UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

MIDDLEWARE PARA AUTORIZAÇÃO DISTRIBUÍDA UTILIZANDO XACML

JAILSON VOLNEI DOS SANTOS

BLUMENAU
2013

2013/1-17

JAILSON VOLNEI DOS SANTOS

MIDDLEWARE PARA AUTORIZAÇÃO DISTRIBUÍDA UTILIZANDO XACML

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciência da Computação — Bacharelado.

Prof. Paulo Fernando da Silva, Mestre - Orientador

BLUMENAU
2013

2013/1-17

MIDDLEWARE PARA AUTORIZAÇÃO DISTRIBUÍDA UTILIZANDO XACML

Por

JAILSON VOLNEI DOS SANTOS

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

Presidente:	Prof. Paulo Fernando da Silva, Mestre, Orientador, FURB
Membro:	Prof. Mauro Marcelo Mattos, Doutor – FURB
Membro:	Prof. Jhony Alceu Pereira, Especialista – FURB

Blumenau, 09 de Julho de 2013

Dedico este trabalho a todos os amigos, familiares em especial a meus pais José e Terezinha e a todos que contribuíram para realização deste.

AGRADECIMENTOS

A Deus, pelo seu imenso amor e graça, por me dar forças nos dias mais difíceis e por me dar fé para continuar sempre em frente.

Aos meus pais, pela educação que recebi, pela formação do meu caráter, pelo amor que sempre me deram, por saberem suportar todos os meus dias de falta de respeito devido à pressão imposta no decorrer do curso e por me darem apoio em todos os momentos.

Aos meus amigos pela esperança depositada em meu potencial.

Ao meu irmão Jardel, pela amizade, carinho e companheirismo de sempre.

Ao meu grande amor, querida amiga, e fiel namorada Fabiana, por me apoiar em todos os instantes que precisei.

Ao meu orientador, Paulo Fernando da Silva, por ter acreditado na conclusão deste trabalho.

E agradeço novamente meus pais, meu irmão e a minha namorada pelo fato de serem peças importantes na minha vida.

Pense positivo e tudo vai dar certo.

Fabiana Kozowski

RESUMO

Este trabalho apresenta o desenvolvimento de um *middleware* para gerir autorização distribuída e controle de acesso através de políticas de controle. O *middleware* é responsável por cadastrar, alterar e excluir políticas de controle de acesso a partir de um diretório especificado. A partir do cadastro de políticas é possível efetuar solicitações de requisição de acesso a recursos e receber respostas sobre se a permissão de acesso de um determinado sujeito a um determinado recurso é autorizada. Foi utilizada interface *web service* para prover heterogeneidade de comunicação, desacoplamento e flexibilidade necessários para aplicações distribuídas, e interoperabilidade no acesso ao *middleware*. Para especificação das políticas de controle de acesso e também um formato para mensagens de pedido e resposta foi utilizado o padrão *XACML* (*eXtensible Access Control Markup Language*), utilizado para definir que sujeito possui direitos de acesso sobre qual recurso. O formato utilizado para especificar pedido e resposta descreve como as consultas/requisições sob as políticas deverão ser realizadas (pedido) e como deverão ser as respostas. A partir de problemas encontrados no compartilhamento e distribuição de recursos nos sistemas distribuídos ligados à necessidade de segurança de informação e autorização e a abrangência do padrão *XACML*, os resultados obtidos na implementação do *middleware* atenderam aos objetivos propostos no trabalho. Como resultado, o desenvolvimento do *middleware* permitiu abstração da sintaxe e funcionamento do *XACML*, gerência de autorização e controle de acesso através de políticas de segurança que comprovam que é possível o desenvolvimento de uma camada de software intermediária para solucionar os problemas comentados até o momento.

Palavras-chave: *Middleware*. Controle de acesso. Autorização. Sistemas distribuídos. XACML.

ABSTRACT

This paper presents the development of a distributed *middleware* to manage authorization and access control through control policies. The *middleware* is responsible for registering, changing, and deleting access control policies from a specified directory. From the registry policy can make requests request access to resources and receive answers about yourself permission to access a particular subject to a particular feature is authorized. *Web service* interface was used to provide communication heterogeneity, decoupling and flexibility needed for distributed applications, interoperability and access to *middleware*. For specifying access control policies and also a format for request and response messages used was the standard *XACML* (*eXtensible Access Control Markup Language*), used to define who has access rights subject on which resource. The format used to specify request and response describes how queries / requests under the policies should be carried out (application) and how the answers should be. From problems encountered in the sharing and distribution of resources in distributed systems linked to the need for information security and authorization and scope of the *XACML* standard, the results obtained in the implementation of *middleware* met the proposed objectives. As a result, the development of *middleware* abstraction allowed the syntax and operation of *XACML*, management of authorization and access control via security policies that demonstrate that it is possible to develop a software layer intermediate to troubleshoot commented yet.

Keywords: *Middleware*. Access control. Authorization. Distributed systems. *XACML*.

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

ACL - *Acess Control Lists*

API - *Application Program Interface*

CORBA - *Common Object Request Broker Architeture*

DAC - *Discretionary Access Control*

DOM - *Document Object Model*

DRBAC - *Distributed Role Based Access Control*

EA - *Enterprise Architect*

HTTP - *HyperText Trasnfer Protocol*

JDom - *Java Document Object Model*

MAC - *Mandatory Access Control*

NIST - *National Institute of Standarts and Technology*

OASIS - *Organization for the Advancement of Structured Information Standards*

PAP - *Policy Administration Point*

PCABP - Política de Controle de Acesso Baseada em Papéis

PCAD - Política de Controle de Acesso Discricionária

PCAO - Política de Controle de Acesso Obrigatória

PDP - *Policy Decision Point*

PEP - *Policy Enforcement Point*

PIP - *Policy Information Point*

POO - Programação Orientada a Objetos

RBAC - *Role-based Access Control*

RPC - *Remote Procedure Call*

SGPCA - Sistema Gerenciador de Políticas de Controle de Acesso

SOA - *Service Oriented Architecture*

SOAP - *Simple Object Access Protocol*

SSTC - *Security Services Technical Commitee*

UDDI - *Universal Description Discovery and Integration*

UML - *Unified Modeling Language*

URL - *Uniform Resource Locator*

WS - *Web Services*

WSDL - *Web Services Description Language*

XACML - *eXtensible Access Control Markup Language*

XML - *eXtensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

Após a explosão da Internet em 1993, uma referência para computação descentralizada que ficou muito popular foram os sistemas distribuídos. A principal motivação na construção de um sistema distribuído é o compartilhamento de recursos tais como: arquivos, páginas web, hardwares diversos, acesso a banco de dados distribuídos, entre muitos outros. Mas não basta citar apenas isso, pois um sistema distribuído é um conjunto de processos concorrentes acessando recursos distribuídos, os quais podem ser compartilhados ou replicados, através de troca de mensagens em um ambiente de rede (MARTINEZ, 2010).

Devido as limitações encontradas na capacidade de processamento que impõem restrições aos muitos tipos de software como: programas de escritório, programas de manipulação de imagens, jogos, científicos e servidores utilizados nas organizações, técnicas que possibilitassem o processamento distribuído para contornar as limitações de processamento começaram a ser aderidas(DANTAS, 2005, p. 3; 4).

Com o aumento exponencial de processamento de tarefas, alta disponibilidade e por apresentar maior tolerância a falhas, os sistemas distribuídos passaram a ser considerados sistemas de alto desempenho ou alta disponibilidade e começaram a atender como solução para sistemas conhecidos como críticos. A necessidade de aumento da capacidade de processamento de informações, compartilhamento de dados e sem grandes custos são fatos que sugerem que esta abordagem seja utilizada por trazer boas vantagens (PIGATTO, 2009, p. 16).

Na implementação de um sistema distribuído surge a dúvida sobre possibilidade de desenvolvimento da escalabilidade e a abertura de sistema de operacional da rede juntamente com transparência e facilidade relacionada ao uso do sistema operacional distribuído. Esta solução deve ser encontrada em uma camada adicional de software que é utilizado no sistema operacional de rede mais ou menos heterogênea de esconder o conjunto de plataformas subjacentes, mas também para melhorar a transparência da distribuição. Muitos sistemas distribuídos modernos são construídos por meio de desta camada adicional chamada de midddleware (TANENBAUM; STEEN, 2006, p. 36). Esta solução é uma camada mediadora que facilita a comunicação e alto nível de abstração entre uma aplicação e sistema operacional de rede.

Atualmente os sistemas distribuídos são muito utilizados em ambientes que necessitam ter escalabilidade, alto desempenho, tolerância a falhas e heterogeneidade. Com isso necessita-se de segurança. Entretanto não é aceito que sistemas robustos que trocam informações entre cliente e servidores possam ser interceptados, modificados ou até que apresentem falta de controle e autorização de acesso e modificação a recursos.

Segundo Rosset (2004, p. 17 apud GOLLMMAN, 1999), “A segurança trata essencialmente de recursos”, por isso devem-se conhecer os recursos e seus valores para melhor proteção. A prevenção, a detecção e a reação são os princípios da segurança computacional.

A segurança das informações em qualquer sistema é importante, pois garante que o sistema conseguirá executar os objetivos para que seja projetado de maneira correta e para os usuários corretos. O mecanismo responsável por garantir que apenas usuários autorizados consumam os recursos protegidos de um sistema computacional nos sistemas de informação (arquivos, programas de computador, dispositivos de hardware e funcionalidades disponibilizadas por aplicações) é a autorização.

A autorização é uma necessidade da segurança para proporcionar diferentes níveis de acesso (por exemplo, negar / permitir) para diferentes partes ou operações em um sistema de computação. O tipo de acesso é definido pela identidade da pessoa e pelo tipo de operação ou parte do sistema requisitado (LEANDRO, 2012, p. 49).

A autorização de acesso e modificação a recursos é estipulada através de políticas de controle de acesso. As políticas de controle de acesso são um meio de restringir o acesso a objetos protegidos e são elaboradas considerando o ambiente em que se está trabalhando, para que as regras e critérios estabelecidos sejam de acordo com as práticas e realidade de segurança vivenciada pela empresa (LIMA, 2008, p. 27).

As políticas de controle de acesso são um meio de especificar ou modificar o comportamento de gerência em um sistema, para isso definem como os serviços em sistemas computacionais podem ser usados (LIMA, 2008, p. 27 apud LORCH et al. 2003)

Geralmente cada sistema utiliza uma linguagem ou padrão próprio para definição de suas políticas de acesso, com isso se torna limitada levando em vista a concepção de sistemas distribuídos e abertos. Visando a transparência entre diversos sistemas o órgão Organization for the Advancement of Structured Information Standards (OASIS) lançou a eXtensible Access Control Markup Language (XACML), uma linguagem padrão para especificação de políticas de controle de acesso de proposito geral em eXtensible Markup Language (XML) (MELLO et al., 2006).

A utilização do padrão XACML para especificação de políticas de controle de acesso em XML proporciona alto poder de flexibilidade na definição de regras de autorização, que permitem tanto a definição de políticas globais genéricas quanto totalmente individuais, além de prover interoperabilidade com aplicações que possam estar em domínios diferentes, proporcionando a capacidade de executar controle de acesso distribuído (ROSSET, 2004, p. 14).

A partir dos cenários abordados até o momento, percebeu-se problemas em relação a preocupação nos dias atuais com a segurança da informação e a autorização ligada ao compartilhamento e distribuição de recursos encontrados nos sistemas distribuídos juntamente a sua capacidade de escalabilidade. Também foi observado que a alta flexibilidade e genericidade do *XACML* requer um alto conhecimento das funcionalidades e sintaxes utilizadas para ser utilizado.

Diante do exposto, neste trabalho propõe-se o desenvolvimento de um *middleware* para implementar a comunicação entre recursos e aplicativos distribuídos para autorização de acesso em XACML, para tanto o *middleware* poderá manter políticas de controle de acesso que serão alocadas em um repositório e através destas políticas o referido *middleware* disponibilizará autorização distribuída a partir de solicitações de acesso e geração de respostas de autorização

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é o desenvolvimento de um *middleware* para gerir a autorização distribuída e controle de acesso através de políticas de controle utilizando a norma extensível e genérica definida pela OASIS – XACML.

Os objetivos específicos do trabalho são:

a) fornecer serviços de autorização distribuída a controles de acesso;
b) padronizar uma interface para utilização dos serviços disponíveis;
c) utilizar norma XACML no âmbito de especificação e gerência de políticas de controle de acesso;
d) fornecer *middleware* para tratar as solicitações (pedidos e respostas) de controle de acesso;

e) utilizar *web service* para oferecer interface de comunicação a aplicação que se utilizará do *middleware*.

1.2 ESTRUTURA DO TRABALHO

Este trabalho está estruturado em quatro capítulos, onde no primeiro foi apresentada a introdução, os objetivos e a estrutura do trabalho. O segundo capítulo apresenta a fundamentação teórica, contextualizando os temas abordados no desenvolvimento do trabalho, tais como: sistemas distribuídos, *middleware*, *web services*, segurança, autorização e controle de acesso, XACML e trabalhos correlatos. No terceiro capítulo é apresentado o desenvolvimento do sistema, contendo desde a especificação até a operacionalidade. O quarto e último capítulo apresentam as conclusões e algumas propostas de extensões para o *middleware*.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo está organizado em três seções: a primeira trata de sistemas distribuídos sendo subdividida em duas etapas: *middleware* e segurança; a segunda descreve autorização em sistemas distribuídos sendo subdividida em duas etapas: políticas de controle de acesso e XACML; e a última seção traz trabalhos correlatos ao trabalho proposto.

2.1 SISTEMAS DISTRIBUÍDOS

Ao decorrer dos anos uma preocupação maior se deu nas comunidades acadêmicas e industrias com relação ao baixo desempenho dos computadores comerciais convencionais. A limitação na capacidade de processamento impõe muitas restrições aos diversos tipos de softwares. Uma forma de contornar a limitação local de processamento é a utilização de técnicas que possibilitem o processamento distribuído. O processamento distribuído é um paradigma computacional interessante pois permite descentralização de componentes de hardware, pacotes de software e uso de instrumentos geográficos dispersos (DANTAS, 2005, p. 04).

Com o avanço tecnológico e com o crescimento da característica de distribuição de tarefas foram criadas as redes de computadores e a sua utilização como solução para problemas que envolviam tarefas exercidas por mais de um indivíduo. Esta necessidade de soluções distribuídas culminou na criação dos sistemas computacionais distribuídos (SOUZA, 2010, p. 41).

É difícil propor uma definição geral e abrangente para sistemas computacionais distribuídos, pois pode atender a muitas entidades com requisitos diferentes, porém, Tanenbaum e Steen (2006, p. 02) definem que “Um sistema distribuído é uma coleção de computadores independentes que se apresenta para o usuário como um sistema único e coerente”. De acordo com Coulouris, Dollimore e Kindberg (2007, p. 17), “Um sistema distribuído é aquele no qual os componentes interligados em rede se comunicam e coordenam suas ações apenas passando mensagens”.

Um sistema distribuído deve ser fácil de se expandir ou escalar. Esta característica é uma consequência direta de ter computadores independentes, mas, ao mesmo tempo, escondendo como esses computadores efetivamente tomam parte no sistema como um todo. Um sistema distribuído normalmente estará continuamente disponível, embora talvez certas partes do sistema podem estar temporariamente indisponíveis ou não utilizados (TANENBAUM; STEEN, 2006, p. 2). Para os usuários e aplicações a distribuição de processamento, a adição ou alteração de componentes, heterogeneidade, entre outras características deve ser transparente.

Os desafios que se encontram na construção de sistemas distribuídos são a heterogeneidade de seus componentes, ser um sistema aberto, que permita que seus componentes sejam substituídos ou adicionados, segurança, escalabilidade, tratamento de falhas, concorrência de componentes e transparência (COULOURIS; DOLLIMORE; KINDBERG, 2007, p. 17). Os desafios citados são explicados abaixo de acordo com Coulouris, Dollimore e Kindberg (2007, p. 36):

a) heterogeneidade: os sistemas distribuídos devem ser construídos a partir de uma variedades de redes, sistemas operacionais, hardwares e linguagens de programação;
b) sistemas abertos: os sistemas distribuídos devem ser extensíveis;
c) escalabilidade: um sistema distribuído é considerado escalável se o custo de adição de um usuário for um valor constante em termos dos recursos que devem ser adicionados, ou seja, por mais que aumente o numero de usuários os sistema continuará evitando gargalos de desempenho e tempo de acesso;
d) segurança: Os recursos compartilhados e informações devem ser mantidos de forma sigilosa quando são transmitidos em rede;
e) tratamento de falhas: qualquer processo, computador ou rede pode falhar, portanto componentes devem saber tratar de maneira apropriada quando um dos componentes que depende falhar;
f) concorrência: com muitos usuários acessando múltiplos recursos, o sistema deve manter a consistência nos estados dos dados dos recursos;
g) transparência: devem se manter a transparência dos detalhes de distribuição e localização dos processos e recursos do sistema.

Tanenbaum e Steen (2006, p. 36) fazem a seguinte pergunta “[...] a questão que vem à mente é se é possível desenvolver um sistema distribuído que tem o melhor dos dois mundos:

a escalabilidade e abertura dos sistemas operacionais de rede e atransparência e facilidade relacionada ao uso do sistema operacional distribuído". A questão citada é solucionada através de uma camada adicional de software que é utilizado no sistema operacional de rede para esconder o conjunto de plataformas subjacentes e melhorar a capacidade de transparência da distribuição. Esta camada adicional é chamada de *midddleware* (TANENBAUM; STEEN, 2006, p. 36).

É complexa a implementação de comunicação entre os computadores de uma rede no topo das primitivas do sistema operacional, por isso suportes de sistemas de *middleware* localizados entre componentes do sistema distribuído e componentes do sistema operacional, facilitam suas interações (BARBOSA, 2001, p. 02).

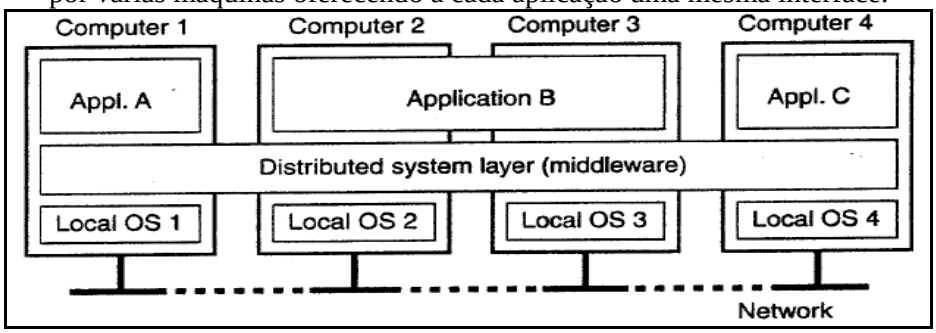
2.1.1 *Middleware*

Para suportar computadores e redes heterogêneas oferecendo uma única visão de sistema, sistemas distribuídos são geralmente organizados por meio de uma camada intermediária de software que é logicamente colocado entre uma camada de nível mais alto que consiste de utilizadores e aplicativos, e uma camada abaixo que consiste dos sistemas operacionais e básicos meios de comunicação (TANENBAUM; STEEN, 2006, p. 03).

Segundo Rodrigues (2007, p. 14) "*middleware* é uma categoria de produtos ou módulos de software que são utilizados por aplicações cliente, para acessar aplicações servidoras, e tentam esconder da aplicação a rede, a comunicação e plataformas especificas", ou seja, abstrai funcionamento, plataforma e tecnologias, evitando o redesenvolvimento.

Na Figura 1 tem-se visão de uma camada de *middleware* sendo utilizado entre aplicações distribuídas e sistemas operacionais.

Figura 1 - Um sistema distribuído organizado como middleware. A camada de middleware se estende por várias máquinas oferecendo a cada aplicação uma mesma interface.



Fonte: Tanenbaum e Steen (2006, p. 03).

A Figura 1 apresenta quatro computadores em rede com três aplicações. A aplicação "B" é distribuída através de computadores 2 e 3. Cada aplicação é oferecida a mesma interface. O sistema distribuído proporciona os meios para que os componentes de uma única aplicação distribuída possam se comunicar uns com os outros, mas também para permitir que diferentes aplicativos se comuniquem. Ao mesmo tempo, ele oculta o máximo possível as diferenças de hardware e sistemas operativos de cada aplicação (TANENBAUM; STEEN, 2006, p. 03).

Compreende-se por *middleware* uma camada intermediária de software, que permite mediar a comunicação entre aplicações distribuídas, assim objetivando a diminuição da complexidade e heterogeneidade de diversos sistemas de forma transparente.

Os pontos importantes na utilização e funcionalidade dos *middlewares* são: padronização, facilidade de uso e gerenciamento, flexibilidade e performance. Porém é impossível o alto nível de performance e flexibilidade, idealizando-se assim um balanceamento (MACIEL; ASSIS, 2004, p. 53).

Além de resolver os problemas de heterogeneidade, o *middleware* fornece um modelo computacional uniforme para ser usado pelos programadores de serviços e aplicativos distribuídos (COULOURIS; DOLLIMORE; KINDBERG, 2007). Os modelos possíveis incluem a invocação remota de objetos, a notificação remota de eventos, o acesso remoto a banco de dados e o processamento de transações distribuídas (COULOURIS; DOLLIMORE; KINDBERG, 2007).

Há um número de serviços comuns a muitos sistemas de *middleware*. Invariavelmente, todos os *middlewares* de uma forma ou de outra tentam implementar a transparência de acesso, oferecendo alto nível de facilidade de comunicação que escondem a mensagem de baixo nível que estão passando pelas redes de computadores. A interface de programação para a camada de transporte oferecidos pelos sistemas operacionais de rede é, portanto, inteiramente substituída por outras instalações. Para dar suporte a comunicação depende-se

muito do modelo de distribuição que o *middleware* oferece para usuários e aplicações (TANENBAUM; STEEN, 2006, p. 39).

Comunicação em sistemas distribuídos é baseada em passar mensagens de baixo nível oferecidas pela rede subjacente. A comunicação através de troca de mensagens é mais difícil do que usar primitivas baseados em memória compartilhada. Modernos sistemas distribuídos, muitas vezes consistem de uma quantidade enorme de processos espalhados em uma rede não confiável como a internet. A menos que as facilidades de comunicação de primitivas de redes de computadores são substituídas por outra coisa, o desenvolvimento de aplicações distribuídas em larga escala é extremamente difícil (TANENBAUM; STEEN, 2006, p. 57).

Diferentes implementações de *middlewares* foram desenvolvidas para integração de sistemas distribuídos com principio de encapsular os detalhes de comunicação. As primeiras plataformas utilizavam técnicas procedurais, com chamada remota de procedimento, do inglês *Remote Procedure Call* (RPC), e baseavam-se no modelo cliente-servidor[1]. Posteriormente, surgiram *middlewares* orientados a objetos, como *Common Object Request Broker Architeture* (CORBA) e Java RMI (CECHINEL, 2009, p. 13).

Atualmente tem surgido no mercado alternativas não orientadas a objetos, porém que utilizam paradigmas nas aplicações desenvolvidas com técnicas de Programação Orientada a Objetos (POO). Estas alternativas são chamadas de *Web Services* (WS) e *Service Oriented Architecture* (SOA). De acordo com ORACLE (2008) “[...] No baixo nível, os *middlewares* orientados a objetos fazem RPC, mas camadas de abstração adicionais oferecem significativas vantagens aos programadores das linguagens orientadas a objetos”.

2.1.1.1 *Web Services*

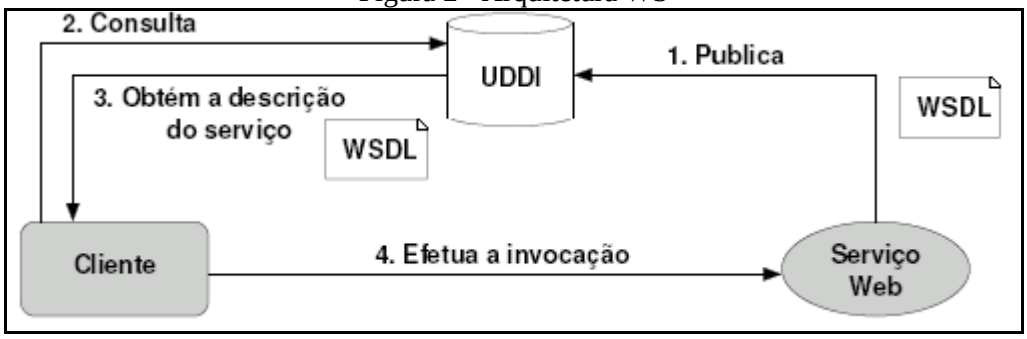
Web Services (WS) é um *middleware* que provê interfaces de comunicação entre aplicações na Web (WEB SERVICES ACTIVITY, 2002) e utilizam o *HyperText Trasnfer Protocol* (HTTP) como protocolo de aplicação (CECHINEL, 2009, p. 31). WS utiliza o protocolo *Simple Object Access Protocol* (SOAP) para troca de mensagens entre sistemas distribuídos dentro da filosofia RPC (EHNEBUSKE et al., 2000).

[1] Modelo onde os processos são divididos em “servidor” (processo de execução de um serviço especifico) e “cliente” (processo que solicita um serviço do servidor enviando-lhe um pedido e, posteriormente esperando resposta) (TANENBAUM; STEEN, 2006, p. 37).

O SOAP é utilizado sobre o HTTP facilitando assim a implantação no próprio servidor web, utilizando os mesmos serviços de autenticação e criptografia e utiliza XML como linguagem de especificação. Com o SOAP o cliente realiza a requisição para o servidor e recebe suas respostas através de estruturas XML. Esta abordagem provê grande flexibilidade ao protocolo (CECHINEL, 2009, p. 31).

Além do SOAP outras especificações como *Web Services Description Language* (WSDL) e a *Universal Description Discovery and Integration* (UDDI) também utilizam XML. A WSDL provê um método padrão para descrever *Web Services* e suas propriedades. A UDDI é o serviço padrão para publicação e localização de Serviços Web (MELLO et al., 2006).

Figura 2 - Arquitetura WS



Fonte: Mello et al. (2006).

De acordo com Mello et al. (2006), a Figura 2 ilustra uma arquitetura de um WS. Para tornar um WS disponível inicialmente o provedor de serviços deve descrever a interface do serviço que deseja prover, utilizando a WSDL. De acordo com passo 1 da Figura 2 a interface do WS deve estar publicada em um serviço de busca público. A partir de então, de acordo com passo 2 e 3 o cliente pode localizar o serviço desejado e obter a sua WSDL. A comunicação entre o servidor e o consumidor de um serviço é realizada através de envelopes SOAP (passo 4).

Apesar de não proverem padrão de procedimentos de segurança pois permitem o acesso direto a aplicações, os WS tem as vantagens de utilizar padrões abertos (permite ser desenvolvido para diferentes linguagens e plataformas), podem ser facilmente implementados e possuem baixo custo de implementação e de integração entre aplicações pelo fato de utilizar estruturas e protocolos de comunicação existentes para efetuar troca de informação (ROSSET, 2004, p. 35).

2.1.2 Segurança

A segurança em sistemas distribuídos pode ser dividida em comunicação entre usuários ou processos, onde os mecanismos que garantem a comunicação segura são autenticação, integridade da mensagem e confidencialidade, e a preocupação com a autorização que lida com a garantia de que um processo ou sujeito recebe apenas os direitos de acesso aos recursos que lhe são permitidos em um sistema distribuído (TANENBAUM; STEEN, 2006, p. 377).

Muitos recursos de informação que se tornam disponíveis nos sistemas distribuídos tem um alto valor para seus usuários. A segurança de recursos de informações tem três componentes: confiabilidade (proteção contra exposição a pessoas não autorizadas), integridade (proteção contra alteração ou dano) e disponibilidade (proteção contra interferência com os meios de acesso aos recursos) (COULOURIS; DOLLIMORE; KINDBERG, 2007, p. 30).

