de ambiente (quaisquer atributos que não façam parte do target).

2.1.4. Elemento Condition

A liberdade que o XACML nos dá, permite-nos criar predicados, tipos de dados e atributos que podem ser usados para a avaliação de condições adicionais nas regras.

Para exemplificar a função do elemento **Condition**, tomemos como exemplo a regra em que o **Target** é

- Subject: Anónimo
- Action: Ler
- Resource: Tópico

e que o campo **Effect="Deny"** se encontra nessa regra. Suponhamos que existe um atributo com o nome *numUtilizadores*, que retorna o número de utilizadores a usar o serviço. Entre muitos predicados que vêm de raiz implementados no XACML[1], existe um com o nome *integer-greater-than*, que segundo o nome indica, avalia se um valor numérico inteiro é maior do que outro valor numérico inteiro. Então colocamos nesta regra uma condição (**Condition**), em que comparamos o valor do atributo *numUtilizadores* com o número inteiro 5. Então temos que,

$$\left.\begin{array}{c} Subject = Anónimo \\ Action = Ler \\ Resource = Tópico \\ numUtilizadores > 5 \end{array}\right\} \Longrightarrow Autorização = "Deny"$$

Isto significa que se o utilizador anónimo tentar ler o tópico, mas estiverem mais do que 5 utilizadores a usar o sistema, a autorização nessa regra é negada.

Assim sendo, o elemento *Condition* tem como argumento uma expressão (por exemplo, um elemento Apply - ver página 9) que tenha como valor de retorno *boolean*.

2.1.5. Algoritmos combinatórios

Agora que está explicado como são estruturadas as políticas e como são identificadas para serem usadas, falta esclarecer como estas formam uma autorização a um pedido. Caso um **PolicySet** tenha apenas um **Policy**, que por sua vez tem apenas um elemento **Rule**, e estes estejam de acordo com o **Target** a ser avaliado, então a decisão de autorização final é o resultado desse **Rule**.

Mas se num **PolicySet** existirem mais do que um **Policy** ou mesmo outros **PolicySet**, que por sua vez têm mais do que um elemento **Rule**, em que é possível que mais do que uma **Policy** ou **Rule** se apliquem a uma determinada autorização, então são precisos algoritmos para tomar decisões com base em todos os resultados obtidos. De raiz, existem quatro algoritmos combinatórios para conjunto de políticas e outros quatro idênticos para conjunto de regras. Estes algoritmos são:

- **permit-overrides** - Se pelo menos um elemento permitir autorização, então a decisão final será "Permit". Podemos pensar no cálculo da decisão da seguinte forma: $ResFinal = R1 \vee R2 \vee \cdots \vee Rn$, em que R é o conjunto de resultados obtidos de elementos aplicáveis, cujo domínio é {verdadeiro (permit), falso (deny)};
- **deny-overrides** - Se pelo menos um elemento negar autorização, então a decisão final será "Deny". Podemos pensar no cálculo da decisão da seguinte forma: $ResFinal = R1 \wedge R2 \wedge \cdots \wedge Rn$, em que R é o conjunto de resultados obtidos de elementos aplicáveis, cujo domínio é {verdadeiro (permit), falso (deny)};
- **first-applicable** - Apenas o primeiro elemento aplicável à autorização é avaliado e a decisão final é o resultado desse elemento. Neste caso, a ordem é importante, porque podem existir resultados diferentes consoante a ordem dos elementos, o que pode ter como consequência resultados indesejados;
- **only-one-applicable** - Caso exista apenas um elemento aplicável a ser avaliado, então o resultado final será o resultado desse elemento. Se existir mais do que um caso aplicável, então o resultado é indeterminado ("Indeterminate").

Em todos estes algoritmos, caso não existam elementos que se apliquem à autorização, então o resultado é "Not applicable", ou seja, não existem políticas aplicáveis a essa autorização.

2.1.6. Elemento Obligations

Algumas políticas têm um conjunto de obrigações que têm de ser cumpridas para que o acesso seja concedido ou negado. Estas obrigações são enviadas juntamente com a resposta da autorização. As obrigações podem ser definidas para quando a decisão da política for de permissão e para quando a decisão for de negação. O campo para definir quando a obrigação deve ser enviada é o **FulfillOn**, que recebe como parâmetro "Permit" ou "Deny", respectivamente. Conforme especificado,

PEPs that conform with v2.0 of XACML are required to deny access unless they understand and can discharge all of the Obligations elements associated with the applicable policy. [1]

Ou seja, caso não seja possível a execução de alguma das obrigações, a decisão de acesso deverá ser sempre de negação (Informações sobre PEP, consultar capítulo 2.3.2).

2.1.7. Atributos, funções e predicados

Para avaliação de condições ou *target*'s, é necessário comparar os valores dos atributos da política com os valores dos atributos do pedido (*request*). Para melhor compreensão do *request*, ver página 14. Para que essas comparações sejam feitas, é necessário recorrer a funções de comparação ou predicados.

Elemento AttributeDesignator

Quando é necessário obter um valor de um atributo que se encontra no pedido, é usado o elemento *AttributeDesignator*. Os *AttributeDesignator*'s são responsáveis por procurar no xml de *request* os atributos indicados.

Como exemplo, se existir no *request* um atributo cujo *AttributeId* é "utilizador" com valor "Anónimo" e na política a ser analisada existir um *AttributeDesignator* com *AttributeId* "utilizador", então o valor atribuído ao *AttributeDesignator* será "Anónimo".

Algo a ter em atenção é que pode existir mais do que um valor encontrado pelo AttributeDesignator, por isso é preciso filtra-lo com o elemento *Apply* quando queremos apenas um valor (ver Elemento Apply).

Elemento AttributeSelector

Caso o atributo desejado não faça parte do *request*, o AttributeSelector é responsável por procura-lo no sítio certo. Por exemplo, se existir um *request* com informação completa de um utilizador em xml dentro de um elemento, o AttributeSelector tem de ler essa informação recorrendo a XPath para obter os valores desejados.

Tal como o AttributeDesignator, este elemento pode encontrar mais do que um valor.

Elemento AttributeValue

Para definir valores estáticos comparáveis, é usado o AttributeValue. É normalmente usado para definir valores esperados das políticas para que estas sejam aplicadas.

Os atributos anteriormente referidos são usados em funções ou predicados de comparação. A seguir são apresentados os elementos onde possam ser utilizados.

Elemento Apply

O elemento *Apply* é usado como função com retorno de um ou mais elementos, cujos argumentos são todos os elementos nele contidos. Normalmente é utilizado para filtrar valores de *AttributeDesignator* ou *AttributeSelector* quando é desejado apenas um dos valores atribuídos - isto acontece porque estes elementos podem encontrar um ou mais valores (ver Elemento AttributeDesignator e Elemento AttributeSelector). A função a ser utilizada é identificada no seu atributo *FunctionId*, cuja implementação se encontra no *Policy Decision Point* (ver capítulo 2.3.1). Para uma lista exaustiva de funções, ver a documentação da especificação do XACML v2 da OASIS [1].

Elemento Match

Para comparar os atributos esperados com os atributos do pedido, são usados elementos de *Match* (comparação). Estes elementos têm como argumentos dois atributos: um Attribute Value juntamente com um Attribute Designator ou Selector. Para uma lista exaustiva de funções de *match*, ver a documentação da especificação do XACML v2 da OASIS [1].

2.2.A implementação dos ficheiros

Com todo o conhecimento adquirido até aqui, é agora possível perceber como é elaborado um ficheiro de políticas em xml, assim como os ficheiros de pedido de autorização e de resposta. Nos subcapítulos seguintes será demonstrado como estes são implementados na prática.

2.2.1. Implementação do ficheiro policy.xml

Começando pela elaboração do ficheiro de políticas, podemos analisar a Ilustração 1 como guia de implementação. O ficheiro de políticas tem sempre como base um **PolicySet**. Em xml, este será o elemento de raiz. Dentro de um **PolicySet** temos elementos **Policy** e outros elementos **PolicySet**. Por sua vez, os elementos **Policy** contêm elementos **Rule**. Então, podemos começar por fazer um esqueleto do ficheiro, como é mostrado na Ilustração 4.

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

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

Ilustração 4 - Esqueleto do ficheiro de politicas em xml

Dentro de cada um destes elementos deve estar um elemento **Target**, que contém os elementos **Subjects**, **Actions** e **Resources**. Caso algum destes três elementos seja irrelevante, ou seja, que pode ser aplicado independente do pedido de autorização, é feito, como já explicado no capítulo 2.1.2, com recurso à tag **<Any**<Nome_Elemento>**/>**. Tomemos como exemplo uma regra em que o utilizador anónimo não pode executar qualquer acção num tópico. Na Ilustração 5 pode-se verificar como ficaria estruturado o elemento **Rule**. Há que ter em atenção que não é a implementação final, uma vez que é apenas um exemplo ilustrativo e não contém todo o detalhe necessário, apenas o essencial para perceber de uma forma mais clara.

```
<Rule Effect="Deny">
        <Target>
                <Subjects>
                        <Subject>
                                <SubjectMatch MatchId="string-equal">
                                        <SubjectAttributeDesignator AttributeId="utilizador" DataType="string"/>
                                        <AttributeValue  DataType="string">
                                                Anónimo
                                        </AttributeValue>
                                </SubjectMatch>
                        </Subject>
                </Subjects>
                <Actions>
                        <AnyAction />
                </Actions>
```

```
<Resources>
    <Resource>
        <ResourceMatch MatchId="string-equal">
            <ResourceAttributeDesignator AttributeId="tipo" DataType="string"/>
            <AttributeValue DataType="string">
                Tópico
            </AttributeValue>
        </ResourceMatch>
    </Resource>
</Resources>
</Target>
</Rule>
```

Ilustração 5 - Exemplificação de implementação do elemento Target num elemento Rule

O mesmo se aplica na implementação do elemento **Target** nos elementos **PolicySet** e **Policy**.

A análise dos elementos contidos no **Target** é feita a partir de condições calculadas com um predicado, no caso do **Subject** e do **Resource** o predicado *string-equal,* para verificar a equidade de duas strings, neste caso para comparar o valor do atributo *utilizador* da regra (Anónimo) com o valor do atributo *utilizador* do pedido.

Podemos verificar neste exemplo como deverá ser usado o atributo **Effect** no elemento **Rule**, que neste caso define a negação da autorização, caso a regra se verifique.

Caso existam outras condições além dos elementos do **Target** que devam ser avaliadas, devem ser acrescentados elementos **Condition** ao elemento **Rule**. A Ilustração 6 exemplifica uma **Condition** implementada.

```
<Condition FunctionId="integer-greater-than">
    <Apply FunctionId="integer-one-and-only">
        <EnvironmentAttributeDesignator DataType="integer"
        AttributeId="numUtilizadores"/>
    </Apply>
    <AttributeValue DataType="integer">
        3
    </AttributeValue>
</Condition>
```

Ilustração 6 - Exemplo de uma condição em xacml

A condição verifica se o número de utilizadores é maior do que 3. O que esta condição faz, quando inserida na regra da Ilustração 5, é negar ao anónimo a autorização de executar qualquer tarefa no tópico apenas quando o número de utilizadores no sistema é maior do que 3. Atenção que a não satisfação desta condição não implica a permissão de autorização. Usando este caso, quando esta condição não é satisfeita (quando o número de utilizadores é menor ou igual a 3), então a regra é não aplicável.

A função *integer-one-and-only* certifica-se que existe apenas um valor inteiro. É necessária esta aplicação porque o **AttributeDesignator** retorna um número indeterminado de valores num elemento **Bag**, e para a função *integer-greater-than* funcionar, esta tem de receber dois argumentos, em que neste caso o segundo argumento é o valor inteiro 3.

A função do **AttributeDesignator** é de encontrar valores de um atributo no xml de *request*, dado o seu nome. Neste exemplo, o atributo é o *numUtilizadores* e é um atributo personalizado.

Algumas políticas podem ter obrigações que devem de ser tidas em conta pelo requisitante da autorização. Mais informações sobre obrigações no capítulo 2.1.6.

```
<Obligations>
        <Obligation FulfillOn="Deny" ObligationId="ob1">
                <AttributeAssignment AttributeId="nextReqInSec" DataType="integer">30</AttributeAssignment>
        </Obligation>
</Obligations>
```

Ilustração 7 - Exemplificação do elemento Obligation no ficheiro xml

Ao colocar esta obrigação numa política, quando um pedido de autorização resultar na negação da autorização vinda dessa política, é enviado ao requisitante a decisão juntamente com esta obrigação. É suposto que o requisitante consiga interpretar a obrigação e a cumpra, mas caso esta obrigação não seja cumprida não implica a alteração do resultado da autorização recebida, conforme explicado anteriormente. Caso o requisitante tivesse uma função associada ao atributo *nextReqInSec*, esta deveria ser executada com o valor 30 no seu argumento de entrada.

Para terminar, e após juntar todas as peças de exemplo, é preciso completar o que está em falta, para que esta seja uma política válida.

