UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

INSTITUTO DE INFORMÁTICA

PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

ESTÊVÃO MIGUEL ZANETTE ROHR

Segurança em Gerenciamento de Redes Baseado em Web Services

Dissertação apresentada como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação

Profa. Dra. Liane Margarida R. Tarouco
Orientadora

Porto Alegre, abril de 2009

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Rohr, Estêvão Miguel Zanette

Segurança em Gerenciamento de Redes Baseado em Web Services / Estêvão Miguel Zanette Rohr – Porto Alegre: Programa de Pós-Graduação em Computação, 2009.

96 p.:il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação. Porto Alegre, BR – RS, 2009. Orientadora: Liane Margarida R. Tarouco.

1. Gerenciamento de redes. 2. Web Services. 3. Segurança. 4. WS-Security. 5. XACML I. Tarouco, Liane Margarida R.. II. Título.

AGRADECIMENTOS

Agradeço à professora Liane Margarida R. Tarouco, pela oportunidade de realizar este trabalho e pelo incentivo com relação ao tema abordado.

Agradeço ao professor Lisandro Granville, pelo auxílio ao longo deste trabalho, sempre tendo consideração em relação ao meu foco dividido entre as responsabilidades profissionais e acadêmicas, e ajudando sempre que solicitado, desde antes da inscrição no mestrado, em que tivemos uma reunião filosófica muito proveitosa, até a conclusão desta dissertação.

Gostaria de agradecer muito ao amor da minha vida, a Fê, por estar sempre ao meu lado e trazer equilíbrio à minha vida. Por entender o tempo que precisei dedicar ao mestrado, e também pelas vezes em que ela me motivou a ir pro computador trabalhar na dissertação quando eu não estava com muita vontade. Te amo!

Agradeço também à minha família, em especial à minha mãe Glória Maria, por me incentivar a realizar o mestrado, ao qual não é dado o devido valor, ao menos fora das instituições acadêmicas, hoje em dia. A decisão final de iniciar o mestrado foi minha, até porque se fosse por minha mãe eu teria começado antes, mas certamente o incentivo dela ajudou nessa decisão, e sempre que eu me questionava se havia decidido corretamente em realizar o mestrado. Agora, na fase de finalização deste trabalho, posso dizer a ela e a todos que sinto-me muito satisfeito em ter cursado o mestrado e com o trabalho que desenvolvi.

Pela seguinte afirmação contrariar minhas expectativas iniciais, faço questão de ressaltar: valeu a pena!

SUMÁRIO

LISTA DE ABREVIATURAS

API	Application Programming Interface
AXIOM	Axis Object Model
CRUD	Create, Read, Update and Delete
DMTF	Distributed Management Task Force
HTTP	Hypertext Transfer Protocol
HTTPS	HTTP Secure (ou HTTP over SSL)
IETF	Internet Engineering Task Force
IP	Internet Protocol
JAXB	Java Architecture for XML Binding
JAX-WS	Java API for XML Web Services
JDK	Java Development Kit
JMX	Java Management Extensions
JRE	Java Runtime Environment
JSR	Java Specification Request
JVM	Java Virtual Machine
JWSDP	Java Web Services Developer Pack
MEP	Message Exhange Pattern
MIB	Management Information Base
MOWS	Management of Web Services
MUWS	Management Using Web Services
OASIS	Organization for the Advancement of Structured Information Standards
OID	Object Identifier
PDP	Policy Decision Point
PDU	Protocol Data Unit
PEP	Policy Enforcement Point
PKI	Public Key Infrastructure
RFC	Request for Comments
RMON	Remote Network Monitoring
SAAJ	SOAP with Attachments API for Java
SAML	Security Assertion Markup Language
SGMP	Simple Gateway Monitoring Protocol
SMI	Structure of Management Information

SMP	Simple Management Protocol
SNMP	Simple Network Management Protocol
SNMP4J	SNMP API for Java
SOAP	Simple Object Access Protocol
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDDI	Universal Description, Discovery, and Integration
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
USM	User-Based Security Model
VACM	View-Based Access Control Model
W3C	World Wide Web Consortium
WS	Web Services
WSDL	Web Services Description Language
WSDM	Web Services Distributed Management
WSRF	Web Services Resource Framework
WSS4J	Web Services Security for Java
XACML	Extensible Access Control Markup Language
X-KISS	XML Key Information Service Specification
XKMS	XML Key Management Specification
X-KRSS	XML Key Registration Service Specification
XML	Extensible Markup Language
XrML	Extensible Rights Markup Language

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

A área de gerência de redes encontra uma série de desafios desde seu príncipio. O protocolo que surgiu como padrão para gerência de redes, o SNMP, possui uma série de limitações, por exemplo, no tocante à segurança, configuração de equipamentos e composição de serviços. Por essa razão, tecnologias alternativas para o gerenciamento de redes têm sido pesquisadas. A tecnologia de Web Services surgiu como forte alternativa, por características como o uso de padrões amplamente suportados (HTTP e XML) e modelo de desenvolvimento orientado a serviços. Pesquisas iniciais demonstraram que os Web Services são uma alternativa viável em termos de desempenho. Assim, o uso de Web Services em áreas específicas de gerência de redes, como notificações e gerência por delegação, tem sido pesquisado. Porém, há carência de estudos sobre o uso de segurança no gerenciamento de redes via Web Services. Os Web Services trazem facilidade para uso de segurança, que é vital para a gerência de redes, e este é o foco deste trabalho. É proposta uma arquitetura de integração de segurança à comunicação de mensagens de gerenciamento de redes via Web Services. Para isso, foram utilizados o padrão WS-Security, para segurança em Web Services, e o padrão WS-Management, para gerenciamento de redes via Web Services. Também foi integrado controle de acesso à arquitetura, com uso do padrão XACML. Uma avaliação de desempenho foi realizada para verificar o impacto do uso de segurança, e comparações com SNMPv3 foram realizadas na solução de controle de acesso via XACML. Os testes mostram que, como é tradicional, a segurança tem impacto considerável no tempo de processamento e tráfego na rede. Porém, a arquitetura e implementação realizadas comprovam que, também na área de segurança, a tecnologia de Web Services tem aplicação eficaz para o gerenciamento de redes.

Palavras-chave: gerenciamento de redes, SNMP, Web Services, segurança, segurança de redes, segurança de Web Services, controle de acesso, WS-Security, XACML, WS-Management, VACM

Security in Web Services-based Network Management

ABSTRACT

The network management field has several challenges since its beginning. The standard protocol for network management, SNMP, has many drawbacks, related to security, device configuration, and service composition. For these reason, alternative technologies for network management have been investigated. Web Services technology emerged as a strong solution, due to advantages such as employing widely supported standards (HTTP and XML) and service-oriented development model. The first performed investigations in the area showed that Web Services are a valid alternative to SNMP in terms of performance. Thus, Web Services usage in specific areas of network management, such as notifications and management by delegation, have been researched. However, there are currently no studies on security aspects of Web Services-based network management. Web Services enable easy integration of security, which is mandatory for network management, and this is the main goal of this work. An architecture is proposed for security integration in a network management message communication using Web Services. The standards used in this architecture were WS-Security, which enables security in Web Services, and WS-Management, which targets Web Services-based network management. Access control integration was also developed, using XACML standard. A performance evaluation was carried out in order to verify security usage impact, and comparisons with SNMPv3 were performed in XACML access control solution. Tests showed that, as expected, security has a considerable impact in processing time and network traffic. However, the architecture and implementation show that, also in the security area, the Web Services technology has effective aplication in network management.

Keywords: network management, SNMP, Web Services, security, network security, Web Services security, access control, WS-Security, XACML, WS-Management, VACM

1 INTRODUÇÃO

As redes de computadores são hoje uma ferramenta essencial para a sociedade. A comunicação entre pessoas físicas e jurídicas em todo o mundo se dá através dessas redes. Grande parte dos serviços fornecidos por qualquer empresa dependem da infra-estrutura de comunicação fornecida por essas redes. O gerenciamento dessas redes é de fundamental importância para a manutenção e melhoria destes serviços, tornando a área de gerência de redes alvo de vasta pesquisa tecnológica na indústria e na comunidade científica.

A área de gerência de redes enfrenta uma série de desafios desde o seu princípio. A diversidade de recursos gerenciáveis (em geral, equipamentos de rede) tornou necessária a definição de protocolos de gerenciamento padronizados, permitindo uma maior integração entre as redes gerenciadas e as aplicações de gerenciamento. Essa necessidade gerou trabalhos que culminaram na criação do SNMP (*Simple Network Management Protocol*) (CASE, 1990), padronizado pelo IETF (*Internet Engineering Task Force*), e que se tornou o padrão de fato para gerência de redes, sendo hoje suportado em uma ampla variedade de equipamentos de rede. Com o passar do tempo, o protocolo foi otimizado, por exemplo, com a agregação de funções de segurança, no SNMPv3 (HARRINGTON, 2002).

Mesmo sendo um protocolo padronizado, há uma série de problemas de gerenciamento de redes para os quais o SNMP não é uma solução adequada ou otimizada. As áreas de gerência por delegação, configuração de equipamentos, composição de serviços e a própria segurança apresentam problemas. Apesar destas áreas serem pesquisadas e melhoradas continuamente em relação ao SNMP, as limitações existentes têm levado a comunidade científica a investigar tecnologias alternativas para a gerência de redes, já que os problemas do SNMP propiciaram o surgimento de diversos padrões proprietários, desenvolvidos por empresas que precisavam resolver adequadamente questões problemáticas no SNMP.

Recentemente, o conjunto de tecnologias denominado Web Services (CURBERA, 2002), padronizado pelo W3C (*World Wide Web Consortium*), inicialmente direcionado para o comércio eletrônico (*e-business*), tornou-se extremamente popular em uma gama variada de aplicações. Isso se deve a uma série de características destas tecnologias. Por ter seu transporte baseado nos mesmos protocolos em que a Internet é baseada, como HTTP e HTTPS, os Web Services podem usufruir de toda a infra-estrutura de segurança disponível nestes protocolos, além de padrões de segurança próprios para Web Services. Outro ponto forte dos Web Services é a sua

independência de plataforma, já que a tecnologia é baseada em padrões largamente aceitos, como XML (*eXtensible Markup Language*) (BRAY, 2006a) e SOAP (*Simple Object Access Protocol*) (GUDGIN, 2007). Além disso, o modelo de desenvolvimento dos Web Services, baseado em XML e com a possibilidade de uso amplo de facilidades de orientação de objetos, simplifica a criação de serviços mais elaborados e a integração de sistemas complexos e diversificados, permitindo que aplicações com diferentes funcionalidades possam ser integradas com o objetivo de construir soluções mais completas.

Todas essas vantagens dos Web Services podem ser diretamente aplicadas na área de gerência de redes, fazendo com que naturalmente a idéia de padronizar Web Services para gerência tenha ganho força e esteja sendo alvo de grande quantidade de pesquisas. O maior receio inicial da utilização de Web Services em gerência de redes foi em relação ao desempenho: por ser baseado em XML, notadamente um protocolo verboso, haviam desconfianças quanto a uma queda grande de desempenho em relação ao SNMP. Assim, os primeiros estudos focaram na comparação de desempenho entre os Web Services e o SNMP. Esses estudos mostraram que, para um número grande de informações a serem recuperadas, e com uso de algoritmos de compressão, os Web Services têm um desempenho similar ou superior ao SNMP.

Partindo das conclusões satisfatórias a respeito do desempenho dos Web Services em gerência de redes, torna-se necessária a avaliação do impacto do uso de Web Services no desempenho de áreas específicas de gerência. Algumas dessas áreas já foram pesquisadas em trabalhos anteriores, como gerência por delegação, composição de serviços e notificações. Porém, uma área crítica para as aplicações de gerência de redes ainda não foi abordada: a segurança. É nesta área que está focado esse trabalho, com o objetivo de analisar o desempenho do uso de padrões de segurança baseados em Web Services no gerenciamento de redes. Para isso, serão realizadas a definição de uma arquitetura e sua implementação, para que testes de desempenho possam ser efetuados. A área de segurança foi escolhida por duas razões: sua relevância no contexto da gerência de redes, e as deficiências de segurança existentes no SNMP.

Quanto à relevância, a segurança é um aspecto central para qualquer solução real de gerência de redes, implicando em dois pontos distintos: sistemas de gerência devem ser capazes de gerenciar a segurança da rede, e também devem ser eles próprios seguros. Uma aplicação de gerência com falhas de segurança poderia dar acesso a recursos de rede para usuários maliciosos, que poderiam realizar tanto ações indevidas simples como aumentar sua cota de impressão, quanto drásticas, como interromper serviços e conexões de rede.

Quanto ao SNMP, a segurança é uma de suas principais deficiências. A falta de segurança no protocolo foi um dos motivos que fizeram com que fabricantes desenvolvessem protocolos proprietários para obter a segurança necessária. Dessa forma, o SNMPv3, que adiciona uma série de aspectos de segurança, não foi popularizado, devido ao legado que os fabricantes têm nas suas soluções proprietárias. Por essas razões, o SNMP é empregado basicamente para funções de monitoramento de redes (SCHÖNWALDER, 2007), já que os riscos de uma falha de segurança nestas funções são baixos, ou mesmo inexistentes.