É de grande importância que se assegure confiabilidade, integridade e disponibilidade em redes de computadores, sistemas operacionais, hardwares e linguagens de programação. Os três termos de garantia de segurança citados são providos ao limitar o acesso a recursos do sistema e permitir apenas autorização ao acesso a recursos a usuários autorizados de acordo com a política de segurança e controle (SOUZA, 2010, p. 25).

Na maioria dos tipos de rede local é fácil construir um programa que obtenha cópias de mensagens transmitidas entre processos, por exemplo, um programa deste tipo pode ser executado em um computador que já está conectado à rede ou em um que está infiltrado nela, através de um ponto de conexão sobressalente, ou até instalar como um servidor de arquivos e obter cópias de informações confidenciais contidas nos dados que os clientes encaminham para armazenamento (MARTINS, 2009).

Para se resguardar das ameaças citadas acima e para garantir a segurança requerida, políticas de segurança devem ser adotadas e mecanismos de segurança devem ser empregados a fim de implementar tais políticas de acesso e autorização (SOUZA, 2010, p. 22).

2.2 AUTORIZAÇÃO EM SISTEMAS DISTRIBUÍDOS

No âmbito de segurança em sistemas distribuídos existe a preocupação com o acesso e gestão de direitos. Nos sistemas centralizados a gestão dos direitos de acesso é relativamente fácil, pois no momento que um novo usuário é adicionado ao sistema, é dado direitos iniciais, ou seja, relato completo de um usuário é configurado para uma máquina específica em que todos os direitos foram previamente especificado pelo sistema administradores (TANENBAUM; STEEN, 2006, p. 440). De acordo com Tanenbaum e Steen (2006, p. 434) “Em um sistema distribuído, as questões de gestão de direitos e acesso são complicadas pelo fato dos recursos estarem espalhados por várias máquinas. Se a abordagem de sistemas não distribuídos fosse seguida, seria necessário criar uma conta para cada usuário em cada máquina.”.

Uma questão importante na segurança de sistemas distribuídos é o controle de acesso, ou autorização. O termo autorização no âmbito de segurança da informação é o artifício que garante a proteção de recursos para que só possam ser utilizados (ou qualquer tipo de acesso) por usuários autorizados. Os recursos incluem arquivos, softwares, funcionalidades ou módulos de um sistema, periféricos e dispositivos de hardware (SOUZA, 2010, p. 30).

> A autorização é o processo de conceder ou negar direitos a usuários ou sistemas, por meio das chamadas listas de controle de acessos (Acess Control Lists – ACL), definindo quais atividades poderão ser realizadas, desta forma gerando os chamados perfis de acesso. (LAUREANO, 2005, p. 20).

Podem ser considerados consumidores de recursos as pessoas que utilizam um sistema através de uma interface, programas e outros dispositivos de um computador.

A propriedade básica de segurança utilizada para determinar se um sujeito tem acesso para executar alguma ação sobre algum recurso é a autorização.

Para que sejam efetuados os controles de acesso e proteção a estes recursos as organizações estipulam um conjunto de regras e requisitos conhecidos como políticas de segurança que tem como intuito garantir sua segurança. Essas políticas devem conter políticas de controle de acesso no qual descreve a relação entre pessoas e recursos de uma organização e devem ser capaz de permitir que pessoas autorizadas utilizem os recursos (SOUZA, 2010, p. 11).

2.2.1 Políticas de controle de acesso

O controle de acesso, como o próprio nome especifica, é uma referência à prática de permitir o acesso a um recurso apenas por pessoas autorizadas. Este controle de relação entre pessoas e recursos é descrito a partir de políticas de controle. O controle de acesso tem sido um dos mecanismos de segurança mais básicos e amplamente utilizados (HU et al., 2007). A decisão de controle de acesso ou autorização é determinada pela política de controle de acesso que é uma descrição da relação entre sujeito, objeto, operação e uma permissão (SOUZA, 2010, p. 30).

Os três conceitos citados acima devem ser esclarecidos no âmbito de políticas de controle de acesso: sujeito, operação e objeto. O sujeito é especificado pela parte que deseja realizar alguma ação no sistema (usuário, programa, processo ou outro sistema). A ação a ser realizada é chamada de operação é pode ser uma escrita, leitura, execução, criação, etc., e o objeto se trata do recurso no qual o sujeito deseja executar a operação (SOUZA, 2010, p. 26).

Segundo Souza (2010, p. 25), "O Controle de Acesso pode ter impacto sobre todos os objetivos da segurança em computação. Contudo a sua implementação restringe-se em geral apenas a dois: integridade e confidencialidade [...]".

A capacidade de limitar o acesso a recursos do sistema faz o primeiro relacionamento com a confidencialidade. As atividades de leitura e cópia só serão permitidas a usuários com autorização, portanto possibilitando a confidencialidade da informação. Já as atividades de criação e modificação passam a ser atividades restritas a usuários de acordo com a política de segurança, contribuindo assim para a integridade (SOUZA, 2010, p. 25).

Em um projeto de sistema seguro, pode-se adotar qualquer mecanismos de controle de acesso que gerenciem as tomadas de decisão a partir das políticas de segurança do sistema. Os mecanismos de controle acesso são classificados em três tipos (ROSSET, 2004, p. 28):

a) *Discretionary Access Control* (DAC): o controle de acesso descricionário permite que um usuário possa definer como suas informações podem ser acessadas por outros usuários, onde o dono da informação determina quem pode acessa-la;
b) *Mandatory Access Control* (MAC): o controle de acesso obrigatório define acesso as informações de maneira centralizada com regras incontornáveis, ou seja, determina acesso as informações através de parâmetros (classificação dos usuários, ou grupos de usuários, em níveis de segurança e rótulos de segurança) estabelecidos por um sistema administrador;

c) *Role-based Access Control* (RBAC): o controle de acesso baseado em papéis define acesso as informações de acordo com a função do usuário (gerente, secretaria, etc.).

Geralmente são utilizadas linguagens próprias, dependendo do sistema, para definição das políticas de controle, tornando assim um fator limitante para a concepção de sistemas distribuídos e abertos.

Para facilitar a gestão e manutenção de controle de acesso, políticas de controle de acesso são cada vez mais escritas em linguagens de especificação, como *eXtensible Access Control Markup Language* (XACML) (HU et al., 2007).

2.2.2 *eXtensible Access Control Markup Language – XACML*

O XACML foi desenvolvido pela OASIS *Security Services Technical Commitee* (SSTC) como um padrão de linguagem de marcação que permite especificar políticas de segurança, requisições e respostas para decisões de autorização e controle de acesso. É um padrão aberto (*Open-Standard)* e pode ser utilizado tanto para prover controle de acesso para sistemas completos bem como a um recurso específico, permitindo que organizações com ambiente e aplicações tanto proprietárias quanto públicas possam utilizar estas políticas para controlar acesso a conteúdos e informações protegidas (ROSSET, 2004, p. 50).

O XACML (OASIS, 2005b) é um padrão que define uma linguagem declarativa, utilizada para a descrição de políticas de controle de acesso e uma linguagem para formular consultas (e obter respostas), utilizada para verificar se uma determinada ação é ou não permitida. Como resposta à consulta, um dos seguintes valores pode ser retornado: *Permit*, *Deny*, *Indeterminate* (um erro ocorreu, ou algum valor não foi especificado, impedindo o prosseguimento da consulta) ou *Not Applicable* (quando a resposta não puder ser retornada pelo serviço solicitado, inexistência de políticas ou regras).

> A XACML (eXtensible Access Control Markup Language) descreve uma linguagem para políticas de controle de acesso e também um formato para mensagens de pedido e resposta. A linguagem para política de controle de acesso é utilizada para definir quem possui direitos de acesso sobre o quê. O formato de pedido e resposta descreve como as consultas sobre o sistema de políticas deverão ser realizadas (pedido) e como deverão ser as respostas. (FERREIRA et al., 2004, p. 05).

O modelo de arquitetura utilizado pelo XACML (ROSSET, 2004, p. 50) é dividido em entidades denominadas: *Policy Enforcement Point* (PEP); *Policy Decision Point* (PDP);

Policy Information Point (PIP) e *Policy Administration Point* (PAP). Estas entidades executam os seguintes papéis (ROSSET, 2004, p. 50):

a) PEP: entidade responsável por receber as requisições de acesso, envia-las para o PDP e aplicar decisões de acesso. Esta entidade encaminha os pedidos de autorização e interpreta as respostas obtidas;
b) PDP: entidade responsável por tomar as decisões de acesso sobre as solicitações encaminhadas pelo PEP. As políticas existentes e que se aplicam ao pedido são avaliadas para permitir ou negar o acesso e são encaminhadas novamente ao PEP;
c) PAP: esta entidade é caracterizada pela criação e gestão das políticas de controle de autorização de acesso e disponibiliza-las ao PDP;
d) PIP: entidade repositório de atributos externos que são disponibilizados ao PDP.

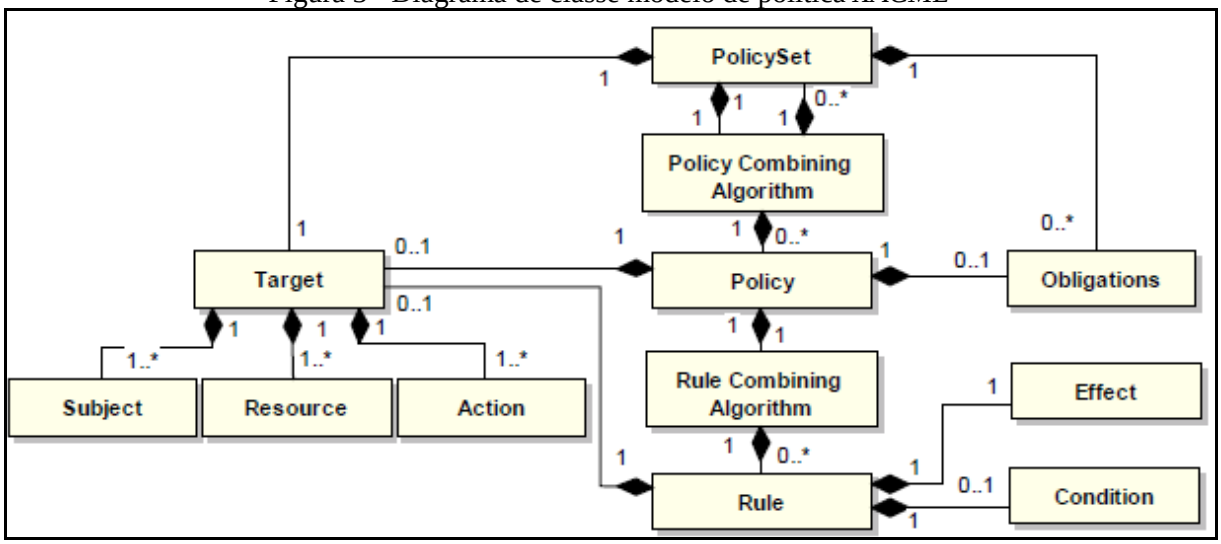
O PIP é uma fonte de informações de determinados atributos. Por exemplo, este pode ser um sistema de informações de dados de utilizadores que não constam no pedido de solicitação e devem ser utilizados pelo PDP para poder analisar a política relacionada. Esta entidade não é especificada quanto a sua implementação e funcionalidade (GOUVEIA, 2011, p. 17).

O formato de pedido e resposta tratados a partir do padrão XACML define as trocas de comunicação ente o PDP (ponto de processamento da política) e o PEP (ponto que efetua as decisões de política). A linguagem XACML é definida por dois esquemas (schema) XML: “xacml context” e “xacml policy”. O “xacml context” define como representar as mensagens de requisição e resposta trocadas entre o PEP e o PDP e o “xacml policy” como representar as políticas de controle de acesso (TOKTAR et al., 2005).

Um pedido ou requisição (*request*) é composto pelos seguintes atributos: sujeito que originou a requisição (*subject*), identificação do recurso desejado (*resource*), ações que serão executadas no recurso (*action*) e atributos do ambiente (*environment* – quaisquer atributos que não façam parte da *Target*). Na resposta (*response*) estarão descritas as tomadas de decisão formalizadas pelo PDP: *permit*; *deny*; *not applicable* ou *indeterminate* (MELLO et al. 2006 apud LORCH et al. 2003)

É representado de acordo com Figura 3 um diagrama UML para o esquema “xacml policy” das classes e associações entre os elementos XACML. De acordo com o mostrado a seguir na Figura 3, uma política determina um conjunto de permissões (ou negações) de acesso através de associações denominadas *Targets* mais conhecido como alvo (TOKTAR et al., 2005).

Figura 3 - Diagrama de classe modelo de política *XACML*



Fonte: Toktar et al. (2005).

A estrutura de todas as políticas *XACML* é iniciada em um elemento raiz que pode ser um *Policy* (política) ou um *PolicySet* (conjunto de políticas). Um *PolicySet* pode conter um conjunto de políticas e uma política pode conter um conjunto de *Rules* (regras). Uma regra é um conjunto de atributos a serem calculados num predicado, que avalia se esses atributos estão de acordo com os parâmetros aceitáveis (ROSSET, 2004, p. 52).

As políticas e as regras são elementos fundamentais para que as decisões sejam tomadas, uma vez que elas contém as informações e algoritmos necessários para determinar se um determinado sujeito (*Subject*) pode executar determinada ação (*Action*) em um determinado recurso (*Resource*). Os elementos citados fazem parte de outro elemento chamado de *Target*. O alvo como é conhecido é utilizado para identificar o sujeito, recurso e ação que determinada regra, política ou conjunto de políticas se aplica (GOUVEIA, 2011, p. 03). Quando o PEP requisita uma autorização referente a uma *Target* (sujeito, recurso e ação), somente políticas ou conjunto de políticas que contenham os elementos de sujeito, recurso e ação, ou seja, o *Target*, que se aplique aos dados da requisição serão processadas. Os *Targets* associados a regras (*Rules*) permitem expressar permissões (ou negações) condicionais (TOKTAR et al., 2005).

Pode ocorrer de elementos como *Subjects* e *Actions* (ou outro) não conterem dados, por isso é utilizada uma lista vazia ou uma tag *<Any<Nome_Elemento>>*. Caso algum elemento não tenha especificação no seu *Target* estas especificações são herdadas do elemento superior, ou seja, uma *regra* que não contenha especificação no elemento *Subject* de sua *Target* herdará as especificações do elemento *Subject* da política, o mesmo se aplica com política e conjunto de políticas (GOUVEIA, 2011, p. 05).

Uma regra (elemento *Rule*) é expressa pela sintaxe: “Se as condições (*Condition*) forem satisfeitas, então aplicar o efeito (*Effect*) sobre o alvo (*Target*)”. Os valores possíveis para Effect são de permissão (*Permit*) ou negação (*Deny*) (TOKTAR et al., 2005).

O elemento *Condition* é utilizado para criação de condições adicionais às regras e implica diretamente no resultado do efeito de avaliação de uma regra. Uma condição é sempre avaliada por uma função lógica que efetua comparações de valores de atributos de sujeito, recurso, ação ou ambiente (valores complementares que não dizem respeito ao sujeito, recurso ou ação). Quando a condição é satisfeita retorna um efeito (*Deny* ou *Permit*). A avaliação de uma condição também pode retornar um erro (*Indeterminate*) e ainda pode retornar (*NotApplicate*), quando uma condição não se aplica a requisição de acesso.

Opcionalmente algumas políticas são criadas com obrigações que têm de ser cumpridas para que o acesso ao recurso seja permitido ou negado, este elemento na estrutura XACML é chamado de *Obligation* (GOUVEIA, 2011, p. 08).

A classe *Obligations*, quando definida, é passada para o PEP juntamente com o resultado da avaliação da política. De acordo com Toktar et al. (2005), a versão 1.0 do XACML (versão utilizada também neste trabalho) não especifica os tipos de ações passadas em *Obligations*, apenas define que o PEP deverá interpretá-las sem especificar o modo de processamento. Referente a especificação do XACML é importante especificar outras limitações como: ausência da definição referente ao protocolo de comunicação para suportar a troca de mensagens entre PDP e PEP e a definição da estratégia a ser adotada para armazenar os documentos XACML que representam as políticas na rede (TOKTAR et al., 2005).

De acordo com a Figura 4, é ilustrado como o modelo UML é representado em um documento XML.

Figura 4 - Representação de estrutura de política *XACML*

```
<Policy PolicyId="..." RuleCombiningAlgId="...">
  <Target>
     <Subjects>...</Subjects>
     <Resources>...</Resources>
     <Actions>...</Actions>
  </Target>
  <Rule RuleId=" " Effect=" ">
     <Target>...</Target>
     <Condition FunctionId=" ">...</Condition>
  </Rule>
  <Obligations>
     <Obligation ObligationId=" " FulfillOn=" "> </Obligation>
  </Obligations> <!-- Em Obligations, o atributo FulfillOn indica se o obligation  -->
</Policy>       <!--deve ser exectuado no caso de um efeito "Permit" ou "Deny" -->
```

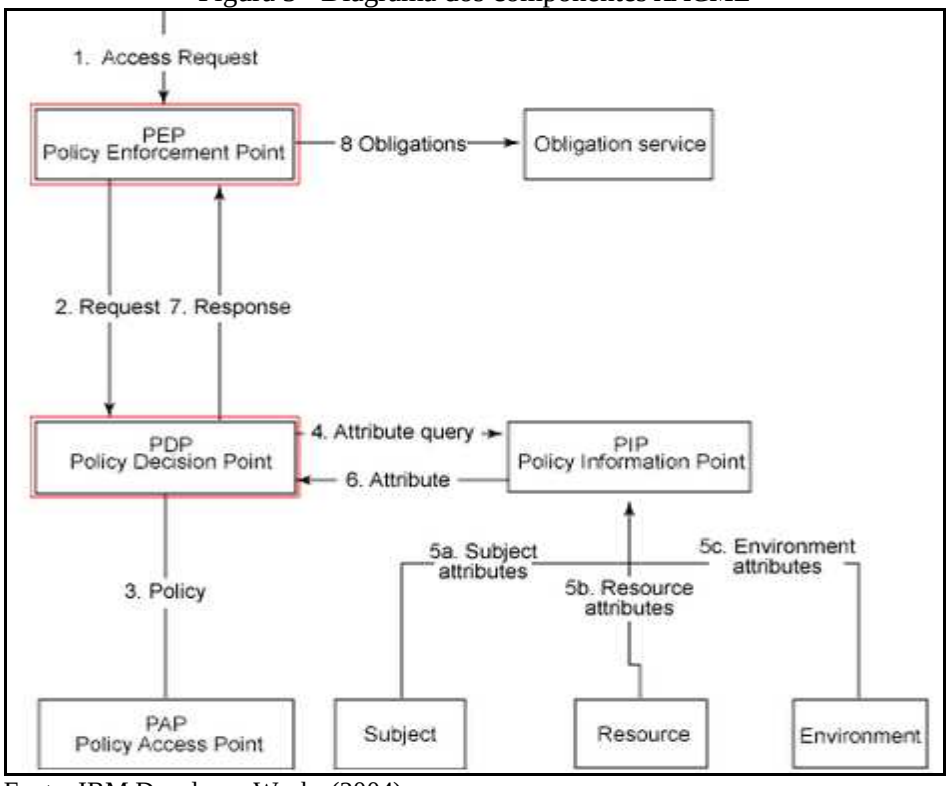
Fonte: Toktar et al. (2005).

Quando um PEP efetua uma requisição ao PDP, ele fornece os atributos que permitem identificar os elementos de uma *Target* (*Subject*, *Resource*, *Action*). O PDP avalia as regras da política e determina se existe uma *Target* com esses

> atributos, e então retorna o efeito correspondente: “permitir” ou “negar”. Se ele não conseguir encontrar uma *Target* em suas políticas que satisfaça os atributos fornecidos pelo PEP, ele retornará como resposta: não-aplicável (*NotApplicable*) (TOKTAR et al. 2005).

O *XACML* é composto por vários componentes de acordo com Figura 5 (IBM DEVELOPER WORKS, 2004).

Figura 5 - Diagrama dos componentes *XACML*



Fonte: IBM Developer Works (2004).

De acordo com a Figura 5, após receber solicitação acesso o PEP cria um pedido XACML e envia para o PDP, que avalia a solicitação e envia de volta uma resposta. A resposta pode ser tanto uma permissão de acesso como negado, com as devidas obrigações. O PDP chega a uma decisão depois de avaliar as políticas e as regras dentro deles. O PAP escreve políticas e conjuntos de políticas, e torna-os disponíveis para o PDP, porém o PDP só avalia as mais relevantes, com base no elemento *Target* da política (IBM DEVELOPER WORKS, 2004).

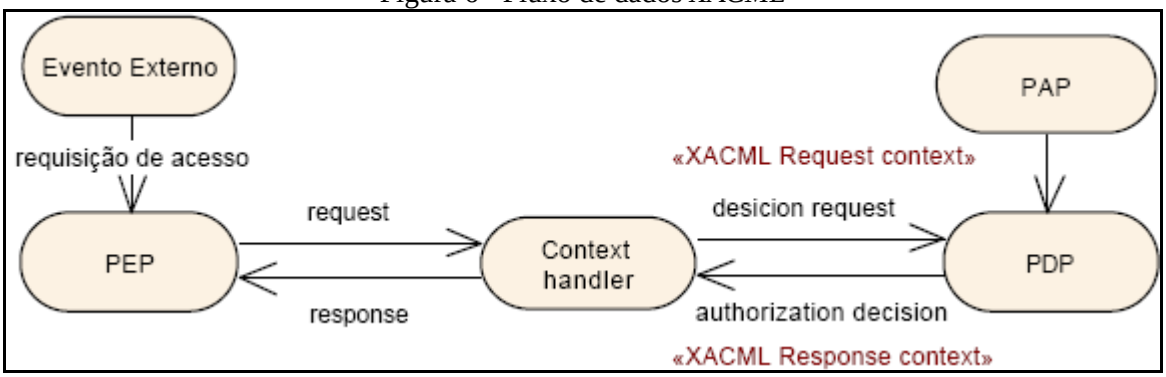
O PDP também pode invocar o PIP para recuperar os valores de atributos relacionados ao sujeito, o recurso, ou o ambiente. PDP envia a decisão de autorização para o PEP. O PEP cumpre as obrigações e, com base na decisão de autorização enviado pelo PDP permite ou nega o acesso ao recurso (IBM DEVELOPER WORKS, 2004).

De acordo com Toktar (2003, p. 57) existe um componente intermediário entre o PEP e o PDP chamado *context handler*. A partir do momento em que um evento externo invoca

um PEP gerando uma requisição de acesso, é enviado ao *context handler*, contendo dados necessários para uma avaliação de política. O *context handler* gera uma requisição no formato XACML *Request context*, chamada *decision request*, e disponibiliza ao PDP para avaliação.

De acordo com o apresentado na Figura 6, Toktar (2003, p. 58) especifica fluxo de comunicação do *context handler*. O *context handler* é uma entidade do sistema responsável para converter os dados requisitados pelo PEP em forma nativa para o modelo XACML *Request context*, possibilitando o processamento do PDP. Após a avaliação da requisição o PDP gera um XACML *Response context* que novamente é encaminhado para o *context handler* para ser convertido em um formato nativo para ser encaminhando ao PEP.

Figura 6 - Fluxo de dados *XACML*



Fonte: Toktar (2003, p. 58).

2.2.2.1 Algoritmos de combinação

Uma política de controle de acesso poderá conter vários elementos *PolicySet*, *Policy* e *Rule* e cada um destes elementos poderá conter seu elemento *Target*. Com vários elementos *Target* contidos em uma política de controle de acesso é necessária a utilização de algoritmos de combinação para tomar a decisão de quais políticas e regras serão avaliadas com base nos dados obtidos. Os algoritmos de combinação são divididos em algoritmos de combinação de políticas: *permit-overrides; deny-overrides; first-applicable* e *only-one-applicable-policy* e algoritmos de combinação de regras: *permit-overrides; deny-overrides* e *first-applicable* (ROSSET, 2004, p. 52).

De acordo com Rosset (2004, p. 52-53) “[...] algoritmos de combinação estão previstos na própria especificação do padrão XACML. A utilização de outros algoritmos de combinação é possível e suportado”.

Os algoritmos citados de acordo com Gouveia (2011) são:

a) permit-overrides – a decisão retornará "*Permit*" se pelo menos um elemento permitir autorização;
e) deny-overrides – a decisão retornará "*Deny*" se pelo menos um elemento negar a autorização;
f) first-applicable – neste caso a ordem dos elementos tem consequência nos resultados, pois o primeiro elemento aplicável a autorização será avaliado e o resultado final da decisão é o resultado deste elemento;
g) only-one-applicable – só retornará resultado caso haja apenas um elemento a ser avaliado e avaliação deste será o resultado. Caso haja mais que um elemento a ser avaliado retornará o resultado indeterminado ("*Indeterminate*").

2.2.2.2 Atributos e valores de atributos

Para que a avaliação das condições ou alvos das políticas é necessário comparar os valores dos atributos da política com os valores dos atributos do pedido (*request*). Estes atributos são características que identificam as particularidades do sujeito, recurso, ação e ambiente onde é feito a solicitação de acesso (GOUVEIA, 2011, p. 08).

Um pedido de acesso tem seu atributos comparados com os dados das políticas através de dois mecanismos: *AttributeSelector* e *AttributeDesignator.* O elemento AttributeDesignator é responsável por procurar no XML do pedido (request) os atributos indicados pois identifica um atributo e seu tipo de dado. São definidos quatro tipos de *AtributeDesignator*, para sujeitos (*Subject*), para recursos (*Resource*), para ações (*Action*) e para ambientes (*Environment*). Segundo Gouveia (2011, p. 08) "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"". Vale a pena ressaltar que o *AttributeDesignator* poderá retornar mais de um valor e deverá ser tratado por funções de filtro definidas no XACML por *Apply*[2].

O elemento *AtributeSelector* especifica um caminho para obter um valor de atributo externo necessário para validar uma regra atributo caso o atributo desejado não faça parte do *request*. Tal como o *AtributeDesignator o AtributeSelector* também pode retornar um

conjunto de valores. Para esse conjunto de valores se dá o nome de *Bag* que poderá conter apenas valores de um único tipo (GOUVEIA, 2011, p. 09).

Outro elemento utilizado é o *AttributeValue*. Este elemento é definido como valor esperado das políticas para que estas sejam aplicadas.

De acordo com trecho de política exemplificada naFigura 7, pode-se visualizar os elementos *AttributeDesignator* e *AttributeValue* do elemento *Resource* da *Target* (ROSSET, 2004, p. 55).

Figura 7 - Trecho de uma política *XACML*

```
01: <Policy PolicyId="SamplePolicy"
02:       RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:permit-overrides">
03:   <Target>
04:     <Subjects>
05:       <AnySubject/>
06:     </Subjects>
07:     <Resources>
08:       <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
09:         <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">SampleServer</AttributeValue>
10:         <ResourceAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema#string"
11:                                      AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"/>
12:       </ResourceMatch>
13:     </Resources>
```

Fonte: Rosset (2004, p. 55).

2.2.2.3 Funções: Elemento *Match*

O XACML utiliza funções para comparar os atributos das políticas com os atributos do pedido. Estas funções são conhecidas pelo elemento *Match* (comparação). Estes elementos têm como argumentos dois atributos: um *AttributeValue* juntamente com um *AttributeDesignator* ou Attribute*Selector* (GOUVEIA, 2011, p. 09).

A Figura 8 apresenta um trecho de política XACML que contém uma função de comparação de texto (`String`) para um elemento *Action.*

---

[2] *Apply*: função retorno de um ou mais elementos e é identificada pelo atributo *FunctionId*. (GOUVEIA, 2011).