O que falta colocar são os algoritmos combinatórios na política, e os URI's nos tipos de dados, nos nomes das funções e nos identificadores. Esses URI's são definidos por convenção para que o PDP consiga interpretar tudo o que se encontra no ficheiro de políticas. Essas convenções encontram-se no documento da especificação do XACML[1]. O resultado final pode ser viso na Ilustração 8.

```
<PolicySet PolicyCombiningAlgId="urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:deny-overrides">
        <Target>
                <Subjects>
                        <AnySubject/>
```

```
                </Subjects>
                <Actions>
                        <AnyAction/>
                </Actions>
                <Resources>
                        <Resource>
                                <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
                                        <ResourceAttributeDesignator AttributeId="tipo"
DataType="http://www.w3.org/2001/XMLSchema#string"/>
                                        <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">Tópico</AttributeValue>
                                </ResourceMatch>
                        </Resource>
                </Resources>
        </Target>
        <Policy RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-overrides">
                <Target/>
                <Rule Effect="Deny">
                        <Target>
                                <Subjects>
                                        <Subject>
                                                <SubjectMatch
MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
                                                        <SubjectAttributeDesignator AttributeId="utilizador"
DataType="http://www.w3.org/2001/XMLSchema#string"/>
                                                        <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">Anónimo</AttributeValue>
                                                </SubjectMatch>
                                        </Subject>
                                </Subjects>
                                <Actions>
                                        <AnyAction/>
                                </Actions>
                                <Resources>
                                        <AnyResource/>
                                </Resources>
                        </Target>
                        <Condition FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-greater-than">
                                <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-one-and-only">
                                        <EnvironmentAttributeDesignator
DataType="http://www.w3.org/2001/XMLSchema#integer" AttributeId="numUtilizadores"/>
                                </Apply>
                                <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#integer">3</AttributeValue>
                        </Condition>
                </Rule>
                <Obligations>
                        <Obligation FulfillOn="Deny" ObligationId="ob1">
                                <AttributeAssignment AttributeId="nextReqInSec"
DataType="http://www.w3.org/2001/XMLSchema#integer">30</AttributeAssignment>
                        </Obligation>
                </Obligations>
        </Policy>
</PolicySet>
```

Ilustração 8 - Exemplo de um ficheiro xml de políticas

2.2.2. Implementação do ficheiro request.xml

Para uma aplicação questionar uma autorização, precisa de comunicar com a entidade que toma decisões de forma a que esta possa interpretar o pedido. Esta comunicação é feita também com um ficheiro xml.

A Ilustração 9 mostra um diagrama com a estrutura do ficheiro xml. Este ficheiro é constituído por três elementos, que juntos formam uma questão: *"O sujeito (**Subject**) pode realizar a acção (**Action**) no determinado recurso (**Resource**)?"*. Estes elementos são os identificadores usados para avaliar o elemento **Target** das políticas e regras.

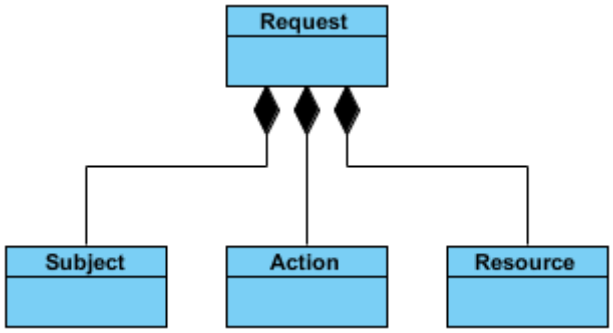

Ilustração 9 - Diagrama com a estrutura do Request

```
<Request>
        <Subject>
                <Attribute AttributeId="utilizador" DataType="http://www.w3.org/2001/XMLSchema#string">
                <AttributeValue>Anónimo</AttributeValue>
        </Attribute>
        </Subject>
        <Action>
                <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
                DataType="http://www.w3.org/2001/XMLSchema#string">
                <AttributeValue>Ler</AttributeValue>
        </Attribute>
        </Action>
        <Resource>
                <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
                DataType="http://www.w3.org/2001/XMLSchema#anyURI">
                <AttributeValue>Tópico</AttributeValue>
        </Attribute>
        </Resource>
</Request>
```

Ilustração 10 - Exemplo de ficheiro request.xml

Na Ilustração 10 está a implementação possível para um pedido que pode ser avaliado na política da Ilustração 8. Neste pedido de autorização, é questionado se o *Anónimo* pode *Ler* o *Tópico*.

2.2.3. Implementação do ficheiro response.xml

Quando é feito um pedido de autorização, é esperado que se receba uma resposta com uma decisão e com obrigações, caso existam. Essa resposta é um ficheiro xml com a estrutura ilustrada na Ilustração 11.

A resposta pode trazer mais do que um resultado, em que cada resultado tem uma decisão referente a um determinado recurso. Podem existir também obrigações a ser cumpridas e um código de estado para indicar se existiram erros durante a avaliação do pedido. Os elementos **AttributeAssignment** servem como argumentos da obrigação para o requisitante interpretar.

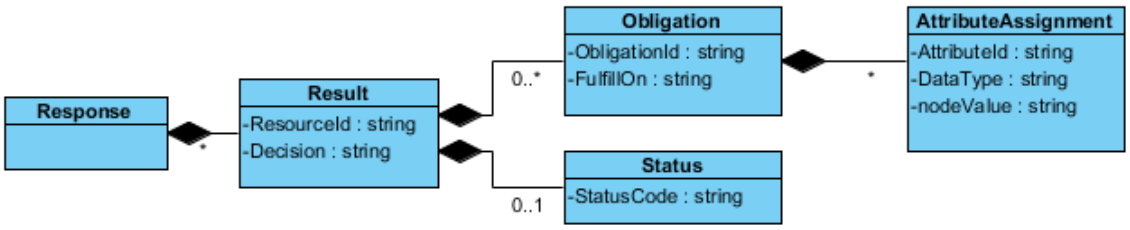

Ilustração 11 - Diagrama com a estrutura do Response

O ficheiro xml de resposta ao pedido será idêntico ao que é apresentado na Ilustração 12.

```
<Response>
	<Result ResourceID="Tópico">
		<Decision>Deny</Decision>
		<Status>
			<StatusCode Value="urn:oasis:names:tc:xacml:1.0:status:ok"/>
		</Status>
		<Obligations>
			<Obligation ObligationId="ob1" FulfillOn="Deny">
				<AttributeAssignment AttributeId="nextReqInSec"
				DataType="http://www.w3.org/2001/XMLSchema#integer">30
				</AttributeAssignment>
			</Obligation>
		</Obligations>
	</Result>
</Response>
```

Ilustração 12 - Exemplo de ficheiro response.xml

Este caso seria uma resposta possível ao pedido da Ilustração 10 se o número de utilizadores no sistema fosse maior do que 3. Neste caso o requisitante teria de ter o acesso negado ao recurso e deveria executar a acção apresentada nas obrigações com o argumento numérico *30*.

2.3.A arquitectura

Tendo a noção de como funciona a linguagem, é importante saber para que propósito e por que componentes esta é utilizada.

Segundo a arquitectura XACML definida pela OASIS, na arquitectura de um motor de políticas devem de existir os seguintes componentes:

- **Policy Decision Point (PDP)** - Ponto onde são calculadas as decisões;
- **Policy Enforcement Point (PEP)** - Implementado na aplicação que pretende aceder a um recurso, é esta que faz o pedido de autorização a um *PDP* e interpreta o resultado recebido;
- **Policy Administration Point (PAP)** - Ponto de criação, gestão e armazenamento de políticas;
- **Policy Information Point (PIP)** - Local onde estão alojados os atributos referentes às políticas.

2.3.1. Policy Decision Point

Este é a entidade que recebe pedidos de autorização de um **PEP** (ver capítulo 2.3.2). Recebidos esses pedidos, são verificadas as políticas que se aplicam e é avaliada a permissão ou negação de autorização. Aqui são também recolhidas todas as obrigações (ver capítulo 2.1.6) a serem enviadas na resposta ao **PEP**, junto com a decisão de autorização.

2.3.2. Policy Enforcement Point

Esta entidade deve de fazer parte da aplicação no qual queremos controlar as autorizações. Esta entidade é responsável por enviar os pedidos de autorização e por interpretar as respostas obtidas, assim como tentar resolver todas as obrigações recebidas nas respostas. Caso esta não consiga executar as obrigações, então esta deve de tomar sempre a decisão de negar a autorização.

2.3.3. Policy Administration Point

Na especificação não é clara a implementação deste ponto, apenas o caracteriza como uma entidade responsável para criação e gestão de políticas.

2.3.4. Policy Information Point

Esta entidade é apenas referenciada como fonte de atributos, mas não é especificada quanto à sua implementação e funcionalidade.

Segundo David Brossard,

Policy Information Points are attribute stores. They can be any format and located anywhere. PIPs are to XACML what the DVLA, the set of police records, the census bureau, etc… are to a nation and its citizens.[2]

Ou seja, o **PIP** é a fonte de informação de determinados atributos. Por exemplo, este pode ser um sistema de informação onde estão guardadas todas as informações dos utilizadores, e essa informação pode ser obtida tendo como base o identificador do sujeito recebida no *request.xml*. Podem existir diferentes **PIP** para diferentes informações.

2.4.Fluxo do processo de autorização

Para perceber como é que as entidades comunicam entre si, e que tipo de informação usam, o diagrama da ilustração X mostra o desenrolar do processo de autorização.

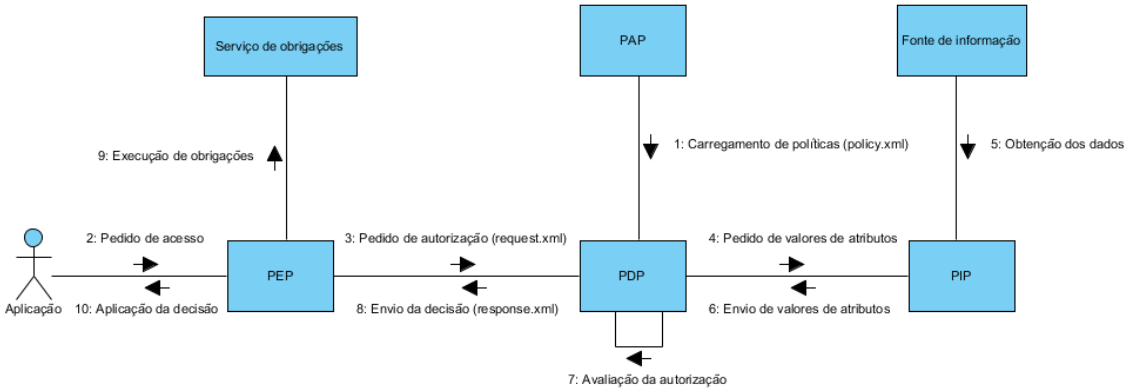

Ilustração 13 - Fluxo do processo de autorização

1 - PAP encarrega-se de carregar as políticas para o PDP;

2 - É pedido ao PEP um acesso de um sujeito executar uma acção a um determinado recurso;

3 - PEP comunica com PDP, enviando um *request.xml* com os dados referentes ao pedido;

4 - PDP recebe o request e pede atributos em falta ao PIP;

5 - PIP obtém dados dos atributos requisitados (diferentes dados podem vir de diferentes fontes de informação);

6 - PIP envia ao PDP os valores dos atributos;

7 - PDP avalia a autorização usando os dados do pedido em conjunto com os dados obtidos do PIP nas políticas existentes;

8 - O PDP comunica a decisão tomada ao PEP, enviando um response.xml com a decisão e as obrigações que devem ser cumpridas;

9 - PEP executa as obrigações.

10 - Concede ou nega acesso, tendo em conta a decisão e a possibilidade de execução de todas as obrigações.

2.4.1. Considerações

Para um bom funcionamento e comunicação entre o PEP e o PDP é necessário que estes estejam em sintonia, ou seja, um PEP não deve de comunicar com um PDP que não esteja preparado para interpretar os seus pedidos de autorização, nem comunicar com um PDP cujas obrigações não sejam interpretáveis por este, conforme esclarecido por Anne Anderson, uma das principais construtoras do XACML [3]:

When a PEP makes use of [i.e. is configured to use] a given PDP, the PEP is assuming that the PDP is supplied with policies by a PAP responsible for issuing policies related to the resource with which the PEP is associated. Both the developer of a resource's PEP, and the developer of the PAP that will provision that PEP's PDP must know the authorization model for protection of the PEP's resources, the meanings of the Attributes and values that the PEP is expected to supply, and the meaning and values of Obligations that the policies used by the PDP may return. [4]

2.5.eXtensible Access Control Markup Language (XACML) versão 3

A versão 3 do XACML ainda é uma especificação em desenvolvimento, existem poucas implementações e das poucas que existem, algumas não são *open source*.

Nos subcapítulos seguintes serão apresentadas algumas diferenças na arquitectura e implementação em relação à versão 2. Numa visão geral, o diagrama apresentado na Ilustração 14 contém a estrutura do XACMLv3.