Devido à criticidade da segurança no gerenciamento, o esforço de padronizar Web Services como protocolo de gerência de redes vem acompanhado da clara

necessidade de uma arquitetura consistente de segurança. Dentre a vasta família de padrões Web Services, há um conjunto de padrões que agregam aos Web Services os principais conceitos de segurança de redes, i.e., identificação, autenticação, autorização, integridade e confidencialidade (STALLINGS, 2005). Essa série de padrões constitui um *framework* de segurança para Web Services, cujo padrão central é o WS-Security (*Web Services Security*) (NADALIN, 2006a). A especificação WS-Security possibilita o projeto de soluções seguras baseadas em Web Services. Portanto, para a gerência de redes via Web Services, essa especificação possibilita a implementação de segurança de forma mais abrangente do que seria possível com as funcionalidades limitadas do SNMPv3. Por exemplo, um dos padrões que constituem o *framework* WS-Security, o XACML (*eXtensible Access Control Markup Language*) (RISSANEN, 2008), integra controle de acesso a comunicações baseadas em mensagens XML com bastante flexibilidade.

Como se sabe, a segurança tem um custo considerável no desempenho dos sistemas, e isso ocorre também nas soluções de segurança baseadas em Web Services. Qualquer arquitetura de segurança para gerenciamento precisa ser avaliada adequadamente no tocante ao desempenho.

Até hoje, não há estudos propondo arquiteturas de segurança para soluções de gerenciamento de redes baseadas em Web Services. Tampouco há estudos avaliando o desempenho de soluções como essa. Isso é crucial, pois dependendo do impacto negativo dessas soluções, o uso das mesmas pode ser inviável. Este trabalho tem por objetivo preencher essa lacuna, estabelecendo uma proposta de arquitetura de segurança, baseada em WS-Security, para soluções de gerência de redes que utilizam Web Services. Como o padrão projetado para controle de acesso no WS-Security, o WS-Authorization, ainda não está implementado, essa arquitetura usará também o XACML, que é o padrão de fato, hoje, para controle de acesso em Web Services.

A arquitetura projetada neste trabalho deve prover facilidades para o gerenciamento de uma rede na qual o administrador utiliza Web Services, mas cujos recursos gerenciáveis podem tanto suportar apenas comunicação via Web Services quanto via SNMP. É substancial que a solução seja válida nesse tipo de rede, já que as redes gerenciadas via Web Services terão em geral esse perfil, pois a implementação de Web Services diretamente nos recursos gerenciáveis é atualmente ainda muito custosa. Outra razão para isso são as redes legadas: mesmo que os Web Services passem a ser empregados em redes reais, em boa parte dessas redes, recursos gerenciados exclusivamente via SNMP continuarão existindo.

Devido ao alto impacto do uso de segurança no desempenho de aplicações em geral, o desempenho da implementação dessa arquitetura será avaliado em termos de tempo de resposta e tráfego de rede. Estes dois parâmetros são os habitualmente considerados em pesquisas focadas no gerenciamento de redes via Web Services, como será visto nos trabalhos relacionados, apresentados em seção específica dentro deste trabalho.

O restante deste trabalho está organizado como segue. No Capítulo 2, é feita uma revisão sobre a tecnologia de Web Services, estudos acadêmicos e padrões de Web Services para gerenciamento de redes, padrões de segurança para Web Services, e

segurança nas diversas versões do padrão SNMP. O Capítulo 3 apresenta a arquitetura proposta para a solução de segurança para gerenciamento via Web Services, dividida em dois módulos: autenticação, confidencialidade e integridade usando WS-Security, e controle de acesso (autorização) via XACML. Os detalhes da implementação desta arquitetura são apresentados no Capítulo 4. Em seguida, no Capítulo 5, são apresentados os resultados da avaliação de desempenho realizada. Nos capítulos 4 e 5, a implementação e avaliação de desempenho também são organizadas nos módulos projetados, com WS-Security e com XACML. Por fim, o Capítulo 6 apresenta as conclusões finais deste trabalho, bem como os possíveis trabalhos futuros que podem ser realizados.

2 WEB SERVICES E GERÊNCIA DE REDES

Neste capítulo, será feita uma revisão sobre a arquitetura da tecnologia dos Web Services, e dos principais componentes e protocolos envolvidos na mesma. Será também discutido o contexto atual do gerenciamento de redes via Web Services. Este contexto é formado pelos trabalhos acadêmicos realizados nessa área, e também por dois padrões desenvolvidos pela indústria para gerência de redes via Web Services, o WS-Management (*Web Services for Management*) (ARORA, 2006) e o MUWS (*Management Using Web Services*) (BULLARD, 2006), que hoje estão em processo de convergência. O processo de convergência destes dois padrões também é detalhado, pois o novo padrão pós-convergência será baseado principalmente no WS-Management, e a partir desse fato, se optou por realizar os experimentos neste trabalho exclusivamente com WS-Management, em detrimento do MUWS. Desta forma, também é apresentada a API Wiseman, implementação em Java do WS-Management. Também serão apresentados detalhes sobre os principais padrões de segurança para Web Services, WS-Security e XACML, e implementações em Java destes padrões, que foram utilizadas neste trabalho. Por fim, será apresentado o enfoque dado à segurança no desenvolvimento dos protocolos SNMPv1 (CASE, 1990), SNMPv2 (CASE, 1993) e SNMPv3.

2.1 Web Services

Os Web Services são um conjunto de tecnologias recentes, especificadas e desenvolvidas pelo World Wide Web Consortium (W3C). Os Web Services possuem uma grande visibilidade na indústria de tecnologia, sendo considerados um grande marco para o desenvolvimento de software distribuído. As maiores empresas da indústria atual têm anunciado e desenvolvido estratégias na área de Web Services, e diversos padrões com foco específico têm sido elaborados com base na estrutura dos Web Services (como por exemplo, os já citados neste trabalho: WS-Security, WS-Management e MUWS).

O termo Web Services é bastante auto-explicativo: ele se refere à utilização de serviços através da Web. O uso corrente do termo abrange a arquitetura, padrões e tecnologias que os tornam possíveis. Os Web Services permitem que aplicações se comuniquem de forma independente de plataforma e de linguagens de programação através da Web.

Um Web Service é uma interface de software que descreve um conjunto de funções que podem ser acessadas via rede através de mensagens XML. Esta descrição contém todos os detalhes necessários para a utilização do serviço, como formatos de mensagem, protocolos de transporte e localização. A interface de um Web Service esconde os detalhes de implementação do serviço, que não são relevantes para quem utiliza o Web Service, e assim o serviço pode ser utilizado de forma independente de plataforma de hardware ou software em que está rodando, bem como da linguagem de programação em que foi escrito.

A arquitetura básica dos Web Services é baseada nas interações entre três entidades: o provedor de serviços, o consumidor de serviços e o registro de serviços, este último opcional. As interações envolvem operações de disponibilização, procura e interação de dois componentes: o serviço e a descrição de serviço. Tipicamente, o provedor de serviços possui um módulo de software acessível através da rede (que é ponto de invocação da própria implementação de um Web Service). O provedor de serviços define a descrição para esse serviço, e a disponibiliza em um registro de serviços, ou diretamente para um consumidor de serviços. O consumidor procura a descrição do serviço no registro, ou localmente, e através dela, interage com o provedor de serviços, invocando a implementação do Web Service. A figura 2.1 ilustra a arquitetura dos Web Services.

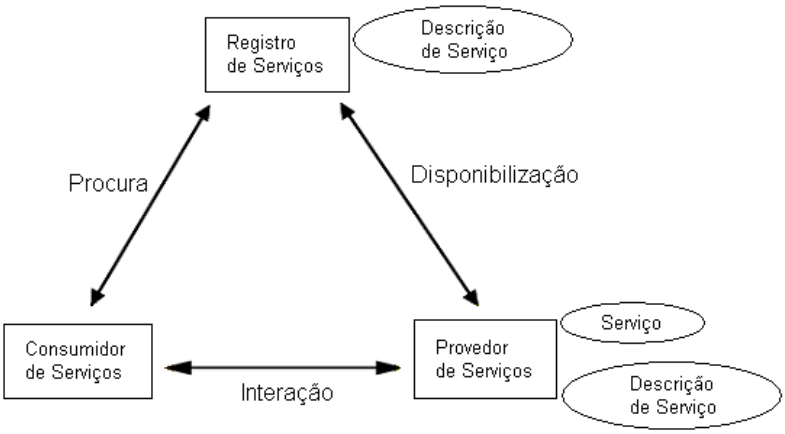


Figura 2.1: Arquitetura dos Web Services

As entidades exibidas na figura 2.1 são as seguintes:

- Provedor de serviços: Plataforma que possui a implementação do serviço.
- Consumidor de serviços: Aplicação que procura e interage com o serviço. O consumidor pode ser, por exemplo, um navegador Web ou um programa sem interface de usuário, como outro Web Service ou uma aplicação qualquer.
- Registro de serviços: Registro com descrições de serviço, publicadas pelos provedores. Os consumidores procuram serviços e informações sobre como interagir com estes serviços através de suas descrições. O registro de serviços é

uma entidade opcional na arquitetura, pois o provedor pode enviar a descrição diretamente para o(s) consumidor(es).

As operações da figura 2.1 são detalhadas a seguir:

- Disponibilização: Para ser acessível, uma descrição de serviço precisa ser disponibilizada pelo provedor em um registro de serviços, para que possa ser encontrada por um consumidor.

- Procura: O consumidor recupera uma descrição de serviço diretamente com o produtor ou no serviço de registros. Esta operação pode aparecer tanto na implementação quanto na execução do consumidor.

- Interação: O consumidor, em tempo de execução, utilizando as informações contidas na descrição de um serviço, o localiza, contata e invoca.

Os componentes existentes nas entidades da figura 2.1 são os seguintes:

- Serviço: Um serviço é um módulo de software disponível em plataformas acessíveis através da rede, fornecido por um provedor de serviços. Além de poder ser invocado por um consumidor, também pode funcionar ele próprio como um consumidor, utilizando outras implementações de serviço.

- Descrição de serviço: Contém todas as informações necessárias de interface do serviço, como tipos de dados utilizados na comunicação, operações disponíveis e localização na rede.

2.1.1 Padrões

A adoção em larga escala dos Web Services requer padrões que possibilitem a interoperabilidade entre plataformas independentes. Hoje, o conjunto de padrões amplamente suportado para a implementação da arquitetura dos Web Services é formado pelos padrões WSDL, SOAP e UDDI.

A WSDL (*Web Services Description Language*) (CHINNICI, 2006) foi desenvolvida com o propósito de descrição de serviços. Através dela, um serviço é descrito como um conjunto de interações suportadas entre uma aplicação (consumidor) e um Web Service (provedor). Essas interações são descritas como operações que devem ter uma mensagem de requisição, e opcionalmente, uma mensagem de resposta.