Figura 8 - Trecho de uma política *XACML*

```
28:     <Actions>
29:       <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
30:         <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">login</AttributeValue>
31:         <ActionAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema#string"
32:                                    AttributeId="ServerAction"/>
33:       </ActionMatch>
34:     </Actions>
```

Fonte: Rosset (2004, p. 55).

2.3 TRABALHOS CORRELATOS

Como trabalhos correlatos pode-se citar Souza (2010), Lima (2008) e Hu et al. (2007).

2.3.1 Controle de Acesso para Sistemas Distribuídos

O trabalho de Souza (2010) trata da especificação e implementação de uma estrutura de controle de acesso a recursos de sistemas distribuídos utilizando política de papéis. A estrutura é denominada *Distributed Role Based Access Control* (DRBAC).

O DRBAC foi obtido a partir da expansão da arquitetura de referência de ferramentas de controle de acesso e a especificação do modelo foi desenvolvida a partir da especificação padronizada pela *National Institute of Standarts and Technology* (NIST) (SOUZA, 2010, p. 06). Esta estrutura averigua a legalidade de operações independente de sua localização no sistema operacional distribuído a partir de requisições. Além disso, é capaz de responder as requisições de operações mesmo com o aumento da quantidade de usuários que acessam o sistema e serviços providos (SOUZA, 2010, p. 12).

No decorrer do trabalho três modelos de políticas de controle de acesso foram descritas e analisadas: Política de Controle de Acesso Discricionária (PCAD), Política de Controle de Acesso Obrigatória (PCAO) e Política de Controle de Acesso Baseada em Papéis (PCABP).

A PCAD é entendida pela capacidade dos usuários permitir ou negar permissões a terceiros ou a grupos. Esta especificação seria impossível de ser controlada em sistemas distribuídos pela quantidade de usuários e grupos que estes sistemas tendem a possuir e pelo controle de conflito de políticas de acesso.

A PCAO foi desenvolvida com o intuito de proteger informações sem prever os elementos de um ambiente distribuído.

Já a PCABP mesmo não sendo desenvolvida propriamente para sistemas distribuídos consegue atender as deficiências das outras políticas citadas acima.

> Ao se considerar que no *Role Based Access Control* – RBAC as permissões são garantidas a papéis, e que apenas a entidade administrativa pode prover acesso a esses papéis, é possível anuir que o número de usuários do sistema não deve impactar na aplicação coercitiva da política de controle de acesso. Além disso, o RBAC enxerga os recursos do sistema como objetos, sem impor qualquer tipo de limitação a especificação destes. (SOUZA, 2010, p. 58).

Para encontrar uma arquitetura e um modelo de controle de acesso que permitisse a construção de uma ferramenta para aplicar as políticas de controle de acesso de forma coercitiva e adequada a um sistema distribuído foi construído a estrutura DRBAC. O modelo *Role Based Access Control* (RBAC) foi escolhido, pois consegue tratar número ilimitado de usuários e escalabilidade, pela facilidade de gerenciamento das permissões concedidas e pela facilidade de se implantar políticas de segurança de alta granularidade (SOUZA, 2010, p. 90).

2.3.2 Sistema Gerenciador de Políticas de Controle de Acesso (SGPCA)

O trabalho de Lima (2008) trata-se de um sistema de gerenciamento de políticas de controle de acesso, bem como o controle de conflitos a políticas cadastradas, contando com algoritmos que o gerenciam automaticamente. O sistema possibilita a criação e edição de políticas de controle de acesso no âmbito de organizações da saúde, sem ter a necessidade de conhecimento específico na linguagem de codificação das políticas.

O trabalho de Lima (2008) avaliou diferentes tecnologias e tipos de linguagens de política para saber qual a linguagem que se adaptaria melhor a determinadas situações. Uma das linguagens observadas foi o XACML que é baseada em XML (linguagem extensível), amplamente utilizada e compatível com diversas linguagens de programação facilitando assim sua especificação e implementação. A linguagem XACML foi utilizada por conter grande quantidade de referencial teórico e por ser destinada a modelos de controle de acesso (LIMA, 2008, p. 37).

> O XACML padronizado pela OASIS tem por objetivo descrever em XML, políticas de controle de acesso. As regras definidas em XACML permitem um aprimorado controle de acesso, onde é possível determinar o modo de acesso, por exemplo, um documento somente pode ser acessado se provar que possui as ações necessárias para tal. (LIMA, 2008, p. 34-35).

A implementação feita utiliza uma *Application Program Interface* (API) denominada `sun.xacml` para leitura das políticas por programas Java, possibilitando assim a implementação de algoritmos de verificação de conflitos, além de apontar o nome da política conflitante e regras que estão conflitando.

A principal qualidade do SGPCA é o fato de possibilitar a geração de políticas de forma simples e correta. O SGPCA efetua a verificação de conflitos no momento em que as políticas de controle de acesso são criadas, além de efetuar a edição de políticas de controle de acesso sem uso de linguagem de política. A maneira utilizada para verificação de conflito e edição de políticas de controle de acesso possibilita maximizar a produtividade na geração e edição de políticas livres de conflitos (LIMA, 2008, p. 86).

Lima (2008) efetua o controle de conflito e criação de políticas de segurança de acesso no âmbito de organizações da saúde utilizando a norma XACML, fornecendo maior segurança em sistemas distribuídos e em controle de acesso.

2.3.3 *Conformance Checking of Access Control Policies Specified in XACML*

Atualmente garantir que a especificação de uma política esteja exata ou conforme contexto especificado é tarefa muito importante e um desafio a ser resolvido. As políticas de controle tornam-se mais complexas e são usadas para gerenciar uma grande quantidade de informações organizadas em estruturas sofisticadas e que requerem segurança (HU et al. 2007).

Para garantir esta exatidão, estas especificações de política de controle de acesso devem ser submetidas a rigorosas verificações e validações.

O trabalho de Hu et al. (2007) trata-se de uma abordagem para a realização de verificação de conformidade de políticas de controle de acesso especificado no XACML, com base em verificações e testes a partir de ferramentas de testes para as políticas XACML.

Por ser muito genérico e flexível, o XACML apresenta divergências entre o quê se pretende especificar e o que é realmente especificado através das políticas.

> Muitas vezes, as propriedades comuns para políticas específicas de controle de acesso não pode ser satisfeita quando essas políticas são especificadas em XACML, fazendo com que a discrepância entre o que os autores de políticas pretendem especificar e o que as políticas especificadas XACML realmente refletem. (HU et al. 2007).

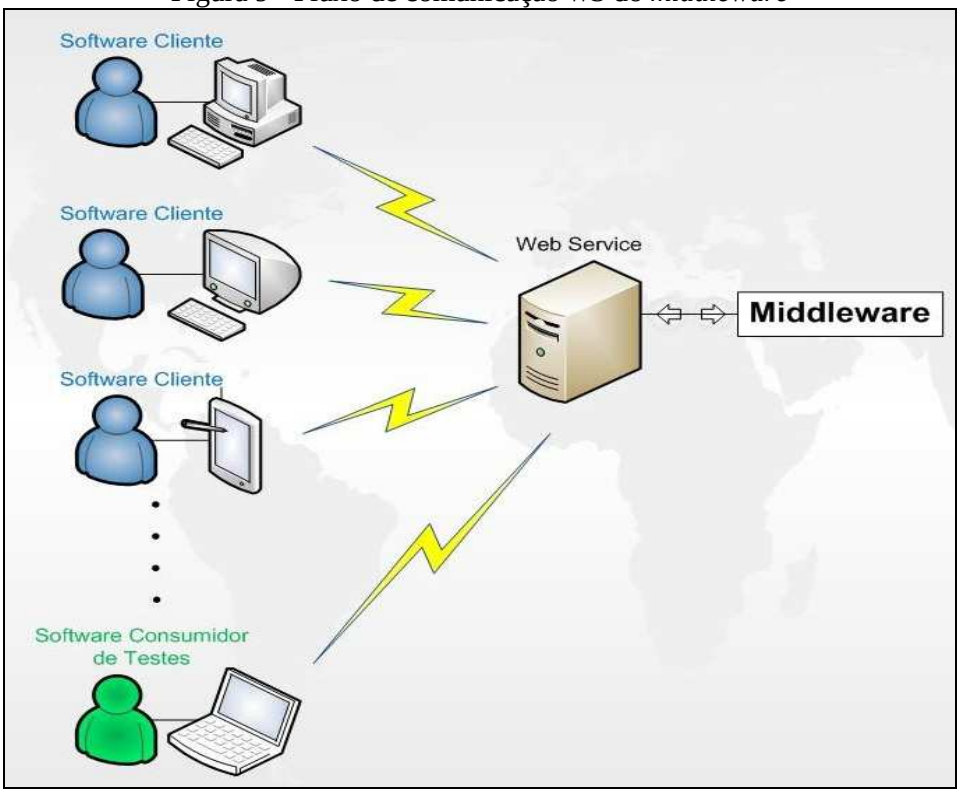
A flexibilidade e a expressividade do XACML acarretam na complexidade e difícil compreensão da linguagem, além disso, não existe função eficiente que efetue a verificação de conformidade e integridade da política especificada (ou o seu modelo) em relação à consistência semântica de uma política de controle de acesso que se pretendia especificar (HU et al., 2007).

No trabalho foi proposto sintetizar propriedades concretas e genéricas da política no âmbito da verificação e, em seguida, alimentar uma ferramenta de verificação de políticas ou ferramentas de teste de política com as propriedades para verificação de conformidades (HU et al., 02007).

3 DESENVOLVIMENTO

Este capítulo apresenta as etapas do desenvolvimento do *middleware*. São abordados os principais requisitos definidos para o desenvolvimento do trabalho, a especificação do *middleware* desenvolvido juntamente com seu software consumidor de testes (software implementado para testar as funções disponibilizadas pelo *middleware*) e a implementação do *middleware* e do software consumidor de testes, mencionando as técnicas e ferramentas utilizadas. Também são comentadas questões sobre a operacionalidade da implementação e os resultados obtidos.

Figura 9 - Fluxo de comunicação *WS* do *middleware*



De acordo com a Figura 9, o *middleware* pode ser consumido por vários softwares clientes, que a partir da comunicação com o *web service* podem consumir os recursos disponíveis no *middleware* através de chamadas remotas.. Conforme apresentado na figura acima, um dos softwares clientes desenvolvido neste trabalho é o software consumidor de testes.

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

Os requisitos da ferramenta estão divididos como requisitos funcionais (RF) e requisitos não funcionais (RNF), os quais são:

a) o *middleware* deverá permitir criar, editar e excluir políticas de controle (RF);
b) o *middleware* deve permitir entrada de dados (solicitações de acesso - *request*) (RF);
c) o *middleware* deve possibilitar a designação de resultados (*response*), a partir de decisões sob as políticas de controle de acesso (RF);
d) o *middleware* disponibilizará de uma interface *web service* para acesso a seus recursos (RF);
e) o *middleware* deverá ser implementado utilizando os conceitos de orientação a objetos (RNF);
f) o *middleware* deverá ser implementado na linguagem de programação Java (RNF);
g) o *middleware* deverá utilizar a norma (padrão de linguagem) XACML para geração de políticas de acesso, pedidos e respostas (RNF);
h) o *middleware* deverá ser implementado utilizando os ambientes de desenvolvimento Netbeans IDE 6.8 e/ou Eclipse 3.5.2 ou superiores (RNF);
i) o *middleware* utilizará bibliotecas para gerir a criptografia das políticas cadastradas, garantindo assim a segurança dos dados (RNF).

3.2 ESPECIFICAÇÃO

O *middleware* implementado e o software consumidor de testes foram especificados utilizando diagramas da *Unified Modeling Language* (UML) usando orientação a objetos. Foi utilizada a ferramenta *Enterprise Architect* (EA) (ENTERPRISE ARCHITECT, 2008) para o desenvolvimento dos diagramas de caso de uso, de classes e de atividades. Na seção 3.2.1 é apresentado a especificação da implementação do *middleware* e na seção 3.2.2 é apresentada a especificação do software consumidor de testes implementado para testar as funcionalidade do *middleware* proposto.

3.2.1 Especificação do *Middleware*

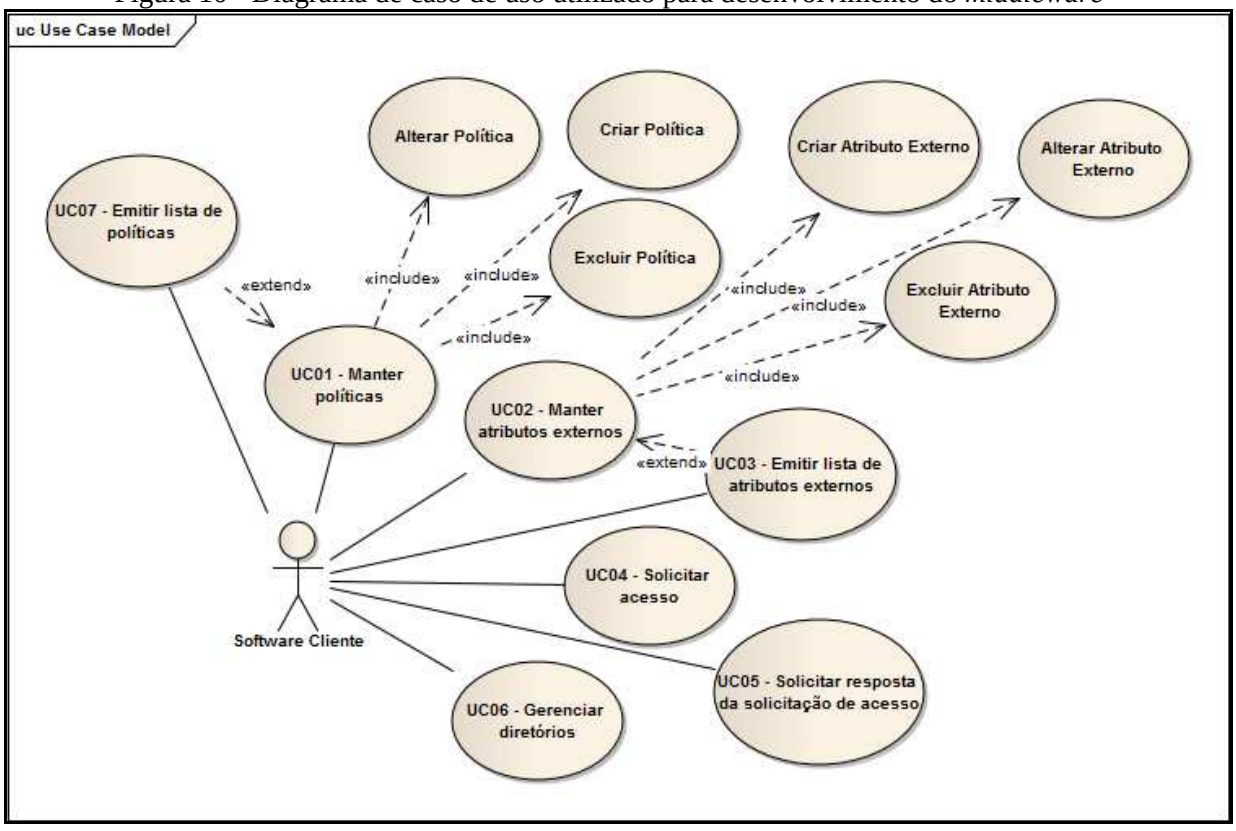
A especificação do *middleware* implementado está dividida em três seções: diagrama de casos de uso, diagrama de classes e diagrama de atividades.

3.2.1.1 Diagrama de casos de uso

A apresentação do diagrama de casos de uso define os requisitos funcionais do *middleware* implementado. Segue abaixo conforme Figura 10 o diagrama de casos de uso que contém sete casos e um ator envolvido (software cliente).

O ator envolvido é considerado qualquer software de aplicação cliente que desejar utilizar as funcionalidades do *middleware*.

Figura 10 - Diagrama de caso de uso utilizado para desenvolvimento do *middleware*



A seguir são demonstrados os detalhamentos dos três principais casos de uso identificados no diagrama da Figura 10: `manter políticas` (criação), `solicitar acesso` e `solicitar resposta de solicitação de acesso`. O detalhamento do caso de uso é

feito a partir da sequência de chamada de métodos que deve ser efetuada ao *middleware*. O detalhamento dos demais casos encontra-se no Apêndice A. No Apêndice E são explicados todos os métodos (funcionalidades e parâmetros) descritos abaixo na descrição dos casos de usos.

No Quadro 1 é demonstrado o detalhamento da criação de uma política a partir de chamadas feitas ao *middleware* pelo software cliente.

Quadro 1 - Caso de Uso 01 – Manter políticas

UC01 – Manter políticas – Criar política.	
Pré-condição	01) O software cliente deve ter uma sessão ativa, ou seja, efetuar a chamada do método `getSession()`.
Cenário principal	01) O software cliente executa a chamada de função `createPolicy()`; 02) O *middleware* instancia um objeto do tipo `Policy` para manipulação; 03) O software cliente executa a chamada de função `createTargetSubject()`; 04) O *middleware* cria um sujeito para o alvo; 05) O software cliente executa a chamada de função `createTargetResource()`; 06) O *middleware* cria um recurso para o alvo; 07) O software cliente executa a chamada de função `createTargetAction()`; 08) O *middleware* cria uma ação para o alvo; 09) O software cliente executa a chamada de função `addTargetInPolicy()`; 10) O *middleware* cria uma `Target` (alvo) com os dados de sujeito, ação e recurso especificados nos métodos dos passos 03, 05 e 07 e adiciona a `Target` a política. 11) O software cliente executa a chamada de função `createRule()`; 12) O *middleware* instancia um objeto do tipo `Rule` para manipulação; 13) O software cliente executa as chamadas de função dos passos 03, 05 e 07; 14) O *middleware* efetua os passos 04, 06 e 08; 15) O software cliente executa a chamada de função `addTargetInRule()`; 16) O *middleware* cria uma `Target` (alvo) com os dados de sujeito, ação e recurso especificados nos métodos dos passos 03, 05 e 07 e adiciona a `Target` a regra; 17) O software cliente executa a chamada de função `createConditiontRule()`; 18) O *middleware* instancia um objeto do tipo `Apply` (condição) para manipulação; 19) O software cliente executa a chamada de função `addConditionInRule()`; 20) O *middleware* finaliza a criação da condição e a adiciona a regra que está em criação. 21) O software cliente executa a chamada de função `addRuleInPolicy()`; 22) O *middleware* finaliza a criação da regra e a adiciona a política que esta em criação. 23) O software cliente executa a chamada de função `finalizaCreatePolicy()`; 24) O *middleware* finaliza a criação da política, gera XML de acordo com a sintaxe XACML, criptografa o XML gerado e salva a política em arquivo ‘”Identificacao_Politica.xmlEncrypted” no diretório padrão ou configurado.
Cenário Alternativo 1	Nos passos 03 e 05 o software cliente pode adicionar atributos complementares ao sujeito ou recurso do alvo: 03.1) O software cliente executa a chamada de função `addAttributesTarget()`; 03.2) O *middleware* adiciona um atributo complementar ao sujeito do alvo; 05.1) O software cliente executa a chamada de função `addAttributesTarget()`; 05.2) O *middleware* adiciona um atributo complementar ao recurso do alvo;
Cenário Alternativo 2	No passo 17 o software cliente poderá adicionar nova condição. Com isso a condição assumirá duas expressões que podem ser analisadas a partir dos operadores lógicos “E” ou “OU”. 17.1) O software cliente executa novamente a chamada de função `createConditiontRule()`; 17.2) O *middleware* adiciona uma nova expressão a `Apply` criada anteriormente;
Pós-condição	O *middleware* atualiza a `PolicySet` “Política de Avaliação” de acordo com políticas cadastradas.

No Quadro 2 é demonstrado o detalhamento de uma solicitação de acesso a um recurso a partir de chamadas ao *middleware* feitas pelo software cliente.

Quadro 2 - Caso de Uso 04 – Solicitar acesso

UC04 – Solicitar acesso.	
Pré-condição	01) O software cliente deve ter uma sessão ativa, ou seja, efetuar a chamada do método `getSession()`.
Cenário principal	02) O software cliente executa a chamada de função `initRequisicao()`; 03) O *middleware* cria uma requisição de acesso a partir dos dados do sujeito, recurso, e ação parametrizados na chamada do método do passo 01;
Cenário Alternativo 1	No passo 02 o software cliente pode adicionar atributos complementares ao sujeito e recurso e também adicionar atributos de ambiente: 02.1) O software cliente executa a chamada de função `requisitaSujeitoPEP()`; 02.2) O *middleware* cria um atributo adicional ao sujeito da requisição; 02.3) O software cliente executa a chamada de função `requisitaRecursoPEP()`; 02.4) O *middleware* cria um atributo adicional ao recurso da requisição; 02.5) O software cliente executa a chamada de função `requisitaAmbientePEP()`; 02.6) O *middleware* cria um atributo de ambiente adicional na requisição;
Pós-condição	O *middleware* cria um arquivo no diretório padrão de requisições ou diretório configurado do *request* gerado - "requestmm dd hh aaaa.xml".

No Quadro 3 é demonstrado o detalhamento de uma solicitação de resposta a uma solicitação de acesso gerada a partir de chamadas ao *middleware* feitas pelo software cliente.

Quadro 3 - Caso de Uso 05 – Solicitar resposta da solicitação de acesso

UC05 – Solicitar resposta da solicitação de acesso.	
Pré-condição	01) O software cliente deve ter uma sessão ativa, ou seja, efetuar a chamada do método `getSession()`. 02) O software cliente deve ter efetuado uma solicitação de acesso a recurso;
Cenário principal	03) O software cliente executa a chamada de função `finalizaRequisicao()`; 04) O *middleware* analisa a requisição de acesso criada, verifica se existem políticas para o sujeito, recurso e ação solicitados e retorna ao software cliente uma permissão ou negação de acesso;
Cenário Alternativo 1	No passo 03 o software cliente pode solicitar maiores detalhes sobre a análise da requisição feita, ou seja, pode receber informação do recurso solicitado e o *status* da avaliação da requisição em relação às políticas existentes: 03.1) O software cliente executa a chamada de função `getJustificativaPEP()`; 03.2) O software cliente executa a chamada de função `getRecursoDecisaoPEP()`; 03.3) O *middleware* gera um texto de informações sobre o recurso solicitado e o *status* da análise da requisição e retorna ao software cliente;
Pós-condição	O *middleware* cria um arquivo no diretório padrão de respostas ou diretório configurado do *response* gerado - "responsemm dd hh aaaa.xml".

3.2.1.2 Diagrama de classes

Os diagramas de classes apresentados a seguir exibem as principais classes do *middleware* e o diagrama das classes que fazem parte da biblioteca *SunXACML* (SUN MICROSYSTEMS, 2006) utilizadas pelo *middleware* para estruturar as políticas de controle

de acesso. Para melhor entendimento, nesta seção será feita uma explanação de cada diagrama.

Segue o detalhamento das classes apresentadas no diagrama da Figura 11 - Diagrama de classes – Estrutura de classes principais do *middleware*.

a) classe `FacadeAccessControlMDW`: é a estrutura principal do *middleware*. Esta classe funciona como uma fachada para as chamadas do *WS* e contém todas as funções para garantir que os objetivos especificados neste trabalho sejam cumpridos;
b) classe `SessionObject`: responsável por gerenciar os objetos das classes `AccessPolicyMDW` e `AccessPolicyEnforcementPointMDW`. Esta classe efetua validação de sessões ativas, caso o tempo ocioso de acesso a chamadas do *WS* seja superior a 45 minutos, os objetos gerenciados serão destruídos e a sessão desativará;
c) classe `SessionTimer: thread` responsável por controlar concorrentemente com a execução do *middleware* o tempo ocioso de acesso (45 minutos);
d) classe `AccessPolicyMDW`: responsável por manter as políticas de acesso. Esta classe cria, altera e exclui políticas, gerencia alvos, regras e condições, e controla os diretórios de cadastro das políticas;
e) classe `Encrypter`: utilizada para gerir criptografia das políticas salvas. Esta classe utiliza a biblioteca `Jasypt` para criptografar e descriptografar as políticas;
f) classe `ListPolAcces`: classe estática responsável por controlar o acesso concorrentemente ao arquivos das política. Esta classe possui métodos sincronizados (`synchronized`) para exclusão, criação e acesso as políticas cadastradas;
g) classe `PolicyInformPointFinderModuleMDW`: responsável por manter os atributos externos (criação, edição e exclusão) e utilizado como módulo de pesquisa da classe `AccessPolicyDecisionPointMDW`. Esta classe herda as características da classe `AttributeFinderModule` (classe da biblioteca *SunXACML*), caso a classe `AccessPolicyDecisionPointMDW` necessite de algum atributo de sujeito ou recurso da política que não esta disponível na solicitação de acesso (*request*), esta classe será utilizada como módulo de procura;
h) classe `AccessPolicyDecisionPointMDW`: responsável por efetuar as tomadas de decisão de acesso, através dela são avaliadas as requisições de acesso recebidas

(`requests`). Esta classe tem acesso à política de avaliação e aos atributos externos (`PolicyInformPointFinderModuleMDW`);

i) classe `AccessPolicyEnforcementPointMDW`: responsável por receber os valores de atributos da requisição de acesso, repassa-los ao `ContextHandlerMDW` e após análise permitir ou negar acesso ao requisitante;

j) classe `ContextHandlerMDW`: classe responsável por criar as requisições de acesso (*request*) de acordo com a sintaxe XACML a partir dos dados recebidos do `AccessPolicyEnforcementPointMDW` e repassá-las ao `AccessPolicyDecisionPointMDW` para que sejam avaliadas. Esta classe recebe a decisão do `AccessPolicyDecisionPointMDW` e novamente traduz a resposta para ser encaminhada para ao `AccessPolicyEnforcementPointMDW`;

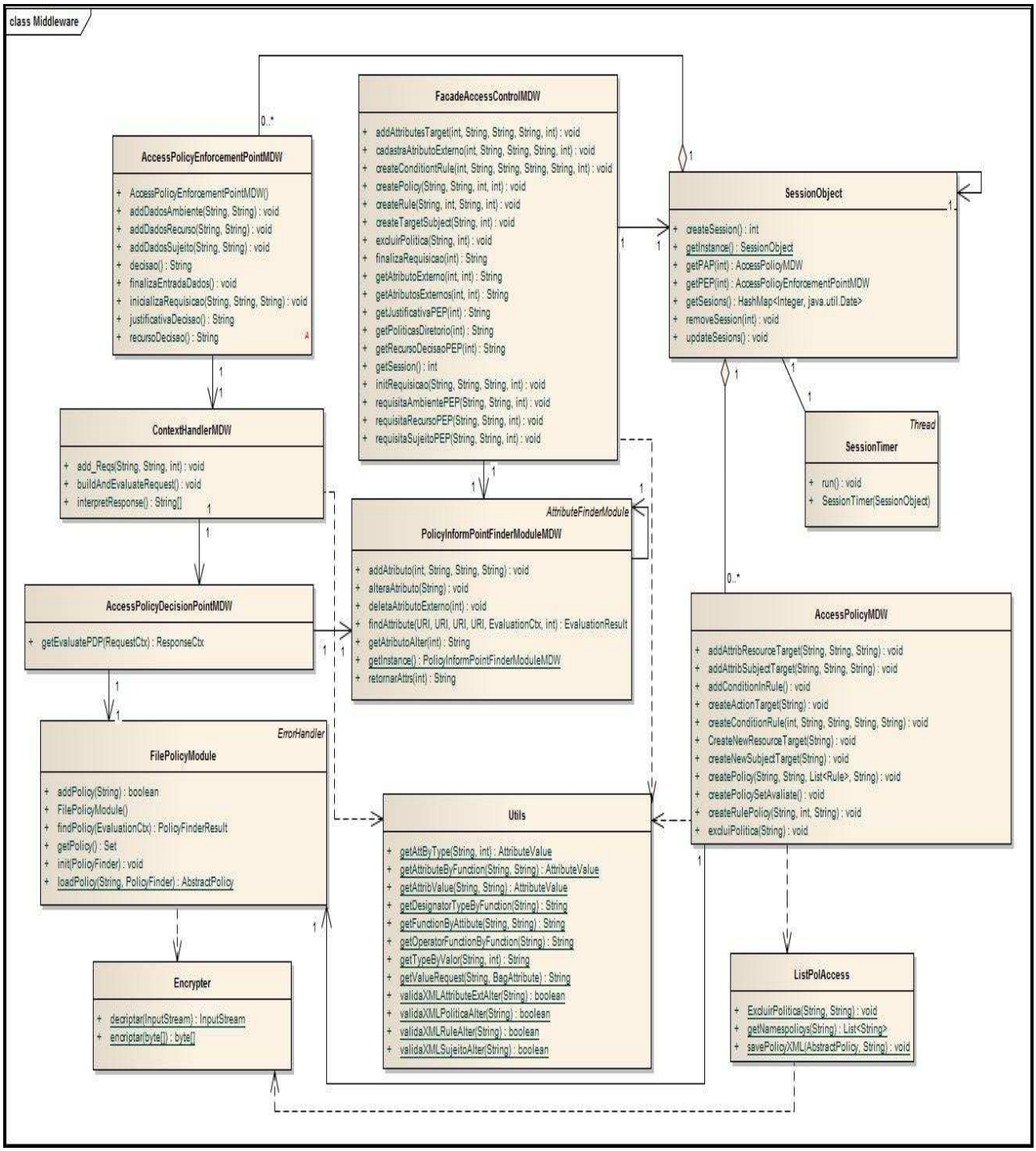
k) classe `Utils`: classe responsável pela conversão de valores da sintaxe XACML para sintaxe utilizada pelo *middleware*. Esta classe também efetua o controle de validação de sintaxe do XML utilizado no *middleware* para alteração de políticas, atributos externos, valores de regras e alvos, XML de configuração de diretórios, etc., ou seja, valida o XML recebido do software cliente. Os XML's utilizados pelo *middleware* para troca de mensagem são explicados no Apêndice B;

l) classe `FilePolicyModule`: classe responsável pela pesquisa de políticas e é utilizada para instanciar uma política (`Policy`) que esta em um documento XML. Esta classe pertence a biblioteca *SunXACML* e sofreu alterações para descriptografar as políticas no momento da instancia.