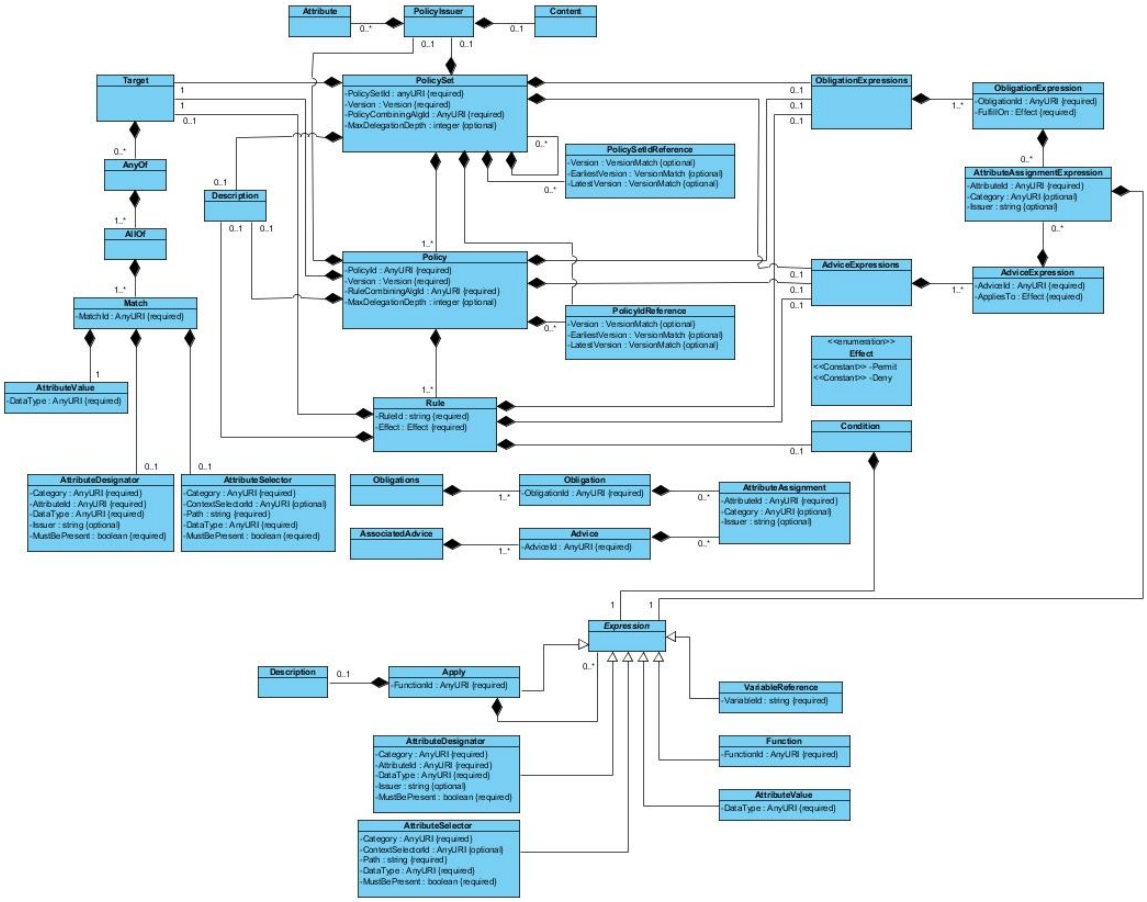

Ilustração 14 - Diagrama com a estrutura do documento de políticas do XACMLv3

2.5.1. Uso obrigatório de Namespace nos documentos

Segundo o XML Schema disponibilizado pela OASIS[5], o uso de prefixos para elementos XACML é obrigatório. Isto significa que nos documentos de políticas, pedidos e respostas o prefixo *xacml* deve ser usado e o namespace *urn:oasis:names:tc:xacml:3.0:core:schema:wd-17* declarado. Na ilustração seguinte está demonstrado como o fazer.

```
<xacml:PolicySet xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
        xmlns:xacml="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17 http://docs.oasis-
open.org/xacml/3.0/xacml-core-v3-schema-wd-17.xsd"
        PolicySetId="RootPolicySet"
        PolicyCombiningAlgId="urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:permit-overrides"
        Version="1">
  <xacml:Target/>
  <xacml:Policy PolicyId="Administrators"
```

```
                Version="1"
                RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-overrides">
    <xacml:Target>
      <xacml:AnyOf>
        <xacml:AllOf>
          <xacml:Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
            <xacml:AttributeValue DataType="http://www.w3.org/2001/XMLSchema#anyURI">
              Administrador
            </xacml:AttributeValue>
            <xacml:AttributeDesignator
              Category="urn:oasis:names:tc:xacml:3.0:attribute-category:access-subject"
              AttributeId="group"
              MustBePresent="true"
              DataType="http://www.w3.org/2001/XMLSchema#anyURI" />
          </xacml:Match>
        </xacml:AllOf>
      </xacml:AnyOf>
    </xacml:Target>
  </xacml:Policy>
</xacml:PolicySet>
```

Ilustração 15 - Uso de namespace em XACMLv3

2.5.2. Elemento Target

Na versão 2 do XACML, o elemento *Target* contém elementos para *Subject*, *Action*, *Resource* e *Environment*. Na versão 3 esses elementos são substituídos por um atributo que identifica a categoria onde estes se inserem. Outra novidade é o facto de existir forma de definir intersecções e reuniões destes elementos.

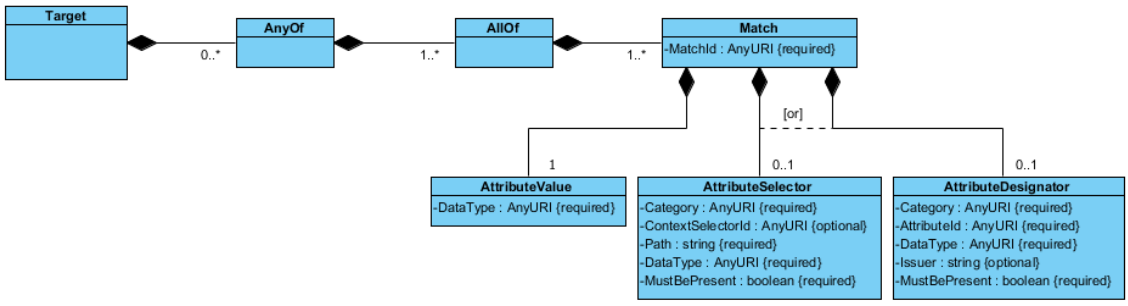

Ilustração 16 - Constituição do elemento Target no XACMLv3

Como é possível verificar na Ilustração 16, os elementos *AttributeSelector* e *AttributeDesignator* têm um atributo *Category*. Este atributo identifica a categoria onde este elemento se insere, e deve ser identificado com um dos seguintes identificadores:

- urn:oasis:names:tc:xacml:3.0:attribute-category:resource
- urn:oasis:names:tc:xacml:3.0:attribute-category:action
- urn:oasis:names:tc:xacml:3.0:attribute-category:environment

Em relação aos identificadores que substituem o elemento *Subject*, são usados os mesmos desde a versão 1 do XACML.

Para identificar a entidade que iniciou o pedido de autorização, é usado o seguinte identificador:

- urn:oasis:names:tc:xacml:1.0:subject-category:access-subject

Para identificar a identidade que irá receber o resultado do pedido, é usado o seguinte identificador:

- urn:oasis:names:tc:xacml:1.0:subject-category:recipient-subject

Para identificar os intermediários por onde o pedido passa, é usado o seguinte identificador:

- urn:oasis:names:tc:xacml:1.0:subject-category:intermediary-subject

Neste último caso, podem existir vários intermediários e a especificação não pressupões como estes devem ser ordenados. Estes intermediários são usados em processos de delegação.

Outros identificadores que podem ser usados para esta categoria são:

- urn:oasis:names:tc:xacml:1.0:subject-category:codebase
- urn:oasis:names:tc:xacml:1.0:subject-category:requesting-machine

Para reunir estes elementos, e criar vários conjuntos de elementos para que um Target seja aplicável, existem os elementos AnyOf e AllOf.

Avaliação de um elemento Target

Um elemento *Target*, *AnyOf* ou *AllOf* quando avaliado, pode ter três tipos de resultado [1]:

- Match - É aplicável no contexto;
- No Match - Não é aplicável no contexto;
- Indeterminate - Existe uma indeterminação quanto à sua aplicabilidade.

No primeiro caso, para que o elemento *Target* seja avaliado como ***Match***, é necessário que **todos** os elementos *AnyOf* nele contido sejam avaliados como ***Match***.

Para que o elemento seja avaliado como ***No Match***, basta que **pelo menos um** dos elementos *AnyOf* nele contidos seja avaliado como ***No Match***.

Em qualquer outro caso, o elemento *Target* é avaliado como ***Indeterminate***.

Avaliação de um elemento AnyOf

O elemento *AnyOf* pode ser avaliado em três tipos de resultados (ver Avaliação de um elemento Target).

Para que o elemento *AnyOf* seja avaliado como ***Match***, basta que **pelo menos um** dos elementos *AllOf* nele contido seja avaliado como ***Match***.

Para que o elemento seja avaliado como ***Indeterminate***, **nenhum dos elementos** *AllOf* é avaliado como ***Match*** e **pelo menos um** avaliado como ***Indeterminate***.

Para que seja avaliado como ***No Match***, **todos** os elementos AllOf nele contido devem ser avaliados como ***No Match***.

Avaliação de um elemento AllOf

O elemento *AllOf* pode ser avaliado em três tipos de resultado resultados (ver Avaliação de um elemento Target).

Para que o elemento *AllOf* seja avaliado como ***Match***, **todos** os elementos *Match* nele contido devem ser avaliados como ***True*** (valor booleano verdadeiro).

Para que o elemento seja avaliado como ***No Match***, **pelo menos um** dos elementos *Match* nele contido deve ser ***False*** (valor booleano falso).

Para que seja avaliado como ***Indeterminate***, basta que **pelo menos um** elemento *Match* nele contido seja ***Indeterminate*** e o resto seja ***True***.

Implementação

A Ilustração 17 exemplifica a aplicação de um elemento Target.

```
<xacml3:Target>
  <xacml3:AnyOf>
    <xacml3:AllOf>
```

```
      <xacml3:Match MatchId="urn:oasis:names:tc:xacml:2.0:function:ipAddress-regexp-match">
        <xacml3:AttributeDesignator Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
                    AttributeId="com:tsiua:computer:ipaddress"
                    DataType="urn:oasis:names:tc:xacml:2.0:data-type:ipAddress"
                    MustBePresent="true" />
        <xacml3:AttributeValue DataType="urn:oasis:names:tc:xacml:2.0:data-type:ipAddress">
          192.168.1.1
        </xacml3:AttributeValue>
      </xacml3:Match>
      <xacml3:Match MatchId="urn:oasis:names:tc:xacml:2.0:function:time-in-range">
        <xacml3:AttributeDesignator Category="urn:oasis:names:tc:xacml:3.0:attribute-category:environment"
                    AttributeId="urn:oasis:names:tc:xacml:1.0:environment:current-time"
                    DataType="http://www.w3.org/2001/XMLSchema#time"
                    MustBePresent="true" />
        <xacml3:AttributeValue DataType="http://www.w3.org/2001/XMLSchema#time">
          08:00:00
        </xacml3:AttributeValue>
        <xacml3:AttributeValue DataType="http://www.w3.org/2001/XMLSchema#time">
          12:30:00
        </xacml3:AttributeValue>
      </xacml3:Match>
    </xacml3:AllOf>
    <xacml3:AllOf>
      <xacml3:Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
        <xacml3:AttributeDesignator Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
                    AttributeId="com:user:group"
                    DataType="http://www.w3.org/2001/XMLSchema#string"
                    MustBePresent="true" />
        <xacml3:AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
          administrator
        </xacml3:AttributeValue>
      </xacml3:Match>
    </xacml3:AllOf>
  </xacml3:AnyOf>
  <xacml3:AnyOf>
    <xacml3:AllOf>
      <xacml3:Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
        <xacml3:AttributeDesignator Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action"
                    AttributeId="com:tsiua:computer:command"
                    DataType="http://www.w3.org/2001/XMLSchema#string"
                    MustBePresent="true" />
        <xacml3:AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
          shutdown
        </xacml3:AttributeValue>
      </xacml3:Match>
    </xacml3:AllOf>
  </xacml3:AnyOf>
</xacml3:Target>
```

Ilustração 17 - Elemento Target em XACMLv3

Segundo o que foi descrito na Avaliação de um elemento Target, podemos passar este elemento para uma expressão lógica:

$$R = ([subject\_ipaddress == 192.168.1.1 \wedge 08:00:00 < currenttime < 12:30:00] \vee [subject\_group == administrator]) \wedge (command == shutdown)$$

Em que R é o resultado da avaliação.

Isto significa que o *Target* é aplicável em dois casos:

- o comando for shutdown e o requisitante pertencer ao grupo de administradores;
- o comando for shutdown, a hora actual for entre 8:00 e 12:30 e o requisitante tiver o endereço ip 192.168.1.1.

2.5.3. Request

Da mesma forma que se suprimem elementos de categorias de *Subject*, *Action*, *Resource* e *Environment* do elemento *Target*, o xml de request passa a ter apenas atributos categorizados através do atributo *Category*.

```
<xacml3:Request  xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
            xmlns:xacml3='urn:oasis:names:tc:xacml:3.0:core:schema:wd-17'
            xsi:schemaLocation='urn:oasis:names:tc:xacml:3.0:core:schema:wd-17 xacml-core-v3-schema-wd-
17.xsd'
            ReturnPolicyIdList="true"
            CombinedDecision="false">
  <xacml3:Attributes Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject">
    <xacml3:Attribute AttributeId="com:tsiua:datacenter:computers"
                IncludeInResult="true">
      <xacml3:AttributeValue DataType="urn:oasis:names:tc:xacml:2.0:data-type:ipAddress">
        192.168.0.1
      </xacml3:AttributeValue>
    </xacml3:Attribute>
  </xacml3:Attributes>
  <xacml3:Attributes Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource">
    <xacml3:Attribute AttributeId="com:tsiua:datacenter:computers"
                IncludeInResult="true">
      <xacml3:AttributeValue DataType="urn:oasis:names:tc:xacml:2.0:data-type:ipAddress">
        192.168.0.124
      </xacml3:AttributeValue>
    </xacml3:Attribute>
  </xacml3:Attributes>
</xacml3:Request>
```

Ilustração 18 - Request em XACMLv3

2.5.4. Multi-request

Embora seja opcional esta implementação, a OASIS criou uma especificação para envio de múltiplos pedidos num só *request*. Isto permite um ganho de performance caso o PEP deseje fazer mais do que um pedido ao PDP, enviando todos os pedidos de uma só vez no num *request*, e recebendo apenas um *response* com os resultados.

2.5.5. Elemento Obligation e elemento Advice

Juntamente com uma tomada de decisão de acesso, por vezes é obrigatório que o Policy Enforcement Point (PEP) execute algumas acções para que a autorização seja

finalmente concedida. Na norma XACMLv2 existe o elemento Obligation para identificar essas acções (ver capítulo 2.1.6).