A descrição inclui detalhes como definição de tipos de dados, operações suportadas pelo serviço, formatos das mensagens de entrada e saída, endereço de rede, protocolo de ligação (*binding*), etc. A Figura 2.2 mostra um exemplo de documento WSDL.

```
<?xml version="1.0"?>

<!-- root element wsdl:definitions defines set of related services -->
<wsdl:definitions name="EndorsementSearch"
  targetNamespace="http://namespaces.snowboard-info.com"
  xmlns:es="http://www.snowboard-info.com/EndorsementSearch.wsdl"
  xmlns:esxsd="http://schemas.snowboard-info.com/EndorsementSearch.xsd"
```

```
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">

  <!-- wsdl:types encapsulates schema definitions of communication types; here using xsd
-->
  <wsdl:types>

    <!-- all type declarations are in a chunk of xsd -->
    <xsd:schema targetNamespace="http://namespaces.snowboard-info.com"
      xmlns:xsd="http://www.w3.org/1999/XMLSchema">

      <!-- xsd definition: GetEndorsingBoarder [manufacturer string, model string] -->
      <xsd:element name="GetEndorsingBoarder">
       <xsd:complexType>
         <xsd:sequence>
           <xsd:element name="manufacturer" type="string"/>
            <xsd:element name="model" type="string"/>
         </xsd:sequence>
       </xsd:complexType>
      </xsd:element>

      <!-- xsd definition: GetEndorsingBoarderResponse [... endorsingBoarder string ...]
-->
      <xsd:element name="GetEndorsingBoarderResponse">
       <xsd:complexType>
         <xsd:all>
           <xsd:element name="endorsingBoarder" type="string"/>
         </xsd:all>
       </xsd:complexType>
      </xsd:element>

      <!-- xsd definition: GetEndorsingBoarderFault [... errorMessage string ...] -->
      <xsd:element name="GetEndorsingBoarderFault">
       <xsd:complexType>
         <xsd:all>
           <xsd:element name="errorMessage" type="string"/>
         </xsd:all>
       </xsd:complexType>
      </xsd:element>

    </xsd:schema>
  </wsdl:types>

  <!-- wsdl:message elements describe potential transactions -->

  <!-- request GetEndorsingBoarderRequest is of type GetEndorsingBoarder -->
  <wsdl:message name="GetEndorsingBoarderRequest">
    <wsdl:part name="body" element="esxsd: GetEndorsingBoarder"/>
  </wsdl:message>

  <!-- response GetEndorsingBoarderResponse is of type GetEndorsingBoarderResponse -->
  <wsdl:message name="GetEndorsingBoarderResponse">
    <wsdl:part name="body" element="esxsd: GetEndorsingBoarderResponse"/>
  </wsdl:message>

  <!-- wsdl:portType describes messages in an operation -->
  <wsdl:portType name="GetEndorsingBoarderPortType">

    <!-- the value of wsdl:operation eludes me -->
    <wsdl:operation name="GetEndorsingBoarder">
      <wsdl:input message="es: GetEndorsingBoarderRequest"/>
      <wsdl:output message="es: GetEndorsingBoarderResponse"/>
      <wsdl:fault message="es: GetEndorsingBoarderFault"/>
    </wsdl:operation>
  </wsdl:portType>

  <!-- wsdl:binding states a serialization protocol for this service -->
  <wsdl:binding name="EndorsementSearchSoapBinding"
               type="es:GetEndorsingBoarderPortType">

    <!-- leverage off soap:binding document style -->
    <soap:binding style="document"
                  transport="http://schemas.xmlsoap.org/soap/http"/>

    <!-- semi-opaque container of network transport details classed by soap:binding
above -->
    <wsdl:operation name="GetEndorsingBoarder">
```

```
      <!-- again bind to SOAP  -->
      <soap:operation soapAction="http://www.snowboard-info.com/EndorsementSearch"/>

      <!-- further specify that the messages in the wsdl:operation "GetEndorsingBoarder"
use SOAP -->
      <wsdl:input>
        <soap:body use="literal"
                   namespace="http://schemas.snowboard-info.com/EndorsementSearch.xsd"/>
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal"
                   namespace="http://schemas.snowboard-info.com/EndorsementSearch.xsd"/>
      </wsdl:output>
      <wsdl:fault>
        <soap:body use="literal"
                   namespace="http://schemas.snowboard-info.com/EndorsementSearch.xsd"/>
      </wsdl:fault>
    </wsdl:operation>
  </wsdl:binding>

  <!-- wsdl:service names a new service "EndorsementSearchService" -->
  <wsdl:service name="EndorsementSearchService">
    <wsdl:documentation>snowboarding-info.com Endorsement Service</wsdl:documentation>

    <!-- connect it to the binding "EndorsementSearchSoapBinding" above -->
    <wsdl:port name="GetEndorsingBoarderPort"
               binding="es:EndorsementSearchSoapBinding">

      <!-- give the binding an network address -->
      <soap:address location="http://www.snowboard-info.com/EndorsementSearch"/>
    </wsdl:port>
  </wsdl:service>

 </wsdl:definitions>
```

Figura 2.2: Exemplo de documento WSDL

O SOAP (*Simple Object Access Protocol*) é um padrão para a troca de informações baseadas em XML entre aplicações distribuídas. Ele permite vários tipos de transporte, sendo tipicamente transmitido sobre HTTP ou HTTPS, provendo assim a plataforma para comunicações com e entre Web Services. O SOAP define o formato de requisição e resposta de documentos XML sobre HTTP. Assim, uma aplicação invoca um serviço através de uma mensagem de requisição SOAP, e o serviço pode retornar uma mensagem de resposta SOAP. O WSDL é encapsulado no SOAP, quando é transmitido de um provedor de serviços para um registro de serviços, e também quando é obtido no registro por um consumidor de serviços.

A figura 2.3 exibe um par de requisição e resposta SOAP, relativos ao serviço descrito pelo documento WSDL apresentado na figura 2.2.

```
<SOAP-EsdsEnvelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
    <m:GetEndorsingBoarder xmlns:m="http://namespaces.snowboard-info.com">
      <manufacturer>K2</manufacturer>
      <model>Fatbob</model>
    </m:GetEndorsingBoarder>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

(a)

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
    <m:GetEndorsingBoarder xmlns:m="http://namespaces.snowboard-info.com">
      <endorsingBoarder>Chris Englesmann</endorsingBoarder>
    </m:GetEndorsingBoarderResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

(b)

Figura 2.3: Exemplo de mensagens de requisição (a) e resposta (b) SOAP

Utilizando WSDL e SOAP, os Web Services podem ser descritos e utilizados por aplicações. Para possibilitar a disponibilização e procura dos mesmos, é utilizado o UDDI (*Universal Discovery, Description and Integration*) (BELLWOOD, 2004), que é uma especificação para registros distribuídos de serviços. Um registro UDDI pode ser acessado via SOAP por uma aplicação que procura um serviço. O UDDI especifica interfaces para aplicações disponibilizar e procurar serviços, através de documentos WSDL.

A maioria dos serviços implementados atualmente não utiliza o registro de serviços, e, por conseqüência, UDDI. Em geral, as descrições de serviços ficam disponíveis no próprio provedor de serviços, ou são conhecidas de antemão pelos consumidores.

2.2 Gerência de redes via Web Services

Uma série de características dos Web Services o tornaram uma alternativa interessante para o gerenciamento de redes, como independência de plataforma, agilidade no desenvolvimento e uso de padrões largamente suportados, baseados em XML. A partir dessa idéia, uma série de estudos acadêmicos foi realizada para verificar a viabilidade do uso dos Web Services em gerência, focando em questões de desempenho. Também surgiram padrões na indústria para gerência via Web Services, o WS-Management e o MUWS. Esta seção aborda os estudos acadêmicos realizados, e detalha os padrões WS-Management e MUWS, bem como uma iniciativa em andamento que visa a convergência destes 2 padrões, além do Wiseman, implementação em Java do WS-Management.

2.2.1 Estudos Acadêmicos

Muitos artigos foram publicados recentemente focando em soluções práticas que possibilitem comparações de desempenho entre a gerência via Web Services e a gerência tradicional, via SNMP. Boa parte dessas soluções é baseada em *gateways* WS/SNMP. *Gateways* WS/SNMP são componentes que realizam a interface entre um sistema de gerência baseado em Web Services e um recurso gerenciável via SNMP. Eles podem se localizar fisicamente junto às aplicações de gerência, junto aos recursos gerenciáveis, ou mesmo em dispositivos independentes. Esses *gateways* traduzem as mensagens Web Services enviadas pela aplicação de gerência para mensagens SNMP, e

as repassam aos recursos gerenciados. A figura 2.4 ilustra o funcionamento de uma rede gerenciada com a presença de *gateways* WS/SNMP.

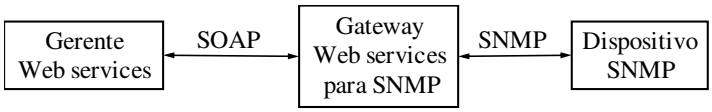


Figura 2.4: *Gateway* Web Services para SNMP

Certamente muitos dispositivos SNMP atuais se manterão nas redes no futuro, seja pelo fato de a atualização para Web Services ser custosa, ou pela falta de recursos no dispositivo que permitam que ele ofereça esses serviços. Portanto, a padronização dos Web Services passa pelo suporte à gerência de redes em que haja recursos que só suportam SNMP. Sendo a hierarquia administrativa dessas redes baseada em Web Services, será necessária a presença de *gateways* para integrar a rede.

Em (NEISSE, 2004), foi realizado um dos primeiros estudos comparando o desempenho da gerência via Web Services com SNMP, com foco em consumo de banda. Foi implementado um *gateway* WS/SNMP e medido o tráfego nos dois lados do *gateway*. Dois mecanismos de tradução de informação foram implementados no *gateway*: orientado a protocolo, em que as mensagens SNMP (por exemplo, *Get*, *Set*, *GetNext*) são mapeadas diretamente para Web Services; e orientado a objeto, em que cada objeto ou tabela (por exemplo, *IfTable*) de uma MIB (*Management Information Base*) (McCLOGHRIE, 1991) SNMP é mapeado para um Web Service. A conclusão desse estudo foi que o *gateway* orientado a protocolo apresentava um consumo de banda maior do que o SNMP. Porém, o *gateway* orientado a objeto apresentou melhor desempenho do que o SNMP quando um grande número de objetos é recuperado, consumindo menos banda.

Em (PRAS, 2004), foi realizada uma comparação levando em conta consumo de banda, tempo de requisição e consumo de memória. Foram comparadas algumas dezenas de agentes SNMP com um agente Web Services, desenvolvido a partir do agente Net-SNMP (SOURCEFORGE, 2009a), com a troca direta do código SNMP pelo uso de funções de Web Services. A conclusão do estudo foi que, com o uso de compressão de dados, e para um número grande de objetos, o desempenho dos Web Services foi superior. Também foi observado que o tempo de processamento para a codificação dos dados em XML e para a compressão dos dados foi mínimo, em comparação com o tempo necessário para recuperar as informações efetivas dos recursos gerenciados.

Em (FIOREZE, 2005), foi estudada a aplicação de Web Services em ambientes de gerência por delegação. Foi comparado o tempo de requisição e o consumo de banda entre um ambiente tradicional de gerência por delegação, com uso de SNMP e da IETF Script MIB (LEVI, 2001) e um ambiente com *gateway* WS/SNMP. Dois mecanismos de tradução foram implementados no *gateway*: orientado a objetos e orientado a serviços. O *gateway* orientado a serviços é desenvolvido analisando os serviços diretamente oferecidos por uma MIB, e não apenas sua estrutura de objetos SNMP. Isso permite a modelagem de Web Services mais eficientes. No estudo realizado, o *gateway* orientado a objetos teve um desempenho pior que o ambiente com SNMP e IETF Script

MIB. Já o *gateway* orientado a serviços teve um consumo de banda menor que o do SNMP, e o tempo da requisição foi praticamente o mesmo.

Em (LIMA, 2006), foi comparada o desempenho de traps SNMP com a de notificações Web Services. Foram implementados três tipos de *gateways* nessa avaliação. Um dos *gateways* traduzia diretamente traps SNMP para notificações WS. O outro encapsulava traps SNMP dentro das notificações WS. Como essas soluções tiveram desempenho inferior ao das traps SNMP, um terceiro mecanismo foi desenvolvido, trazendo do gerente para o *gateway* o tratamento das traps SNMP: após receber uma ou mais traps SNMP, o próprio *gateway* realiza requisições SNMP relevantes para o agente SNMP, e envia uma notificação Web Services mais completa para o gerente, tornando assim desnecessária a busca dessas informações no agente pelo gerente. Essa solução apresentou menor consumo de banda e menor uso de memória do que as traps SNMP.

2.2.2 *Web Services for Management* (WS-Management)

O WS-Management (*Web Services for Management*) é uma especificação desenvolvida por um consórcio formado por um conjunto de empresas, entre as quais Sun, Microsoft, NEC, Intel e Dell. Esta especificação foi submetida à DMTF (*Distributed Management Task Force*) em setembro de 2005, e teve seu *release* final publicado em 12 de fevereiro de 2008 (ARORA, 2008).

A especificação WS-Management está organizada em um único documento, e tem por objetivo definir como um recurso gerenciável pode ser abstraído em uma estrutura XML, e gerenciado através de uma comunicação baseada nessa estrutura. O WS-Management é construído com o uso de uma série de outros padrões da família Web Services: XML Schema (THOMPSON, 2004) e (BIRON, 2004), WS-Addressing (*Web Services Addressing*) (BOX, 2004), WS-Eventing (*Web Services Eventing*) (BOX, 2006), WS-Enumeration (*Web Services Enumeration*) (ALEXANDER, 2006a) e WS-Transfer (*Web Services Transfer*) (ALEXANDER, 2006b). O foco destas especificações é o seguinte:

- XML Schema: Linguagem para descrição em XML da estrutura de documentos XML.

- WS-Addressing: Especificação para mecanismos independentes de transporte (exemplo: HTTP, HTTPS) para comunicação de informações de endereçamento de Web Services.

- WS-Eventing: Protocolo para envio e subscrição de eventos entre Web Services (mecanismo similar às traps SNMP).

- WS-Enumeration: Base para iterações em membros de coleções disponibilizadas como retorno de Web Services.

- WS-Transfer: Especificação que define transferência de representações em XML de recursos disponibilizados via Web Services, bem como a criação e remoção de tais recursos.

No WS-Management, um recurso gerenciável qualquer pode ser representado por uma estrutura XML arbitrariamente definida que represente seu conjunto de propriedades. As operações podem ser realizadas sobre um recurso como um todo, sobre parte das propriedades de um recurso (operação definida como *Fragment-Level Transfer*, e que pode ser realizada usando XPath (*XML Path Language*) (BERGLUND, 2007) ou algum outro dialeto específico), ou sobre um conjunto de recursos (através do WS-Enumeration). No caso de um conjunto de recursos, as operações também podem ser realizadas sobre todas informações do conjunto de recursos, ou sobre parte das propriedades de cada recurso do conjunto.

As operações suportadas no WS-Management são as seguintes:

- DISCOVER: Utilizada para descobrir a presença de recursos de gerenciamento e navegar entre esses recursos.

- GET, PUT, CREATE, DELETE (operações derivadas do WS-Transfer) e RENAME (operação do próprio WS-Management): Manipulação dos recursos de gerenciamento e de suas propriedades.

- ENUMERATE: Usado para retornar o conteúdo coleções, como tabelas e logs volumosos.

- SUBSCRIBE: Recepção de eventos emitidos por recursos gerenciados.

- EXECUTE: Executação de métodos de gerenciamento específicos, incluindo parâmetros de entrada e saída fortemente tipados.

A figura 2.5 exibe um exemplo de mensagem de requisição de um recurso gerenciável WS-Management:

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"
        xmlns:jb="http://test.foo"
        xmlns:rtentry="http://labcom.inf.ufrgs.br/schemas/routeentry.xsd"
        xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
        xmlns:wse="http://schemas.xmlsoap.org/ws/2004/08/eventing"
        xmlns:wsen="http://schemas.xmlsoap.org/ws/2004/09/enumeration"
        xmlns:wsman="http://schemas.dmtf.org/wbem/wsman/1/wsman.xsd"
        xmlns:wxf="http://schemas.xmlsoap.org/ws/2004/09/transfer"
        xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <env:Header>
        <wsa:Action env:mustUnderstand="true">
            http://schemas.xmlsoap.org/ws/2004/09/transfer/Get
        </wsa:Action>
        <wsa:ReplyTo>
            <wsa:Address env:mustUnderstand="true">
                http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous
            </wsa:Address>
        </wsa:ReplyTo>
        <wsa:MessageID env:mustUnderstand="true">
            uuid:9ceab391-198f-4cb5-b69b-4ae5420f6b7a
        </wsa:MessageID>
        <wsa:To env:mustUnderstand="true">
            http://143.54.12.157:8080/wsman/
        </wsa:To>
        <wsman:ResourceURI>
            wsman:test/entry
        </wsman:ResourceURI>
        <wsman:SelectorSet>
            <wsman:Selector Name="destination">
                00000000
```

```
            </wsman:Selector>
        </wsman:SelectorSet>
    </env:Header>
    <env:Body/>
</env:Envelope>
```