A partir da classe `FacadeAccessControlMDW` serão efetuadas todas as chamadas de métodos através de interface *WS*.

Para efetuar qualquer chamada de função do *middleware* primeiramente deve-se chamar o método `getSession()`. Este método retorna uma nova sessão (`Integer`) utilizada para controlar os objetos instanciados no *WS* para cada instancia software cliente. A partir da chamada do método citado pode-se executar as funções descritas após a Figura 11 de acordo com a necessidade de gerenciamento.

Figura 11 - Diagrama de classes – Estrutura de classes principais do *middleware*



No ambito de criação de políticas são efetuadas chamadas de função sequencialmente dos métodos `createPolicy()`, `createTargetSubject()`, `createTargetResource()`, `createTargetAction()`, `addAttributesTarget()`, `addTargetInPolicy()`, `createRule()`, `addTargetInRule()`, `createConditiontRule()`, `addConditionInRule()`, `addRuleInPolicy()` e `finalizaCreatePolicy()`.

Para alteração ou exclusão de uma política é primeiramente solicitado ao *middleware* o XML das políticas disponíveis através do método `getPoliticasDiretorio()`. A partir do retorno XML (Apêndice B) podem ser efetuadas as seguintes chamadas de funçao: `excluirPolitica()`, `getPoliticaAltera()`, `alteraPolitica()`, `getNextRegraAltera()`, `alteraRegra()`, `getNextSujeitoAltera()`, `getNextRecursoAltera()`, `getNextAcaoAltera()`, `alteraSujeito()`, `alteraRecurso()`, `alteraAcao()` e `finalizaAlterPolitica()`.

Após a criação de políticas o *middleware* pode receber solicitações de acesso a recursos e interpretar respostas através dos métodos: `initRequisicao()`, `requisitaSujeitoPEP()`, `requisitaRecursoPEP()`, `requisitaAmbientePEP()`, `finalizaRequisicao()`, `getJustificativaPEP()` e `getRecursoDecisaoPEP()`.

No caso do software cliente optar por gerir os atributos externos o *middleware* disponibiliza os métodos: `cadastraAtributoExterno()`, `getAtributosExternos()`, `getAtributoExterno()`, `alteraAtributoExterno()`, `excluiAttrExterno()`.

Na Figura 12 são apresentadas as classes da biblioteca *SunXACML* que são utilizadas na estruturação das políticas e que sofreram alterações nos seus métodos de acesso a atributos (`getters` e `setters`) para melhor manipulação dos dados a partir das classes do *middleware.*

A classe `PolicySet` constitui um conjunto de objetos `Policy`. Ela representa um conjunto de políticas e é utilizada no *middleware* implementado no momento de avaliação das políticas cadastradas. Cada vez que uma política é alterada, criada ou excluída uma `PolicySet` chamada “Política de Avaliação” é recriada com o conjunto de políticas vigentes, com o algoritmo de combinação padrão `DenyOverridesPolicyAlg` (a não ser que seja alterada no arquivo de configuração) para analise das políticas e com seu alvo com a *tag* <Any<*Nome_Elemento*>> para sujeito, recurso e ação, ou seja, com essa *tag* todas as solicitações serão avaliadas a partir da “Política de Avaliação” e os alvos a serem comparados serão somente os das políticas cadastradas e suas devidas regras;

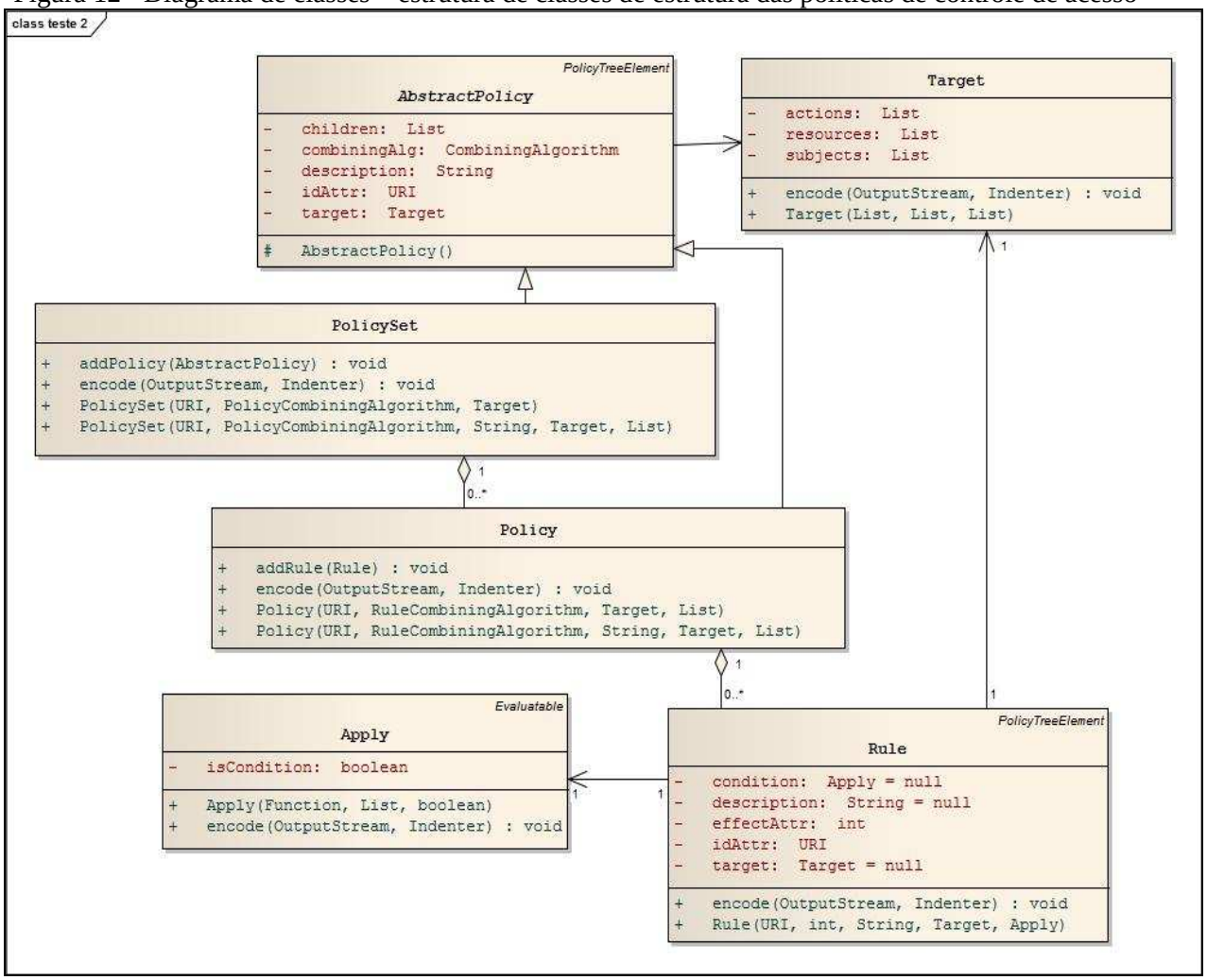
A classe `Policy` é utilizada como objeto de política. A partir da classe `Policy` são criadas as políticas internamente no *middleware*. A classe `Policy` possui um `Target` que representa o alvo das políticas, conjunto de políticas ou regras. A classe `Target` possui três listas para manipulação de sujeitos, recursos e ações;

A classe `Policy` pode conter regras, ou seja, um conjunto de objetos `Rule`. As regras são utilizadas como atributos a serem calculados num predicado que avalia se estes atributos estão dentro de um parâmetro aceitável, ou seja, se o sujeito que solicitou acesso é autorizado

ou não. Os objetos `Rule` contam com o objeto `Apply` que representa uma condição adicional a regra.

As classes especificadas até o momento contam com o método `encode()` que permite gerar o objeto instanciado em XML.

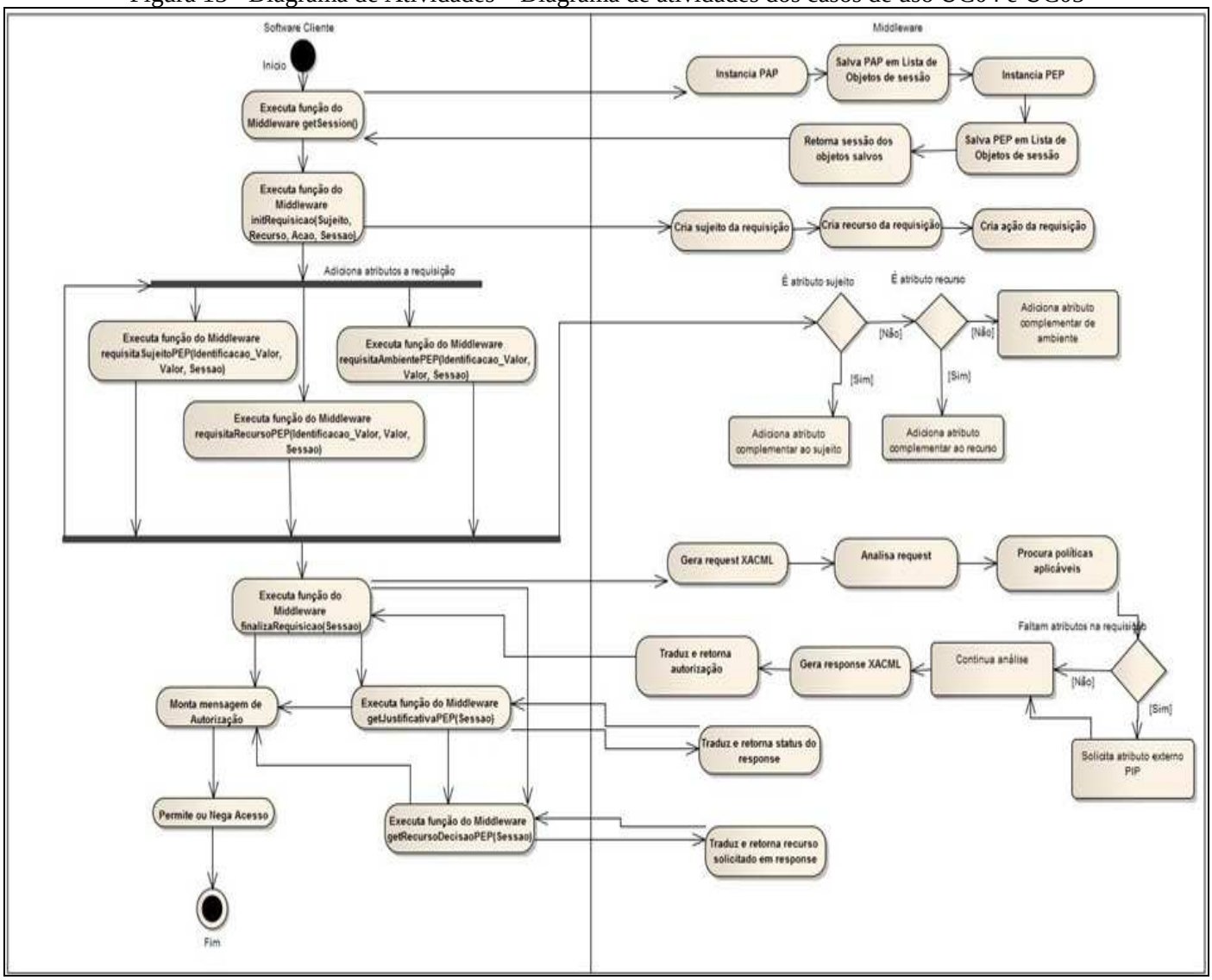
Figura 12 - Diagrama de classes – estrutura de classes de estrutura das políticas de controle de acesso



3.2.1.3 Diagrama de atividades

O diagrama de atividades apresentado a seguir exibe o fluxo de uma solicitação de acesso a um recurso a partir do software cliente e resposta de autorização.

Figura 13 - Diagrama de Atividades – Diagrama de atividades dos casos de uso UC04 e UC05



Na Figura 13 é exibido o diagrama de atividades onde o software cliente solicita acesso de um determinado sujeito a um determinado recurso para executar uma ação específica. Primeiramente o software cliente solicita uma sessão para a execução de funções do *middleware*, em seguida inicia uma requisição de acesso. O software cliente pode adicionar atributos auxiliares ao sujeito e recurso da solicitação e ainda atributos de ambiente. O *middleware* monta um request (requisição) em arquivo para ser analisado posteriormente.

O software cliente finaliza a requisição de acesso, com isso o *middleware* analisa o request gerado, gera um arquivo de response (que logo pode ser utilizado para requisição de maiores informações da análise) e retorna ao software cliente a permissão ou negação de acesso.

3.2.2 Especificação do software consumidor de testes

O software consumidor de testes foi implementado para testar e demonstrar as funcionalidades que o *middleware* desenvolvido dispõe e será disponibilizado como parte do pacote de implementação deste trabalho. A especificação do software será demonstrada a partir do diagrama de classes da Figura 14, o qual segue a explanação das classes a seguir.

A classe `FacadeAccessControlMDW (Cliente)` é uma interface para os métodos de serviços web disponíveis. Esta classe define os métodos publicados no *web service* que podem ser utilizados.

A classe `FacadeAccessControlMDWProxy` implementa a interface da classe `FacadeAccessControlMDW (Cliente)` e, portanto, age como um *proxy* de cliente invocando as chamadas para os serviços web. Esta é a classe que é instanciada na classe `ControlerCliente` para chamar os métodos *WS*.

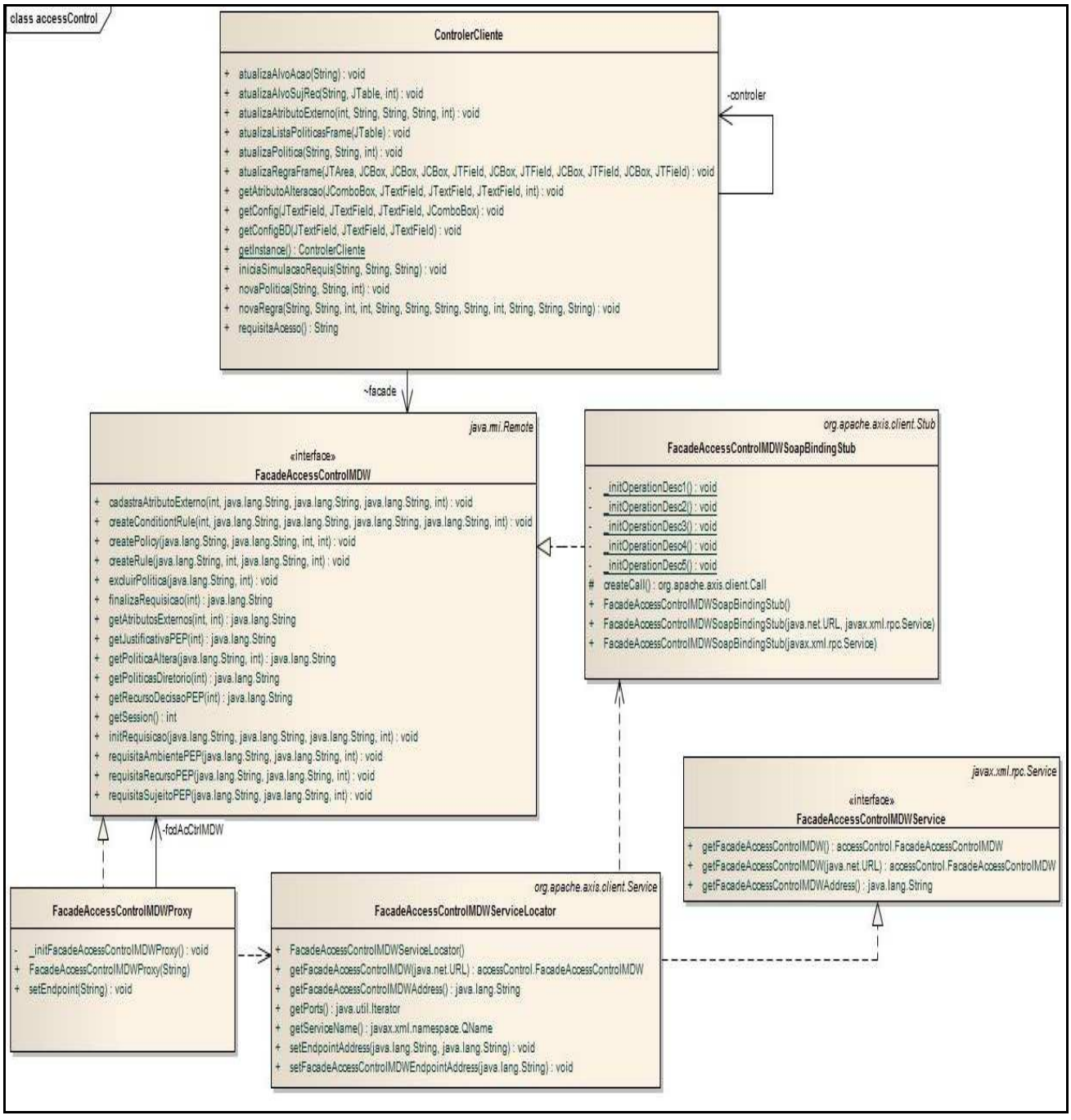
A classe `ControlerCliente` efetua a ponte entre as telas (`Frames`) do software consumidor de testes e a classe `FacadeAccessControlMDW (Cliente)`. Ela recebe as entradas de dados da interface cliente, efetua chamadas ao métodos da interface `FacadeAccessControlMDW (Cliente)` para que sejam efetuadas as chamadas remotas ao *WS* e após receber as respostas atualiza as interfaces de tela.

A classe `FacadeAccessControlMDWService` é uma interface que define os métodos do serviço de localização do *WS*, que é implementado pela interface `FacadeAccessControlMDWServiceLocator` para conectar ao ponto de extremidade do *WS* (*endPoint*).

A classe `FacadeAccessControlMDWServiceLocator` é responsável por localizar o serviço de web e conexão para o terminal, ela armazena e configura a *URL* (Uniform Resource Locator) do *WS*.

A classe `FacadeAccessControlMDWSoapBindingStub` é responsável pela troca de mensagens SOAP entre o cliente e o serviço web. Esta classe invoca as chamadas para os métodos do serviço web, ou seja, ele envia os pedidos SOAP e recebe as respostas SOAP.

Figura 14 - Diagrama de Classes – Diagrama de classes software consumidor de testes



3.3 IMPLEMENTAÇÃO

Nesta seção são apresentados os aspectos a respeito da implementação do *middleware* e sobre a implementação do software consumidor de testes, bem como as técnicas e ferramentas utilizadas. Por último, é descrita a operacionalidade da implementação onde será

demonstrado fluxo de utilização do software consumidor de testes do *middleware* e as chamadas de função utilizadas.

3.3.1 Técnicas e ferramentas utilizadas

O *middleware* e o software consumidor de testes foram desenvolvidos utilizando técnicas de orientação a objetos e o ambiente de desenvolvimento Eclipse Java *Enterprise Edition e* NetBeans IDE 6.8. Para persistência dos dados foi utilizado o banco de dados MySQL na versão 5.1 e o ambiente *MySQL Workbench* (MYSQL, 2012) para testes e execução dos comandos DDL e DML.

Para efetuar o controle de acesso a partir do XACML 1.0 o *middleware* utilizou a *API SunXACML 1.2* (SUN MICROSYSTEMS, 2006). O pacote *SunXACML*, foi projetado e desenvolvido pela empresa *Sun Microsystems*, e apresenta classes que implementam os processos de criação de requisições e respostas XACML bem como a avaliação de políticas. O pacote também apresenta classes para geração de políticas XACML.

A linguagem de programação utilizada é o JAVA e as principais bibliotecas de apoio utilizadas são: jDom-2.0.4.jar, Jasypt-1.9.0.jar e Axis1.4.jar.

O jDom-2.0.4 (JDOM, 2012) é uma biblioteca para representação Java de um documento XML. jDom fornece uma maneira de representar um documento XML para leitura, manipulação e escrita. É uma biblioteca alternativa ao *DOM* e *SAX*, embora consiga fácil integração. Esta biblioteca foi utilizada pelo *middleware* e pelo software cliente para manipulação dos XML's gerados pelo *middleware* para alteração de dados.

Apache Axis 1.4 (APACHE SOFTWARE FOUNDATION, 2013) é uma implementação Open Source cliente/servidor do SOAP ("Simple Object Access Protocol") padrão do W3C. Essencialmente é um mecanismo SOAP - um *framework* para a construção de processadores de SOAP, tais como clientes, servidores, gateways, etc. O Axis suporta WSDL, que permite gerar facilmente *stubs* para acessar serviços remotos, e também para exportar automaticamente descrições legíveis por máquina de seus serviços implantados. Este framework foi utilizado para gerar o *WS* a partir de uma classe Java, gerar o *WSDL*, processar e interpretar as mensagens SOAP e para apoiar na criação dos *stubs* de acesso do cliente.

Como servidor container do *WS*, foi utilizado o servidor Apache Tomcat 7.0.4 (APACHE SOFTWARE FOUNDATION, 2009), mais especificamente, um container de

servlets.

O Tomcat é um servidor de aplicações *Java Enterprise Edition* (JEE). Foi desenvolvido pela *Apache Software Foundation*. Ele tem a capacidade de atuar como servidor web e provê um servidor web HTTP puramente em Java.

O Jasypt-1.9.0 (JASYPT, 2011) é uma biblioteca Java *Open Source*, que permite adicionar capacidades básicas de criptografia, possui alta segurança, baseada em padrões de criptografia tanto unidirecional quanto bidirecional, pode criptografar senhas, textos e números binários. Esta biblioteca foi utilizada pelo *middleware* para criptografia e descriptografia das políticas de controle de acesso cadastradas.

3.3.2 Implementação do *Middleware*

Nesta seção é demonstrado o núcleo do *middleware* desenvolvido, explanando algumas classes com os códigos fonte das principais funções.

Para explicação dos códigos serão apresentados alguns fontes implementados para criação de uma política, uma solicitação de acesso a um recurso e o controle de sessões feito pelo *middleware* para gerenciar os objetos de cada software cliente no âmbito das chamadas *WS*.

A classe `FacadeAccessControlMDW` centraliza todas as chamadas do *middleware* e efetua as chamadas de funções aos objetos correspondentes, assim, todas as funções explanadas nas seções abaixo iniciam nesta classe.

3.3.2.1 Criação de uma política de controle de acesso

A criação de uma política de controle de acesso no *middleware* inicia na chamada do método `createPolicy()` da classe `FacadeAccessControlMDW`. A partir desta chamada a classe `AccessPolicyMDW` executa o método demonstrado no Quadro 4.

Quadro 4 - Método `createPolicy()` - Classe `AccessPolicyMDW`

```
01 public void createPolicy(String aPolicyId, String aPolicyDescription,
02                          List<Rule> aPolicyRules, String aAlgCombId)
03                          throws ExceptionGen {
04    URI policyId;
05    URI combiningAlgId;
06    RuleCombiningAlgorithm combiningAlg;
07    try {
08       policyId = new URI(aPolicyId);
09       combiningAlgId = new URI(aAlgCombId);
10       CombiningAlgFactory factory = CombiningAlgFactory.getInstance();
11
12 // Cria algoritmo de combinacao para as regras dispostas na politica
13       combiningAlg = (RuleCombiningAlgorithm)
14                      (factory.createAlgorithm(combiningAlgId));
15    } catch (URISyntaxException e) {
16             throw new ExceptionGen(EnumErros.ERRO_SINTAXE_XACML,
17           "Erro ao criar Politica - Metodo createPolicy() - PAP "+
18            e.getMessage());
19    } catch (UnknownIdentifierException e) {
20             throw new ExceptionGen(EnumErros.ERRO_SINTAXE_XACML,
21           "Erro ao criar Politica - Metodo createPolicy() - PAP "+
22          e.getMessage());
23    }
24    String description = aPolicyDescription;
25    Target tg = new Target(new ArrayList<Object>(),new ArrayList<Object>(),
26                           new ArrayList<Object>());
27       Policy policy = new Policy(policyId, combiningAlg, description, tg,
28                                  aPolicyRules);
29    policyAtual = policy;
30    }
```

De acordo com o Quadro 4, o método exemplificado instancia os valores para a identificação, algoritmo de combinação e descrição da política nas linhas 08, 09 e 24. Na linha 25 instancia um objeto `Target` com valores nulos (pois será criado posteriormente e adicionado á política) e na linha 27 cria um objeto `Policy`.

Este objeto é atribuído a um objeto global que mantém a política para ser acessada posteriormente para adição de atributos (regras e alvo).

Após a criação da política pode ser efetuada a criação de uma regra e sua adição a política vigente a partir da chamada dos métodos `createRule()` e `addRuleInPolicy()` da classe `FacadeAccessControlMDW`.

Na sequência, o Quadro 5 exemplifica o método `createRulePolicy()` que instancia os valores para a identificação, e efeito da regra e os parametriza juntamente com a descrição efetuando a criação de uma regra de acordo com a linha 15. Na instanciação do objeto Rule é parametrizado um objeto Target com seus valores também nulos, assim poderá ser criado e atribuído posteriormente.

O objeto `Rule` também é atribuído a um objeto global (de acordo com linha 20) para que possam ser adicionados suas condições e alvos posteriormente.

A regra criada pode conter condições para os atributos que serão recebidos de uma requisição de acesso.