Entretanto surgiu uma dúvida quanto ao que fazer quando o PEP não consegue executar alguma acção, sendo esta irrelevante em relação à tomada de decisão (como por exemplo, um pedido de *logging*, quando este está desactivado no PEP). Uma solução seria tratar esse tipo de obrigações como um esforço de executar uma acção, ficando nesse caso ao critério do PEP decidir se determinada obrigação é relevante ou não à tomada de decisão, respondendo ao PDP que a tarefa foi executada, mesmo tento esta falhado.

Na norma do XACMLv3 esse problema é resolvido com um novo elemento idêntico ao *Obligation*, o elemento *Advice*. O objectivo do elemento *Advice* é de identificar acções que devem ser realizadas pelo PEP, caso este as consiga realizar. O insucesso da realização destas acções não influencia a decisão final, ao contrário das *Obligations*.

3. Policy Administration Point

Com base no que foi estudado, será desenvolvida uma aplicação com interface web para gestão de políticas. Este será um dos componentes da arquitectura do XACML, mais especificamente um Policy Administration Point (PAP). Nos capítulos seguintes será abordada toda a arquitectura e implementação, baseada em objectivos e respectivas soluções.

3.1.Objectivos

3.1.1. Interface web intuitiva e usável

Neste momento existem gestores de políticas para XACML que se encontram desactualizados (apenas compatíveis com a versão 2 do XACML) entre outros que oferecem uma interface de muito baixo nível. O principal objectivo deste projecto é criar uma aplicação com interface web intuitiva para administração de políticas em XACML v3. A partir desta interface será possível ao utilizador escolher a informação disponibilizada pelos Policy Information Points (PIP) para preencher campos das políticas e regras, sem que as tenha de saber de cor, assim como também escolher algoritmos combinatórios e tipos de dados que estejam no padrão do XACML v3 sem

que tenha de os escrever. Para que seja possível obter informações a partir de vários PIP, o objectivo seguinte consiste, além da extensibilidade, na abstracção na recolha de informação.

3.1.2. Arquitectura extensível

Outro objectivo é tornar a aplicação extensível, para que quem a utilizar possa colocar módulos personalizados para retribuição de políticas, retribuição de informação de um ou mais PIP e para mapeamento de informações por categorias. Desta forma será indiferente para a aplicação se as políticas estão armazenadas em ficheiros, base de dados ou outro tipo de armazenamento. O mesmo acontece com a ligação ao PIP, permitindo que as informações sejam recebidas de um servidor LDAP, de uma base de dados ou mesmo de um ficheiro.

No capítulo 3.2 serão abordadas as soluções estudadas que permitirão o desenvolvimento da aplicação, seguindo os critérios anteriormente descritos.

3.1.3. Conformidade com as regras do XACMLv3

O resultado final de cada política tem de estar de acordo com as regras descritas na especificação da OASIS, de uma forma transparente para o administrador. O XML gerado a partir dos formulários de inserção e edição terão de estar de acordo com o esquema do XACMLv3.

3.2.Soluções

3.2.1. Interface web

Sendo o primeiro o objectivo tornar a administração de políticas algo usável e intuitivo, planeei usar um motor de políticas para gerar automaticamente regras dado um determinado triplo constituído por Sujeito, Acção e Recurso. O processo decorreria avaliando a partir da raiz de políticas até ao limite onde esse triplo seria aplicável, e inserir a esse nível a informação restante para a criação da nova regra. Como o XACMLv3 ainda está em estudo, não existem muitas implementações de motores de políticas, não tendo encontrado uma solução viável para aplicar esta ideia com sucesso.

Como alternativa, optei por avançar com uma ideia de mais baixo nível e representar as políticas em nós, ligados entre si conforme estiver definido na sua hierarquia. Desta forma é mais fácil visualizarmos como estão definidas as políticas, abstraindo a complexidade do XML. A Ilustração 19 exibe uma sugestão de apresentação desses nós para representar a informação das políticas.

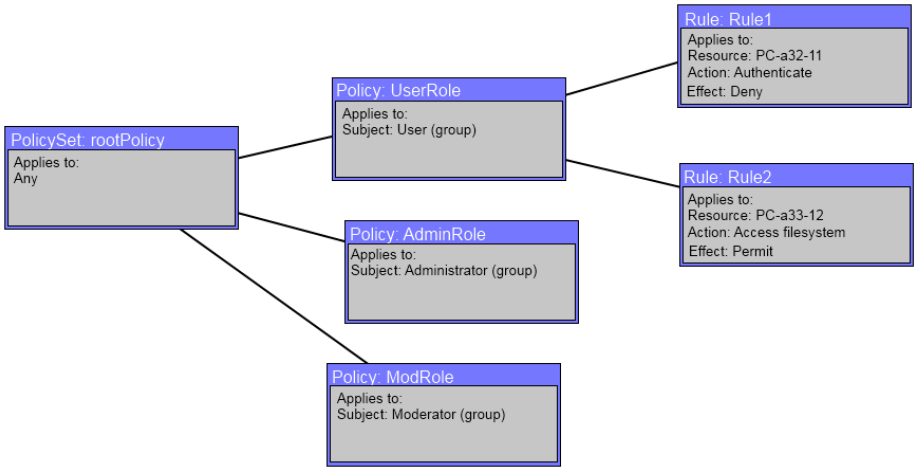

Ilustração 19 - Sugestão de apresentação dos nós

Desta forma, a criação de políticas e regras passa a estar ao cargo do administrador, mas com a facilidade de poder visualizar melhor como estas estão organizadas.

3.2.2. Extensibilidade

Para que exista abstracção na fonte de informação e na gestão de políticas, é necessário que exista uma forma de importar módulos externos. Esses módulos terão de seguir um padrão para que todos tenham um comportamento semelhante independentemente do que é e como é executado. Tem de existir forma de importar esses módulos sem que seja para isso preciso recompilar o código da aplicação. A solução encontrada foi fazer importação de classes contidas em ficheiro com extensão ".jar" (arquivos Java com bibliotecas ou aplicações) recorrendo a um objecto nativo do Java, o URLClassLoader[3]. Além disso, para que os módulos possam ser usados pelo sistema de informação de uma forma semelhante entre eles, recorrerei ao uso de interfaces, só aceitando instanciar objectos de classes que as implementem. Desta forma, independentemente da implementação, o comportamento esperado deverá ser o mesmo. Para identificar essas classes, planeio ter um ficheiro de configuração em XML com informação quanto às classes a importar.

[3] http://download.oracle.com/javase/1.4.2/docs/api/java/net/URLClassLoader.html

3.2.3. Conformidade

No motor de políticas implementado pela SUN para XACMLv2, todos os elementos presentes no esquema do XACMLv2 são traduzidos em classes, transformando assim todas as políticas em objectos. Para administração de políticas optei por não fazê-lo, uma vez que todas as alterações feitas nas políticas são guardadas em tempo real usando XQuery para persistir e fazer a leitura dos dados, apenas transformando os resultados em objectos Node[4]. A validação dos dados em relação ao esquema irá ser feita do lado do componente externo que retribui e persiste as políticas.

3.3.Arquitectura

Neste capítulo será apresentada a arquitectura do sistema de informação. O problema está dividido em várias partes, por sua vez a solução vai ser constituída por diferentes módulos.

Começando pela camada de apresentação, o capítulo seguinte aborda a arquitectura da interface do o utilizador.

3.3.1. Interface do administrador

A interface do administrador deve abranger a gestão dos elementos principais das políticas XACML, sem que este os tenha de decorar, oferecendo para isso as opções indicadas conforme o contexto onde se encontra.

Casos de uso - visão geral

O digrama apresentado na Ilustração 20 estão representados todos os casos de uso a implementar para a interface do administrador, tendo em conta a sugestão apresentada na Ilustração 19. Nos capítulos seguintes este diagrama será explicado por partes.

[4] http://download.oracle.com/javase/1.5.0/docs/api/org/w3c/dom/Node.html

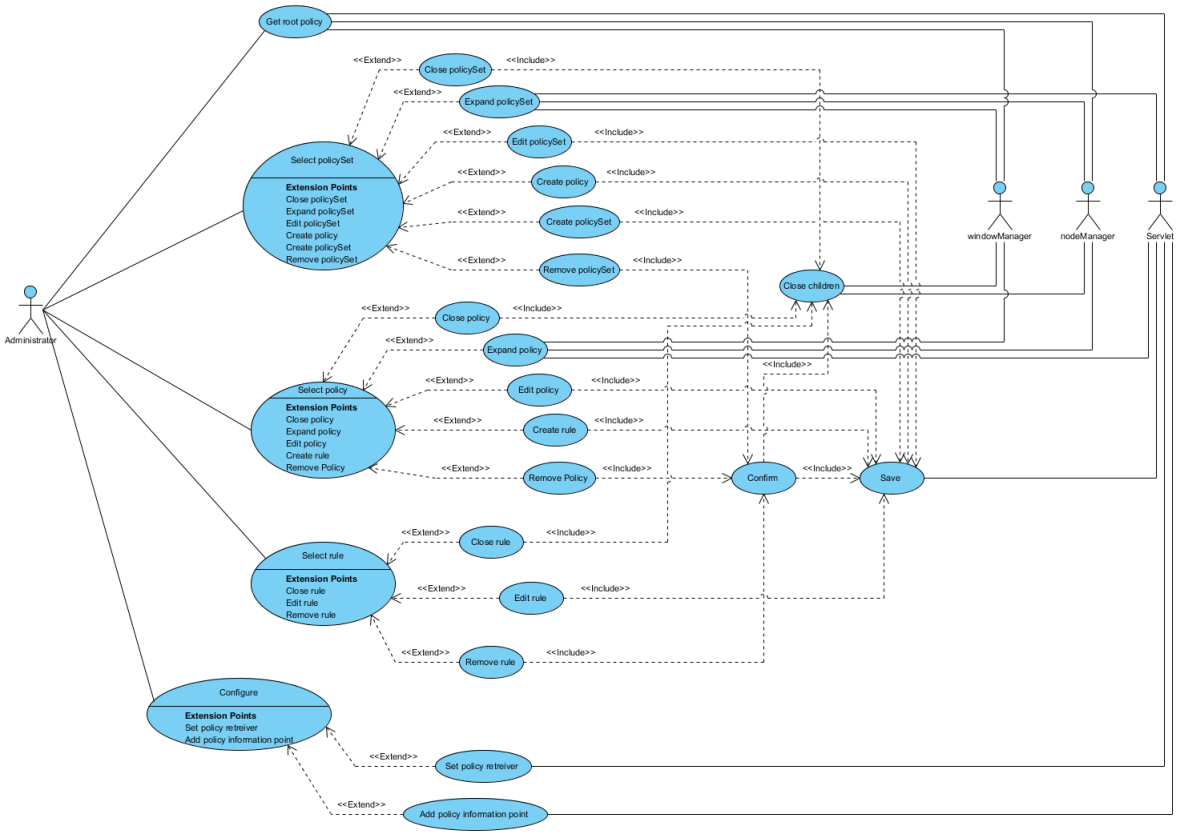

Ilustração 20 - Visão geral dos casos de uso da interface do utilizador

Actores

Administrador

Pessoa responsável pela gestão de políticas. Esta pessoa interage com a interface web a partir de um browser.

windowManager

Componente responsável pela criação e manutenção de janelas/nós (ver capítulo 3.4.1).

nodeManager

Componente responsável pela manutenção de ligações entre nós.

Servlet

Componente responsável pela persistência e obtenção de políticas e informações relevantes, tais como informação do *PIP*.

Caso de uso - get root policy

Este caso de uso serve para o carregamento da política raiz. É a partir deste elemento carregado que se expandirão as políticas e regras.

Caso de uso - Select PolicySet

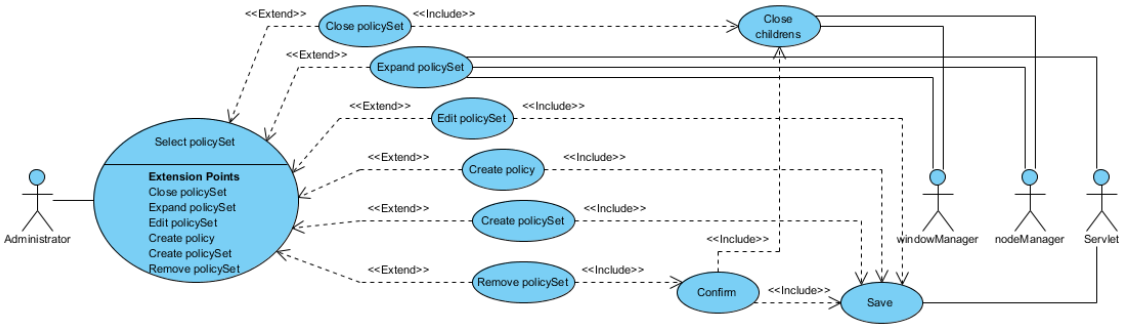

Ilustração 21 - Caso de uso - Select PolicySet

Quando o administrador selecciona um *PolicySet*, são apresentadas as opções exequíveis referentes a este elemento. Essas opções são:

Close policySet

Fecha o *Policy Set*. Ao fechar, o elemento não é apagado do xml, apenas deixa de estar visível no interface do administrador. Todos os elementos descendentes a este elemento são fechados também.

Expand policySet

Apresenta na interface do administrador os descendentes directos deste *PolicySet*.

Edit policySet

Abre uma janela de edição deste elemento. Para que as alterações sejam guardadas é necessário que a operação *Save* (guardar) seja executada.

Create policy

Abre uma janela de criação de um elemento *Policy*. O elemento *Policy* é um elemento válido a ser criado dentro de um *PolicySet*. Para que este elemento seja guardado é necessário que a operação *Save* (guardar) seja executada.

Create policySet

Abre uma janela de criação de um elemento *PolicySet*. O elemento *PolicySet* é um elemento válido a ser criado dentro de um *PolicySet*. Para que este elemento seja guardado é necessário que a operação *Save* (guardar) seja executada.