Figura 2.5: Exemplo de mensagem WS-Management de requisição

A resposta à requisição da figura 2.5 é exibida na figura 2.6:

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"
        xmlns:jb="http://test.foo"
        xmlns:rtentry="http://labcom.inf.ufrgs.br/schemas/routeentry.xsd"
        xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
        xmlns:wse="http://schemas.xmlsoap.org/ws/2004/08/eventing"
        xmlns:wsen="http://schemas.xmlsoap.org/ws/2004/09/enumeration"
        xmlns:wsman="http://schemas.dmtf.org/wbem/wsman/1/wsman.xsd"
        xmlns:wxf="http://schemas.xmlsoap.org/ws/2004/09/transfer"
        xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <env:Header>
        <wsa:Action env:mustUnderstand="true">
            http://schemas.xmlsoap.org/ws/2004/09/transfer/GetResponse
        </wsa:Action>
        <wsa:MessageID env:mustUnderstand="true">
            uuid:d40897f6-4c05-4218-8558-62b1869fe56b
        </wsa:MessageID>
        <wsa:RelatesTo>
            uuid:9ceab391-198f-4cb5-b69b-4ae5420f6b7a
        </wsa:RelatesTo>
        <wsa:To env:mustUnderstand="true">
            http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous
        </wsa:To>
    </env:Header>
    <env:Body>
        <RouteEntry xmlns="http://labcom.inf.ufrgs.br/schemas/routeentry.xsd">
            <ipCidrRouteDest>00000000</ipCidrRouteDest>
            <ipCidrRouteMask>00000000</ipCidrRouteMask>
            <ipCidrRouteTos>0</ipCidrRouteTos>
            <ipCidrRouteNextHop>0F00A8C0</ipCidrRouteNextHop>
            <ipCidrRouteIfIndex>eth0</ipCidrRouteIfIndex>
            <ipCidrRouteType>local</ipCidrRouteType>
            <ipCidrRouteProto>local</ipCidrRouteProto>
            <ipCidrRouteAge>000000</ipCidrRouteAge>
            <ipCidrRouteInfo>.0.0</ipCidrRouteInfo>
            <ipCidrRouteNextHopAS>0</ipCidrRouteNextHopAS>
            <ipCidrRouteMetric1>0</ipCidrRouteMetric1>
            <ipCidrRouteMetric2>-1</ipCidrRouteMetric2>
            <ipCidrRouteMetric3>-1</ipCidrRouteMetric3>
            <ipCidrRouteMetric4>-1</ipCidrRouteMetric4>
            <ipCidrRouteMetric5>-1</ipCidrRouteMetric5>
            <ipCidrRouteStatus>1</ipCidrRouteStatus>
        </RouteEntry>
    </env:Body>
</env:Envelope>
```

Figura 2.6: Exemplo de mensagem WS-Management de resposta

Para melhor compreensão dos exemplos apresentados, abaixo são detalhados os campos possíveis no cabeçalho de mensagens WS-Management. Os campos que iniciam por *wsa* são campos definidos na especificação WS-Addressing; os que iniciam por *wsman* são definidos na própria especificação WS-Management. Os prefixos *wsa* e *wsman* se referem aos *namespaces* XML (BRAY, 2006b) das respectivas especificações.

- wsa:To: Indica o endereço WS-Addressing do Web Service que representa o recurso gerenciado.

- wsman:ResourceURI: URI (*Uniform Resource Identifier*) (BERNERS-LEE, 2005) do recurso gerenciado.

- wsman:SelectorSet: Usado para referenciar ou selecionar uma instância específica de um recurso, se há mais de uma instância.

- wsa:ReplyTo: Endereço válido para resposta, ou a URI *http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous*, que indica que a resposta deve ser enviada através da mesma conexão pela qual a requisição foi feita.

- wsa:MessageID: Deve ser gerada por um algoritmo que garanta que duas mensagens distintas não tenham o mesmo ID.

- wsa:RelatesTo: Indica à qual mensagem de requisição se refere uma resposta.

- wsa:Action: Indica qual a ação sendo realizada sobre o recurso (por exemplo, GET, PUT, CREATE, DELETE).

- wsa:From: Indica a origem da mensagem. Quando a mesma conexão é usada para a requisição e a resposta, é desnecessário, mas pode ser usado em casos em que a resposta é enviada através de uma conexão diferente da requisição.

No tocante a coleções de recursos gerenciáveis, o WS-Management define o uso do WS-Enumeration. A enumeração de uma coleção é realizada em três operações. A primeira é a *wsen:Enumerate*, que estabelece o contexto da enumeração. Depois, são utilizadas sucessivas *wsen:Pull*, para iterar entre os membros da coleção enumerada. Quando todos os membros foram obtidos, ou a enumeração não é mais necessária, usa-se a operação *wsen:Release*. Os elementos desejados de uma coleção podem ser especificados por XPath ou através dos próprios *SelectorSets* do WS-Management.

O WS-Management suporta a execução de ações específicas, diferentes das operações básicas (como GET e PUT), através das *Custom Actions*, que são ações específicas de cada aplicação, e que aparecem no campo *wsa:Action* das mensagens WS-Management.

Por fim, o suporte a eventos no WS-Management se dá através das seguintes operações:

- Subscribe: Requisita que eventos de um recurso sejam enviados.

- Unsubscribe: Cancela a subscrição aos eventos de um recurso.

- Renew: Extende o tempo de vida de uma subscrição.

- SubscriptionEnd: Notifica o fim do tempo de vida de uma subscrição.

- Acknowledgement of Delivery: Confirmação da recepção de eventos.

- Refusal of Delivery: Falha na recepção de eventos, seja por razões de segurança ou por alterações em políticas.

2.2.3 *Management Using Web Services* (MUWS)

O MUWS faz parte da especificação WSDM (*Web Services Distributed Management*). A WSDM foi desenvolvida e padronizada em março de 2005 pelo OASIS (*Organization for the Advancement of Structured Information Standards*), e é composta pelo MUWS (*Management using Web Services*) e pelo MOWS (*Management of Web Services)* (WILSON, 2006). O objetivo do WSDM como um todo é mais abrangente que o do WS-Management, sendo tanto a gerência de recursos através das Web Services, quanto a gerência dos próprios Web Services, utilizados ou não para fins de gerenciamento. O objetivo de cada parte do WSDM pode ser resumido da seguinte forma:

- MUWS: Define como acessar e representar recursos através de Web Services. Consiste em um conjunto de capacidades de gerenciamento que estes Web Services devem atender, como identidade, configuração e relações que permitam composição de recursos. O MUWS também prevê o tratamento de eventos dos recursos gerenciados.

- MOWS: Define como descrever os próprios Web Services como recursos gerenciáveis via MUWS.

Conforme o exposto acima, é um dos padrões componentes do WSDM, o MUWS, que faz paralelo com WS-Management, definindo como abstrair um recurso gerenciável arbitrariamente via XML. Assim, focamos em apresentar os detalhes do MUWS neste trabalho.

O MUWS se baseia em uma série de padrões baseados em Web Services: XML Schema, WS-Addressing, WS-Resource (*Web Services Resource*) (GRAHAM, 2006a) e WS-Notification (*Web Services Notification*) (GRAHAM, 2004a).

No MUWS, um Web Service, denominado *endpoint*, provê acesso a um recurso gerenciável. Um consumidor (ou sistema de gerenciamento) descobre o *endpoint* e troca mensagens com o mesmo. A tabela 2.1 exibe os MEPs (*Message Exchange Patterns*) existentes no MUWS para interação entre o consumidor e o recurso.

Tabela 2.1: MEPs (*Message Exchange Patterns*) do MUWS

Requisições de Informação	
GetResourceProperty	Retorna o valor de uma propriedade de um recurso
GetMultipleResourceProperties	Retorna os valores de um conjunto de propriedades de um recurso
QueryResourceProperties	Retorna parte do documento XML de propriedade de um recurso (por exemplo, usando XPath)
QueryRelationshipsByType	Retorna informações de um tipo de relação da qual um recurso participa
Comandos de alteração de informação	
SetResourceProperties	Insere, altera ou remove propriedades de um recurso
Notificações	
Subscribe	Subscrição às notificações de um recurso
GetCurrentMessage	Requisita última notificação de um recurso
PauseSubscription	Requisita pausa temporária numa subscrição
ResumeSubscription	Reativa uma subscrição interrumpida
Notify	Recebe notificações
RegisterPublisher	Regista um recurso como *Publisher* de notificações
Destroy	Destrói registro de um recurso como *Publisher*

Três tipos de interação são possíveis: recuperar as informações do recursos gerenciável; alterar estas informações; ou a notificação de eventos a partir do recurso gerenciável. A figura 2.7 exibe um exemplo de mensagem de requisição do MUWS:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
    <soapenv:Header>
        <add:To xmlns:add="http://schemas.xmlsoap.org/ws/2004/08/addressing">
            http://143.54.12.157:8080/muse/services/route
        </add:To>
        <add:Action xmlns:add="http://schemas.xmlsoap.org/ws/2004/08/addressing">
            http://getResourceProp
        </add:Action>
        <svr:ResourceIdentifier xmlns:svr="http://labcom.inf.ufrgs.br/wsdl/route.wsdl"
                xmlns="http://schemas.xmlsoap.org/ws/2004/08/addressing">000C368F
        </svr:ResourceIdentifier>
    </soapenv:Header>
    <soapenv:Body>
        <wsrf:GetMultipleResourceProperties xmlns:wsrf=
                "http://docs.oasis-open.org/wsrf/2004/06/
                wsrf-WS-ResourceProperties-1.2-draft-01.xsd">
            <wsrf:ResourceProperty xmlns:rout=
                "http://labcom.inf.ufrgs.br/schemas/routeentry.xsd">rout:ipCidrRouteDest
            </wsrf:ResourceProperty>
            <wsrf:ResourceProperty xmlns:rout=
                "http://labcom.inf.ufrgs.br/schemas/routeentry.xsd">rout:ipCidrRouteMask
            </wsrf:ResourceProperty>
            <wsrf:ResourceProperty xmlns:rout=
                "http://labcom.inf.ufrgs.br/schemas/routeentry.xsd">rout:ipCidrRouteTos
            </wsrf:ResourceProperty>
        </wsrf:GetMultipleResourceProperties>
    </soapenv:Body>
</soapenv:Envelope>
```