Quadro 5 - Método `createRulePolicy()` - Classe `AccessPolicyMDW`

```
01  public void createRulePolicy(String URIId, int effect, String desc)
02                                throws ExceptionGen {
03    URI ruleId;
04    try {
05       ruleId = new URI(URIId);
06         } catch (URISyntaxException e) {
07                  throw new ExceptionGen(EnumErros.ERRO_SINTAXE_XACML,
08                         "Erro ao criar Regra – Metodo createRulePolicy() –"
09                            + " PAP "  + e.getMessage());
10          }
11    int effectRule = 1;
12    if (effect == 0 || effect == 1)
13      effectRule = effect;
14
15    Rule rulePolicy = new Rule(ruleId, effectRule, desc, new Target(
16                                  new ArrayList<Object>(),
17                                   new ArrayList<Object>(),
18                                  new ArrayList<Object>()), null);
19
20    ruleAtual = rulePolicy;
21   }
```

A seguir no Quadro 6 é especificada a criação de uma condição para a regra criada anteriormente. As condições criadas pelo *middleware* podem conter uma ou duas expressões. De acordo com condição verificada na linha 24, caso o operador da condição (parâmetro do método) estiver vazio, a condição será criada com apenas uma expressão, entretanto, se o operador contiver valores como “E” ou “OU”, a negação desta condição (linha 44) disponibilizará outra chamada ao mesmo método do *middleware* que adicionará uma segunda expressão a condição.

Como a condição criada pode conter duas expressões, o objeto `Apply` (condição) também é atribuído a um objeto global (de acordo com as linhas 42 e 73) para poder atribuir nova expressão e para ser adicionado ao objeto regra criado.

Como pode-se notar no Quadro 6, são efetuadas algumas chamadas a funções da classe `Utils`. Está classe foi criada para efetuar conversões de valores utilizado pelo *middleware* em valores XACML e inverso, e também efetua controle de sintaxe XML utilizada pelo *middleware* para comunicação com o software cliente.

Quadro 6 - Método `createConditiontRule()` - Classe `AccessPolicyMDW`

```
01 public void createConditionRule(int member, String ident, String funcao,
02                                 String valor_Comparacao, String operadorLogico)
03           throws ExceptionGen {
04  int imember = 0;
05  if (member == 0 || member == 1 ||  member == 3)
06    imember = member;
07
08  if (operadorLogico != null)
09    if (!operadorLogico.trim().equalsIgnoreCase(oper_log) &&
10        !operadorLogico.trim().equalsIgnoreCase("") )
11      oper_log = operadorLogico;
12
13  List<Object> conditionArgs = new ArrayList<Object>();
14  List<Object> applyArgs = new ArrayList<Object>();
15  Function conditionFunction = null;
16  String desigType = Utils.getTypeByValor(valor_Comparacao, imember);
17  URI designatorType = null;
18  URI designatorId = null;
19  AttributeDesignator designator = null;
20  Apply apply = null;
21  FunctionFactory factory = FunctionFactory.getConditionInstance();
22  Function applyFunction;
23
24   if (oper_log.trim().equalsIgnoreCase("")
25       && (apply1Oper == null && apply2Oper == null)) {
26
27      factory = FunctionFactory.getGeneralInstance();
28      applyFunction = factory.createFunction(Utils
29                              .getFunctionApplyAttrDesignator(desigType));
30      conditionFunction = factory.createFunction(Utils
31                      .getFunctionXACMLByTypeAndOperateFunc(desigType,funcao));
32      designatorType = new URI(desigType);//
33      designatorId = new URI(ident);
34      designator = new AttributeDesignator(imember, designatorType,
35                                           designatorId, false, null);
36      applyArgs.add(designator);
37      apply = new Apply(applyFunction, applyArgs, false);
38      conditionArgs.add(apply);
39      AttributeValue value = Utils.getAttribValue(desigType,
40                                                  valor_Comparacao);
41      conditionArgs.add(value);
42      conditionrule = new Apply(conditionFunction, conditionArgs,true);
43
44    } else {
45       factory = FunctionFactory.getGeneralInstance();
46       applyFunction = factory.createFunction(Utils
47                                 .getFunctionApplyAttrDesignator(desigType));
48       conditionFunction = factory.createFunction(Utils
49                       .getFunctionXACMLByTypeAndOperateFunc(desigType,funcao));
50
51       designatorType = new URI(desigType);
52       designatorId = new URI(ident);
53       designator = new AttributeDesignator(imember, designatorType,
54                                            designatorId, false, null);
55         applyArgs.add(designator);
56         apply = new Apply(applyFunction, applyArgs, false);
57         conditionArgs.add(apply);
58       AttributeValue value = Utils.getAttribValue(
59                       desigType,valor_Comparacao);
60        conditionArgs.add(value);
61        if (apply1Oper == null)
62          apply1Oper = new Apply(conditionFunction, conditionArgs,false);
63       else if (apply2Oper == null)
64                  apply2Oper = new Apply(conditionFunction, conditionArgs,
65                                         false);
66    }
67    if (apply1Oper != null && apply2Oper != null) {
68    conditionFunction = factory.createFunction(Utils
69                                              .getOperadorLogico(oper_log));
70    conditionArgs.clear();
71    conditionArgs.add(apply1Oper);
72    conditionArgs.add(apply2Oper);
73    conditionrule = new Apply(conditionFunction, conditionArgs, true);
74    apply1Oper = null;
75    apply2Oper = null;
```

Abaixo no Quadro 7 segue exemplo de um método da classe `Utils` que retorna um atributo XACML a partir do valor e tipo suportado pelo *middleware*.

Quadro 7 - Método `getAttributeByFunction():AttributeValue`

```
01 public static synchronized AttributeValue getAttributeByFunction(
02                                                 String valor,
03                                                 String function) {
04  try {
05    if (function.equalsIgnoreCase("=String")
06       || function.equalsIgnoreCase("=startString")
07       || function.equalsIgnoreCase("=domainEmail")) {
08      return new StringAttribute(valor);
09
10    } else if (function.equalsIgnoreCase("=Integer")) {
11     return new IntegerAttribute(Integer.valueOf(valor));
12    } else if (function.equalsIgnoreCase("=Double")) {
13       return new DoubleAttribute(Double.valueOf(valor));
14    } else if (function.equalsIgnoreCase("=Data")) {
15      try {
16        if (valor.trim() != "") {
17        SimpleDateFormat format = new SimpleDateFormat("dd/MM/yyyy");
18          Date data = (Date) format.parse(valor);
19          return new DateAttribute(data, 0, 0);
20      } else
21          return new DateAttribute();
22           } catch (Exception e) {
23             return new DateAttribute();
24           }
25    } else if (function.equalsIgnoreCase("=Email")) {
26     return new RFC822NameAttribute(valor);
27    } else if (function.equalsIgnoreCase("=URI")) {
28     return new AnyURIAttribute(new URI(valor));
29    }
30      } catch (Exception e) {
31       return new StringAttribute(valor);
32      }
33  return new StringAttribute(valor);
34 }
```

Conforme explicado na seção 3.2.1.2 “*Diagrama de classes*” do *middleware*, a cada criação, alteração ou exclusão das políticas do diretório, uma `PolicySet` é criada ou atualizada para ser utilizada como política de avaliação. A seguir no Quadro 8 é apresentada a implementação do método `createPolicySetAvaliate()`.

Quadro 8 - Método `createPolicySetAvaliate()` - Classe `AccessPolicyMDW`

```
01 public void createPolicySetAvaliate() throws ExceptionGen {
02
03  File dir = new File(".\\PolicySetAvaliate");
04  if (!dir.exists())
05    dir.mkdirs();
06
07  URI policyId;
08  URI combiningAlgId;
09  CombiningAlgFactory factory;
10  PolicyCombiningAlgorithm combiningAlg;
11  try {
12     policyId = new URI("AvaliaPoliticas");
13     combiningAlgId = new URI(Utils.getRegraCombin(
14                     Integer.parseInt(getAlgPadraoAnalise()), "Policy"));
15     factory = CombiningAlgFactory.getInstance();
```

Quadro 8 – Continuação método `createPolicySetAvaliate()` - Classe `AccessPolicyMDW`

```
16
17      combiningAlg = (PolicyCombiningAlgorithm) (factory
18                                  .createAlgorithm(combiningAlgId));
19        } catch (URISyntaxException e) {
20        throw new ExceptionGen(EnumErros.ERRO_SINTAXE_XACML,
21        "Erro ao criar Conjunto de Politica - Metodo createPolicySet()-PAP"
22          + e.getMessage());
23      } catch (UnknownIdentifierException e) {
24         throw new ExceptionGen(EnumErros.ERRO_SINTAXE_XACML,
25         "Erro ao criar Conjunto de Politica - Metodo createPolicySet()-PAP"
26           + e.getMessage());
27      }
28   String description = "Policy Set criada para avaliacao de politicas "+
29                        "cadastradas";
30   PolicySet policySet = new PolicySet(policyId, combiningAlg,description,
31                                       new Target(null, null, null),
32                                       new ArrayList<Object>());
33   FilePolicyModule file = new FilePolicyModule();
34   List<String> politicas = ListPolAccess.getNamespolicys(getPathPolicy());
35
36   for (int i = 0; i < politicas.size(); i++) {
37     file.addPolicy(getPathPolicy() + politicas.get(i));
38   }
39   file.init(new PolicyFinder());
40   Iterator<?> it = file.getPolicy().iterator();
41   while (it.hasNext()) {
42     Policy po = (Policy) it.next();
43     policySet.addPolicy(po);
44   }
45   ListPolAccess.savePolicyXML(policySet, DIRECTORY_POLICY_AVALIATE);
46 }
```

No método mostrado no quadro acima é criado um objeto `PolicySet` com algoritmo de combinação padrão e com `Target` genérico e um objeto `FilePolicyModule` (classe *SunXACML*). O objeto `PolicySet` será a política de avaliação e nele serão adicionadas todas as políticas cadastradas. Já o objeto `FilePolicyModule` percorre todas as políticas XML cadastradas no diretório e através do método `loadPolicy()`decripta o XML das políticas (alteração feita para atender ao proposto no trabalho) e transforma-os em objetos `Policy`. Estes objetos `Policy` são adicionados a lista de políticas da `PolicySet` (linha 43).

Na linha 45 é utilizada a classe `ListPolAccess` (método `savePolicyXML()`) para encriptar o XML da `PolicySet` e salvá-lo em arquivo.

Segue abaixo no Quadro o método `loadPolicy()`da classe `FilePolicyModule` onde é demostrado a instancia e descriptografia (classe `Encrypter`) de uma politica a partir de um arquivo XML.

Quadro 9 - Método `loadPolicy()` - Classe `FilePolicyModule`

```
01 public static AbstractPolicy loadPolicy(String filename,
02           PolicyFinder finder, File schemaFile, ErrorHandler handler) {
03  try {
04    // create the factory
05    DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
06    factory.setIgnoringComments(true);
```

Quadro 9 – Continuação método `loadPolicy()` - Classe `FilePolicyModule`

```
01 public static AbstractPolicy loadPolicy(String filename,
02           PolicyFinder finder, File schemaFile, ErrorHandler handler) {
03  try {
04    // create the factory
05    DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
06    factory.setIgnoringComments(true);
07    DocumentBuilder db = null;
08
09    factory.setNamespaceAware(true);
10    if (schemaFile == null) {
11      factory.setValidating(false);
12      db = factory.newDocumentBuilder();
13     } else {
14              factory.setValidating(true);
15              factory.setAttribute(JAXP_SCHEMA_LANGUAGE, W3C_XML_SCHEMA);
16           factory.setAttribute(JAXP_SCHEMA_SOURCE, schemaFile);
17              db = factory.newDocumentBuilder();
18           db.setErrorHandler(handler);
19           }
20    File file = new File(filename);
21    InputStream inn = file.toURL().openStream();
22    Document doc = db.parse(Encrypter.decriptar(inn));
23
24    Element root = doc.getDocumentElement();
25    String name = root.getTagName();
26    if (name.equals("Policy")) {
27      return Policy.getInstance(root);
28     } else if (name.equals("PolicySet")) {
29        return PolicySet.getInstance(root, finder);
30     } else {
31        throw new Exception("Unknown root document type: " + name);
32     }
33  } catch (Exception e) {
34       if (logger.isLoggable(Level.WARNING))
35           logger.log(Level.WARNING, "Error reading policy from file "
36                                     + filename, e);}
37  return null;
38 }
```

3.3.2.2 Requisição de acesso a recurso

Para que ocorra uma requisição de acesso necessita-se de três principais valores: sujeito, recurso e ação. Quando é efetuada a chamada ao *middleware* pelo método `initRequisicao()`, estes três valores são parametrizados para iniciar a solicitação de acesso.

O *middleware* por sua vez efetua a comunicação com classe `AccessPolicyEnforcementPointMDW` para transferência dos valores recebidos.

A classe `AccessPolicyEnforcementPointMDW` recebe as requisições de acesso em um formato em que o software cliente consiga interpretar e parametrizar. Logo esta classe encaminha os pedidos de autorização a classe `ContextHandlerMDW` e ao final após todo o processo de análise responde a decisão ao software cliente.

A classe `ContextHandlerMDW` efetua o papel de transformar a solicitação expressa pelo `AccessPolicyEnforcementPointMDW` em um formato de *request XACML* e ao final interpretar o *response XACML* para retornar à classe `AccessPolicyEnforcementPointMDW`.

O processo efetuado para criar um sujeito para a solicitação está mostrado no Quadro 10.

Quadro 10 - Método `add_subject()` - Classe `ContextHandlerMDW`

```
01 private void add_subject(String subject_AttributeID,String
02                          subject_AttributeValue)throws ExceptionGen {
03  URI subjectID = null;
04  try {
05    subjectID = new URI(subject_AttributeID);
06    subject_attributes.add(new Attribute(subjectID, null, null, Utils
07                                  .getAttByType(subject_AttributeValue,
08                                 AttributeDesignator.SUBJECT_TARGET)));
09    } catch (URISyntaxException e) {
10        throw new ExceptionGen(EnumErros.ERRO_SINTAXE_XACML,"Erro ao criar"
11                        + "sujeito - Metodo add_subject()-ContextHandlerMDW "
12                        + e.getMessage());
13    }
14 }
```

Conforme mostrado na figura acima é criado um atributo do sujeito com a identificação e o valor para a requisição. Esse atributo é adicionado em uma lista de atributos do sujeito (*subject_atributes* – linha 06) e todos os atributos relacionados ao sujeito que forem adicionados posteriormente também serão alocados a lista como atributos complementares.

A mesma lógica é implementada para o recurso e seus atributos complementares, para a ação e para atributos de ambiente.

O primeiro atributo adicionado para cada uma das listas (a partir da chamada de função `initRequisicao()`) terá seu valor de identificação fixo, como se fosse o valor principal da identificação do sujeito, recurso ou ação. Logo após, as chamadas das funções `requisitaSujeitoPEP()`, `requisitaRecursoPEP()` e `requisitaAmbientePEP()` farão chamadas a função especificada no Quadro 10 normalmente, parametrizando identificação e valor que serão adicionados a lista como complementares.

Segue conforme Quadro 11 exemplo dos valores de uma requisição.

Quadro 11 - Valores de uma requisição de acesso

	Sujeito	Recurso	Ação	Ambiente
	Valores			
	Jailson	Arquivo	Excluir	X
Identificação	**Valores**			
Idade	21	X		X
Dominio	Admin	X		X
Nome	X	Tcc.txt		X
Localização	X	//C:/		X
Data ultimo acesso	X	X		21/05/2013

De acordo com quadro mostrado acima, o inicio da requisição solicitaria acesso para o sujeito `Jailson`, ao recurso `Arquivo` e com a ação `Excluir`. Posteriormente seriam adicionados valores complementares ao sujeito, recurso e atributos de ambiente, como: `idade` do sujeito igual a `21`, `domínio` do sujeito igual a `admin`, `nome` do recurso igual a `Tcc.txt`, `localização` do recurso igual a `//C:/`, e como atributo de ambiente a `data de ultimo acesso` do sujeito ao recurso igual a `21/05/2013`.

Após a passagem dos atributos de requisição, ao finalizar a requisição de acesso através da chamada de função do *middleware* `FacadeAccessControlMDW.finalizaRequisicao()` o `ContextHandlerMDW` monta o *request* de solicitação e faz a chamada da classe `AccessPolicyDecisionPointMDW` para efetuar a análise do *request* conforme Quadro 12.

No método `buildAndEvaluateRequest()` (Quadro 12) é criado um objeto `RequestCtx` (classe *SunXACML*) que representa o objeto de um *request XACML* (linha 11). Este objeto além de ser salvo em arquivo XML (linhas 17 e 20), também é parametrizado no método `getEvaluatePDP()`(linha 30).

O método `getEvaluatePDP()`da classe `AccessPolicyDecisionPointMDW` analisa o *request* parametrizado, gera um objeto `ResponseCtx` (classe *SunXACML*) – representação do objeto *response XACML*, salva `ResponseCtx` em arquivo e o retorna. O *response* é atribuído a um atributo global para ser interpretado posteriormente para o `AccessPolicyEnforcementPointMDW`.

Quadro 12 - Método `buildAndEvaluateRequest()` - Classe `ContextHandlerMDW`

```
01 public void buildAndEvaluateRequest() throws ExceptionGen {
02  AccessPolicyDecisionPointMDW PDP = null;
03  try {
04    PDP = new AccessPolicyDecisionPointMDW();
05     } catch (Exception e) {
06         throw new ExceptionGen(EnumErros.ERRO_INSTANCIA,
07                     "Erro ao instanciar PDP - Metodo ContextHandlerMDW()-
08                 + "ContextHandlerMDW "
09                 + e.getMessage());
10      }
11  RequestCtx request = new RequestCtx(getSubject_attributes(),
12                                      getResource_attributes(),
13                                      getAction_attributes(),
14                                      getEnvironment_attributes());
15  File file;
16  String nome = (new Date().toString()).replace(":", "-");
17  file = new File(getPathRequest() + "request" + nome + ".xml");
18  FileOutputStream fOutStream;
19  try {
20    fOutStream = new FileOutputStream(file);
21     } catch (FileNotFoundException e) {
22               throw new ExceptionGen(EnumErros.ERRO_ARQUIVO,
23                                  "Erro na criação de arquivo 'Request'-"
24                                   +"Metodo buildAndEvaluateRequest()-
25                                   +"ContextHandlerMDW "
26                                   + e.getMessage());
27     }
28  request.encode(fOutStream, new Indenter());
29  this.request = request;
30  this.response = PDP.getEvaluatePDP(this.request);
31  subject_attributes.clear();
32  action_attributes.clear();
33  resource_attributes.clear();
34  environment_attributes.clear();
35 }
```

A classe `AccessPolicyDecisionPointMDW` analisa o *request* a partir de todas as políticas cadastradas. Como já foi citado anteriormente a análise se dá sob a política de avaliação (`PolicySet`) e não sob todos os arquivos de políticas salvos.

No momento da avaliação o `AccessPolicyDecisionPointMDW` pode não conter todos os atributos relacionados ao sujeito ou recurso, ou seja, pode ser criado uma requisição sem os valores complementares necessários. Caso ocorra esta falta de atributos o `AccessPolicyDecisionPointMDW` requisita ao `PolicyInformPointFinderModuleMDW` o atributo necessário.

O `PolicyInformPointFinderModuleMDW` foi implementado para pesquisar os valores faltantes sob os atributos externos. Pode-se observar no Quadro 13 o código de pesquisa de um atributo não informado no *request* (requisição de acesso).

Quadro 13 - Método `findAttribute()` - Classe `PolicyInformPointFinderModule`

```
01 public synchronized EvaluationResult findAttribute(URI attributeType, URI
02             attributeId, URI issuer, URI subjectCategory, EvaluationCtx
03             context, int designatorType) {
04  if (!isSupportedDesig(designatorType))
05    return new EvaluationResult(BagAttribute.createEmptyBag(attributeType));
06  switch (designatorType) {
07    case 0:
08     ID = "urn:oasis:names:tc:xacml:1.0:subject:subject-id";
09     break;
10    default:
11     ID = "urn:oasis:names:tc:xacml:1.0:resource:resource-id";
12     break;
13  }
14  URI _ID = new URI(ID);
15  URI IDTYPE = new URI(ID_TYPE);
16
17  EvaluationResult result = null;
18    switch (designatorType) {
19      case 0:
20        result = context.getSubjectAttribute(IDTYPE, _ID,subjectCategory);
21     break;
22     default:
23       result = context.getResourceAttribute(IDTYPE, _ID,subjectCategory);
24     break;
25    }
26  if (result.indeterminate())
27    return result;
28  BagAttribute bagAttrRequest = (BagAttribute) (result.getAttributeValue());
29  if (bagAttrRequest.size() != 1) {
30    ArrayList<String> code = new ArrayList<String>();
31    code.add(Status.STATUS_PROCESSING_ERROR);
32    Status status = new Status(code, "couldn't find user's identifier");
33    return new EvaluationResult(status);
34  }
35  String valorPIP = null;
36  valorPIP = getAttribPIPDB(attributeId, attributeType,
37                          designatorType, Utils.getValueRequest(result
38                        .getAttributeValue().getType().toString(),
39                          bagAttrRequest));
40
41  if (valorPIP == null)
42    return new EvaluationResult(BagAttribute.createEmptyBag(attributeType));
43  AttributeValue str = null;
44  str = Utils.getAttribValue(attributeType.toString(), valorPIP);
45
46  Collection<AttributeValue> attrs = new HashSet<AttributeValue>();
47  attrs.add(str);
48  BagAttribute bagReturn = new BagAttribute(str.getType(), attrs);
49  return new EvaluationResult(bagReturn);
50 }
```

No quadro acima é possível notar o porquê de uma requisição de acesso conter seus primeiros valores de sujeito, recurso e ação sem suas identificações. Levando em consideração que o `PolicyInformPointFinderModuleMDW` necessita de uma identificação e um valor de pesquisa para que sejam vinculados os atributos externos ao seus devidos sujeitos e recursos, todo inicio de requisição atribui como identificação do sujeito e recurso o valor XACML implementado nas linhas 08 e 11. Com isso se a classe de pesquisa necessitar do atributo externo “identificação = `email`, valor = `jailsvs@email.com`” do sujeito do *request*,

ela saberá que o valor do sujeito que necessita dos atributos complementares está na identificação "urn:oasis:names:tc:xacml:1.0:subject:subject-id" do sujeito do *request.*

Caso o atributo faltante seja do sujeito, de acordo com a linha 20 do Quadro 13, é obtido o valor do sujeito do *request* com identificação "*urn:oasis:names:tc:xacml:1.0:subject:subject-id*" e na linha 36 é efetuado uma pesquisa em banco de dados (Apêndice C – detalhamento da tabela utilizada para consistência de dados) a partir da identificação do valor faltante (identificação do atributo externo parametrizado do método `findAttribute - attributeId`). Segue no quadro abaixo com exemplificação da pesquisa.

Quadro 14 - Valores utilizados na pesquisa de atributo externo

			Atributo externo	
Membro Sujeito/Recurso	**Identificação de pesquisa em *request***	**Valor do *request***	**Identificação**	**Valor**
`Sujeito`	"*urn:oasis:names:tc:xacml:1.0:subject:subject-id*"	`jailson`	`Idade`	`21`

A partir do Quadro 14 é evidenciado que o atributo faltante no *request* foi a “Idade” (identificação do atributo externo) do “Sujeito” (designador do atributo externo) “jailson” (valor da identificação "urn:oasis:names:tc:xacml:1.0:subject:subject-id" do *request*).

Após o retorno dos atributos necessários e da avaliação da requisição de acesso estar completa, a classe `ContextHandlerMDW` interpreta o *response* gerado em um texto que o `AccessPolicyEnforcementPointMDW` possa retornar ao software cliente em uma linguagem inteligível.

A partir das informações interpretadas pelo `ContextHandlerMDW` o `AccessPolicyEnforcementPointMDW` permite ou nega o acesso.

3.3.2.3 Controle de sessões

Foi necessária uma implementação para tratamento de objetos de sessão no *middleware* para cada instancia de software cliente. A cada acesso ao *WS* instanciava-se novos objetos `AccessPolicyMDW` e `AccessPolicyEnforcementPointMDW` fazendo com que o processo de criação de uma política, uma requisição de acesso ou qualquer outro processo que exigisse uma sequência linear de execução de métodos fosse finalizado.

Para este tratamento foi implementado a classe `SessionObject` e `SessionTimer`. A classe `SessionObject` é um *Singleton* que gerencia os objetos `AccessPolicyMDW` e `AccessPolicyEnforcementPointMDW` de cada sessão a partir de duas listas de objetos e a classe `SessionTimer` é uma classe que herda da classe `Thread` e que executa paralelamente com a execução do *middleware* verificando o tempo de ociosidade de cada sessão de objetos.

No contrutor da classe `FacadeAccessControlMDW` é instanciada a classe `SessionObject` (Quadro 15). Como esta classe possui uma instancia única (*Singleton*) a cada execução do *WS* os objetos de sessão da classe continuarão os mesmos.

Quadro 15 - Construtor e método `getInstance()` - Classe `SessionObject`

```
01 public class SessionObject {
02  private static SessionObject sessionOBJ = null;
03  private int sessoesCount = 0;
04  HashMap<Integer, AccessPolicyEnforcementPointMDW> PEPs = null;
05  HashMap<Integer, AccessPolicyMDW> PAPs = null;
06  HashMap<Integer, java.util.Date> ultAcesso = null;
07
08  private SessionObject() throws ExceptionGen {
09    PAPs = new HashMap<Integer, AccessPolicyMDW>();
10    PEPs =  new HashMap<Integer, AccessPolicyEnforcementPointMDW>();
11    ultAcesso = new HashMap<Integer, java.util.Date>();
12    new SessionTimer(this).start();
13  }
14
15  public synchronized static SessionObject getInstance()
16    throws ExceptionGen{
17    if(sessionOBJ == null)
18      sessionOBJ =  new SessionObject();
19      return sessionOBJ;
20  }
```

Pode-se observar no construtor da classe da figura acima que são criadas três estruturas de dados `HashMap<Sessao:Integer,Objeto>`: objetos PAP, PEP e horário do último acesso. As três estruturas de dados possuem como chave a sessão do software cliente, ou seja, valor de sessão solicitado ao *middleware* através do método `getSession()` que efetua a chamada ao método interno `createSession()` (Quadro 16).

Quadro 16 - Método `createSession()` - Classe `SessionObject`

```
01 public synchronized int createSession() throws ExceptionGen{
02  PAPs.put(sessoesCount, new AccessPolicyMDW());
03  PEPs.put(sessoesCount, new AccessPolicyEnforcementPointMDW());
04  ultAcesso.put(sessoesCount, new java.util.Date());
05  return sessoesCount++;
06 }
```

A data e hora do último acesso é atualizada de acordo com as chamadas efetuadas ao *middleware* que manipulam os objetos que estão nas estruturas de dados, de acordo com o Quadro 17 mostrado a seguir.

Quadro 17 - Método `getPAP()` e `getPEP()` - Classe `SessionObject`

```
01 public synchronized AccessPolicyMDW getPAP(int sessao)
02     throws ExceptionGen {
03  validateSession(sessao);
04  ultAcesso.remove(sessao);
05  ultAcesso.put(sessao, new java.util.Date());
06  return PAPs.get(sessao);
07 }
08
09 public synchronized AccessPolicyEnforcementPointMDW getPEP(int sessao)
10  throws ExceptionGen {
11  validateSession(sessao);
12  ultAcesso.remove(sessao);
13  ultAcesso.put(sessao, new java.util.Date());
14  return PEPs.get(sessao);
15 }
```

Juntamente no construtor da classe `SessionObject` (Quadro 15), é instanciada e iniciada a `Thread SessionTimer`. Esta `Thread` executa o controle de tempo ocioso a cada 45 minutos efetuando chamadas ao método `updateSessions()` da classe `SessionObject` (Quadro 18).