Remove policySet

Apresenta uma mensagem de confirmação para remoção do elemento *PolicySet* seleccionado. Após confirmação positiva, o elemento seleccionado é removido do xml. Caso esta operação seja executada com sucesso, o elemento é removido da interface do administrador, juntamente com todos os descendentes deste elemento.

Caso de uso - Select Policy

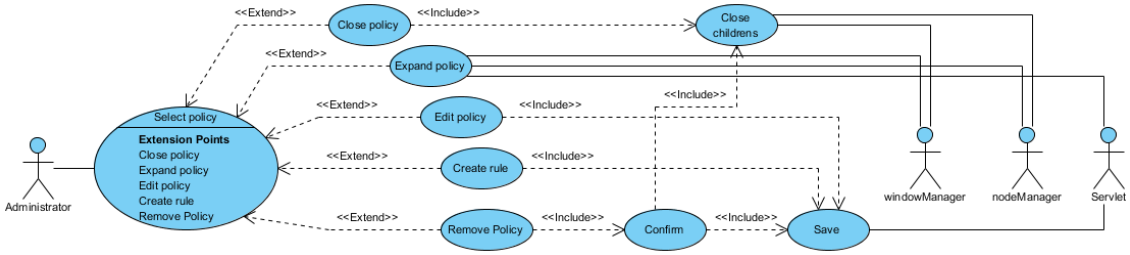

Ilustração 22 - Caso de uso - Select polcy

Quando o administrador selecciona um *PolicySet*, são apresentadas as opções exequíveis referentes a este elemento. Essas opções são:

Close policy

Fecha o *Policy*. Ao fechar, o elemento não é apagado do xml, apenas deixa de estar visível no interface do administrador. Todos os elementos descendentes a este elemento são fechados também.

Expand policy

Apresenta na interface do administrador os descendentes directos deste *Policy*.

Edit policy

Abre uma janela de edição deste elemento. Para que as alterações sejam guardadas é necessário que a operação *Save* (guardar) seja executada.

Create rule

Abre uma janela de criação de um elemento *Rule*. O elemento *Rule* é um elemento válido a ser criado dentro de um elemento *Policy*. Para que este elemento seja guardado é necessário que a operação *Save* (guardar) seja executada.

Remove policy

Apresenta uma mensagem de confirmação para remoção do elemento *PolicySet* seleccionado. Após confirmação positiva, o elemento seleccionado é removido do xml.

Caso esta operação seja executada com sucesso, o elemento é removido da interface do administrador, juntamente com todos os descendentes deste elemento.

Caso de uso - Select Rule

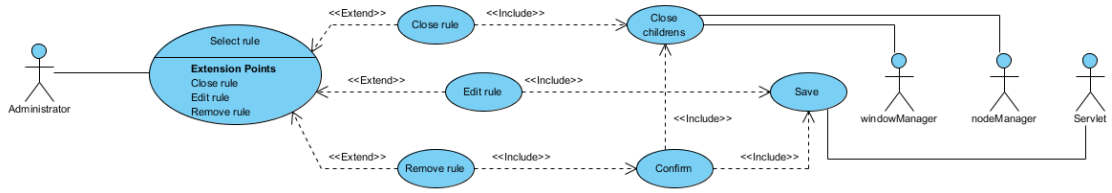

Ilustração 23 - Caso de uso - Select rule

Quando o administrador selecciona um elemento *Rule*, são apresentadas as opções exequíveis referentes a este elemento. Essas opções são:

Close rule

Fecha o elemento *Rule*. Ao fechar, o elemento não é apagado do xml, apenas deixa de estar visível no interface do administrador.

Edit rule

Abre uma janela de edição deste elemento. Para que as alterações sejam guardadas é necessário que a operação *Save* (guardar) seja executada.

Remove policy

Apresenta uma mensagem de confirmação para remoção do elemento *PolicySet* seleccionado. Após confirmação positiva, o elemento seleccionado é removido do xml. Caso esta operação seja executada com sucesso, o elemento é removido da interface do administrador, juntamente com todos os descendentes deste elemento.

Caso de uso - Configure

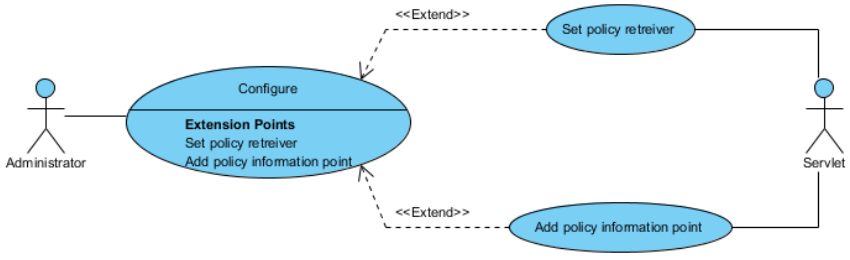

Ilustração 24 - Caso de uso - Configure

Ao executar a opção *Configure* o utilizador poderá configurar as fontes de informação e de políticas. O Servlet é neste caso responsável por obter as opções disponíveis para que o administrador as possa seleccionar.

3.3.2. Componente Java Class Loader

Para que este projecto cumpra os objectivos propostos, é necessário que o sistema de informação seja extensível. Para isso, o sistema tem de permitir importação de módulos externos, sem que seja preciso recompilar o código fonte. A solução encontrada foi usar o URLClassLoader do java para importar classes, dado um URL com a localização de ficheiros com extensão *jar* (Java ARchive). Para melhor organização dos módulos, é necessário que seja possível organizar os módulos por pastas. O método usado para abrir todos os ficheiros com extensão *jar* de uma pasta e de todas as subpastas nele contidas, foi percorrer recursivamente todos os directórios locais a partir de uma dada localização. O capítulo seguinte ilustra a arquitectura do Java Class Loader a implementar.

Casos de uso

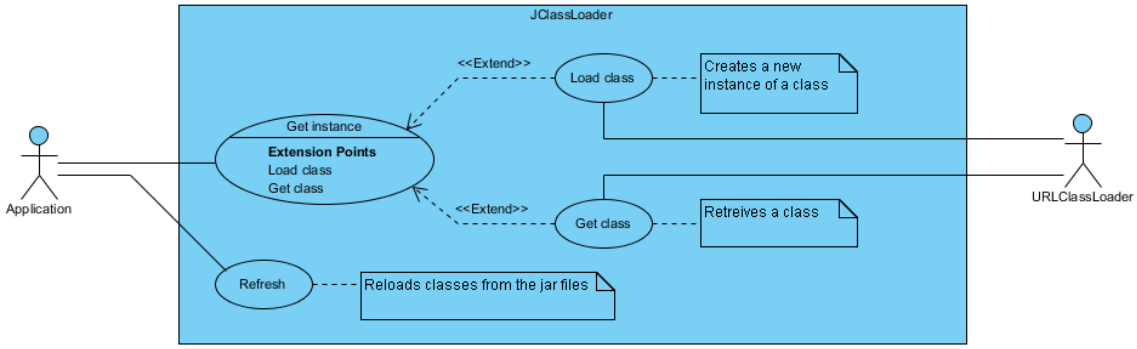

Ilustração 25 - Casos de uso do Java Class Loader

Caso de uso - Get instance

A ideia deste módulo é ter uma instância única para cada contexto a ser utilizado. Dentro desse contexto, deverão existir diferentes localizações onde procurar por módulos a carregar.

Tendo essa instância, deveremos ter a possibilidade de obter uma classe carregada (Get class), e também de obter uma instância dessa classe (Load class). A primeira poderá servir o propósito de verificar se os módulos carregados implementam determinada interface, sem que para isso seja necessário instanciar um objecto.

Caso de uso - Refresh

Para o caso de serem acrescentados módulos em tempo de execução, o método Refresh quando chamado deverá ser responsável por recarregar o URLClassLoader com todos os ficheiros encontrados.

Diagrama de classes

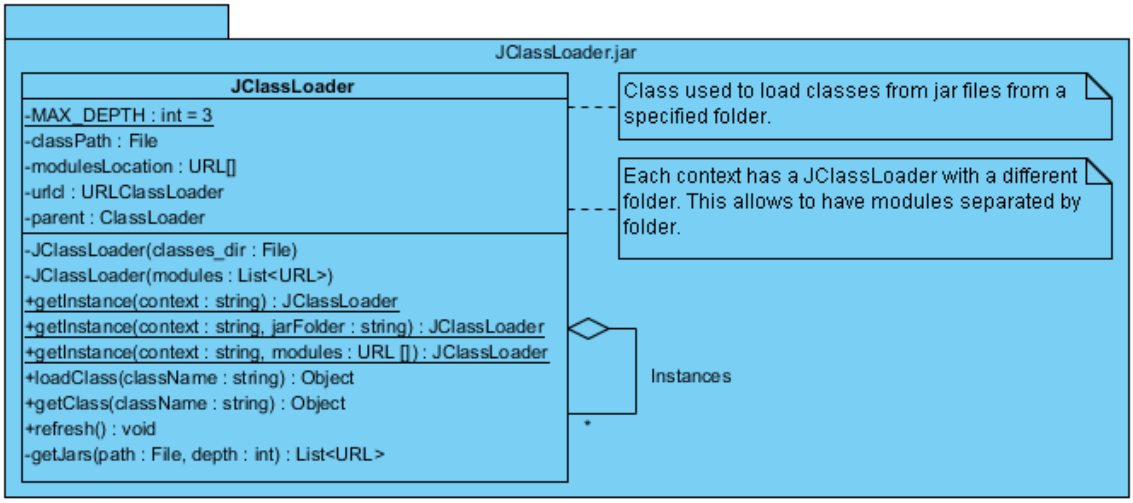

Ilustração 26 - Diagrama de classes do componente JClassLoader

Tal como referido nos casos de uso, este componente terá uma única instância por contexto. Essas instâncias serão armazenadas num objecto Map<String, JClassLoader>, permitindo assim a obtenção de uma instância pelo contexto definido (neste caso o contexto é identificado com um String).

As razões para a existência do atributo *parent* e da existência de um *array* de *URL*s estão explicadas no capítulo 3.4.2 em Problemas encontrados na implementação.

Para documentação em JavaDoc deste componente, ver Anexo B deste documento.

Interfaces de módulos externos

Para garantir que os módulos externos cumpram os requisitos, é necessário que contenham classes que implementem as interfaces impostas nesta arquitectura (ver capítulo 3.2.2). Os módulos serão usados para retribuição de informação dos *Policy Information Point* e para persistência e obtenção de políticas XACMLv3. Os subcapítulos seguintes ilustram as interfaces que serão utilizadas.

Interface IInfoRetreiver

Para obtenção de informação, é necessário que os módulos implementem obrigatoriamente os métodos apresentados no diagrama da Ilustração 27.

Estes métodos permitem obter listas de recursos existentes, obter recursos de uma determinada categoria, obter informações adicionais sobre cada recurso e mapeamento para futura utilização.

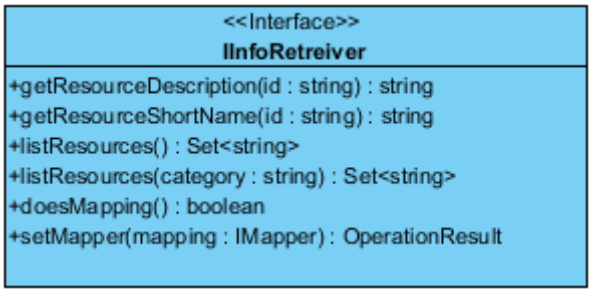

Ilustração 27 - Interface IInfoRetreiver

Para a implementação de um módulo que implemente esta interface, os resultados esperados estão especificados nos subcapítulos seguintes.

getResourceDescription

Dado o identificador de um recurso, deverá o método devolver uma descrição sobre esse recurso.

getResourceShortName

Dado o identificador de um recurso, o método deverá devolver um nome de fácil leitura para ser usado na interface de administração. Por exemplo, ser apresentado o nome "ELearning server" em vez do identificador "pt.ua.servers.ElearningServer".

listResources

Este método deverá devolver uma lista com todos os recursos existentes neste *Policy Information Point*. Caso o argumento *category* seja utilizado, então a lista a ser devolvida deverá ser de todos os recursos existentes e que pertençam a essa categoria.

doesMapping

O módulo não é obrigado a fazer o trabalho de mapeamento de valores por categorias a serem reconhecidas pelo *Policy Administration Point* (ver Interface IMapper). Caso não suporte esse mapeamento, este método deverá devolver o valor *false*.

setMapper

Neste método é injectado o objecto que contém as funções para mapeamento. É neste objecto que os valores são mapeados por categorias reconhecidas pelo *PAP*. Caso a operação não tenha sucesso, deverá devolver um objecto OperationResult com valor OPERATION_FAILURE, juntamente com uma mensagem de erro. caso contrário devolve um objecto OperationResult com o valor OPERATION_SUCCESS (ver capítulo 0).

Interface IPolicyRetreiver

O módulo de gestão e persistência de políticas tem de cumprir a interface da Ilustração 28.

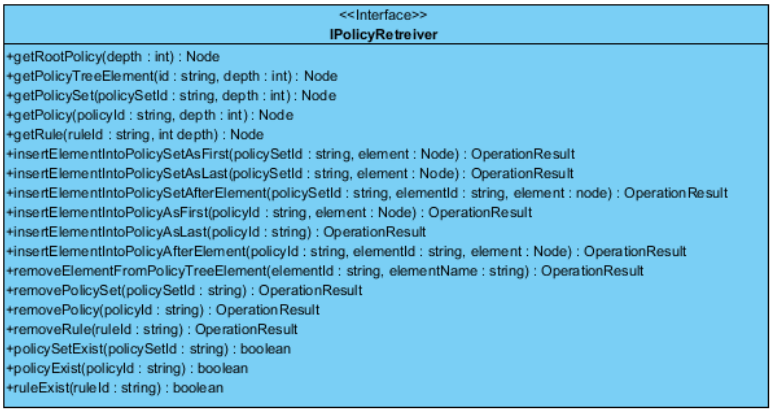

Ilustração 28 - Interface IPolicyRetreiver

Este módulo tem de ser capaz de persistir as políticas ao inserir, apagar e editar elementos. Este é responsável por validar o xml gerado, segundo o esquema especificado pela OASIS.