Figura 2.7: Exemplo de mensagem MUWS de requisição

A figura 2.8 exibe a resposta MUWS correspondente à requisição da figura 2.7:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
.....
    <soapenv:Body>
        <wsrf:GetMultipleResourcePropertiesResponse xmlns:wsrf=
                "http://docs.oasis-open.org/wsrf/2004/06/
                wsrf-WS-ResourceProperties-1.2-draft-01.xsd">
            <rout:ipCidrRouteDest xmlns:rout=
                "http://labcom.inf.ufrgs.br/schemas/routeentry.xsd">000C368F
            </rout:ipCidrRouteDest>
            <rout:ipCidrRouteMask xmlns:rout=
                "http://labcom.inf.ufrgs.br/schemas/routeentry.xsd">00FFFFFF
            </rout:ipCidrRouteMask>
            <rout:ipCidrRouteTos xmlns:rout=
                "http://labcom.inf.ufrgs.br/schemas/routeentry.xsd">0
            </rout:ipCidrRouteTos>
        </wsrf:GetMultipleResourcePropertiesResponse>
    </soapenv:Body>
</soapenv:Envelope>
```

Figura 2.8: Exemplo de mensagem MUWS de resposta

2.2.4 Convergência entre WS-Management e MUWS

Conforme exposto, há hoje dois padrões na indústria para gerenciamento via Web Services, o WS-Management e o MUWS. Estes padrões cobrem os mesmos objetivos. Diante disso, surgiu a idéia (CLINE, 2006) de desenvolver um padrão único a partir da convergência destes dois padrões. O trabalho de desenvolvimento deste novo padrão está sendo realizado de forma conjunta entre Intel, Microsoft, IBM e HP.

Há três objetivos principais para o novo padrão:

- Recursos: possibilidade de criar, ler, alterar e deletar informações de recursos via Web Services

- Eventos: possibilidade de comunicação entre Web Services através de uma arquitetura orientada a eventos, no modelo *publisher/subscriber*.

- Gerenciamento: prover um modelo baseado em Web Services para a construção de sistemas e aplicações de gerenciamento de recursos.

O trabalho de convergência consiste em dois blocos de padrões: gerenciamento de recursos e eventos/notificações. A intenção é criar um conjunto de novos padrões, baseados nos padrões que são utilizados no WS-Management e MUWS, e que permitam compatibilidade entre serviços que suportem o novo padrão, e serviços que suportam WS-Management ou MUWS.

No tocante ao gerenciamento de recursos, o WS-Management é baseado no WS-Transfer, utilizado para operações CRUD (*Create, Read, Update, Delete*) sobre recursos, e no WS-Enumeration, utilizado para coleções de recursos. O MUWS é baseado no WSRF (*WS-Resource Framework*), que consiste em diversos padrões específicos: além do próprio WS-Resource, há o WSRF-RP (*WS-ResourceProperties*) (GRAHAM, 2004b), o WSRF-RL (*WS-ResourceLifetime*) (SRINIVASAN, 2006), o WSRF-SG (*WS-ServiceGroup*) (MAGUIRE, 2006), e o WSRF-BF (*WS-BaseFaults*) (TUECKE, 2004). Os novos padrões que possibilitarão a convergência utilizam o WS-Transfer: são o WS-Transfer Addendum (ainda não existente) e o WS-Metadata Exchange 1.1 (BALLINGER, 2006a).

O WS-Transfer Addendum define extensões às mensagens Get, Put e Create, padronizando a especificação de um sub-conjunto de informações dos recursos gerenciados. Nas extensões das mensagens Put e Create, também passa a permitir que seja alterado o *endpoint* WS-Addressing do recurso.

O WS-Metadata Exchange 1.1 permite que meta-dados dos recursos gerenciados sejam gerenciados eles próprios como recursos, e altera a mensagem Get do padrão original para uso do Get do WS-Transfer.

Referenciando este conjunto de padrões novos e padrões que seguirão sendo utilizados, será definido o WS-RT (*Web Services Resource Transfer*), adicionado aos padrões WS-Transfer, WS-Enumeration, WS-Transfer Addendum e WS-Metadata Exchange 1.1 conceitos avançados existentes no WS-Resource Framework, como gerenciamento de sub-elementos e ciclo de vida.

A figura 2.9 exibe um panorama geral da relação entre os padrões citados. A pilha à direita indica os padrões que são usados para formar os padrões subseqüentes. Os padrões em paralelo, à esquerda, referenciados através de uma seta, indicam conceitos que foram do padrão origem da seta que foram utilizados no padrão destino da seta.

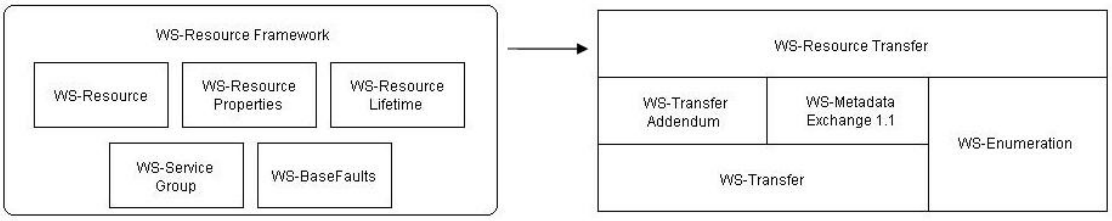


Figura 2.9: Convergência WS-Management/MUWS - Padrões para gerenciamento de recursos

Com relação à gerência de eventos, o WS-Management tem por base o WS-Eventing. O MUWS utiliza o WS-Notification, que é, por sua vez, composto por três padrões: o WS-BaseNotification (GRAHAM, 2006b), o WS-Topics (VAMBENEPE, 2006), e o WS-BrokeredNotification (CHAPPELL, 2006). Para a convergência, será criado um novo padrão, chamado WS-EventNotification, que é baseado no WS-Eventing, e que incorpora conceitos do WS-Notification, como: políticas de subscrição, linguagens complexas para filtragem de eventos, tratamento de subscrições como recursos gerenciados, e pausa de subscrições. O WS-EventNotification também utilizará o WS-ResourceTransfer, para manipulação de subscrições como recuros.

A figura 2.10 demonstra a relação entre esses padrões. Assim, como na figura 2.9, há pilha de padrões que forma o novo padrão, WS-EventNotification, é exibida à direita, e os padrões cujos conceitos foram utilizados são exibidos à esquerda.

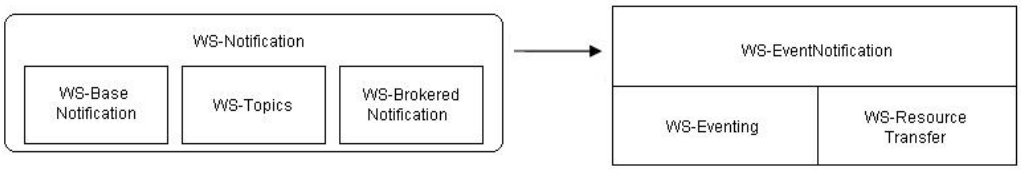


Figura 2.10: Convergência WS-Management/MUWS - Padrões para gerenciamento de eventos

O trabalho desenvolvido para convergência entre WS-Management e MUWS será formado por dois padrões, o WS-ResourceTransfer e o WS-EventNotification. A arquitetura proposta para estes padrões é baseada em WS-Eventing, WS-Transfer e WS-Enumeration, que são a base do WS-Management. Isso leva a conclusão de que o novo padrão terá grande similaridade com o WS-Management do que com o MUWS.

Essa análise foi considerada na implementação da arquitetura proposta neste trabalho, de forma que a implementação foi realizada apenas com base no WS-Management. Esta conclusão também foi utilizada na implementação do padrão "JSR (*Java Specification Request*) 262 – A Web Services Connector for JMX Agents" (SUN, 2008). Esta especificação Java consiste na definição de um formato para comunicação via Web Services com agentes JMX (*Java Management Extensions*), usados para o gerenciamento de aplicações Java. Foi decidido usar o WS-Management como protocolo de gerenciamento dos agentes JMX. O WS-Management foi escolhido em detrimento do MUWS, conforme exposto em (McMANUS, 2006), exatamente pelo fato do trabalho de convergência em andamento usar os padrões base do WS-Management.

2.2.5 Wiseman

Neste trabalho, será proposta uma arquitetura de segurança para gerenciamento de redes baseadas em Web Services. Há dois padrões na indústria para gerência via Web Services, o WS-Management e o MUWS. Como há trabalho em andamento para convergência entre esses padrões, e a indicação é de que o novo padrão será baseado no WS-Management, a implementação da arquitetura proposta neste trabalho usará o WS-Management. Existe uma API (*Application Programming Interface*) em Java para o WS-Management, o Wiseman (SUN, 2009), e que foi usada neste trabalho.

O Wiseman é uma biblioteca de código livre, com a licença Apache versão 2.0 (APACHE, 2004). O Wiseman encontra-se na versão 1.0, versão disponível desde 22 de junho de 2007. O escopo do Wiseman são as implementações cliente e servidor (aplicação de gerência e agente do recurso gerenciado) do WS-Management e das demais especificações WS utilizadas pelo WS-Management.

A distribuição do Wiseman, além das implementações de cliente e servidor, traz ferramentas auxiliares para o desenvolvimento de agentes Wiseman para recursos gerenciados, e de clientes (gerentes) que irão se comunicar com os agentes. A distribuição também traz um ambiente de execução para os Web Services que serão os pontos de comunicação com os recursos gerenciados. O ambiente de execução roda as implementações dos serviços oferecidos pelos agentes, e pode ser integrado a qualquer *container* Java EE (*Enterprise Edition*) para Web, como o Apache Tomcat (APACHE, 2009a).

A arquitetura do Wiseman é exibida na figura 2.11:

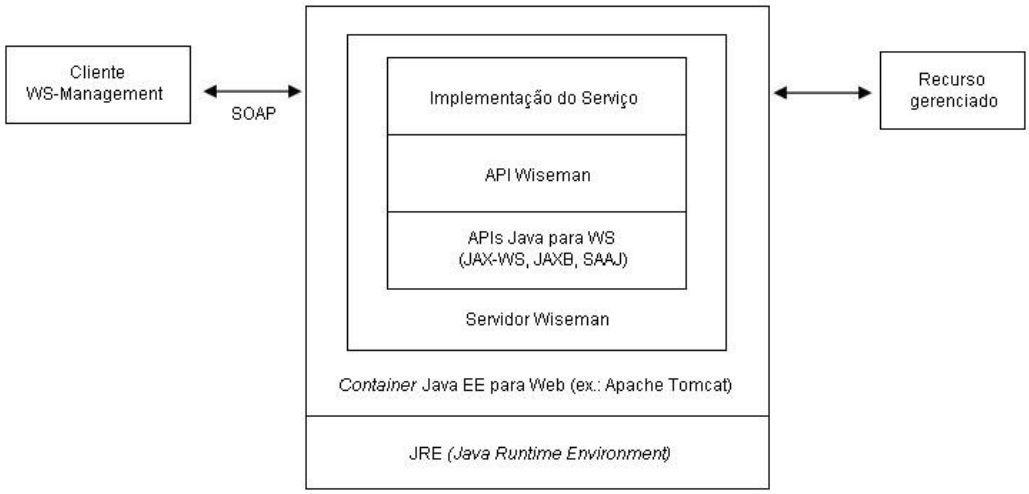


Figura 2.11: Arquitetura do Wiseman

O servidor Wiseman é executado sobre um servidor Java EE para aplicações Web comuns. Ele não pode ser integrado a servidores específicos para Web Services atualmente existentes, como o Apache Axis2 e o JWSDP (*Java Web Services Developer Pack*). Isso ocorre porque o endereçamento dos serviços implementados, no WS-Management, é necessariamente via WS-Addressing. Nos servidores de Web Services existentes, é utilizado o corpo dos documentos SOAP, conforme o WS-I (*Web Services Interoperability*) Basic Profile for Web Services (BALLINGER, 2006b). Porém, no WS-Management, algumas mensagens, como Get, possuem corpo vazio na mensagem SOAP, o que não é sequer permitido pelo WS-I Basic Profile. Além disso, o Axis2 e o JWSDP não suportam o endereçamento dos serviços via WS-Addressing, não podendo ser utilizados.

As ferramentas auxiliares de desenvolvimento que fazem parte da distribuição do Wiseman são as seguintes:

- Ferramenta para geração de descrição WSDL para serviços WS-Management, a partir de definição em XML Schema do recurso gerenciado.
- Ferramenta para geração de classes Java a partir de descrição WSDL de serviços WS-Management.
- Visualizador de meta-dados de serviços WS-Management.
- *Template* para projetos de agentes de recursos gerenciados.
- Classes auxiliares para a construção de clientes WS-Management.
- Utilitário para teste de uso de serviços WS-Management via linha de comando.

- Aplicações de exemplo demonstrando as facilidades do Wiseman.

2.3 Segurança em Web Services

O uso extensivo da tecnologia de Web Services em diversas áreas da computação, incluindo a gerência de redes, faz com que seja necessário que a troca de informações via Web Services seja segura. Um nível adequado de segurança para o gerenciamento de redes baseadas em Web Services tem grande relevância, pois redes trafegando dados que fazem parte do conhecimento estratégico das empresas de tecnologia possuem altos requisitos de segurança.

Esta seção aborda os conceitos e padrões utilizados na segurança em Web Services. No tocante a padrões, o WS-Security e o XACML são descritos com maior detalhamento, já que são os padrões utilizados neste trabalho. Também são detalhados nesta seção as implementações em Java utilizadas para WS-Security (combinação Axis2 + Rampart + WSS4J) e para XACML (SunXACML).

Os padrões existentes para segurança em XACML e Web Services são relativamente novos. Porém, os conceitos utilizados na construção desses padrões são tradicionais. As características básicas de uma arquitetura segura para ambientados orientados a serviços são as mesmas de aplicações distribuídas tradicionais, sendo:

- Identidade: o destinatário de uma mensagem deve ser capaz de identificar o remetente, ou seja, responder a pergunta “quem é você?”.

- Autenticação: o destinatário deve ser capaz de verificar que a identidade alegada pelo remetente é válida, ou seja, responder a pergunta “como sei que você é quem diz ser?”.

- Autorização: o destinatário deve poder determinar os níveis de autorização do remetente, por exemplo, no tocante às operações e dados aos quais o remetente possui direito de acesso. Em outros termos, responder a pergunta “o que você tem permissão para fazer?”.

- Integridade: uma mensagem deve permancer inalterada por todo o seu caminho de transmissão, desde o envio até a entrega.

- Confidencialidade: o conteúdo de uma mensagem não pode ser visualizado durante a transmissão, exceto por entidades autorizadas.

2.3.1 Padrões para segurança em Web Services

Há uma série de padrões relacionados a segurança em XML e Web Services. A tabela 2.2 exibe a relação entre esses padrões e os conceitos de segurança apresentados previamente.

Tabela 2.2: Padrões de segurança para Web Services

Identificação **Autenticação** **Autorização**	WS-Security *eXtensible Access Control Markup Language* (XACML) *eXtensible Rights Markup Language* (XrML) *XML Key Management Specification* (XKMS) *Security Assertion Markup Language* (SAML)
Confidencialidade	WS-Security XML-Encryption *Transport Layer Security* (TLS) *Secure Sockets Layer* (SSL)
Integridade	WS-Security XML-Digital Signature

Os padrões da tabela 2.2 são detalhados a seguir, com exceção do WS-Security e XACML, que são apresentadas com mais abrangência em seções posteriores deste trabalho:

- *eXtensible Rights Markup Language* (XrML) (XRML, 2001): utilizada quando arquivos com conteúdo sujeito a *copyright* são transportados via Web Services. Arquivos podem ser anexados a Web Services, por exemplo, através de SOAP Attachments (NIELSEN, 2004).

- *XML Key Management Specification* (XKMS) (FORD, 2001): estabelece uma forma de obter e gerenciar chaves públicas via XML. O XKMS é compatível com diversas tecnologias de infra-estrutura de chave primaria, mas não exige o uso de nenhuma delas, tornando desnecessária a integração de produtos de *Public Key Infrastructure* (PKI) proprietários. Esta especificação consiste em 2 padrões complementares: a *XML Key Registration Service Specification* (X-KRSS) e a *XML Key Information Service Specification* (X-KISS). O X-KRSS é um protocolo para registro de pares de chaves, e as funções do X-KISS são a localização das chaves, e a associação destas chaves a informações de identificação.

- *Security Assertion Markup Language* (SAML) (CANTOR, 2007): o SAML é um padrão do tipo denominado *single sign-on*, ou seja, voltado ao acesso a várias aplicações correlacionadas através de uma única identificação do usuário. Tecnologias *single sign-on* são importantes em sistemas com aplicações distribuídas e controles de acesso independentes. O SAML trata da autenticação e autorização com *single sign-on* via Web Services, o que tem alta relevância, pois os Web Services contribuem para ao aumento dos canais de acesso a um sistema. Mensagens de requisição e resposta são definidas para facilitar a transmissão de credenciais de usuário entre Web Services.

- XML-Encryption (EASTLAKE, 2002): modelo em XML para encriptar tanto dados contidos no próprio XML, quanto dados textuais e binários referenciados por documentos XML. Esta especificação também aborda a comunicação de

informações essenciais para que os destinatários das mensages possam decriptá-las.

- XML-Digital Signature (EASTLAKE, 2008): formato para representar assinaturas digitais em XML. Assinaturas digitais acrescentam credibilidade às mensagens, assegurando que foram enviadas pelo remetente esperado. Além disso, garantem que o conteúdo da mensagem não foi alterado na transmissão. Conforme o XML-Encryption, é possível assinar digitalmente o conteúdo de um arquivo XML, assim como dados binários e textuais externos.

- *Transport Layer Security* (TLS) (DIERKS, 2008) e seu predecessor, *Secure Sockets Layer* (SSL) (FREIER, 1996): protocolos criptográficos que provêem segurança ao transporte de dados na Internet. A combinação de uma conexão HTTP através de uma conexão segura TLS ou SSL é denominada HTTPS, que é o protocolo padrão para troca de mensagens seguras na Internet. Desta forma, o transporte mais seguro para Web Services é o HTTPS.

No tocante à segurança que o uso de HTTPS traz para os Web Services, surge a necessidade de um padrão de segurança específico para Web Services. Isso ocorre porque o HTTPS provê segurança para uma conexão ponto-a-ponto entre 2 Web Services. Para uma interação Web comum, entre um browser e um servidor Web, isso é suficiente, pois o servidor Web age diretamente na lógica de negócio da aplicação executada, ou se comunica com esta lógica de alguma outra forma.

A conexão ponto-a-ponto HTTPS é usada como transporte para troca de mensagens entre Web Services. Caso hajam serviços intermediários envolvidos nesta troca, o uso de HTTPS não garante a segurança da comunicação como um todo. A segurança existe nas conexões HTTPS entre os intermediários, mas para realizar a retransmissão da mensagem para o local necessário, os serviços intermediários precisam abrir toda a mensagem XML trafegada. Esses serviços intermediários só precisariam conhecer as informações de endereçamento da mensagem, que pode estar no *header* da mesma, em *tags* WS-Addressing; não é necessário que os serviços intermediários tenham acesso ao conteúdo completo das mensagens.

É necessário que haja segurança fim-a-fim para o conteúdo das mensagens XML trocadas entre os Web Services finais; isso é possibilitado pela especificação WS-Security, que é apresentada na próxima seção. A figura 2.12 demonstra claramente a diferença entre segurança ponto-a-ponto e fim-a-fim em uma comunicação entre Web Services envolvendo intermediários.

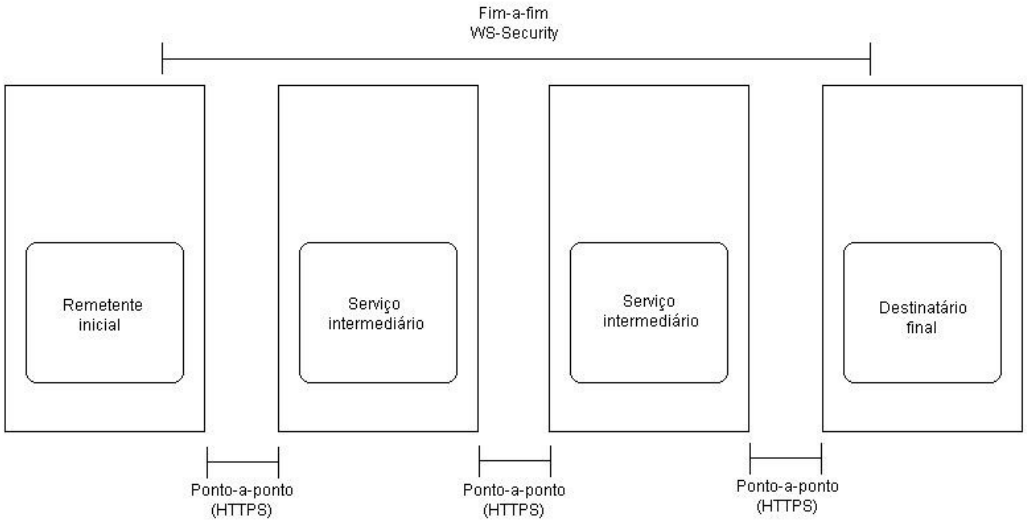


Figura 2.12: Segurança ponto-a-ponto e fim-a-fim para Web Services

2.3.2 WS-Security

A especificação *Web Services Security* (WS-Security) foi desenvolvida inicialmente pelas empresas IBM, Microsoft e VeriSign. Ela foi padronizada pelo OASIS em 19 de abril 2004, em sua versão 1.0. A versão atual é a 1.1, lançada em 17 de fevereiro de 2006.

O WS-Security traz um modelo de segurança acima da tradicional segurança no nível de transporte (HTTP), provendo um padrão fim-a-fim para segurança no nível das mensagens SOAP. Isso é feito através de blocos de segurança no *header* das mensagens SOAP, com informações de criptografia e assinaturas digitais (geradas com XML-Encryption e XML-Digital Signature), por exemplo. O conteúdo da mensagem se torna um texto criptografado e/ou assinado digitalmente, entre *tags* XML comuns. O WS-Security também descreve como acrescentar *tokens* de autenticação às mensagens SOAP, como certificados X.509 (COOPER, 2008) e *tickets* Kerberos (NEUMAN, 2005), ou mesmo informação em texto livre de usuário e senha. Todas estas informações são organizadas num elemento chamado `Security`, que é utilizado para identificar os atributos WS-Security no *header* de uma mensagem SOAP.

A figura 2.13 exibe um exemplo de mensagem WS-Security:

```
(001) <?xml version="1.0" encoding="utf-8"?>
(002) <S11:Envelope xmlns:S11="..." xmlns:wsse="..." xmlns:wsu="..." xmlns:xenc="..."
xmlns:ds="...">
(003)   <S11:Header>
(004)      <wsse:Security>
(005)         <wsu:Timestamp wsu:Id="T0">
(006)            <wsu:Created>2001-09-13T08:42:00Z
(007)            </wsu:Created>
(008)         </wsu:Timestamp>
(009)
(010)         <wsse:BinarySecurityToken
ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-
1.0#X509v3" wsu:Id="X509Token" EncodingType="...#Base64Binary">
(011)            MIIEZzCCA9CgAwIBAgIQEmtJZc0rqrKh5i...
```

```
(012)         </wsse:BinarySecurityToken>
(013)         <xenc:EncryptedKey>
(014)            <xenc:EncryptionMethod Algorithm=
"http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
(015)            <ds:KeyInfo>
(016)               <wsse:SecurityTokenReference>
(017)                  <wsse:KeyIdentifier EncodingType="...#Base64Binary"
ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-
1.0#X509v3">MIGfMa0GCSq...
(018)                  </wsse:KeyIdentifier>
(019)               </wsse:SecurityTokenReference>
(020)            </ds:KeyInfo>
(021)            <xenc:CipherData>
(022)               <xenc:CipherValue>d2FpbmdvbGRfE0lm4byV0...
(023)               </xenc:CipherValue>
(024)            </xenc:CipherData>
(025)            <xenc:ReferenceList>
(026)               <xenc:DataReference URI="#enc1"/>
(027)            </xenc:ReferenceList>
(028)         </xenc:EncryptedKey>
(029)         <ds:Signature>
(030)            <ds:SignedInfo>
(031)               <ds:CanonicalizationMethod
Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
(032)               <ds:SignatureMethod WSS: SOAP Message Security (WS-Security 2004) 01
November 2006 Copyright © OASIS Open 2002-2006. All Rights Reserved. Page 55 of 76
Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
(033)               <ds:Reference URI="#T0">
(034)                  <ds:Transforms>
(035)                     <ds:Transform
Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
(036)                  </ds:Transforms>
(037)                  <ds:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
(038)                  <ds:DigestValue>LyLsF094hPi4wPU...
(039)                  </ds:DigestValue>
(040)               </ds:Reference>
(041)               <ds:Reference URI="#body">
(042)                  <ds:Transforms>
(043)                     <ds:Transform
Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
(044)                  </ds:Transforms>
(045)                  <ds:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
(046)                  <ds:DigestValue>LyLsF094hPi4wPU...
(047)                  </ds:DigestValue>
(048)               </ds:Reference>
(049)            </ds:SignedInfo>
(050)            <ds:SignatureValue>
(051) Hp1ZkmFZ/2kQLXDJbchm5gK...
(052)            </ds:SignatureValue>
(053)            <ds:KeyInfo>
(054)               <wsse:SecurityTokenReference>
(055)                  <wsse:Reference URI="#X509Token"/>
(056)               </wsse:SecurityTokenReference>
(057)            </ds:KeyInfo>
(058)         </ds:Signature>
(059)      </wsse:Security>
(060)   </S11:Header>
(061)   <S11:Body wsu:Id="body">
(062)      <xenc:EncryptedData Type="http://www.w3.org/2001/04/xmlenc#Element"
wsu:Id="enc1">
(063)         <xenc:EncryptionMethod
Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc"/>
(064)         <xenc:CipherData>
(065)            <xenc:CipherValue>d2FpbmdvbGRfE0lm4byV0...
(066)            </xenc:CipherValue>
(067)         </xenc:CipherData>
(068)      </xenc:EncryptedData>
(069)   </S11:Body>
(070) </S11:Envelope>
```

Figura 2.13: Exemplo de mensagem WS-Security

O padrão WS-Security é a base de um conjunto de especificações suplementares visando sistemas seguros usando Web Services. Este conjunto pode ser visualizado na figura 2.14.

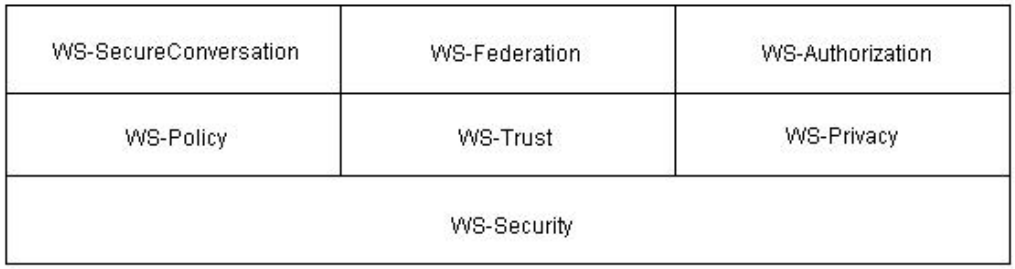


Figura 2.14: Especificações baseadas em WS-Security

O conjunto de especificações da figura 2.14 inclui tanto especificações finalizadas e passando por revisões, quanto especificações em planejamento. Estas especificações são detalhadas a seguir:

- *Web Services Policy* (WS-Policy) (VEDAMUTHU, 2007): esta especificação não é limitada ao escopo da segurança. É um padrão para expressar políticas gerais de sistemas, podendo ser usada, portanto, para definir políticas de segurança.

- *Web Services Trust Language* (WS-Trust) (NADALIN, 2007): padrão para modelos de confiança, usados para validar *tokens* de segurança trocados entre sistemas. O padrão prevê um mecanismo para requisitar um auxílio de autoridades de confiança para auxiliar na validação.

- WS-Privacy (em planejamento): usada para comunicar políticas de privacidade entre sistemas, e para verificar se serviços usuários concordam com estas políticas.

- *Web Services Secure Conversation Language* (WS-SecureConversation) (NADALIN, 2008): provê definições para criação de contextos de segurança, permitindo a troca de informações de segurança, como as chaves associadas à comunicação.

- WS-Federation (em planejamento): padrão para estabelecimento de uma federação entre sistemas distintos, ou seja, um ambiente em que um nível de confiança é acordado entre domínios de confiança distintos.

- WS-Authorization (em planejamento): controle de acesso à leitura e escrita de informações.

2.3.3 *eXtensible Access Control Markup* Language (XACML)

Conforme exposto na seção anterior, há um conjunto de padrões baseados em WS-Security para a construção de sistemas seguros usando Web Services. Entre esses padrões, está previsto o WS-Authorization, para controle de acesso. Como esta é uma área relevante da segurança, e o padrão WS-Authorization está em planejamento, este trabalho usará o *eXtensible Access Control Markup Language* (XACML) para implementação e testes relativos à controle de acesso. Atualmente, este é o padrão de fato para controle de acesso em XML, e portanto, pode ser usada no gerenciamento de redes via Web Services.

O XACML é um padrão estabelecido pelo OASIS, cuja primeira versão foi publicada em 6 de fevereiro de 2003. Atualmente, a especificação encontra-se na versão 2.0, de 28 de fevereiro de 2008.

O XACML é composto por duas linguagens: uma para definição de políticas de acesso, e outra para requisição e resposta relativas a decisões de acesso. A idéia básica para definição dessas linguagens é o cenário tradicional utilizada para controle de acesso: algum sujeito deseja realizar alguma ação sobre um recurso. Este sujeito encaminha uma requisição com a ação desejada para uma entidade responsável pelo controle de acesso ao recurso. Esta entidade é denominada *Policy Enforcement Point* (PEP).

O PEP constrói uma requisição com uma série de informações: sujeito que requisitou a ação, o recurso em questão, a ação desejada, e outras informações relevantes. Esta requisição é enviada pela PEP ao *Policy Decision Point* (PDP), que procura em um conjunto de políticas a política aplicável à requisição. Com essa política, o PDP obtém uma resposta definindo se a ação deve ser permitida ou negada. Esta resposta é enviada ao PEP, que permite ou nega a ação com base na resposta.

A definição das políticas XACML se dá através de documentos `Policy` e `PolicySet`. Os componentes destes documentos podem ser visualizados na figura 2.15. Os componentes são apresentados em inglês para que possa ser feita correspondência direta com a especificação.

Conforme a figura 2.15, um `PolicySet` é composto por um ou mais documentos `Policy`. O conjunto de políticas de um PDP sempre precisa ser definido a partir de documentos `Policy`, e opcionalmente, pode-se combiná-las através de `PolicySets`. As políticas são combinadas em um `PolicySet` através de algoritmos de combinação de políticas, como `deny-overrides`, `permit-overrides` e `first-applicable`. Os documentos `Policy` são compostos por `Rules`, que podem ser combinada por algoritmos similares. Cada regra possui um `Effect` (`Deny` ou `Permit`) e, opcionalmente, uma `Condition`. As políticas podem conter `Obligations`, que são similares às `Conditions`. `Obligations` e `Conditions` podem ser definidas com operadores lógicos, aritméticos e customizados, e operam sobre parâmetros dos `Subjects`, `Resources`, `Actions`, e `Environment`. Estes parâmetros também são usados para a definição do `Target`, que é o conjunto de informações que determinam quando a política deve ser aplicada. Um determinado `Target` deve ter uma única política associada em um PDP.

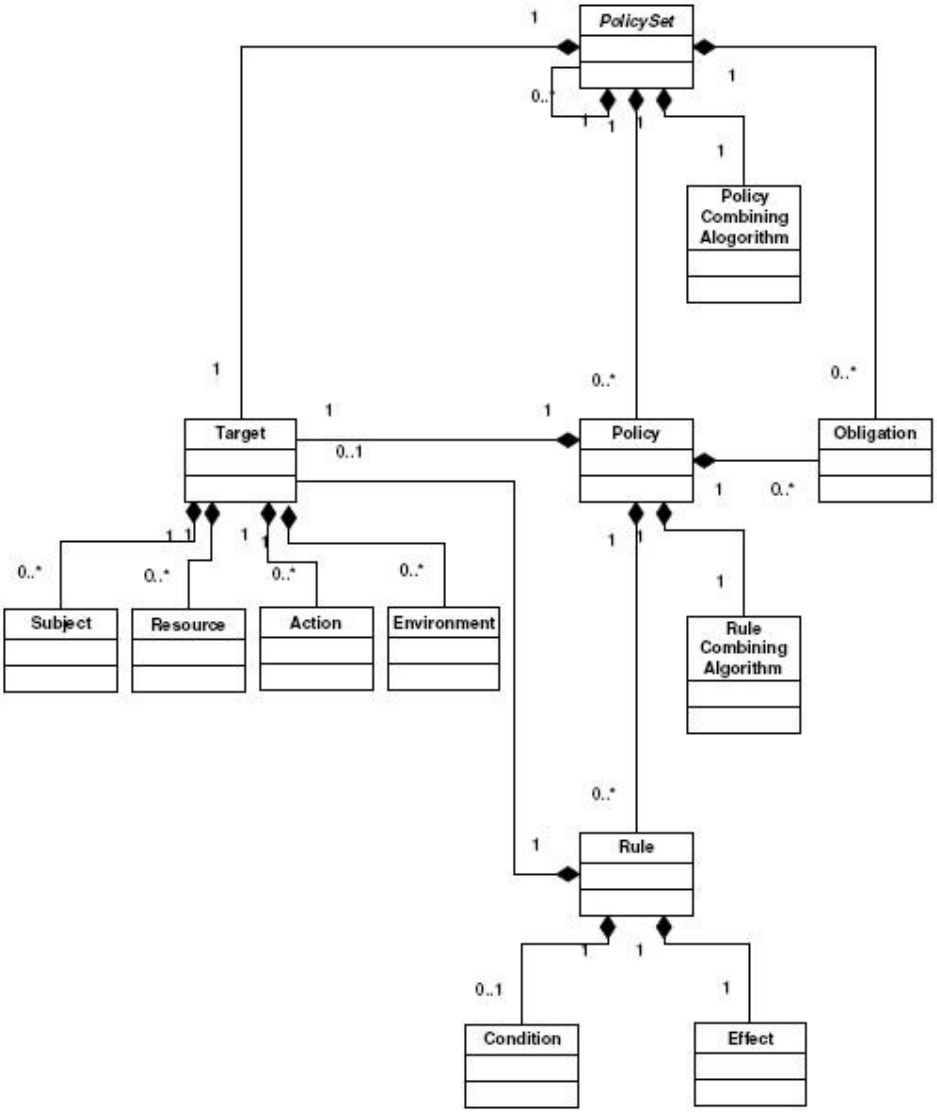


Figura 2.15: Definição de políticas XACML

A definição de uma requisição de acesso se dá através do documento `Request`, e decisão é informada através do documento `Response`. Uma `Request` é composta por informações que levam a um `Target`, ou seja, definição de `Subject`, `Resource`, `Action` e/ou `Environment` da requisição. A resposta contém a informação de `Permit`, `Deny`, ou `NotApplicable`.

A figura 2.16 é um exemplo de política XACML:

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

  <Description>
    This policy applies to any accounts at users.example.com accessing
    server.example.com. The one Rule applies to the specific action of
```

```
    doing a CVS commit, but other Rules could be defined that handled
    other actions. In this case, only certain groups of people are
    allowed to commit. There is a final fall-through rule that always
    returns Deny.
  </Description>

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

  <Rule RuleId="CommitRule" Effect="Permit">
    <Target>
      <Subjects>
        <AnySubject/>
      </Subjects>
      <Resources>
        <AnyResource/>
      </Resources>
      <Actions>
        <Action>
          <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
            <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">commit</AttributeValue>
            <ActionAttributeDesignator   DataType=http://www.w3.org/2001/XMLSchema#string
AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"/>
          </ActionMatch>
        </Action>
      </Actions>
    </Target>

    <Condition FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
        <SubjectAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema#string"
                                    AttributeId="group"
                                    Issuer="admin@users.example.com"/>
      </Apply>
      <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">developers</AttributeValue>
    </Condition>
  </Rule>

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

</Policy>
```

Figura 2.16: Exemplo de política XACML

A figura 2.17 exemplifica uma requisição XACML:

```
<Request xmlns="urn:oasis:names:tc:xacml:1.0:context"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <Subject>
```

```
    <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
               DataType="urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name">
      <AttributeValue>seth@users.example.com</AttributeValue>
    </Attribute>
    <Attribute AttributeId="group"
               DataType="http://www.w3.org/2001/XMLSchema#string"
               Issuer="admin@users.example.com">
      <AttributeValue>developers</AttributeValue>
    </Attribute>
  </Subject>
  <Resource>
    <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
               DataType="http://www.w3.org/2001/XMLSchema#anyURI">
      <AttributeValue>http://server.example.com/</AttributeValue>
    </Attribute>
  </Resource>
  <Action>
    <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
               DataType="http://www.w3.org/2001/XMLSchema#string">
      <AttributeValue>commit</AttributeValue>
    </Attribute>
  </Action>
</Request>
```

Figura 2.17: Exemplo de requisição XACML

2.3.4 Axis2, Rampart e WSS4J

Este trabalho tem por objetivo a definição de uma arquitetura de segurança para gerenciamento de redes via Web Services, e também a implementação desta arquitetura. Para o tratamento de segurança em Web Services, serão utilizados o WS-Security e o XACML. A implementação de XACML usada será o SunXACML, abordado na seção 2.3.5. A implementação para WS-Security se dará através de um conjunto de ferramentas: Axis2, Rampart e WSS4J. A seguir essas três ferramentas são detalhadas.

O Apache Axis2 (AXIS2, 2008) é uma arquitetura de Web Services baseada em SOAP. Há implementações desta arquitetura em Java e em C. O Apache Axis2 em Java (doravante denominado simplesmente Axis2) encontra-se atualmente na versão 1.4.1, disponível desde 25 de agosto de 2008. Esta arquitetura é uma evolução de um projeto anterior com o mesmo foco, denominado Apache Axis.

O Axis2 implementa os lados cliente e servidor para comunicação entre Web Services. A arquitetura é modular, de forma a possibilitar a integração de quaisquer padrões baseados em Web Services. O Axis2 pode ser utilizado para envio e recepção simples de mensagens SOAP, bem como para a criação de clientes e servidores para Web Services específicos.