Quadro 18 - Método `updateSessions()` - Classe `SessionObject`

```
01 public synchronized void updateSesions() throws ExceptionGen {
02  ArrayList<Integer> sessions = new ArrayList<Integer>();
03  for (Integer sess : ultAcesso.keySet()) {
04       java.util.Date data = ultAcesso.get(sess);
05    if (new java.util.Date().getMinutes() - data.getMinutes() >= 45) //45min
06      sessions.add(sess);
07    }
08  for (Integer i : sessions){
09    removeSession(i);
10    }
11 }
```

A partir do quadro apresentado acima pode-se observar que qualquer sessão (representado como chave das estruturas de dados) que estiver com seu último acesso superior ou igual a 45 minutos será removida.

3.3.3 Implementação da interface *WS* – Servidor e Cliente

A partir da implementação da regra de negócio do *middleware*, através de uma classe principal (no caso utilizado um *Facade* - `FacadeAccessControlMDW`) foi possível gerar o módulo servidor e o módulo de cliente de acesso através do *framework* Axis (APACHE SOFTWARE FOUNDATION, 2013).

Ao gerar o *WS*, o *framework* disponibilizou um processador interno de *request* e *response* do protocolo SOAP, efetuando todo o tratamento de comunicação cliente-servidor

para as chamadas de procedimentos remotos por meio de parâmetros XML sobre o protocolo HTTP.

O servidor *web service* gerado pelo *framework* descreve seu próprio arquivo WSDL como segue na quadro abaixo. O WSDL é usado por qualquer agente que queira ter acesso ao *WS*, ou seja, é sua interface, encontrando no mesmo, todos os métodos, parâmetros e respostas definidas.

Quadro 19 - *WSDL*

```
<?xml version="1.0" encoding="UTF-8"?>
- <wsdl:definitions xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
                    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:tns1="http://utils"
xmlns:intf="http://accessControl" xmlns:impl="http://accessControl"
xmlns:apachesoap="http://xml.apache.org/xml-soap" targetNamespace="http://accessControl">
    <!--WSDL created by Apache Axis version: 1.4 Built on Apr 22, 2006 (06:55:48 PDT)-->
- <wsdl:types>
  - <schema targetNamespace="http://accessControl" xmlns="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
      <import namespace="http://utils"/>
    - <element name="createRule">
      - <complexType>
        - <sequence>
            <element name="idRule" type="xsd:string"/>
            <element name="effect" type="xsd:int"/>
            <element name="desc" type="xsd:string"/>
            <element name="session" type="xsd:int"/>
          </sequence>
        </complexType>
      </element>
    - <element name="createRuleResponse">
        <complexType/>
      </element>
      <element name="fault" type="tns1:ExceptionGen"/>
    - <element name="getSession">
        <complexType/>
      </element>
    + <element name="getSessionResponse">
    + <element name="getConexaoBD">
    + <element name="getConexaoBDResponse">
    + <element name="setConfigsBD">
    + <element name="setConfigsBDResponse">
    + <element name="initRequisicao">
    + <element name="initRequisicaoResponse">
    + <element name="finalizaRequisicao">
    + <element name="finalizaRequisicaoResponse">
```

O Quadro 19 ilustra parte do WSDL gerado. A partir deste arquivo que contém todos os métodos implementados na classe `FacadeAccessControlMDW` de acordo com a explanação dos método no Apêndice E, o cliente saberá quais as funções que poderá utilizar. As chamadas do cliente são codificadas em SOAP e transmitidas por HTTP. Quando o serviço recebe a requisição acontece o inverso, as chamadas são decodificadas e executadas e a resposta é codificada novamente para ser enviada para o cliente.

Com base no arquivo WSDL disponibilizado pelo próprio *web service*, é possível criar as classes necessárias para a comunicação entre a aplicação cliente desenvolvida e o serviço que se deseja acessar (*middleware*).

No cliente, para a comunicação com o servidor, são geradas as seguintes classes que também estão representadas no diagrama de classes da seção 3.2.2 *Especificação do software consumidor*: `FacadeAccessControlMDW (Cliente)`, `FacadeAccessControlMDWProxy`, `FacadeAccessControlMDWService`, `FacadeAccessControlMDWSoapBindingStub` e `FacadeAccessControlMDWServiceLocator`.

Apresenta-se abaixo o *Locator* (interface localizadora `FacadeAccessControlMDWServiceLocator`) onde está contida a URL para o *web service* e que oferece métodos para acessar a classe *stub* que faz o acesso *proxy* ao *web service*. Nas linhas 13 e 14 do Quadro 20 é possível visualizar a atribuição da URL de localização do WS (`http://localhost:8080/WSMddAccessPol/services/FacadeAccessControlMDW`) a partir da interface `FacadeAccessControlMDWServiceLocator`.

Quadro 20 - Interface `FacadeAccessControlMDWServiceLocator`

```
01 public class FacadeAccessControlMDWServiceLocator
02    extends org.apache.axis.client.Service
03    implements accessControl.FacadeAccessControlMDWService {
04
05 public FacadeAccessControlMDWServiceLocator() {}
06 public FacadeAccessControlMDWServiceLocator(EngineConfiguration config) {
07              super(config);
08  }
09  public FacadeAccessControlMDWServiceLocator(String wsdlLoc,QName sName)
10                      throws javax.xml.rpc.ServiceException {
11    super(wsdlLoc, sName);
12  }
13  private String FacadeAccessControlMDW_address =
14     "http://localhost:8080/WSMddAccessPol/services/FacadeAccessControlMDW";
15  public String getFacadeAccessControlMDWAddress() {
16        return FacadeAccessControlMDW_address;
17  }
18  private String FacadeAccessControlMDWWSDDServiceName =
19                            "FacadeAccessControlMDW";
20  public String getFacadeAccessControlMDWWSDDServiceName() {
21        return FacadeAccessControlMDWWSDDServiceName;
22 }
```

A classe `FacadeAccessControlMDWSoapBindingStub` é considerada classe raiz pois faz a comunicação entre o Java e o *web service*, ou seja, a classe que faz o acesso *proxy* entre a aplicação cliente e o *web service*. No Quadro 21 pode-se visualizar o método `createRule()` setando configuração e propriedades do objeto `Call` como o método a ser invocado (linha 29) e invocando a chamada de operação ao *WS* (linha 34).

Quadro 21 - Interface FacadeAccessControlMDWSoapBindingStub

```
01 public class FacadeAccessControlMDWSoapBindingStub
02     extends org.apache.axis.client.Stub
03     implements accessControl.FacadeAccessControlMDW {
04
05  private java.util.Vector cachedSerClasses = new java.util.Vector();
06  private java.util.Vector cachedSerQNames = new java.util.Vector();
07  private java.util.Vector cachedSerFactories = new java.util.Vector();
08  private java.util.Vector cachedDeserFactories = new java.util.Vector();
09
10  static org.apache.axis.description.OperationDesc [] _operations;
11
12  public FacadeAccessControlMDWSoapBindingStub(URL endpointURL,
13             Service service) throws AxisFault{}
14  protected Call createCall() throws java.rmi.RemoteException{}
15
16  public void createRule(String idRule, int effect, String desc,int session)
17             throws java.rmi.RemoteException, utils.ExceptionGen {
18    if (super.cachedEndpoint == null) {
19      throw new org.apache.axis.NoEndPointException();
20    }
21    org.apache.axis.client.Call _call = createCall();
22    _call.setOperation(_operations[0]);
23    _call.setUseSOAPAction(true);
24    _call.setSOAPActionURI("");
25    _call.setEncodingStyle(null);
26    _call.setProperty(Call.SEND_TYPE_ATTR, Boolean.FALSE);
27    _call.setProperty(AxisEngine.PROP_DOMULTIREFS, Boolean.FALSE);
28    _call.setSOAPVersion(SOAPConstants.SOAP11_CONSTANTS);
29    _call.setOperationName(new QName("http://accessControl",
30                                     "createRule"));
31   setRequestHeaders(_call);
32   setAttachments(_call);
33   try {
34     Object _resp = _call.invoke(new java.lang.Object[] {idRule,
35        new Integer(effect), desc, new Integer(session)});
36     if (_resp instanceof java.rmi.RemoteException) {
37           throw (java.rmi.RemoteException)_resp;
38      }
39     extractAttachments(_call);
…
```

A partir das classes citadas acima o software consumidor de testes explicado na seção posterior efetua as chamadas ao *middleware* e faz o tratamento do retorno do *web service* para a interface da aplicação.

3.3.4 Implementação do software consumidor de testes

Nesta seção será demonstrado o software consumidor de testes desenvolvido para exemplificar as chamadas de uma aplicação cliente ao *middleware* e para testar e demonstrar as funcionalidades que o *middleware* desenvolvido dispõe, explanando as classes com os códigos fonte das principais funções.

Deixa-se claramente explicito nesta seção que o software consumidor de testes não é o cliente fixo do *middleware* implementado, pois o *middleware* pode receber solicitações de

qualquer aplicação cliente implementada a partir das chamadas de procedimentos já explicadas.

A classe `ControlerCliente` é um *Singleton* (instancia única entre as telas) e foi criada para efetuar a ponte de comunicação entre as telas (demonstradas na seção 3.3.5 "*Operacionalidade da implementação*") e o *stub* cliente do *WS*. Segue abaixo a Quadro 22 mostrando a classe `ControlerCliente`.

Quadro 22 - Classe `ControlerCliente`

```
01 public class ControlerCliente {
02  private static ControlerCliente controler = null;
03  static FacadeAccessControlMDW facade = null;
04  private static int SESSAO = -1;
05  private String xmlSujeito = "";
06  private String xmlRecurso = "";
07  private String xmlAcao = "";
08
09  public static ControlerCliente getInstance() {
10    if (controler == null) {
11     controler = new ControlerCliente();
12     facade = new FacadeAccessControlMDWProxy();
13     try {
14       SESSAO = facade.getSession();
15     } catch (ExceptionGen e) {
16        JOptionPane.showMessageDialog(null, e.getFaultString());
17          facade.clearMemoryAll(SESSAO);
18     } catch (RemoteException e15) {
19        JOptionPane.showMessageDialog(null, e15.getMessage());
20     }
21  }
…
```

Como se pode notar no método `getInstance()` da classe mostrada acima, no momento em que a instancia da classe é efetuada também é instanciado um `FacadeAccessControlMDWProxy` (linha 12). A instancia do `FacadeAccessControlMDWProxy` é demonstrada através do Quadro 23 .

No construtor da classe `FacadeAccessControlMDWProxy`, é executada a chamada de método `_initFacadeAccessControlMDWProxy()`. Este método instancia um `FacadeAccessControlMDWServiceLocator` que configura a localização do *WS* e retorna uma interface Java para os métodos do *WS* disponíveis (`FacadeAccessControlMDW (Cliente)`) a partir do endereço (`endpoint.address`).

Com a instancia da interface Java para os métodos do *WS* disponíveis, a classe `ControlerCliente` (Quadro 22) efetua a chamada `getSession()` conforme linha 14. Este retorno será atribuído a uma variável global pois será utilizado em todas as posteriores chamadas ao *WS* como sessão da aplicação cliente.

Quadro 23 - Classe `FacadeAccessControlMDWProxy`

```
01 public class FacadeAccessControlMDWProxy implements
02             accessControl.FacadeAccessControlMDW {
03  private String _endpoint = null;
04    private accessControl.FacadeAccessControlMDW
05            facadeAccessControlMDW = null;
06
07  public FacadeAccessControlMDWProxy(String endpoint) {
08    _endpoint = endpoint;
09    _initFacadeAccessControlMDWProxy();
10  }
11
12  private void _initFacadeAccessControlMDWProxy() {
13    try {
14      facadeAccessControlMDW = (
16                  new accessControl.FacadeAccessControlMDWServiceLocator())
17                     .getFacadeAccessControlMDW();
18      if (facadeAccessControlMDW != null) {
19        if (_endpoint != null)
20          ((javax.xml.rpc.Stub)facadeAccessControlMDW)
21          ._setProperty("javax.xml.rpc.service.endpoint.address",
22                        _endpoint);
23      else
24        _endpoint = (String)((javax.xml.rpc.Stub)facadeAccessControlMDW)
25                    ._getProperty("javax.xml.rpc.service.endpoint.address");
26      }
27    }
28    catch (javax.xml.rpc.ServiceException serviceException) {}
29  }
```

Para explicação dos códigos da aplicação consumidora de testes serão apresentados a seguir a execução de alguns métodos da classe `ControlerCliente`.

Conforme Quadro 24, o método `atualizaListaAtributosExtFrame` é chamado para atualização da lista de atributos externos na tela da classe `ManageGetAtributosExt` (Figura 21). De acordo com o observado nas linhas 11 e 12 é efetuada a chamada de função do *middleware* `getAtributosExternos()`. A função é chamada duas vezes para buscar os atributos externos de sujeito e recurso.

Após o retorno do XML (Apêndice B), com a utilização das classes: `Document` que faz o mapeamento em memória da árvore *Document Object Model* (DOM) responsável pela representação do documento XML, da classe `SAXBuilder` que faz a compilação do documento XML e da classe `Element` utilizada para representar cada elemento contido na árvore, é feita a manipulação do XML. Conforme as linhas 26 e 40 são montados os documentos XML, é atribuído o `element root` do `Document` a um `Element` e são solicitados de acordo com as linhas 28 e 42 os elementos filhos para iterar sobre eles.

A lista de elementos filhos é percorrida e assim é feita a atualização da `JTable` dos atributos.

Quadro 24 - Método `atualizaListaAtributosExtFrame()` - Classe `ControlerCliente`

```
01 public void atualizaListaAtributosExtFrame(JTable jtableLstAttr) {
02  String atributosExternosSuj = "";
03  String atributosExternosRec = "";
04  Element AtributoInfo;
05  List<Element> elements;
06  DefaultTableModel tabelaAttrs =
07                    (DefaultTableModel)jtableLstAttr.getModel();
08  tabelaAttrs.setNumRows(0);
09
10  try {
11      atributosExternosSuj = facade.getAtributosExternos(0, SESSAO);
12      atributosExternosRec = facade.getAtributosExternos(1, SESSAO);
13    } catch (ExceptionGen e) {
14       JOptionPane.showMessageDialog(null, e.getFaultString());
15       facade.clearMemoryAll(SESSAO);
16    } catch (RemoteException e) {
17       JOptionPane.showMessageDialog(null, e.getMessage());
18    }
19
20  SAXBuilder sb = new SAXBuilder();
21  Document d = null;
22  ByteArrayInputStream strSuj = new ByteArrayInputStream(
23                                          atributosExternosSuj.getBytes());
24  ByteArrayInputStream strRec = new ByteArrayInputStream(
25                                          atributosExternosRec.getBytes());
26  d = sb.build(strSuj);
27  AtributoInfo = d.getRootElement();
28  elements = AtributoInfo.getChildren();
29  Iterator<Element> i = elements.iterator();
30
31  while (i.hasNext()) {
32    Element Atributo = (Element) i.next();
33    tabelaAttrs.addRow(new Object[] { (Atributo.getChildText("ID")),
34                                      (Atributo.getChildText("Desig")),
35                                      (Atributo.getChildText("VlrEnt")),
36                                      (Atributo.getChildText("IdentSai")),
37                                      (Atributo.getChildText("VlrSai")) });
38
39  }
40  d = sb.build(strRec);
41  AtributoInfo = d.getRootElement();
42  elements = AtributoInfo.getChildren();
43  i = elements.iterator();
44
45  while (i.hasNext()) {
46    Element Atributo = (Element) i.next();
47    tabelaAttrs.addRow(new Object[] { (Atributo.getChildText("ID")),
48                                    (Atributo.getChildText("Desig")),
49                                    (Atributo.getChildText("VlrEnt")),
50                                    (Atributo.getChildText("IdentSai")),
51                                    (Atributo.getChildText("VlrSai")) });
52
53  }
54 }
```

Para alteração de uma política do *middleware* é necessário solicitar a função `getPoliticaAltera()` que retornará à aplicação cliente um XML da política solicitada para alteração (Apêndice B). A partir deste XML segue método `atualizaPolitica()`que efetua chamada de função ao *middleware* para atualização dos dados da política.

Quadro 25 - Método `atualizaPolitica()` - Classe `ControlerCliente`

```
01 public void atualizaPolitica(String ident, String desc, int alg) {
02  Element Politica = new Element("Politica");
03  Element ID = new Element("ID");
04  Element Descricao = new Element("Descricao");
05  Element Algoritmo = new Element("Algoritmo_Combinacao");
06
07  Politica.addContent(ID);
08  Politica.addContent(Descricao);
09  Politica.addContent(Algoritmo);
10
11  ID.setText(ident);
12  Descricao.setText(desc);
13  Algoritmo.setText(String.valueOf(alg));
14
15  XMLOutputter xout = new XMLOutputter();
16  try {
17    facade.alteraPolitica(xout.outputString(Politica), SESSAO);
18     } catch (ExceptionGen e) {
19        JOptionPane.showMessageDialog(null, e.getFaultString());
20        facade.clearMemoryAll(SESSAO);
21     } catch (RemoteException e) {
22        JOptionPane.showMessageDialog(null, e.getMessage());
23     }
24
25 }
```

A partir da tela `ManagePolitica` (Figura 17) são recebidos os parametros necessários para a execução do método acima. No método `atualizaPolitica()` é criada uma estrutura XML a partir do elementos `Politica`, `ID`, `Descricao` e `Algoritmo`. São atribuídos aos valores dos elementos os parâmetros de entrada do método e através da classe `XMLOutputter` é gerado a `String` XML (linha 15 e 17).

Na linha 17 o método `atualizaPolitica()` efetua a chamada da função do *middleware* `alteraPolitica()`, e assim o *middleware* se encarrega da alteração XML do arquivo de política do diretório.

No Quadro 26 demostrado na sequência, segue o método `novaRegra()` responsável por criar uma nova regra, criar uma nova condição e adiciona-la a regra juntamente com o alvo criado, e atribuir a regra na política em fase de criação.

A partir dos dados cadastrados na tela `ManageRegra` (Figura 20) é executada a função da classe `ControlerCliente novaRegra()`. Neste método é efetuada a chamada de função do *middleware* `createRule()` (linha 07) responsável por criar uma regra.

A partir da criação da regra, o método `novaRegra()` verifica se os parâmetros recebidos referente aos atributos da condição estão preenchidos. Se a condição for verdadeira é efetuada a chamada de função do *middleware* `createConditionRule()` para criar uma condição que logo em seguida é atribuída a regra pela chamada `addConditionInRule()`.

Não foi especificado aqui, porém poderia ter sido criado um alvo (`Target`) para a regra, portanto na linha 18 é executada a chamada de função `addTargetInRule()` que por sua vez adiciona o alvo a regra.

Após a criação, a regra deve ser adicionada a política em fase de criação, portanto efetuou-se a chamada de função `addRuleInPolicy`.

Quadro 26 - Método `novaRegra()` - Classe `ControlerCliente`

```
01 public void novaRegra(String ident, String desc, int efeito,
02    int memberCond1, String identCond1, String funcCond1,
03    String valorCond1, String operator, int memberCond2,
04    String identCond2, String funcCond2, String valorCond2) {
05
06  try {
07    facade.createRule(ident, efeito, desc, SESSAO);
08    if (!identCond1.trim().equalsIgnoreCase("")
09      && !valorCond1.trim().equalsIgnoreCase("")) {
10      facade.createConditiontRule(memberCond1, identCond1, funcCond1,
11                                  valorCond1, operator, SESSAO);
12
13      if (!operator.trim().equalsIgnoreCase(""))
14        facade.createConditiontRule(memberCond2, identCond2,
15                                    funcCond2, valorCond2, "", SESSAO);
16      facade.addConditionInRule(SESSAO);
17    }
18    facade.addTargetInRule(SESSAO);
19    facade.addRuleInPolicy(SESSAO);
20    } catch (ExceptionGen e) {
21      JOptionPane.showMessageDialog(null, e.getFaultString());
22      facade.clearMemoryAll(SESSAO);
23    } …
```

3.3.5 Operacionalidade da implementação

Nesta seção são apresentadas as principais funcionalidades do *middleware* desenvolvido através de chamadas de função efetuadas do software consumidor de testes. As chamadas de função serão exibidas na ordem de fluxo de utilização do software consumidor de testes, onde o software conectado ao *middleware* efetua gerência de uma política e logo em seguida efetua uma simulação de solicitação de acesso a um recurso.

Segue a partir da Figura 15 a tela principal do software consumidor de testes implementado para testar as funcionalidades do *middleware* mostrando a composição dos menus e botões e posteriormente fluxo de utilização.

Figura 15 - Tela principal do software consumidor de testes

3.3.5.1 Gerência de política de controle de acesso

Na tela principal do software consumidor de testes, ao clicar no botão `gerenciar política`, é efetuada a chamada de função do *middleware* `getPoliticasDiretorio()` (função que retorna as políticas cadastradas no *middleware*), instanciando e atualizando os dados da tela visualizada a seguir.

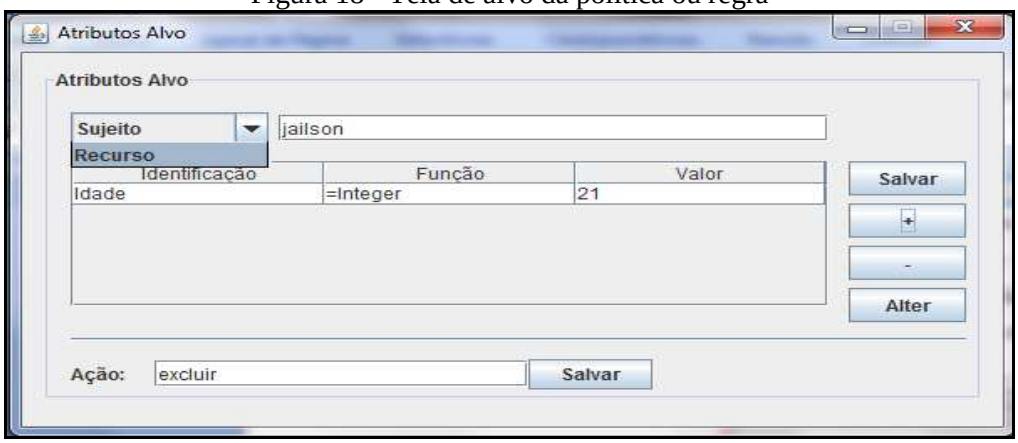
Figura 16 - Tela de gerência de políticas

A tela de gerência de políticas (Figura 16) permite criar, alterar e excluir uma política de controle de acesso. Para o exemplo teste, ao clicar no botão `novo` o software cliente abre a tela da Figura 17 e inicia a criação de uma política.

Na tela de criação de política (Figura 17), após o preenchimento dos campos: (1) identificação da política; (2) algoritmo de combinação (algoritmo utilizado para avaliação das regras da política posteriormente); e (3) descrição, ao clicar no botão `alvos` o software consumidor de testes abre a tela apresentada na Figura 18 para os cadastros de alvo da política de controle de acesso.

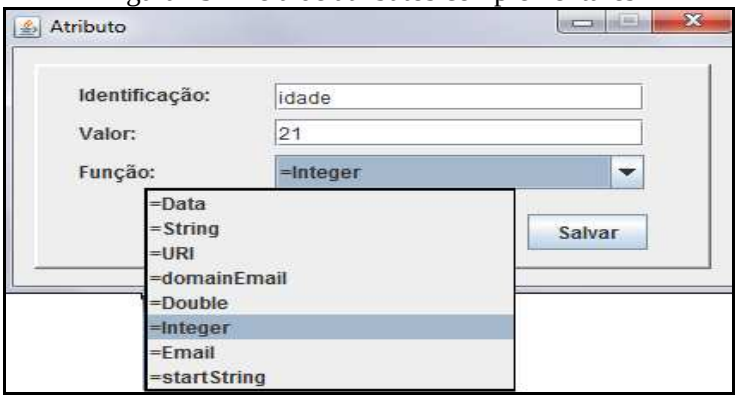
Figura 17 - Tela de criação de política

Figura 18 - Tela de alvo da política ou regra



Para inserir, alterar ou excluir atributos complementares ao sujeito ou recurso utiliza-se os botões `+`, `-` ou `alter`, que abrem a tela de atributos abaixo.

Figura 19 - Tela de atributos complementares



Ao clicar no botão `salvar` da tela de atributos complementares o valor cadastrado será atualizado na `JTable` da tela de alvo (Figura 18) e será fechada a tela de cadastro.

Ao clicar no botão `salvar` dos `Atributos Alvo` da tela da Figura 18, o software consumidor de testes executa as chamadas de função do *middleware*: `createTargetSubject()` para criar o sujeito do alvo da política ou

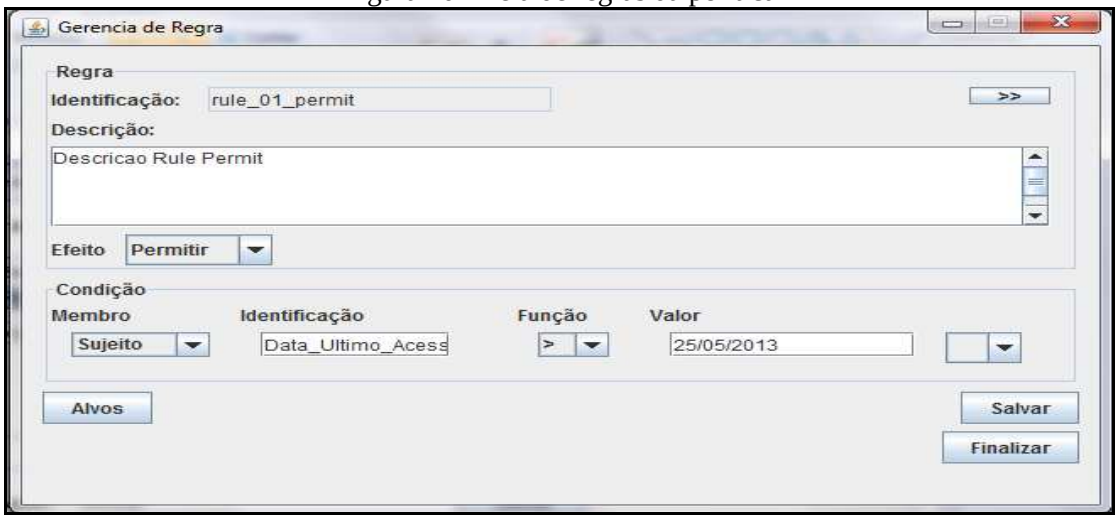
`createTargetResource()` para criar o recurso do alvo da política e `addAttributesTarget()` para adicionar os atributos complementares que são mostrados a partir da `JTable` ao recurso ou sujeito.

Ao clicar no botão `salvar` referente a `Ação` da tela da Figura 18 será efetuada a chamada de função `createTargetAction()` para criar a ação do alvo da política em fase de criação.

Ao sair da tela de alvo da política ou regra, ao clicar no botão `salvar` da tela de criação de política (Figura 17) o alvo criado anteriormente (sujeito, recurso e ação) será adicionado a política através da chamada de função do *middleware* `addTargetInPolicy()` e será habilitado o botão `gerenciar regras/finalizar`.

Ao clicar no botão `gerenciar regras/finalizar` abrirá a tela de criação de regras para a política que também contém a opção de cadastro de condições a regra.

Figura 20 - Tela de regras da política



Após preencher os dados da regra (`identificação`, `descrição`, `efeito` e dados relativos a condição opcional) poderá ser criado e adicionado um alvo também à regra (botão `alvos`), que segue a mesma lógica implementada para a criação e adição de alvo a política.

Ao clicar no botão `salvar` da tela acima, o software consumidor de testes executará em sequência as seguintes funções do *middleware*:

a) `createRule()`: efetua a criação da regra no *middleware* a partir dos valores: `identificação`, `descrição` e `efeito` da figura acima;
b) `createConditionRule()`: efetua a criação de uma condição para a regra no *middleware* a partir dos valores: `membro`, `identificação`, `função` e `valor` da figura acima. Se o valor de operador da condição estiver selecionado (“E” ou

"OU") será efetuada novamente a chamada da função, pois a condição terá duas expressões lógicas;

c) `addConditionInRule()`: o *middleware* adiciona a condição criada a regra em fase de criação;

d) `addTargetInRule()`: caso tenha sido criado um alvo para a regra, o *middleware* adicionará o alvo a regra;

e) `addRuleInPolicy()`: o *middleware* adiciona a regra criada a política em fase de criação.