O comportamento esperado pelos métodos impostos pela interface são especificados nos subcapítulos seguintes.

getRootPolicy

Este método deve devolver a raiz das políticas xml, ou seja, o elemento inicial do documento xml.

getPolicyTreeElement, getPolicySet, getPolicy, getRule

Dado o identificador do elemento principal do documento XACML (sendo os elementos principais o *PolicySet*, o *Policy* e o *Rule*), e a profundidade de descendentes, este método tem de ser capaz de construir um elemento Node com o resultado, limitando os elementos principais descendentes à profundidade imposta. Todos os outros elementos deverão estar presentes, seja qual for a sua profundidade.

insertElementIntoPolicySetAsFirst, InsertElementIntoPolicyAsFirst

Insere um elemento na primeira posição, dentro do elemento *PolicySet* ou *Policy*. O estado da operação é então devolvido com o objecto OperationResult (ver capítulo 0).

insertElementIntoPolicySetAsLast, InsertElementIntoPolicyAsLast

Insere um elemento na última posição, dentro do elemento *PolicySet* ou *Policy*. O estado da operação é então devolvido com o objecto OperationResult.

insertElementIntoPolicySetAfterElement, InsertElementIntoPolicyAfterElement

Insere um elemento, após outro elemento, dentro do elemento *PolicySet* ou *Policy*. O estado da operação é então devolvido com o objecto OperationResult.

removeElementFromPolicyTreeElement

Remove o elemento que se encontrar dentro do elemento principal (*PolicySet, Policy* ou *Rule*), caso exista. O estado da operação é então devolvido com o objecto OperationResult.

removePolicySet, RemovePolicy, RemoveRule

Remove o elemento principal que tenha o identificador igual ao do argumento. . O estado da operação é então devolvido com o objecto OperationResult.

policySetExist, policyExist, ruleExist

Verifica se o elemento principal com o dado identificador existe.

Interface IMapper

O propósito da interface IMapper é de construir um módulo de categorização de recursos. Este será responsável por categorizar os recursos, seja por meio de interface de administrador, em que é o administrador a categorizar manualmente um determinado recurso, seja automaticamente pelas implementações do IInfoRetreiver que suportem mapeamento.

3.3.3. Componente Class Factory

Tendo as interfaces criadas e módulos implementados, é necessário carregar as classes desses módulos, confirmar se estas implementam as interfaces correctas e instancia-las. Para o fazer, optei por implementar uma *Factory*, cuja função é não só de criar

objectos das classes pedidas, mas também de verificar antes quais as classes que podem ser instanciadas (sendo estas consideradas válidas).

Diagrama de classes

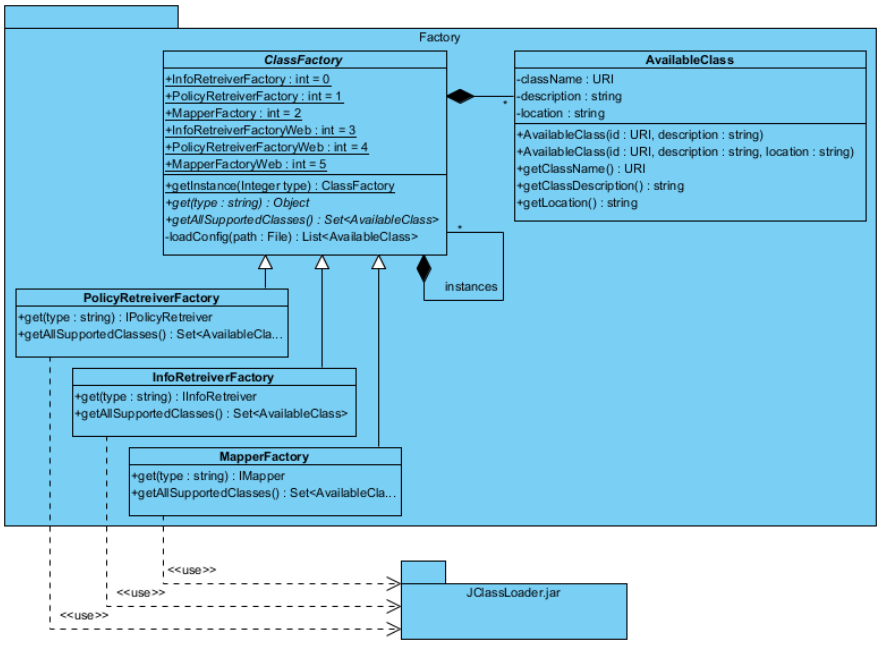

Ilustração 29 - Diagrama de classes do ClassFactory

Como é possível constatar na Ilustração 29, existe uma classe *ClassFactory* abstracta cuja função é devolver instâncias únicas de cada implementação dessa classe. As implementações criadas são extensões *ClassFactory* para cada tipo de módulo a ser procurado (por exemplo, no caso do *PolicyRetreiverFactory* apenas procura por implementações da interface *IPolicyRetreiver*). Cada uma das instâncias dessas classes usam o JClassLoader documentado no capítulo 3.3.2 para carregamento desses módulos.

A razão pela qual é necessário ter instâncias únicas de cada implementação do *ClassFactory* é pela limitação que pode ser imposta pelos módulos de só poderem ser carregados uma vez. Este caso acontece quando um módulo usa uma biblioteca externa e esta permanece bloqueada enquanto este módulo estiver activo.

Ficheiro de configuração de módulos externos

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        targetNamespace="pt.fg.classfactory.modules"
        xmlns:cf="pt.fg.classfactory.modules"
        elementFormDefault="unqualified">

        <xsd:element name="Modules" type="cf:ModulesType" />

        <xsd:complexType name="ModulesType">
                <xsd:choice maxOccurs="unbounded" minOccurs="0">
                        <xsd:element name="InfoRetreiver" type="cf:ModuleType" />
                        <xsd:element name="PolicyRetreiver" type="cf:ModuleType" />
                        <xsd:element name="Mapping" type="cf:ModuleType" />
                </xsd:choice>
        </xsd:complexType>

        <xsd:complexType name="ModuleType">
                <xsd:simpleContent>
                        <xsd:extension base="xsd:string">
                                <xsd:attribute name="name" use="required" />
                        </xsd:extension>
                </xsd:simpleContent>
        </xsd:complexType>
</xsd:schema>
```

Ilustração 30 - Schema do documento xml de configuração de módulos externos

Para obter classes carregadas de ficheiros jar, é necessário saber o caminho relativo dessas classes (por exemplo: pt.fg.dbxml.PolicyRetreiverClass). Para isso, existe um documento xml que usa como *schema* o apresentado na Ilustração 30. Neste documento constam as seguintes informações:

- Tipo de módulo (IPolicyRetreiver, IInfoRetreiver ou IMapper);
- Caminho relativo ao ficheiro *jar* e nome da classe;
- Descrição da classe;
- URL do ficheiro *jar* a carregar (opcional).

A Ilustração 31 mostra um exemplo de um ficheiro de configuração possível.

```
<ns0:Modules  xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
        xmlns:ns0='pt.fg.classfactory.modules'
        xsi:schemaLocation='pt.fg.classfactory.modules schema.xsd'>

        <PolicyRetreiver name="pt.fg.xacml.bdbxml.policyretreiver.PolicyRetreiver">
                Retreives policies from a native database (Oracle Barkeley DBXML).
        </PolicyRetreiver>

        <InfoRetreiver name="pt.fg.xpto.FileInfoRetreiver" location='http://xpto.pt/ir.jar'>
                Retreives information from a file.
        </InfoRetreiver>
</ns0:Modules>
```

Ilustração 31 - Documento xml com a indicação dos módulos externos

3.3.4. Componente Operation Result

Como visto anteriormente nos diagramas de classes, muitos dos métodos devolvem um elemento *OperationResult*. Este elemento foi desenhado para dar uma resposta mais completa do que simplesmente um valor *boolean*. Desta forma, pode ser feita uma resposta positiva ou negativa juntamente com uma mensagem que pode ser implementada para incluir outros objectos.

Diagrama de classes

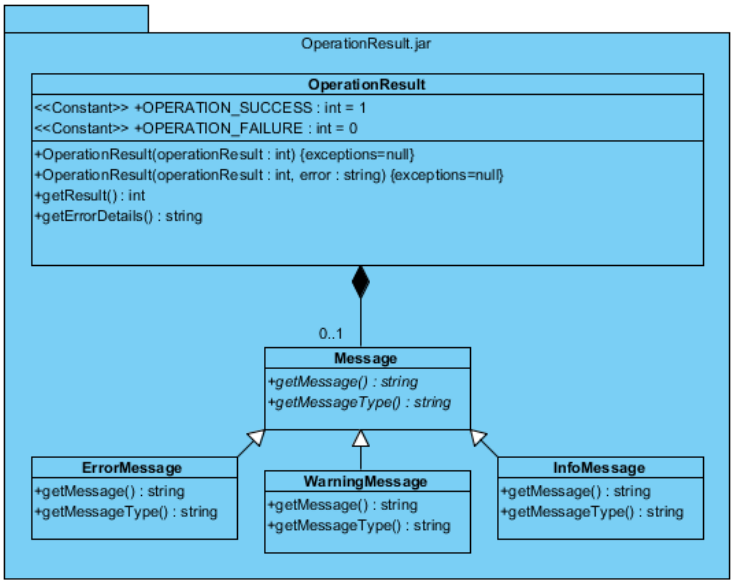

Ilustração 32 - Diagrama de classes do componente OperationResult

Tal como apresentado na Ilustração 32, este componente pode ter o valor *OPERATION_SUCCESS*, para quando uma operação é realizada com sucesso, e *OPERATION_FAILURE* para quando algum erro ocorreu. Em ambos os casos é possível ter uma mensagem associada. Isto poderá ser útil para quando uma acção é executada, mas com alguns problemas, podendo criar uma *WarningMessage* informando dos problemas ocorridos, sendo o estado da operação *OPERATION_SUCCESS*.

3.3.5. Diagrama de componentes

Ao juntar as peças todas, compõe-se a arquitectura da camada lógica e camada de apresentação do sistema de informação. Falta agora ligar as duas camadas para que ambas possam comunicar e trabalhar em conjunto.

Uma vez que tenciono usar AJAX para interface de administração, preciso de arranjar forma de enviar os dados para o browser em *xml*. Para isso recorrerei a *Servlets* para gerar *xml* de acordo com o pedido do browser. Para que o *Servlet* possa comunicar com a camada lógica, usarei *Session Beans* com interface remota.

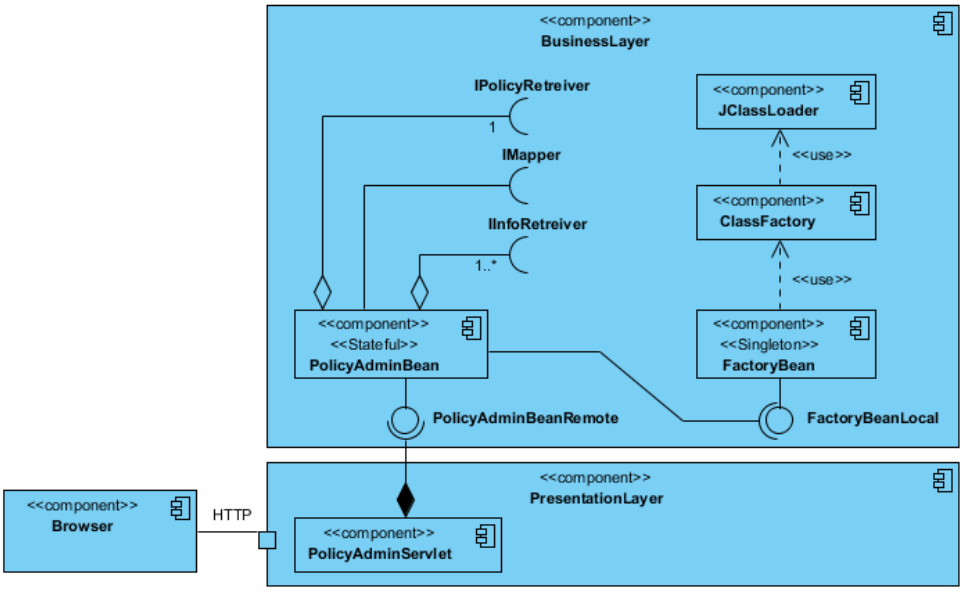

Ilustração 33 - Diagrama de componentes

Conforme apresentado no diagrama de componentes da Ilustração 33, o Servlet contém um *PolicyAdminBean*. Este *bean* é stateful, ou seja, mantém o seu estado durante a sessão. Para mantê-lo entre vários pedidos do browser, este é mantido no objecto *HTTPSession*.

Quanto ao PolicyAdminBean, este será o ponto central do sistema de informação. Este contém os módulos carregados referentes à sessão existente, sendo uma ponte para obter informações dos recursos do *InformationRetreiver*, obter e persistir políticas do *PolicyRetreiver*. Para obter as instâncias desses módulos externos, este usa um *bean singleton* local (instância única), o *FactoryBean*. Embora o componente *ClassFactory* tenha instâncias únicas das *Factories* para cada tipo de módulo a carregar, quando a aplicação web está num servidor de aplicações, esta pode estar a ser executada em vários containers, e nada garante que a instância usada num momento seja a mesma que esteja a ser utilizada na operação seguinte. Isto é importante pelas razões mencionadas no capítulo 3.3.3.

3.4.Implementação

Esta aplicação está implementada na tecnologia J2EE. Desta forma, a camada de apresentação está separada da camada lógica. Os subcapítulos seguintes descrevem a implementação segundo a arquitectura e problemas encontrados.

3.4.1. Camada de apresentação

Para a representação dos nós, foi criado no documento XHTML um elemento *div* com o identificador "windowContainer", cujos elementos filho serão outros elementos *div*, com a propriedade CSS (Cascade Style Sheet) "***position**:absolute;*" (posicionamento absoluto, em relação ao elemento parente), que representam os nós (a que intitulo de "windows" (janelas) por terem um aspecto semelhante às janelas existentes no interface de utilizador de vários sistemas operativos, e por terem algumas das suas características, como a possibilidade de mover). Para a criação e gestão desses nós foram criados componentes em JSON. Para ligação entre nós, foi criado um componente que desenha linhas entre dois pontos (neste caso entre os nós, usando a localização destes no elemento *windowContainer*). Para isso, este módulo guarda todas as ligações entre dois nós e desenha-os num elemento *canvas* do HTML5. Este elemento permite desenhar usando javascript.

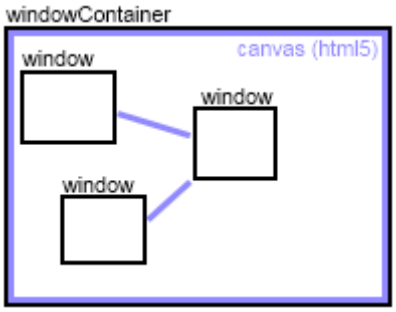

Ilustração 34 - windowContainer

Sendo a interface do administrador criada ao longo das interacções, é necessário que a página web não seja recarregada. Para que seja possível apresentar nova informação sem que a página seja recarregada, utilizo AJAX para estabelecer comunicação com o *servlet* e receber as respostas deste.

A documentação e exemplos de código javascript dos componentes criados encontra-se no Anexo A deste documento.

Organização da informação

Por motivos de usabilidade e melhor compreensão, apenas os elementos *PolicySet*, *Policy* e *Rule* são considerados nós. Toda a outra informação associada é representada dentro do nó referente.

Para que o nó de raiz não seja fechado, este é o único que não contém um botão de fechar na janela correspondente, como é possível ver na Ilustração 35.

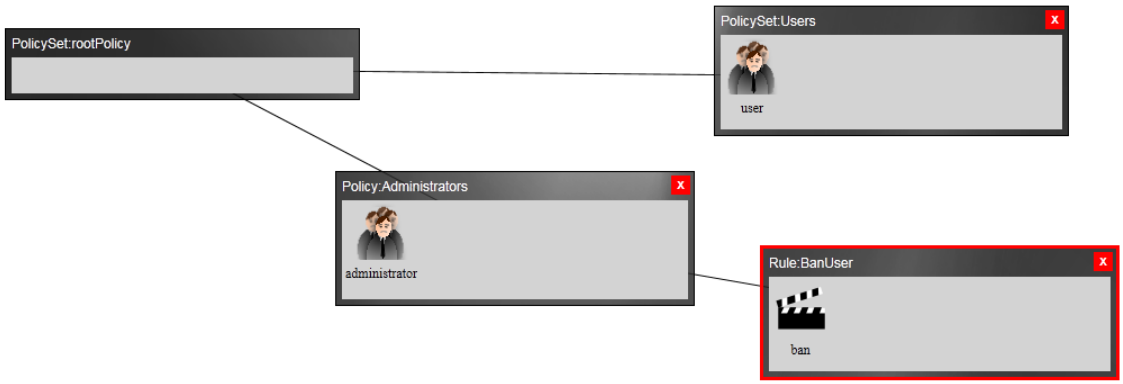

Ilustração 35 - Implementação dos nós

Representação do nó

Sendo o identificador de cada nó único, este é usado para dar nome à janela que representa o nó, juntamente com o tipo de nó (por exemplo, se um elemento *Rule* tiver o identificador *AccessFiles*, o nome da janela será *Rule:AccessFiles*). Identificando um nó pelo seu nome, torna-se trivial evitar que existam repetições de nós na interface do administrador, carregando o nó existente em vez de o recriar. É possível ver na Ilustração 35 o resultado.

Representação do *Target* de cada nó

De forma a conseguir saber de forma directa o *Target* de cada elemento, este tem de ser apresentado sempre em cada elemento, sem que seja necessário fazer algo para exibi-lo. Desta forma é possível saber, não só o *Target* de um elemento, mas também o dos elementos ascendentes e descendentes. Outra forma de o fazer seria apresentar na forma de herança o *Target* de um elemento junto com o dos seus ascendentes, mas isso levaria a um grande volume de informação num só nó o que, por questões de espaço, não se tornaria muito usável.

Ainda assim, a representação do *Target* de cada elemento revelou ser uma tarefa não tão trivial, devido à forma como a norma do XACMLv3 o organiza. A solução estudada foi segmentar as uniões dos seus sub-elementos e representar essas divisões por grupo, ou seja, considerar os elementos AllOf como grupos possíveis para aplicação de um *Target* (ver Avaliação de um elemento Target, na página 21). Cada segmento será

então representado por linhas, separados por um "OR" (em português seria "OU"), tal como apresentado na Ilustração 36.

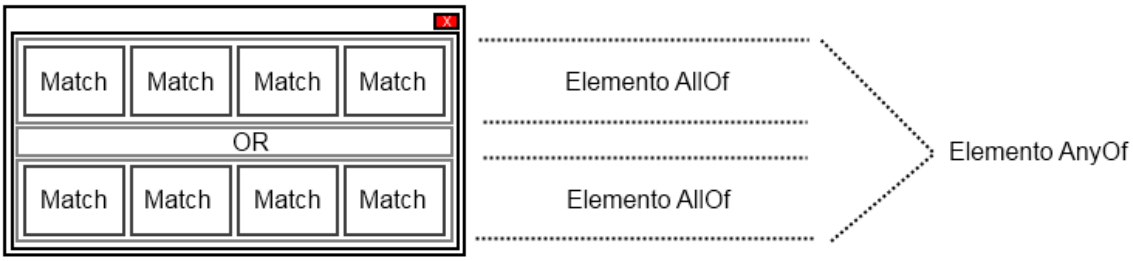

Ilustração 36 - Representação do elemento Target

Dentro de cada grupo, os vários elementos *Match* serão representados lado a lado.

Para uma melhor leitura, cada elemento *Match* poderá ter uma imagem associada à categoria onde se insere, juntamente com o valor esperado para essa categoria. Para o caso da imagem não ser ilustrativa, ao passar o ponteiro do rato por cima desse elemento, uma *tooltip* com uma descrição textual será apresentada. A Ilustração 37 apresenta o resultado final.

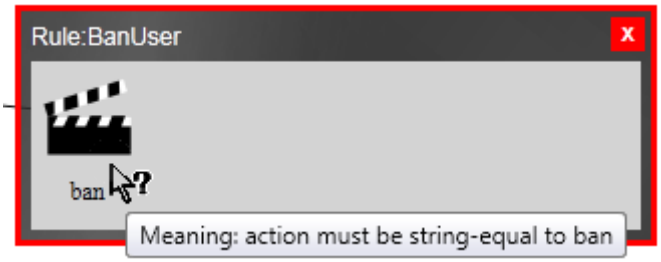

Ilustração 37 - Representação do elemento Match

Representação do efeito de uma regra

Um elemento *Rule* deverá ter um de dois efeitos, sendo eles *Permit* ou *Deny*. Uma forma simples de o representar será colocar a seguir à representação do elemento Target o efeito correspondente a essa *Rule*, com uma cor de fundo verde (no caso do valor ser *Permit*) ou vermelho (no caso do valor ser *Deny*).

Interacção com os elementos principais

Para a interacção com os elementos principais, é usada uma barra de ferramentas que muda de acções conforme um elemento é seleccionado. Dependendo do tipo elemento, as acções mudam conforme as acções possíveis, definidas nos casos de uso (ver capítulo 3.3.1). Na Ilustração 38 está um exemplo de acções de um elemento *PolicySet*.

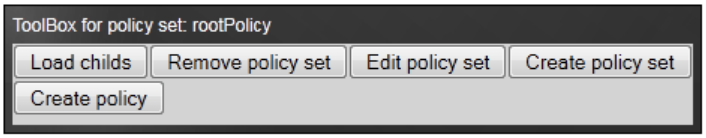

Ilustração 38 - Barra de ferramentas

Criação de um elemento

Para a criação e edição de um elemento, é aberta uma janela com um fundo que bloqueia qualquer tentativa de tentar alterar outros elementos ou de apagar o próprio elemento enquanto este é editado. Na criação de um *PolicySet* ou de um *Policy*, é obtida a informação dos algoritmos combinatórios existentes a partir do *Servlet*, que por sua vez obtém a informação a partir do módulo externo *Info Retreiver*. Na criação de elementos, existe o cuidado de verificar se o identificador dado já existe, e caso isso aconteça é apresentado um aviso na tentativa de guardar, e a janela permanece aberta com a informação para alterar.

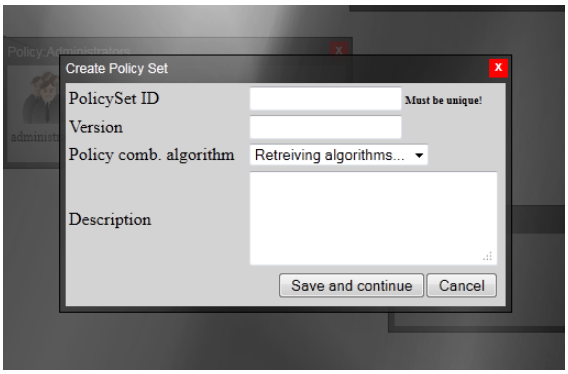

Ilustração 39 - Janela de criação de um *PolicySet*

3.4.2. Camada lógica

A implementação da camada lógica foi baseada na arquitectura estudada, exceptuando nos elementos cuja arquitectura sofreu alterações apresentadas no subcapítulo seguinte. As interfaces dos módulos externos foram sendo modelados conforme as necessidades apresentadas para um correcto funcionamento do sistema.

Problemas encontrados na implementação

Após a confirmação do funcionamento do componente JClassLoader (ver capítulo 3.3.2) numa aplicação *Java* através de vários testes, incluindo a instanciação e verificação de implementação de interfaces dos objectos instanciados (ver Interfaces de módulos externos), surgiram problemas quando utilizado numa aplicação J2EE.

Quando era pedido ao *ClassFactory* uma instância ou uma classe de um determinado módulo implementado, este lançava uma excepção *ClassNotFoundException*.

Numa primeira hipótese considerou-se o facto da aplicação não estar a recolher os módulos do sistema de ficheiros por razões de segurança. Para tentar verificar esse problema, foi acrescentado ao ficheiro de configuração de módulos um atributo com o URL absoluto do módulo a ser carregado, permitindo que esse módulo fosse carregado de uma página web por http (hypertext transfer protocol) ou através de *ftp* (file