Com a regra salva, o software consumidor de testes habilitará novamente a tela de regras para poder efetuar o cadastro de uma nova. Caso não seja mais requerido a criação de novas regras, ao clicar no botão `finalizar` o software consumidor de testes executa a chamada de função `finalizaCreatePolicy()` ao *middleware* para que a política seja salva e executa também a chamada de função `getPoliticasDiretorio()` para atualizar a lista de políticas cadastradas apresentadas na tela da Figura 16.

É apresentado no Quadro 27, como uma política XACML é criada pelo *middleware* antes de sofrer a criptografia.

Quadro 27 - Sintaxe de política XACML criada pelo *Middleware*

```
<?xml version="1.0"?>
- <Policy RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:first-applicable" PolicyId="policy_simulada">
    <Description>Descricao Simulacao de Politica</Description>
  - <Target>
    - <Subjects>
      - <Subject>
        - <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">jailson</AttributeValue>
            <SubjectAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema#string" AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"/>
          </SubjectMatch>
        - <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:integer-equal">
            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#integer">21</AttributeValue>
            <SubjectAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema#integer" AttributeId="idade"/>
          </SubjectMatch>
        </Subject>
      </Subjects>
    + <Resources>
    - <Actions>
      - <Action>
        - <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">excluir</AttributeValue>
            <ActionAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema#string" AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"/>
          </ActionMatch>
        </Action>
      </Actions>
  </Target>
- <Rule Effect="Permit" RuleId="rule_01_permit">
    <Description>Descricao Rule Permit</Description>
  - <Target>
    + <Subjects>
    - <Resources>
      - <Resource>
        - <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">arquivo_01</AttributeValue>
            <ResourceAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema#string" AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"/>
          </ResourceMatch>
        </Resource>
      </Resources>
    + <Actions>
    </Target>
  - <Condition FunctionId="urn:oasis:names:tc:xacml:1.0:function:date-greater-than">
    - <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:date-one-and-only">
        <SubjectAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema#date" AttributeId="Data_Ultimo_Acesso"/>
      </Apply>
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#date">2013-05-25+00:00</AttributeValue>
    </Condition>
  </Rule>
```

Pode-se observar na figura acima uma política *XACML* com identificação igual a "policy_simulada", que possui um algoritmo de combinação *first-applicable*, ou seja, que as

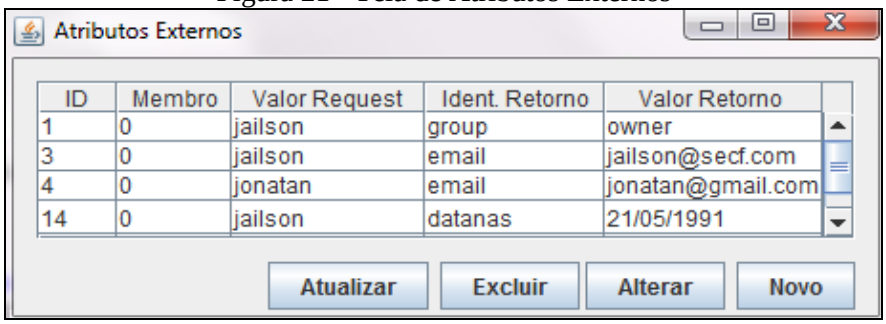
regras são avaliadas em sequência, assim a primeira regra que atender a solicitação será a resposta de autorização e com a descrição igual a “Descricao Simulacao de Politica”. Na política *XACML* demonstrada é possível notar um alvo (`Target`) com valores de sujeito igual a “jailson” e com `idade` igual a “21” e a ação a ser executada no recurso – “excluir” .

A política possui uma regra chamada “rule_01_permit” com efeito de permissão, ou seja, os sujeitos que obedecerem a condição desta regra (onde a `Data_ultimo_acesso` do sujeito for maior que “25/05/2013”) receberão permissão de acesso. A regra da política contem também um alvo com o recurso da solicitação “arquivo_01”.

Para que não seja necessário enviar juntamente na requisição de acesso o atributo complementar relacionado a `idade` do sujeito (conforme demonstrado na política do Quadro 27), pode-se incluir um atributo externo a partir do botão `Gerenciar Atributos Externos` da tela principal do software consumidor de testes (Figura 15).

Ao clicar no botão `Gerenciar Atributos Externos` é efetuada a chamada de função do *middleware* `getAtributosExternos()`. Esta função retorna os atributos externos cadastrados para instanciação e atualização dos valores da tela demonstrada abaixo, que permite a inclusão, exclusão e alteração de um atributo externo.

Figura 21 - Tela de Atributos Externos



Ao clicar no botão `Novo` pode-se visualizar a tela abaixo para incluir um novo atributo externo.

Figura 22 - Tela para cadastro de Atributos Externos

3.3.5.2 Simulação de requisição de acesso a recurso

Para simular uma requisição de acesso a partir do software consumidor de testes, é necessário clicar no botão `Simular requisição de acesso` da tela principal (Figura 15), com isso será instanciada a tela da Figura 23 e Figura 24.

Para efetuar uma requisição de acesso, podem ser apenas enviados os dados do sujeito, recurso ou ação, ou também adicionar atributos complementares ao sujeito, recurso e ambiente. Para situação de teste a seguir será especificado a sequência de chamadas de função de duas solicitações de acesso adicionando atributos complementares de sujeito.

Figura 23 - Tela Simulador de requisições para solicitação 01

Figura 24 - Tela Simulador de requisições para solicitação 02

Após inserir ao valores relacionados a sujeito, recurso e ação, ao clicar no botão `salvar`, caso seja o primeiro atributo complementar adicionado o software consumidor de testes executará a chamada de função do *middleware* `initRequisicao()` parametrizando os

valores do `sujeito`, `recurso` e `ação` das telas de solicitação de acesso (Figura 23 e Figura 24), e efetuará a chamada `requisitaSujeitoPEP()` (atributo complementar de sujeito selecionado) parametrizando os valores `identificação` e `valor`.

Caso já tenha cadastrado atributos complementares, a cada ação do botão `salvar` serão adicionados novos atributos complementares a partir das chamadas de função: `requisitaAmbientePEP()`, `requisitaSujeitoPEP()`ou `requisitaRecursoPEP()`, de acordo com tipo selecionado no `ComboBox` (`Sujeito`, `Recurso` ou `Ambiente`).

A partir das duas solicitações de acesso criadas, o *middleware* cria dois *requests XACML* de acordo com o Quadro 28 e Quadro 29.

Quadro 28 - Exemplo de *request* XACML criado pelo *middleware* da solicitação 01

```
<?xml version="1.0"?>
- <Request>
  - <Subject SubjectCategory="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject">
    - <Attribute DataType="http://www.w3.org/2001/XMLSchema#string"
      AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id">
        <AttributeValue>jailson</AttributeValue>
      </Attribute>
    - <Attribute DataType="http://www.w3.org/2001/XMLSchema#date" AttributeId="data_ultimo_acesso">
        <AttributeValue>2013-05-26+00:00</AttributeValue>
      </Attribute>
    </Subject>
  - <Resource>
    - <Attribute DataType="http://www.w3.org/2001/XMLSchema#string"
      AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id">
        <AttributeValue>arquivo_01</AttributeValue>
      </Attribute>
    </Resource>
  - <Action>
    - <Attribute DataType="http://www.w3.org/2001/XMLSchema#string"
      AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id">
        <AttributeValue>excluir</AttributeValue>
      </Attribute>
    </Action>
</Request>
```

Quadro 29 - Exemplo de *request* XACML criado pelo *middleware* da solicitação 02

```
<?xml version="1.0"?>
- <Request>
  - <Subject SubjectCategory="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject">
    - <Attribute DataType="http://www.w3.org/2001/XMLSchema#string"
      AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id">
        <AttributeValue>jailson</AttributeValue>
      </Attribute>
    - <Attribute DataType="http://www.w3.org/2001/XMLSchema#date" AttributeId="data_ultimo_acesso">
        <AttributeValue>2013-05-24+00:00</AttributeValue>
      </Attribute>
    </Subject>
  - <Resource>
    - <Attribute DataType="http://www.w3.org/2001/XMLSchema#string"
      AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id">
        <AttributeValue>arquivo_02</AttributeValue>
      </Attribute>
    </Resource>
  - <Action>
    - <Attribute DataType="http://www.w3.org/2001/XMLSchema#string"
      AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id">
        <AttributeValue>excluir</AttributeValue>
      </Attribute>
    </Action>
</Request>
```

Após a inserção dos dados referentes a solicitação de requisição de acesso ao recurso, ao clicar no botão `enviar` o software consumidor de testes efetua as chamadas de função ao

middleware: `finalizaRequisicao()`, `getRecursoDecisaoPEP()` e `getJustificativaPEP()`;

No momento em que são efetuadas estas chamadas, o *middleware* analisa a requisição, gera um arquivo de *response XACML* (conforme *responses* das duas solicitações que são mostradas abaixo) e o interpreta para o software consumidor de testes.

Quadro 30 - Exemplo de *response* XACML da solicitação 01 criado pelo *middleware*

```
<?xml version="1.0"?>
- <Response>
  - <Result ResourceID="arquivo_01">
      <Decision>Permit</Decision>
    - <Status>
        <StatusCode Value="urn:oasis:names:tc:xacml:1.0:status:ok"/>
      </Status>
    </Result>
</Response>
```

Quadro 31 - Exemplo de *response* XACML da solicitação 02 criado pelo *middleware*

```
<?xml version="1.0"?>
- <Response>
  - <Result ResourceID="arquivo_02">
      <Decision>Deny</Decision>
    - <Status>
        <StatusCode Value="urn:oasis:names:tc:xacml:1.0:status:ok"/>
      </Status>
    </Result>
</Response>
```

O *middleware* após interpretar a resposta, retorna ao software consumidor de testes a decisão de autorização de acesso (“Acesso permitido” ou “Acesso Negado”), o recurso que foi solicitado acesso e uma justificativa da análise da requisição, ou seja, um status de como se sucedeu o processo de análise.

Com estes valores o software consumidor de testes mostra em tela a resposta à solicitação de acesso feita de acordo com Figura 25.

Figura 25 - Resposta as duas solicitações de acesso apresentadas no software consumidor de testes

3.4 RESULTADOS E DISCUSSÃO

O trabalho através das aplicações desenvolvidas alcançou os resultados esperados, assim, através do serviço de controle de acesso e autorização qualquer aplicação pode incorporar funções para gerir a permissão de acesso a recursos sem a necessidade de maiores implementações ou conhecimento amplo de padrões e linguagens normatizadas para gerenciar políticas de controle de acesso.

O *middleware* implementado neste trabalho atingiu os resultados esperados frente aos objetivos a que se propôs. Como demonstrado na operacionalidade da implementação, há uma abstração das questões de sintaxe e funcionamento do *XACML*, é possível cadastrar e alterar as políticas de controle de maneira simples e solicitar acesso a determinado recurso recebendo a resposta de autorização para o sujeito requisitante.

Diante dos testes realizados no software consumidor de testes, ao efetuar as chamadas de funções do *middleware* foi possível efetuar a gerência de políticas de controle de acesso para recursos variados, ou seja, pode-se criar políticas para gerenciar o controle tanto de *hardware* quanto de arquivos ou casos de uso de um sistema e efetuar a autorização de permissão ou negação para usuários, grupos, domínios entre outros tipos de sujeito requeridos.

A partir do módulo de procura de atributos externos implementado (PIP – Classe `PolicyInformPointFinderModuleMDW`) é possível efetuar solicitações de acesso simples parametrizando apenas a informação do `sujeito`, `recurso` e `ação`, assim os valores complementares dos sujeitos ou recursos serão resolvidos a partir do cadastro dos atributos em banco de dados.

Em relação aos trabalhos correlatos, Souza (2010) assemelha-se ao trabalho de conclusão no âmbito de controle de acesso a recursos em sistemas distribuídos e pelo fato de não tratar problemas oriundos de conflitos de políticas de segurança e mais de uma entidade administradora ou organizações distintas, porém não utiliza o padrão XACML. O trabalho utiliza políticas de controle de acesso baseadas em papeis (RBAC).

Em relação a Lima (2008) e Hu et al. (2007), a semelhança com o trabalho esta na utilização de XACML para gerir as políticas de controle, porém difere-se no fato de que os trabalhos tratam do controle de conflito, conformidade e criação de políticas de segurança de acesso, não tratando da autorização e controle de acesso em sistemas distribuídos. O

middleware desenvolvido neste projeto em relação aos trabalhos de Lima (2008) e Hu et al. (2007) obteve como uma de suas diferenças principais o desacoplamento em relação aos softwares, tornando-o mais flexível.

O trabalho de Lima (2008) trata de um sistema centralizado que tem como modelo de referência direcionado a área da saúde e além de não requerer o conhecimento da linguagem de codificação das políticas como o trabalho de conclusão, propôs algoritmos que gerenciam automaticamente em tempo de criação ou edição das políticas, controles de conflitos, por exemplo, a criação de políticas que permitem e negam a autorização para o mesmo sujeito e recurso ao mesmo tempo.

O trabalho de Hu et al. (2007) apresenta uma abordagem para a realização de verificação de conformidade de políticas de controle de acesso especificadas em XACML, ou seja, com base em verificações e testes efetuam-se consistências sintáticas e semânticas de uma política de controle de acesso, encontrando discrepâncias entre a política especificada e o que realmente o autor de criação da política esperava controlar.

O Quadro 32 apresenta uma comparação entre os trabalhos de Souza (2010), Lima (2008) e Hu et al. (2007) em relação a este trabalho (*middleware*).

Quadro 32 – Comparação entre o *Middleware* e os trabalhos correlatos

	Middleware	Souza (2010)	Lima (2008)	Hu et al. (2007)
Padrão *XACML*	X		X	X
Web Service	X			
Solução distribuída	X	X		
Controle de conformidade de políticas			X	X
Controle de conflitos de políticas			X	
Gerência de autorização e controle de acesso	X	X		
Biblioteca *SunXACML*	X		X	
Multi-Corporativo (entidades distintas)				

4 CONCLUSÕES

Visto que atualmente a informação é um dos bens mais importantes para uma empresa e que o valor de uma organização é descrito pelos recursos que ela possui, sejam bens materiais, documentos, etc., é de extrema importância que se garanta proteção, logo deve-se ter meios para gerenciar e proteger estes recursos. Nos sistemas distribuídos existem a preocupação com o acesso e gestão de direitos.

Como em um sistema distribuído os recursos encontram-se espalhados por várias máquinas, as questões de gestão de direitos e acesso são complicadas, pois se seguida a abordagem de sistemas centralizados seria necessário criar uma conta para cada usuário em cada máquina. Com esta distribuição de recursos notou-se a necessidade da implementação de uma camada intermediária de software – *middleware* para garantir autorização de controle de acesso a recursos em sistemas distribuídos.

Os softwares que utilizam recursos de um *middleware* têm sua complexidade reduzida fazendo com que seus principais objetivos sejam o acesso a esta camada intermediária e a partir dos objetivos deste trabalho consegue-se centralizar as políticas de controle de acesso a fim de possibilitar a consolidação das regras utilizadas por todos os elementos do sistema, o armazenamento e administração de um grande número de sujeitos e recursos e o atendimento de requisições de elementos diversos.

Durante este trabalho desenvolveu-se um estudo na área de segurança da informação, autorização e controle de acesso em sistemas distribuídos e XACML. Este estudo trouxe subsídios para a aplicação dos requisitos referentes à autorização e controle de acesso no *middleware*.

Como foi visto no decorrer do trabalho, a abordagem utilizada para controle de acesso e políticas de controle foi o XACML através da *API SunXACML*. Como o padrão XACML é muito flexível e genérico e a *API* utilizada possui muitas classes e cada uma possui muitos itens, foi feito um estudo sobre esses itens e escolhido os mais prioritários para se conseguir garantir controle de acesso e autorização a varios tipos de recursos sem que se tenha um conhecimento aprofundado do funcionamento do padrão utilizado.

O *middleware* implementado propôs uma maneira de efetuar o controle de acesso a recursos abstraindo o conhecimento de sintaxe e funcionamento do XACML. O uso da *API* utilizada no desenvolvimento do *middleware* diretamente em um software cliente não

forneceria desacoplamento e flexibilidade necessários para aplicações distribuídas e integração de sistemas heterogêneos e forçaria o desenvolvedor a implementar em baixo nível, forçando-o a tratar os diversos tipos de dados, algoritmos de combinação, funções de comparação de valores, entre outras funcionalidades que o XACML possui.

A *API* utilizada *SunXACML* foi de grande utilidade, pois apresenta classes que implementam os processos de criação de requisições e respostas XACML bem como a avaliação de políticas e o pacote também apresenta classes para geração de políticas XACML.

Para garantir a heterogeneidade de comunicação e interoperabilidade no acesso ao *middleware* foi utilizada a tecnologia de *Web Services*. Os *Web Services* têm como vantagens utilizar padrões abertos, permitindo que componentes sejam escritos para diferentes linguagens e plataformas e podem ser facilmente implementados e possuem baixo custo de implementação pelo fato de utilizar estruturas e protocolos de comunicação existentes para efetuar troca de informação.

O *framework* Axis 1.4 possibilitou uma enorme facilidade na criação do *WS*, pois foi muito simples de ser utilizado e muito eficiente em seu objetivo. O *framework* garantiu a implementação do protocolo SOAP para processar e interpretar as mensagens de comunicação entre cliente e servidor de uma maneira abstraída e permitiu criar um *web service* a partir de classes Java.

Com o término do desenvolvimento do *middleware*, conclui-se que os objetivos e os fluxos de trabalho previstos foram alcançados. Conforme discutido acima a partir de *frameworks* e bibliotecas foi possível gerir a autorização distribuída e controle de acesso através de políticas de controle utilizando o padrão XACML, efetuando a gerência de políticas e tratando solicitações (pedidos e respostas) de controle de acesso a recursos e criar o *web service* para comunicação com o middleware. Como resultado, subentende-se que o *middleware* desenvolvido é importante na realização de controle de acesso a sistemas distribuídos e que o desenvolvimento de uma camada intermediária de software que atenda aos itens de autorização e controle de acesso e torne o desenvolvimento mais ágil e mais prático é viável.

Como o padrão XACML é muito extenso, o *middleware* implementado pode ser incrementado para que possa utilizar mais funcionalidades e que possa usufruir de maiores recursos do padrão.

4.1 EXTENSÕES

Para dar continuidade ao *middleware* desenvolvido, propõe-se como extensão:

a) atualmente o *middleware* utiliza os objetos `PolicySet` apenas no âmbito de avaliação das políticas cadastradas. Sugere-se permitir com que o cadastro de políticas seja feito não apenas em objetos `Policy`, mas também em `PolicySet`, ou seja, permitir o cadastro e todo o processo de gerência de conjuntos de políticas (`PolicySet`) para um dado recurso, sujeito ou ação;
b) permitir que o *middleware* incorpore os novos recursos utilizados por versões XACML recentes;
c) implementar no *middleware* algoritmos de verificação de conflitos, possibilitando a verificação de políticas ou regras que estão conflitando em relação a autorização do acesso;
d) efetuar melhorias no âmbito do caso de uso de alteração das políticas, ou seja, possibilitar uma melhor maneira de alterar os dados da política, permanecendo a abstração atual em relação ao XML do XACML, porém alterar a implementação utilizada `getNext()`, fazendo com que a solicitação das regras e alvos das políticas não seja sequencial, mas sim indexada;
e) possibilitar que uma condição para regra tenha um número dinâmico de expressões lógicas;
f) implementar interface web para gerenciamento dos repositórios, listagem e manutenção das políticas e seus elementos, *responses* e *requests*, permitindo a visualização em árvores para maior legibilidade estrutural.

REFERÊNCIAS BIBLIOGRÁFICAS

APACHE SOFTWARE FOUNDATION. **Apache Axis**. [S.l], 2013. Disponível em: <http://axis.apache.org/axis/java/releases.html>. Acesso em: 30 jan. 2013.

APACHE SOFTWARE FOUNDATION. **Apache Tomcat**. [S.l.], 2009. Disponível em: <http://tomcat.apache.org/>. Acesso em: 14 fev. 2013.

BARBOSA, Álvaro C P. **Middleware para integração de dados heterogêneos baseado em composição de frameworks**. 2001. 154 f. Tese de Doutorado (Doutor em Informática) – PUC Rio de Janeiro, Rio de Janeiro. Disponível em: <ftp://ftp.inf.puc-io.br/pub/docs/theses/01_PhD_barbosa.pdf>. Acesso em: 12 ago. 2012.

MARTINS, Vidal. **Segurança em sistemas distribuídos**. [S.l], 2009. Disponível em: <http://www.batebyte.pr.gov.br/modules/conteudo/conteudo.php?conteudo=602>. Acesso em: 21 ago. 2012.

CAMARGO, Edson T. et al. Autenticação e autorização em arquiteturas orientadas a serviço através de identidades federadas. In: SIMPÓSIO BRASILEIRO DE REDES DE COMPUTADORES E SISTEMAS DISTRIBUÍDOS (SBRC), 25., 2007, Belem/PA. **Anais...** Belem: [S.l.], 2007, Não paginado. Disponível em: <http://www.lbd.dcc.ufmg.br/colecoes/sbrc/2007/006.pdf>. Acesso em: 20 ago. 2012.

CECHINEL, Abel L. **HMI:** um middleware para objetos distribuídos sobre o protocolo HTTP. 2008. 87 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Universidade Regional de Blumenau, Blumenau.

COULOURIS, George; DOLLIMORE, Jean; KINDBERG, Tim. **Sistemas distribuídos**: conceitos e projeto. 4. ed. Tradução João Tortello. Porto Alegre: Bookman, 2007.

DANTAS, Mario. **Computação distribuída de alto desempenho**: redes, clusters e grids computacionais. Rio de Janeiro: Axcel Books do Brasil, 2005.

EHNEBUSKE, David et al. **Simple Object Access Protocol (SOAP) 1.1**. [S.l.], 2000. Disponível em: <http://www.w3.org/TR/2000/NOTE-SOAP-20000508/#RFC2774>. Acesso em: 15 mar. 2013.

ENTERPRISE ARCHITECT. Austrália, 2008. Disponível em: <http://www.sparxsystems.com.au/about.html>. Acesso em: 25 jan. 2013.

FERREIRA, Silvia et al. LARCES_Seg - uma arquitetura para gerenciamento de VPNs baseada em políticas utilizando XACML, In: WORKCOMP SUL – WORKSHOP DE COMPUTAÇÃO DA REGIÃO SUL, 1., 2004, Florianópolis/SC. **Anais...** Florianópolis: [S.l.], 2004, Não paginado. Disponível em: <http://jeri.larces.uece.br/larces/images/publicacoes/2004/larces_seg_2004.pdf >. Acesso em: 07 set. 2012.

GOUVEIA, Francisco A. **Policy administration – OASIS XACML v.3**. 2011. 54 f. Trabalho de Conclusão de Curso (Licenciatura em Tecnologias e Sistemas de Informação) – Universidade Aveiro, Aveiro/Portugal.

HU, Vincent C. et al. Conformance checking of access control policies specified in XACML. In: COMPUTER SOFTWARE AND APPLICATIONS CONFERENCE (COMPSAC), 31., 2007, Pequim/China. **Anais...** Carolina do Norte EUA: IEEE Computer Society, 2007. p. 275 280, Disponível em: <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=4291136&contentType=Conference+Publications&queryText%3Dxacml>. Acesso em: 10 ago. 2012.

IBM DEVELOPER WORKS. **XML Security**: control information access with XACML. [S.l.], 2004. Disponível em <http://www.ibm.com/developerworks/xml/library/x-xacml/>. Acesso em: 15 mar. 2013.

JASYPT. **Java simplified encryption**. [S.l], 2011. Disponível em: <http://www.jasypt.org/ >. Acesso em: 03 mar. 2013.

ORACLE. **Java remote method invocation**. [S.l.], 2008. Disponível em: <http://java.sun.com/javase/technologies/core/basic/rmi/whitepaper/index.jsp#1>. Acesso em: 15 mar. 2013.

JDOM. **JDOM project**. [S.l.], 2012. Disponível em: <http://www.jdom.org/involved/index.html>. Acesso em: 12 fev. 2013.

LAUREANO. Marcos A. P. **Gestão de segurança da informação**. [S.l.], 2005. Disponível em: <http://pt.scribd.com/doc/100687499/18/Autenticacao-e-autorizacao>. Acesso em: 15 ago. 2012.

LIMA, Paulo R B D. **SGPCA – Sistema Gerenciador de Políticas de Controle de Acesso**. 2008, 91 f. Dissertação (Mestre em Engenharia de Produção) – Centro de Tecnologia Universidade Federal de Santa Maria, Santa Maria.

MACIEL, Rita S P; ASSIS, Semirames R. Middleware uma solução para desenvolvimento de aplicações distribuídas. **Revista CienteFico**, Salvador, ano 4, v. 1, p. 51 56, jan. 2004. Disponível em: <http://www4.fct.unesp.br/ronaldo/uploads/uma%20solucao%20para%20o%20desenvolvimento%20de%20aplica%C3%A7oes%20distribuidas.pdf >. Acesso em: 22 ago. 2012.

MARTINEZ, Marina. **Sistema de informação distribuído**. [S.l.], 2010. Disponível em <http://www.infoescola.com/informatica/sistema-de-informacao-distribuido>. Acesso em: 16 set. 2012.

MELLO, E R. et al. Segurança em serviços web. In: MINICURSOS DO SIMPÓSIO BRASILEIRO EM SEGURANÇA DA INFORMAÇÃO E SISTEMAS COMPUTACIONAIS (SBSeg/SBC), 6. , 2006, Santos. **Anais...** Santos: [S.l.], 2006, 48 f. Disponível em: <http://tele.sj.ifsc.edu.br/~mello/artigos/mellomcsbseg06.pdf>. Acesso em: 08 ago. 2012.

MYSQL. **Mysql workbench**. [S.l.]. 2012. Disponível em: <https://www.mysql.com/products/workbench/>. Acesso em: 13 ago. 2012.

OASIS. **eXtensible Access Control Markup Language (XACML).** [S.l.], 2005a. Disponível em: <http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf>. Acesso em: 01 ago. 2012.

_____. **eXtensible Access Control Markup Language (XACML).** [S.l.], 2005b. Disponível em: <http://docs.oasis-open.org/xacml>. Acesso em: 01 ago. 2012.

PIGATTO, Daniel F. **Estudo e implementação de uma solução de softwares aplicativos utilizando computação nas nuvens**. 2009. 101 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Universidade Regional Integrada do Alto Uruguai e das Missões, Erechim.

RODRIGUES, Gustavo V. **Disponibilização de serviços de segurança para sistemas distribuídos através de web services**. 2007. 47 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Centro de Ciências Exatas e Naturais Universidade Regional de Blumenau, Blumenau.

ROSSET, Valerio. **Um modelo de autorização e distribuição de direitos de acesso sobre conteúdos digitais**. 2004. 101 f. Dissertação (Mestre em Ciências da Computação) – Centro de Ciências Exatas Universidade Federal de Santa Catarina, Florianópolis.

SOUZA, Marcos T. **Controle de acesso para sistemas distribuídos**. 2010. 96 f. Dissertação (Mestre em Engenharia) – Escola Politécnica da Universidade de SP, São Paulo.

SUN MICROSYSTEMS. **Sun's XACML implementation**. [S.l.], 2006. Disponível em: <http://sunxacml.sourceforge.net >. Acesso em: 15 mar. 2013.