transfer protocol) em vez de a partir do sistema de ficheiros local. Não existiram conclusões quanto a esta solução, uma vez que o problema persistiu.

Outra hipótese considerada seria o facto do ClassLoader não estar a instanciar uma classe que fosse considerada válida. Desta forma optei por acrescentar uma nova lista de valores a cada implementação da *ClassFactory*, usada para complementar informação quanto às classes carregadas válidas, guardando as classes consideradas inválidas. Desta forma, ao tentar obter uma classe ou instanciação dessa classe a partir da implementação da *ClassFactory*, caso essa classe conste nessa lista, uma excepção *InstantiationException* será lançada em vez de *ClassNotFoundException*. Após esta alteração, foi possível verificar que não só a classe era carregada com sucesso, tanto a partir sistema de ficheiros local como por http, mas também foi possível confirmar que o problema residia no facto do *J2EE* não considerar que a classe implementa a interface pretendida (sabendo que esta o fazia, tendo sido testada numa aplicação normal com sucesso).

Após alguma pesquisa sobre o funcionamento do *ClassLoader* do J2EE, foi possível perceber a razão do problema e chegar a uma solução.

Segundo o capítulo 21 do livro *J2EE Projecto - Survival Guide*, o *ClassLoader* do Java funciona por hierarquias[6]. Desta forma ClassLoader que estiver mais a baixo da hierarquia consegue ver as classes carregadas dos *ClassLoaders* ascendentes, mas não os dos descendentes.

Outro aspecto a ter em conta é o facto de que se for carregada uma classe no *ClassLoader* A, e a mesma for carregada no *ClassLoader* B, em que A é descendente de B, estas classes são consideradas implementações diferentes. Caso a ordem de carregamento fosse diferente (primeiro carregar no B e depois no A), ao tentar carregar a classe no A, o ClassLoader encontraria uma classe já carregada num dos seus parentes. A Ilustração 40[6] exemplifica de uma forma clara esse funcionamento. Aqui se encontra a raiz do problema e a solução encontrada foi acrescentar mais um argumento ao JClassLoader, o parentClassLoader (ClassLoader parente). Desta forma, é possível carregar as classes externas no ClassLoader descendente ao dos objectos do EJB e ao fazê-lo, as interfaces não são carregadas em ClassLoaders diferentes, porque no parente a interface já está carregada.

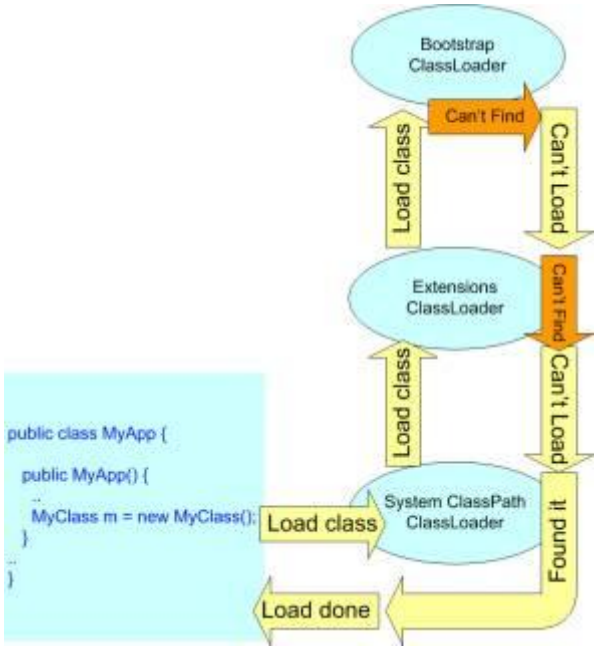

Ilustração 40 - Funcionamento do ClassLoader

3.5.Implementação de módulo externo

Por questões de exemplificação, foram implementados dois módulos que usam diferentes fontes de informação. Um deles é um *Policy Retreiver* e o outro é um *Info Retreiver*, cuja arquitectura e implementação estão descritas nos subcapítulos seguintes.

Para a documentação das interfaces, ver documento do anexo D.

3.5.1. Policy Retreiver

Para a elaboração deste módulo, uso como fonte de persistência de informação uma base de dados nativa de *xml*, no sistema de gestão de base de dados Berkeley's DB XML da ORACLE. Este sistema não tem o padrão para ligação a base de dados ODBC (Open Data Base Connectivity), tornando-o num sistema de baixo nível em relação a outras bases de dados, como por exemplo, o MySQL[7].

Este sistema funciona por *containers* (contentores), onde são armazenados os documentos *xml*. Existem duas formas de armazena-los, de forma integral ou por nós separados. A primeira tem a vantagem de possibilitar obter o documento integral, tal como ele era quando foi originalmente armazenado. A segunda possibilita ganhos de eficiência em relação a espaço e rapidez, já que nem toda a informação é armazenada, tais como espaços em branco[8]. Por questões de eficiência, foi usada a segunda opção.

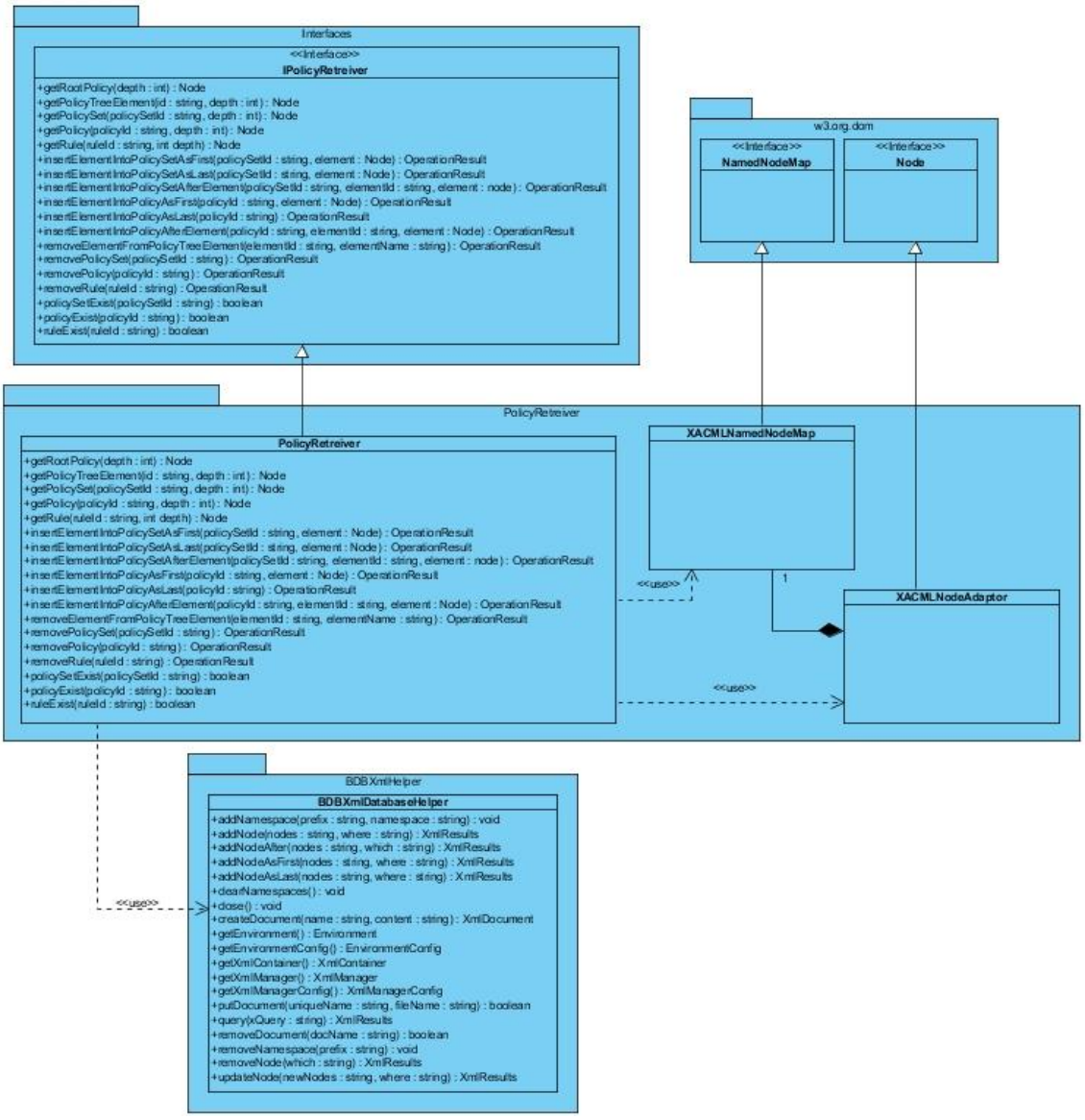

Ilustração 41 - Diagrama de classes do Policy Retreiver

Para facilitar o manuseamento da base de dados, foi criada uma camada de abstracção, como é possível ver na Ilustração 41, a classe BDBXmlDatabaseHelper.

Um problema do Berkeley's DB Xml é o facto de não obedecer a padrões, tais como implementações do Node do w3c. Para colmatar essa falha, foram feitos adaptadores que são implementações de padrões, como é possível ver na Ilustração 41, sendo o XACMLNodeAdaptor uma implementação da interface org.w3c.dom.Node e o XACMLNamedNodeMapAdaptor uma implementação do org.w3c.dom.NamedNodeMap.

Outra falha descoberta foi o facto de a implementação do Berkeley's DB em Java não suportar arquitecturas 64bit. Desta forma, para aplicação funcionar é necessário que a JVM (máquina virtual do java) seja 32bits.

Para documentação do BDBXmlDatabaseHelper, ver documento do anexo C.

3.5.2. Info Retreiver

A implementação do *Info Retreiver* contém duas implementações da interface IInfoRetreiver, uma para obter informações sobre algoritmos combinatórios disponíveis e outra para obter informações sobre tipos de dados e funções disponíveis. Estas classes poderiam estar em projectos diferentes, mas serve de exemplo que pode ser feita mais do que uma implementação em cada projecto/ficheiro *jar*.

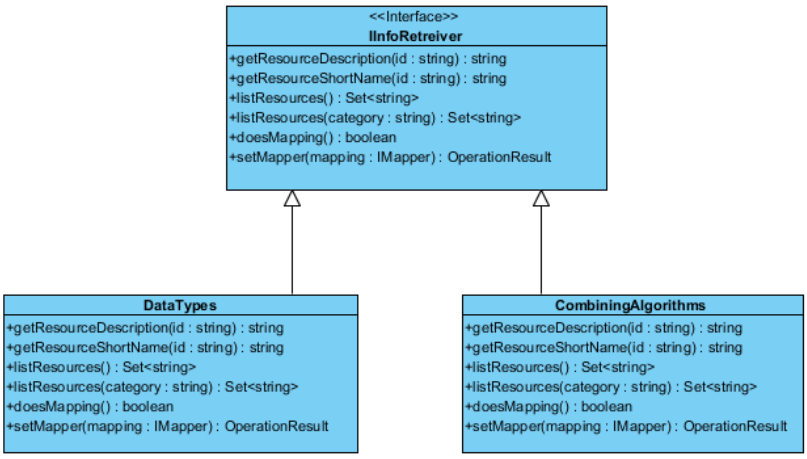

Ilustração 42 - Implementação da interface IInfoRetreiver

Ambas as implementações lêem a informação de ficheiros *xml*.

4. Conclusões

Com base no estudo da norma XACMLv2, deu para perceber como é possível fazer gestão de autorizações de forma genérica e extensível, embora complexa recorrendo a avaliação de *Targets* de políticas compostas por regras e efeitos associados. Havendo flexibilidade de criação de algoritmos combinatórios, funções e tipos de dados torna esta tecnologia aplicável a qualquer contexto. A versão XACMLv3 corrigiu algumas das limitações na arquitectura da versão anterior e acrescentou novas funcionalidades, tendo sido uma evolução no bom sentido, embora ainda esteja em fase de construção. Ainda há falhas a corrigir, e problemas que a nova arquitectura trouxe, tais como a organização dos *Target* em os elementos de intersecção e união não têm identificadores que os distinga ou que os identifique como grupos. A principal dificuldade com que me deparei nesta tecnologia foi a falta de implementações open-source existentes, talvez por ser uma tecnologia ainda em desenvolvimento, o que me limitou nas possibilidades de desenvolvimento que previa inicialmente.

Durante a elaboração do software foi-me possível aprofundar os conhecimentos na arquitectura J2EE em relação à utilização de *JavaBeans* e *Servelets* e como estes comunicam entre eles. Além disso, fiquei a conhecer o modo de funcionamento dos *ClassLoaders* do *Java*, ficando a saber que estes funcionam de uma forma hierárquica e que faz diferença em qual *ClassLoader* as classes são carregadas. Quanto à implementação dos módulos, existiu uma particular dificuldade de adaptação ao sistema de gestão de base de dados Berkeley DB Xml, por ser uma base de dados de baixo nível e por não trabalhar com a linguagem SQL, tendo sido por outro lado positivo ao ter aprendido a usar XQuery.

Quanto à implementação da camada de apresentação, foi interessante trabalhar com AJAX e criar uma interface de administrador fora do habitual, usando HTML5 para desenhar linhas entre elementos.

Para futuro poderá ser considerada a implementação de mais módulos para extender a compatibilidade deste software.

O mesmo se aplica em relação às fontes de informação. Neste momento apenas existe uma implementação que lê ficheiro *xml* a partir do sistema de ficheiros, mas outras

implementações com leitura de valores de um servidor LDAP para colocar utilizadores e recursos em *Targets* ou comunicação com o PDP para obter funções e tipos de dados suportados.

Algo a ter em conta é a implementação de uma base de dados relacional para criação de utilizadores e projectos associados, em que esses projectos terão configurações sobre quais módulos utilizar.

Em relação à gestão de políticas, é necessário ir actualizando de acordo com a norma XACMLv3, uma vez que esta não está na sua versão final e poderá sofrer alterações que impliquem uma reestruturação na forma como os elementos são representados e persistidos. Por outro lado, há elementos que fazem parte da norma que não foram tomados em consideração neste projecto, mas embora sejam opcionais, podem ser uma mais valia para este software.

5. Bibliografia

[1] OASIS. (2005, Fevereiro) OASIS. [Online]. http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf

[2] David Brossard. (2010, Setembro) Harvesting Web Technologies. [Online]. http://www.webfarmr.eu/2010/09/xacml-101-a-quick-intro-to-attribute-based-access-control-with-xacml/

[3] Oracle. ORACLE. [Online]. http://labs.oracle.com/people/mybio.php?uid=74233

[4] Anne Anderson. (2007, Abril) MarkMail. [Online]. http://markmail.org/message/pkastt5jrkziophr

[5] OASIS. (2010, Agosto) OASIS. [Online]. http://docs.oasis-open.org/xacml/3.0/xacml-core-v3-schema-wd-17.xsd

[6] Srikanth Shenoy, Viswanath Krishnan, Nithin Mallya, and Jagmohan Bhasin. Object Source - Chapter 21 Classloaders and J2EE. [Online]. http://www.objectsource.com/j2eechapters/Ch21-ClassLoaders_and_J2EE.htm

[7] MySQL. MySQL Connector/ODBC. [Online]. http://dev.mysql.com/doc/refman/5.1/en/connector-odbc.html

[8] Selim Mimaroglu. (2005, Dezembro) Open Source Database Special Feature: An Introduction to Berkeley DB XML. [Online]. http://xml.sys-con.com/node/164567?page=0,0

[9] Seth Proctor. (2004, Julho) Sun's XACML Implementation. [Online]. http://sunxacml.sourceforge.net/guide.html

[10] Oracle. (2003, Junho) ORACLE. [Online]. http://java.sun.com/developer/technicalArticles/Security/xacml/xacml.html

[11] Mark Kahn. How to drag and drop in javascript. [Online]. http://www.webreference.com/programming/javascript/mk/column2/

[12] Mihai Sucan. (2009, Janeiro) Dev.Opera HTML5 canvas - the basics. [Online]. http://dev.opera.com/articles/view/html-5-canvas-the-basics/

6. Anexos

A. Documentação do código javascript para criação e gestão de janelas e nós

B. Documentação JavaDoc do componente JClassLoader

C. Documentação JavaDoc do módulo BDBXmlDatabaseHelper

D. Documentação JavaDoc das interfaces dos módulos