TANENBAUM, A S.; STEEN, M V. **Distributed system:** principles and paradigms. 2., Nova Jersey: Prentice Hall, 2006. 704 f. Disponível em: <http://www.inf.ufsc.br/~bertoni/download/Tanenbaum.pdf >. Acesso em: 05 out. 2012.

TOKTAR, Emir. **Controle de admissão de RSVP utilizando XACML.** 2003. 142 f. Dissertação (Mestre em Informática Aplicada) – Centro de Ciências Exatas e Tecnologia Pontifícia Universidade Católica do Paraná, Curitiba. Disponível em: <http://www.ppgia.pucpr.br/lib/exe/fetch.php?media=dissertacoes:2005:emir_toktar-2003.pdf>. Acesso em: 10 set. 2012.

TOKTAR, Emir; JAMHOUR, Edgard; SANTIN, A O. Uma arquitetura baseada em XML para políticas RSVP. In: SIMPÓSIO BRASILEIRO DE REDES DE COMPUTADORES, 23., 2005, Fortaleza. **Anais…**, Fortaleza: [S.l.], 2005, Não paginado.

WEB SERVICES ACTIVITY. **Web services activity**. [S.l.], 2002. Disponível em: <http://www.w3.org/2002/ws/>. Acesso em: 15 mar. 2013.

APÊNDICE A – Detalhamento de casos de uso

O Quadro 33 demonstra o detalhamento de uma criação, alteração e exclusão de um atributo externo gerado a partir de chamadas ao *middleware* feitas pelo software cliente.

Quadro 33 - Caso de Uso 02 – Manter atributos externos

UC06 – Manter atributos externos.	
Pré-condição	01) O software cliente deve ter uma sessão ativa, ou seja, efetuar a chamada do método `getSession()`. 02) O software cliente pode efetuar a chamada de método `getAtributosExternos()` de acordo com UC03, para obter sobre quais atributos poderá executar o UC06;
Cenário principal	03) Após executar o UC03 o *middleware* aguarda método de solicitação para manter os atributos externos; 04) Em caso de cadastro, o software cliente executa a chamada de função `cadastraAtributoExterno()`; 04.1) O *middleware* efetua o cadastro do atributo externo parametrizado no método citado no passo 04; 05) Em caso de exclusão, o software cliente executa a chamada de função `excluiAttrExterno()`; 05.1) O *middleware* efetua a exclusão do atributo externo parametrizado no método citado no passo 06 de acordo com o ID (referencia da chave primária da tabela do banco de dados); 06) Em caso de alteração, o software cliente executa a chamada de função `getAtributoExterno()`; 06.1) O *middleware* retorna um XML (Apêndice B) do atributo solicitado a ser alterado; 06.2) O software cliente executa a chamada de função `alteraAtributoExterno()` repassando um XML com os dados do atributo já alterados; 06.3) O *middleware* efetua alteração do atributo a partir de XML recebido;
Pós-condição	O *middleware* consiste as informações em banco de dados de acordo com a solicitação requerida.

O Quadro 34 demonstra o detalhamento de uma solicitação dos atributos externos cadastrados gerada a partir de chamadas ao *middleware* feitas pelo software cliente.

Quadro 34 - Caso de Uso 03 – Emitir lista de atributos externos

UC03 – Manter atributos externos.	
Pré-condição	01) O software cliente deve ter uma sessão ativa, ou seja, efetuar a chamada do método `getSession()`.
Cenário principal	02) O software cliente efetua a chamada de método `getAtributosExternos()`; 03) O *middleware* verifica se existem atributos externos para sujeito ou recurso (de acordo com parâmetro do método do passo 02) e monta XML de resposta ao software cliente (Apêndice B);
Pós-condição	Se existirem atributos externos cadastrados o software cliente receberá um XML com dados dos atributos externos, senão o *middleware* retornará um XML com suas *tags* vazias;

O Quadro 35 demonstra o detalhamento de uma solicitação das políticas cadastradas gerada a partir de chamadas ao *middleware* feitas pelo software cliente.

Quadro 35 - Caso de Uso 07 – Emitir lista de políticas

UC07 – Emitir lista de políticas.	
Pré-condição	01) O software cliente deve ter uma sessão ativa, ou seja, efetuar a chamada do método `getSession()`.
Cenário principal	02) O software cliente efetua a chamada de método `getPoliticasDiretorio()`; 03) O *middleware* verifica se existem políticas cadastradas no diretório padrão ou diretório configurado e monta XML de resposta ao software cliente (Apêndice B);
Pós-condição	Se existirem políticas cadastradas o software cliente receberá um XML com as identificações das políticas cadastradas, senão o *middleware* retornará um XML com suas *tags* vazias;

No Quadro 36 é apresentado o detalhamento de uma exclusão e alteração de uma política gerada a partir de chamadas ao *middleware* feitas pelo software cliente.

Quadro 36 - Caso de Uso 01 – Manter políticas

UC01 – Manter políticas.	
Pré-condição	01) O software cliente deve ter uma sessão ativa, ou seja, efetuar a chamada do método `getSession()`.
Cenário principal	02) O software cliente executa o UC07 e o *middleware* aguarda método de solicitação para manter as políticas; 03) Em caso de exclusão, o software cliente executa a chamada de função `excluirPolitica()`; 03.1) O *middleware* efetua a exclusão do arquivo com identificação da política que foi solicitada a ser excluída; 04) Em caso de alteração, o software cliente executa a chamada de função `getPoliticaAltera()`; 04.1) O *middleware* verifica se existe política com a identificação solicitada cadastrada no diretório padrão ou diretório configurado e monta XML de resposta ao software cliente com os dados da política solicitada(Apêndice B); 04.2) O software cliente efetua a chamada de método `alteraPolitica()` parametrizando o XML com os dados alterados; 04.3) O *middleware* altera os dados da política de acordo com o XML recebido; 04.4) O software cliente efetua a chamada do método `getNextRegraAltera();` 04.5) O *middleware* verifica se existem regras cadastradas na política solicitada e monta XML de resposta ao software cliente com os dados da regra solicitada(Apêndice B); 04.6) O software cliente efetua a chamada do método `alteraRegra()` parametrizando o XML com os dados alterados; 04.7) O *middleware* altera os dados da regra de acordo com o XML recebido; 04.8) O software cliente efetua a chamada de método `finalizaAlterPolitica();` 04.9) O *middleware* efetua a alteração do arquivo XML da política alterada com os novos dados;
Cenário alternativo 1	No passo 04.1, o software cliente pode alterar o alvo da política; 04.1.1) O software cliente solicita ao middleware o XML para alteração efetuando as chamadas aos métodos: `getNextSujeitoAltera(), getNextRecursoAltera()` e `getNextAcaoAltera();` 04.1.2) O software cliente efetua a chamada dos métodos `alteraSujeito()`, `alteraRecurso()`ou `alteraAcao()`parametrizando o XML com os dados alterados do sujeito, recurso ou ação para posterior alteração dos atributos; 04.1.3) O *middleware* efetua a alteração no sujeito, recurso ou ação de acordo com os dados recebidos no XML;

Cenário alternativo 2	No passo 04.5, o software cliente pode alterar o alvo da regra; 04.5.1) O software cliente solicita ao middleware o XML para alteração efetuando as chamadas aos métodos: `getNextSujeitoAltera()`, `getNextRecursoAltera()` ou `getNextAcaoAltera()`; 04.5.2) O software cliente efetua a chamada dos métodos `alteraSujeito()`, `alteraRecurso()`ou `alteraAcao()`parametrizando o XML com os dados alterados do sujeito, recurso ou ação para posterior alteração dos atributos; 04.1.3) O *middleware* efetua a alteração no sujeito, recurso ou ação de acordo com os dados recebidos no XML;
Pós-condição	O *middleware* atualiza a `PolicySet` “Política de Avaliação” de acordo com políticas cadastradas.

APÊNDICE B – Estrutura e sintaxe XML

No Apêndice B serão demonstradas as estruturas XML utilizadas pelo *middleware* para retornar ao software cliente dados referentes a políticas, regras, alvos, configurações de diretórios e base de dados, alteração de atributos externos entre outros. A sintaxe apresentada é a mesma no qual o *middleware* deve receber do software cliente para alteração de dados, conforme mostrado nos quadros abaixo.

Quadro 37 - XML retorno da chamada de função `getPoliticasDiretorio()`

```
<Politicas>
      <Politica>Politica_01</Politica>
      <Politica>Politica_02</Politica>
      <Politica>Politica_03</Politica>
</Politicas>
```

Quadro 38 - XML retorno da chamada de função `getPoliticaAltera()`

```
<Politica>
      <ID>Politica_01</ID>
      <Descricao>Teste de XML Politica_01</Descricao>
      <Algoritmo_Combinacao>0</Algoritmo_Combinacao>
</Politica>
```

Quadro 39 - XML retorno da chamada de função `getNextSujeitoAltera()`

```
<Sujeito>
    <ID>jailson</ID>
    <Atributos>
        <Atributo>
            <Identificacao>email</Identificacao>
            <Valor>secf.com</Valor>
            <Funcao>=domainEmail</Funcao>
        </Atributo>
    </Atributos>
</Sujeito>
```

Quadro 40 - XML retorno da chamada de função `getNextRecursoAltera()`

```
<Recurso>
    <ID>arquivo</ID>
    <Atributos>
        <Atributo>
            <Identificacao>uri</Identificacao>
            <Valor>//C:/ArquivosdePrograma s/MDD</Valor>
            <Funcao>=URI</Funcao>
        </Atributo>
        <Atributo>
            <Identificacao>tamanho</Identificacao>
            <Valor>1.512</Valor>
            <Funcao>=Double</Funcao>
        </Atributo>
    </Atributos>
</Recurso>
```

No Quadro 39 e no Quadro 40, caso o sujeito ou recurso não contiverem atributos complementares a tag Atributos não existirá nos elementos Sujeito ou Recurso.

Quadro 41 - XML retorno da chamada de função `getNextAcaoAltera()`

```
<Acao>
    <ID>Ler</ID>
</Acao>
```

Quadro 42 - XML retorno da chamada de função `getNextRegraAltera()`

```
<Regra>
       <ID>Regra_01</ID>
       <Descricao>Regra_teste_XML</Descricao>
       <Efeito>1</Efeito>
       <Condicao>
              <Operador>E<Operador/>
              <Expressao>
                     <Membro>0</Membro>
                     <Identificacao>group</Identificacao>
                      <Valor_Comparacao>owner</Valor_Comparacao>
                     <Funcao>=</Funcao>
              </Expressao>
              <Expressao>
                     <Membro>1</Membro>
                     <Identificacao>tamanho</Identificacao>
                      <Valor_Comparacao>2024</Valor_Comparacao>
                     <Funcao>>=</Funcao>
              </Expressao>
       </Condicao>
</Regra>
```

No Quadro 42 caso a *tag* `Condicao` contenha apenas uma expressão a *tag* `Operador` terá seu valor vazio (`<Operador/>`), ou se a regra não contiver condição não existirá a *tag* `Condicao` na *tag* `Regra.`

Quadro 43 - XML retorno da chamada de função `getAtributosExternos()`

```
<Informacoes>
       <AtributoInfo>
              <ID>1</ID>
              <VlrEnt>jailson</VlrEnt>
              <IdentSai>group</IdentSai>
              <VlrSai>owner</VlrSai>
              <Desig>0</Desig>
       </AtributoInfo>
       <AtributoInfo>
              <ID>3</ID>
              <VlrEnt>jailson</VlrEnt>
              <IdentSai>email</IdentSai>
              <VlrSai>jailson@secf.com</VlrSai>
              <Desig>0</Desig>
       </AtributoInfo>
</Informacoes>
```

Quadro 44 - XML retorno da chamada de função `getXMLConfiguracoes()`

```
<Configs>
       <Algoritmo_Padrao>0<Algoritmo_Padrao/>
       <Paths>
              <PathPolicy>C://<PathPolicy/>
              <PathRequests>C://<PathRequests/>
              <PathResponses>C://<PathResponses/>
              <PathBKP/><PathBKP/>
       </Paths>
</Configs>
```

No Quadro 44, caso o *middleware* esteja utilizando as configurações de diretórios e algoritmo de combinação padrão o XML será gerado porém com as *tags* vazias ou seja: `<Algoritmo_Padrao/>`,`<PathPolicy/>`,`<PathRequests/>`,`<PathResponses/>`,`<PathBKP/>`.

Quadro 45 - XML retorno da chamada de função `getConfigBD()`

```
<Config>
    <URL>//localhost/controle<URL/>
    <User>root<User/>
    <Password>root<Password/>
<Config/>
```

No Quadro 45, caso o *middleware* não tenha recebido nenhuma configuração de banco de dados o retorno será vazio, ou seja, não retornará nenhuma *tag* apenas uma *`String`* vazia.

APÊNDICE C – Dicionário de dados

O dicionário de dados é exibido no Quadro 46, ele descreve o nome da tabela utilizada no *middleware* implementado, os campos e uma breve descrição. A seguir o detalhamento da tabela `AttributePIPObj` utilizada para cadastro de atributos externos.

Quadro 46 - Dicionário de dados – tabela `AttributePIPObj`

Tabela: ATTRIBUTEPIPOBJ			
Nome da Coluna	**Tipo de Dados**	**Comentários**	**Chave**
ID	Int	Identificação Atributo Externo	Sim
DESIGTYPE	Varchar(80)	Membro do atributo (Sujeito = 0 ou Recurso = 1)	
VALUE	Varchar(80)	Valor do atributo do sujeito ou recurso do *request*	
IDATTR	Varchar(80)	Identificação do valor do atributo de retorno	
ATTRVALUE	Varchar(80)	Valor do atributo de retorno	
TYPE	Varchar(80)	Tipo de dados do valor de retorno (de acordo com sintaxe XACML)	

APÊNDICE D – Recursos suportados

Os recursos suportados são exibidos no Quadro 47 que descreve os recursos que foram adicionados a implementação do *middleware* para apresentar os tipos de atributos XACML suportados, os algoritmos de combinação utilizados, os valores retornados em uma solicitação de acesso e as sintaxes utilizadas pelo *middleware* para formalizar uma maneira do software cliente não necessitar do conhecimento sobre o XACML.

Quadro 47 - Recursos suportados

Padrão XAML	**Sintaxe *Middleware***
Algoritmos de combinação Regras	
DenyOverridesRuleAlg	0
FirstApplicableRuleAlg	1
PermitOverridesRuleAlg	2
Algoritmos de combinação Politicas	
DenyOverridesPolicyAlg	0
FirstApplicablePolicyAlg	1
OnlyOneApplicablePolicyAlg	2
PermitOverridesPolicyAlg	3
Tipos de Atributos	
StringAttribute	String
AnyURIAttribute	String com formatação URI
DateAttribute	Date
DoubleAttribute	Double
IntegerAttribute	Integer
RFC822NameAttribute	String com formatação email
Funções de comparação de valores de atributos do alvo	
urn:oasis:names:tc:xacml:1.0:function:string-equal	=String
urn:oasis:names:tc:xacml:1.0:function:regexp-string-match	=startString
urn:oasis:names:tc:xacml:1.0:function:rfc822Name-match	=domainEmail
urn:oasis:names:tc:xacml:1.0:function:integer-equal	=Integer
urn:oasis:names:tc:xacml:1.0:function:double-equal	=Double
urn:oasis:names:tc:xacml:1.0:function:date-equal	=Data
urn:oasis:names:tc:xacml:1.0:function:rfc822Name-equal	=Email
urn:oasis:names:tc:xacml:1.0:function:anyURI-equal	=URI

Funções de comparação de valores de atributos da condição	
Tipo "Data": urn:oasis:names:tc:xacml:1.0:function:date-greater-than	>
Tipo "Data": urn:oasis:names:tc:xacml:1.0:function:date-greater-than-or-equal	>=
Tipo "Data": urn:oasis:names:tc:xacml:1.0:function:date-less-than	<
Tipo "Data": urn:oasis:names:tc:xacml:1.0:function:date-less-than-or-equal	<=
Tipo "Data": urn:oasis:names:tc:xacml:1.0:function:date-equal	=
Tipo "Email": urn:oasis:names:tc:xacml:1.0:function:rfc822Name-equal	=
Tipo "Integer": urn:oasis:names:tc:xacml:1.0:function:integer-greater-than	>
Tipo "Integer": urn:oasis:names:tc:xacml:1.0:function:integer-greater-than-or-equal	>=
Tipo "Integer": urn:oasis:names:tc:xacml:1.0:function:integer-less-than	<
Tipo "Integer": urn:oasis:names:tc:xacml:1.0:function:integer-less-than-or-equal	<=
Tipo "Integer": urn:oasis:names:tc:xacml:1.0:function:integer-equal	=
Tipo "Double": urn:oasis:names:tc:xacml:1.0:function:double-greater-than	>
Tipo "Double": urn:oasis:names:tc:xacml:1.0:function:double-greater-than-or-equal	>=
Tipo "Double": urn:oasis:names:tc:xacml:1.0:function:double-less-than	<
Tipo "Double": urn:oasis:names:tc:xacml:1.0:function:double-less-than-or-equal	<=
Tipo "Double": urn:oasis:names:tc:xacml:1.0:function:double-equal	=
Tipo "String": urn:oasis:names:tc:xacml:1.0:function:string-equal	=
Tipo "URI": urn:oasis:names:tc:xacml:1.0:function:anyURI-equal	=
Operadores Lógicos das condições	
urn:oasis:names:tc:xacml:1.0:function:and	E
urn:oasis:names:tc:xacml:1.0:function:or	OU
Decisões de autorização	
Permit	Acesso Permitido
Deny	Acesso Negado
Not Applicable	Acesso Negado
Indeterminate	Acesso Negado

APÊNDICE E – Explanação dos Métodos da classe `FacadeAccessControlMDW`

A explanação dos métodos da classe `FacadeAccessControlMDW` é mostrada no Apêndice E para especificar quais os métodos que poderão ser utilizados pelo software cliente que utilizar o *middleware*.

Os referidos métodos com especificação de parâmetros e utilização são:

a) método `getSession()`: método executado no início das chamadas de função pelo software cliente que retorna uma nova sessão (`Integer`). A partir desta execução todas as chamada posteriores serão efetuadas parametrizando o valor da sessão. Esta sessão controla os objetos instanciados no *WS* para cada instancia software cliente;

b) método `createPolicy(String: idPolicy, descPolicy, int: algCombId, session)`: método executado para efetuar criação de nova política. Nele é parametrizado a identificação da política, uma descrição, o algoritmo de combinação para avaliação de regras (0 = `DenyOverridesRuleAlg`; 1 = `FirstApplicableRuleAlg`; 2 = `PermitOverridesRuleAlg`) conforme explicado na fundamentação teórica e a sessão da aplicação cliente;

c) métodos `createTargetSubject`, `createTargetResource`, `createTargetAction (String valor, int session)`: métodos utilizados para criar sujeit, recurso e ação para o alvo da política ou regra em criação. Estes métodos são parametrizados apenas com o valor do sujeito, recurso ou ação e com a sessão da aplicação cliente;

d) método `addAttributesTarget(int memberTarg, String: funcao, valor, identificacao, int session)`: método utilizado para criar atributos adicionais ao sujeito ou recurso do alvo criados a partir dos métodos acima (`createTargetSubject` e `createTargetResource`). Este método é parametrizado com o membro do alvo que deve ser adicionado o atributo (0 = sujeito e 1 = recurso), com uma função de comparação de valores para ser utilizada em comparações com o *request* (Apêndice D) (`=String` utilizado para comparação de igualdade entre `strings`, `=startString` compara o inicio de `strings`, `=domainEmail` compara valores endereços de email a partir de igualdade de domínios, `=Integer` comparação de igualdade de números inteiros, `=Double` comparação de igualdade de números reais, `=Data` comparação de igualdade de

data, `=Email` comparação de igualdade de email, `=URI` comparação de igualdade de identificadores uniformes de recursos), valor do atributo, identificação do valor do atributo, e sessão da aplicação cliente;

e) método `addTargetInPolicy(int session)`: método utilizado para atribuir o alvo criado a política criada. Este método é parametrizado apenas com a sessão da aplicação cliente;

f) método `createRule(String: idRule, int effect, String desc, int session)`: método utilizado uma nova regra a política. Nele é parametrizado a sua identificação, efeito (0 = Permitir, 1 = Negar), descrição e a sessão da aplicação cliente;

g) método `addTargetInRule(int session)`: método utilizado para atribuir o alvo criado a regra criada. Este método é parametrizado apenas com a sessão da aplicação cliente;

h) método `createConditiontRule(int member, String: ident, funcao, valor_Comparacao, operadorLogico, int session)`: método utilizado para criar uma condição para a regra criada. Este método é parametrizado com o membro atributo da regra que se deseja efetuar a condição (0 = sujeito, 1 = recurso, 3 = ambiente), a identificação do valor de atributo a ser comparado, a função de comparação (Apêndice D), o valor do atributo, o operador lógico da condição (‘E’ ou ‘OU’ pois pode ser uma condição com duas expressões) e a sessão da aplicação cliente;

i) método `addConditionInRule(int session)`: método utilizado para atribuir a condição criada a regra criada. Este método é parametrizado apenas com a sessão da aplicação cliente;

j) método `addRuleInPolicy(int session)` : método utilizado para atribuir a regra criada a política criada. Este método é parametrizado apenas com a sessão da aplicação cliente;

k) método `finalizaCreatePolicy(int session)` : método utilizado para finalizar a criação da política e salvá-la em diretório, caso não seja executado nenhuma criação será feita. Este método é parametrizado apenas com a sessão da aplicação cliente;

l) método `getPoliticasDiretorio(int session)`: método que retorna XML com identificações das políticas cadastradas no diretório. Este método é parametrizado apenas com a sessão da aplicação cliente;

m) método `excluirPolitica(String nome, int session)`: método para excluir uma política do diretório a partir de parâmetro de identificação da política e a sessão da aplicação cliente;

n) método `getPoliticaAltera(String idPolitica, int session)`: método retorna um XML com a política solicitada de acordo com parâmetro de identificação e sessão da aplicação cliente. A sintaxe do XML recebido é a mesma que deverá ser enviada no método de alteração e que será exemplificado no Apêndice B;

o) método `alteraPolitica(String xmlPolitica, int session)`: método utilizado para alterar a política solicitada. Este método é parametrizado com o XML da política (Apêndice B) e com a sessão da aplicação cliente;

p) método `getNextRegraAltera(int session)`: método que retorna XML da regra da política que foi solicitada para alteração, levando em consideração sempre a próxima regra, no caso de encontrar a última reinicia-se do início. A sintaxe do XML recebido é a mesma que deverá ser enviada no método de alteração e que será exemplificado no Apêndice B. Este método é parametrizado apenas com a sessão da aplicação cliente;

q) método `alteraRegra(String xmlRegra, int session)` método utilizado para alterar a regra solicitada. Este método é parametrizado com o XML da regra (Apêndice B) e com a sessão da aplicação cliente;

r) métodos `getNextSujeitoAltera`, `getNextRecursoAltera` e `getNextAcaoAltera (int session)`: métodos que retornam XML com sujeito, recurso e ação do alvo da política ou regra que foi solicitada para alteração. O método retorna sempre o próximo sujeito, recurso ou ação do alvo, no caso de encontrar o último reinicia-se do início. A sintaxe do XML recebido é a mesma que deverá ser enviada no método de alteração e que será exemplificado no Apêndice B. Este método é parametrizado apenas com a sessão da aplicação cliente;

s) métodos `alteraSujeito`, `alteraRecurso` e `alteraAcao (String XML, int session)`: métodos que alteram sujeito, recurso ou ação de acordo com ordem de

solicitação dos métodos citados acima. Este método é parametrizado com o XML do sujeito, recurso ou ação (Apêndice B) e com a sessão da aplicação cliente;

t) método `finalizaAlterPolitica(int session)`: utilizado para finalizar a alteração da política solicitada, caso não seja executado nenhuma alteração será feita. Este método é parametrizado apenas com a sessão da aplicação cliente;

u) método `clearMemoryAll(int session)`: método criado para limpar toda e qualquer alteração, criação ou requisição feita ao *middleware* e destruir objetos. Utilizado também no caso de erro lançado pelo *middleware*;

v) método `initRequisicao(String: sujeito, recurso, acao, int session)`: método utilizado para iniciar uma solicitação de acesso. Nele é parametrizado o sujeito, o recurso, a ação e a sessão da aplicação cliente;

w) métodos `requisitaSujeitoPEP`,`requisitaRecursoPEP` e `requisitaAmbientePEP(String ident, valor, int session)`: métodos utilizados para adicionar atributos ao sujeito ou recurso da requisição e também valores de atributos ambiente. Neste métodos são parametrizados a identificação do valor de comparação, o valor de comparação e a sessão da aplicação cliente;

x) método `finalizaRequisicao(int session)`: método utilizado para finalizar uma solicitação de acesso. Neste método é parametrizado apenas a sessão da aplicação cliente e retorna uma resposta a solicitação que pode ser “Acesso Permitido” ou “Acesso Negado”;

y) métodos `getJustificativaPEP` e `getRecursoDecisaoPEP (int session)`: estes métodos são utilizado para retornar maiores detalhes da avaliação de solicitação, como: recurso requerido e o status de execução da avaliação das políticas no momento da finalização de requisição. Neste método é parametrizado apenas a sessão da aplicação cliente;

z) método `cadastraAtributoExterno(int member, String: valorRequisitante, ident, valorIdent, int session)`: método utilizado para efetuar o cadastro de atributos externos, ou seja, atributos que não vão estar presentes nas solicitações de acesso (*request*). Neste método são parametrizados os atributos de membro do atributo (0 = sujeito e 1 = recurso), valor recebido pelo atributo do *request* (valor do sujeito ou recurso) que vai ser utilizado para efetuar a procura do atributo externo, identificação do valor do atributo externo, valor do atributo externo, e sessão da aplicação cliente;

aa) método `getAtributosExternos(int desig, int session)`: método que retorna XML com os atributos externos cadastrados; A sintaxe do XML recebido será exemplificado no Apêndice B. Este método é parametrizado com o membro do atributo (0 = sujeito e 1 = recurso) e com a sessão da aplicação cliente;

bb) método `getAtributoExterno(int id, int session)`: método que retorna XML com o atributo externo solicitado a partir do parâmetro ID (identificação do banco de dados retornado a partir do método `getAtributosExternos()`) e sessão da aplicação cliente. A sintaxe do XML recebido será exemplificada no Apêndice B e será a mesma utilizada para alteração do atributo;

cc) método `alteraAtributoExterno(String xml, int session)`: método que efetua a alteração do atributo externo. Este método é parametrizado com o XML do atributo externo (Apêndice B) e com a sessão da aplicação cliente;

dd) método `excluiAttrExterno(int id, int session)`: método utilizado para excluir atributo externo a partir do parâmetro ID (identificação do banco de dados retornado a partir do método `getAtributosExternos()`) e sessão da aplicação cliente;

ee) método `setConfigsBD(String url, String user, String pass)`: este método é utilizado para configurar um banco de dados utilizado para o cadastro dos atributos externos caso a aplicação cliente não queira utilizar banco padrão do WS;

ff) método `getConfigBD()`: método que retorna as configurações de banco de dados a partir de XML (Apêndice B);

gg) método `getXMLConfiguracoes(int session)`: caso não seja utilizada a configuração padrão, este método retorna XML de configurações como diretório de cadastro de políticas e Beckup, requisições e respostas, e algoritmo de combinação padrão para análise de políticas. Neste método é apenas parametrizado a sessão da aplicação cliente. A sintaxe do XML recebido será exemplificada no Apêndice B;

hh) método `alteraXMLConfiguracoes(String xmlConfig, int session)`: método utilizado para alterar as configurações de diretório de cadastro de políticas e Beckup, requisições e respostas, e algoritmo de combinação padrão para análise de políticas. A sintaxe deste XML é a mesma gerada pelo método `getXMLConfiguracoes()`e que será exemplificada no Apêndice B.