A figura 2.18 exibe a arquitetura cliente/servidor utilizada no Axis2. Nos blocos mais externos, “Aplicação” e “Lógica de negócio”, estão as aplicações que realmente tratam o conteúdo das mensagens trocadas. Os demais blocos são preenchidos pelo Axis2. O Axis2 pode estar rodando apenas no cliente, apenas no servidor, ou em ambos. As funções executadas por esses blocos são as seguintes:

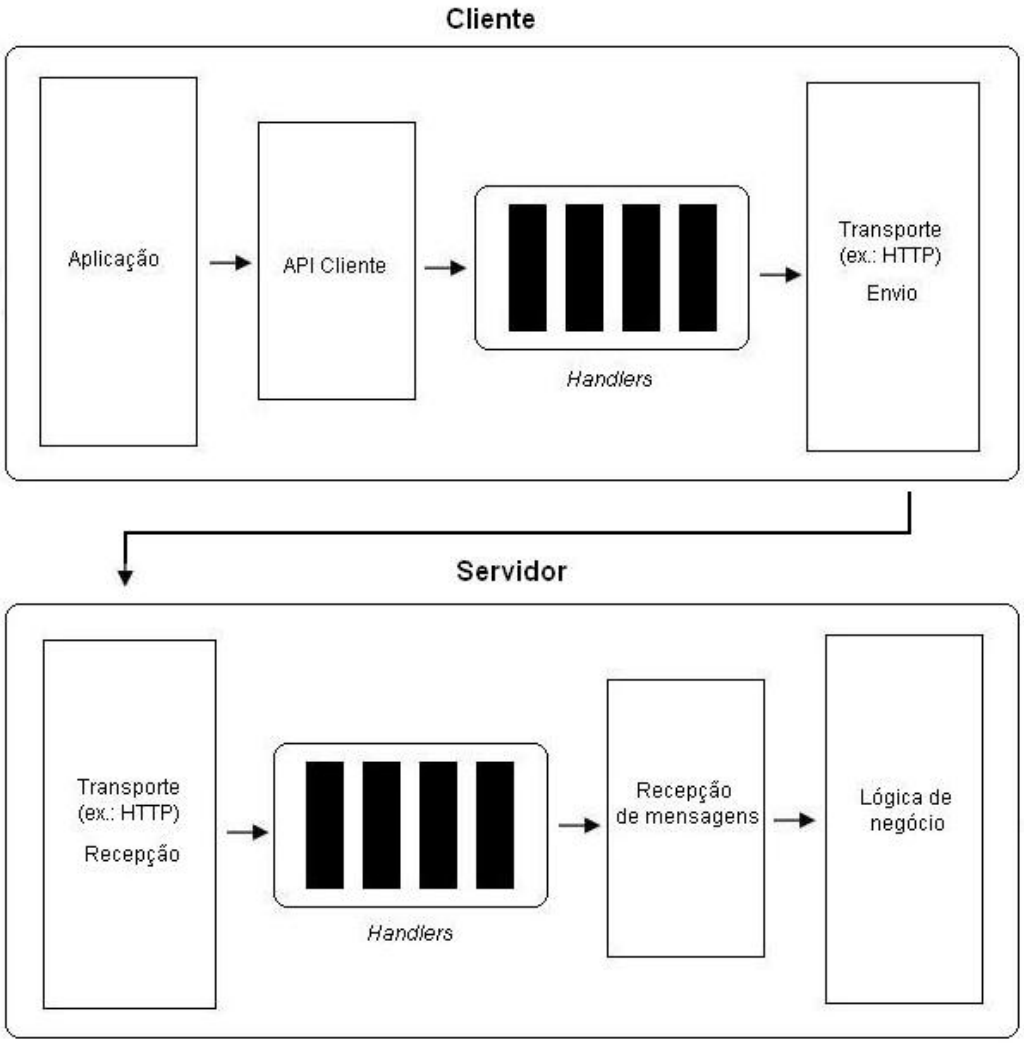


Figura 2.18: Arquitetura do Apache Axis2

- Cliente: a API cliente constrói a mensagem SOAP. Os *Handlers* no cliente realizam quaisquer ações necessárias sobre a mensagem, como por exemplo, criptografia via WS-Security. Por fim, a mensagem é enviada através do protocolo de transporte, comumente HTTP/HTTPS.

- Servidor: a mensagem é recebida no servidor via protocolo de transporte. *Handlers* no servidor executam ações necessárias, como decriptação via WS-Security. A mensagem é então processada na recepção, para que o conteúdo relevante seja repassado à lógica de negócio.

Estas ações, no Axis2, são executadas através de uma série de fases. Algumas são pré-definidas, como *pre-dispatch*, *dispatch* e *message processing*, mas também é possível criar novas fases e incorporá-las a um determinado cliente ou servidor. Cada fase é composta por *handlers* (o que poderia ser traduzido por "serviços de tratamento"), e os mesmos podem ser reordenados, retirados ou incorporados. O desenvolvimento de *handlers* próprios também é possível.

A flexibilização no desenvolvimento e manipulação de fases e *handlers* customizados no processamento das mensagens é o que facilita a integração do suporte a diferentes padrões baseados em Web Services. A integração de novas fases e *handlers* se dá através de conjuntos lógicos denominados módulos. Os módulos são o mecanismo principal de extensão no Axis2.

O projeto Apache Axis2, além de contemplar o desenvolvimento do próprio Axis2, também possui alguns módulos que podem ser integrados ao Axis2 conforme as necessidades de cada aplicação. Os módulos disponíveis são os seguintes:

- *Addressing*: implementação do WS-Addressing, atualmente na versão 1.4.

- *SOAP Monitor*: possibilita que os desenvolvedores monitorem as mensagens SOAP recebidas e enviadas sem necessidade de configurações específicas ou reinicialização do servidor. Atualmente está na versão 1.4.

- *Sandesha2*: implementação do padrão *Web Services Reliable Messaging* (WS-RM) (DAVIS, 2008), que define entrega confiável de mensagens entre Web Services na presença de falhas. Este módulo encontra-se na versão 1.3.

- *Rampart*: implementação para WS-Security e WS-SecureConversation. A configuração dos *handlers* pode ser feita através de um modelo baseado no padrão *Web Services Security Policy Language* (WS-SecurityPolicy) (NADALIN, 2006b). Este módulo está na versão 1.3.

O módulo de interesse para este trabalho é o Rampart (APACHE, 2009c), que possibilita o uso de criptografia, assinaturas digitais e autenticação conforme o padrão WS-Security. Configurando o Axis2 para utilizar determinados *handlers* do Rampart, pode-se facilmente ativar ou desativar pontualmente cada uma destas ações de segurança do WS-Security. Isso é particularmente útil para avaliações de desempenho, em que é necessário apenas alterar arquivos de configuração que definem se a mensagem deve ter criptografia, autenticação, etc. Assim, os diferentes comportamentos de segurança são facilmente chaveados.

O Rampart permite a configuração de uma série de parâmetros em dois grupos. Os grupos de parâmetros são denominados *InflowSecurity* e *OutflowSecurity* (ou seja, pode-se configurar parâmetros diferente para o envio e recepção de mensagens, seja no cliente ou no servidor). Através de cada parâmetro, se define se deve ser empregada criptografia, assinaturas digitais e autenticação. Também são definidas informações necessárias para a realização dessas funções, como algoritmo de criptografia a utilizar, chaves públicas, como as senhas de usuário devem ser obtidas, etc.

As principais funções implementadas no Rampart dizem respeito à flexibilização de configurações e à integração de funções do WS-Security ao Axis2. A implementação das funções do WS-Security em si, usadas no Rampart, fazem parte de outra API independente, o *Web Services Security for Java* (WSS4J) (APACHE, 2009b).

O WSS4J é uma API que pode ser utilizada para uso de WS-Security em aplicações que não utilizam Axis2, e está atualmente na versão 1.5.4. O WSS4J possui

implementação para o uso de XML-Encryption, XML-Signature, *timestamps* e *tokens* de autenticação. Os *tokens* podem ser baseados em usuário/senha ou em SAML.

2.3.5 SunXACML

O SunXACML (SOURCEFORGE, 2009b) é uma implementação em código aberto do padrão XACML. O projeto iniciado pela Sun, por isso o nome da ferramenta. O SunXACML está na versão 1.2.

Todas as definições obrigatórias do XACML são implementadas. Ou seja, estão inclusos *parsing* de documentos *Policy, Request* e *Response*, determinação de aplicabilidade de políticas a requisições, e avaliação de requisições sobre as políticas aplicáveis. Os tipos, funções e algoritmos de combinação de políticas e regras são suportados, e há API para definições de novas funcionalidades (por exemplo, definir um novo tipo de dado, como um OID SNMP). Também há APIs para escrita de mecanismos específicos de procura e obtenção de políticas e atributos de recursos. O SunXACML também possibilita a criação de PEPs e PDPs, que são o mecanismo básico para aplicação do controle de acesso via XACML.

2.4 SNMP e Segurança

Este trabalho tem como foco a segurança em gerência de redes baseadas em Web Services. Para que uma arquitetura de segurança seja planejada e implementada, e resultados de uma análise de desempenho possam ser analisados, é importante o conhecimento das características de segurança do protocolo tradicional de gerência de redes, o SNMP. Nesta seção, é traçado um panorama do tratamento de segurança existente no SNMP.

Nas versões iniciais do protocolo, SNMPv1 e SNMPv2, não há praticamente nenhum suporte à segurança. Todas as necessidades de uma comunicação segura, como autenticação, autorização e confidencialidade, são abstraídas em um mecanismo trivial e pouco robusto: as *communities* (comunidades) SNMP.

Uma *community* SNMP é uma relação entre um agente e um gerente SNMP. O conceito da *community* é local, definido de forma independente no agente e no gerente. A *community* possui um nome que a identifica. Podem ser definidas *communities* com direto apenas de leitura, ou direito de leitura e escrita. Sempre que um gerente faz uma requisição para um agente, envia na mensagem SNMP a *community* conhecida para aquele agente (comumente, os gerentes SNMP possuem uma *community* default de leitura, outra de escrita, e *communities* específicas para agentes cujas *communities* são diferentes das *default*). Caso a *community* enviada pelo gerente não corresponda à exigida pelo agente para realizar a operação desejada, a operação é negada.

Este mecanismo é claramente limitado, sendo que ainda adiciona-se o fato de que a comunicação SNMPv1 e SNMPv2 é em claro, não usando nenhuma forma de criptografia. Portanto, estes protocolos, na prática, não podem ser usados em redes onde a segurança é necessária. Por isso, na indústria, a gerência de configurações, por exemplo, raramente é feita via SNMP, pois isso daria margem para indivíduos mal-intencionados alterarem as configurações de uma rede, denegrindo ou inviabilizando seu funcionamento.

A versão SNMPv3 do protocolo atende os problemas de segurança das versões anteriores. Nesta seção, é apresentado um breve histórico da evolução das versões de SNMP, mostrando como a segurança foi tratada nessa evolução. Também é apresentada uma visão geral das funcionalidades de segurança do SNMPv3.

2.4.1 Evolução das versões de SNMP

A origem do padrão SNMP é um padrão pré-existente, o *Simple Gateway Monitoring Protocol* (SGMP) (DAVIN, 1987). O objetivo do SGMP é a monitoração de *gateways* de rede (roteadores, tipicamente). O SNMP surgiu a partir do SGMP, com a extensão do mesmo para a gerência de dispositivos de rede em geral. A versão final do SNMP é de 1990, mas há versões preliminares de 1988 e 1989. (CASE, 1990) obsoletou (CASE, 1989), que por sua vez, obsoletou (CASE, 1988), a primeira versão padronizada do SNMP. A versão hoje conhecida por SNMPv1 é a de (CASE, 1990).

Uma evolução subseqüente ao SNMPv1 foi a MIB *Remote Network Monitoring* (RMON) (WALDBUSSER, 2000), cuja primeira versão foi publicada em 1991. Esta MIB resolveu um problema do SNMPv1: próprio para monitoração de nós de uma rede, mas não para a monitoração de uma rede como um todo.

As falhas de segurança foram inicialmente tratadas num conjunto de documentos denominado Secure SNMP, em julho de 1992. No mesmo mês, também foi publicado, fora da estrutura de padrões da Internet, o *Simple Management Protocol* (SMP). O SMP continha uma série de melhorias em relação ao SNMPv1, no tocante à performance e funcionalidades, incluindo conceitos do RMON e do Secure SNMP. O SMP foi utilizado como ponto de partida para o desenvolvimento do SNMPv2.

O trabalho de padronização do SNMPv2 foi dividido em um grupo de trabalho com foco em segurança, e outro grupo para os demais assuntos (SMI, MIB, estruturação, coexistência com SNMPv1). Estes grupos de trabalho geraram a publicação original do SNMPv2, em (CASE, 1993). Porém, não houve adoção prática desse padrão. O mesmo foi revisto em 1996, e publicado sem as funcionalidades de segurança, devido à conclusão de que o trabalho de segurança realizado não era adequado. As RFCs 1901 à 1908 contêm a revisão final do SNMPv2, conhecida por SNMPv2c (*Community-based SNMPv2*).

Em seguida, novos grupos independentes iniciaram trabalho pra adição de segurança. Os principais foram o SNMPv2u e o SNMPv2*. Ambos foram base para o SNMPv3.

A figura 2.19 resume a evolução das versões do SNMP, conforme (STALLINGS, 1999):

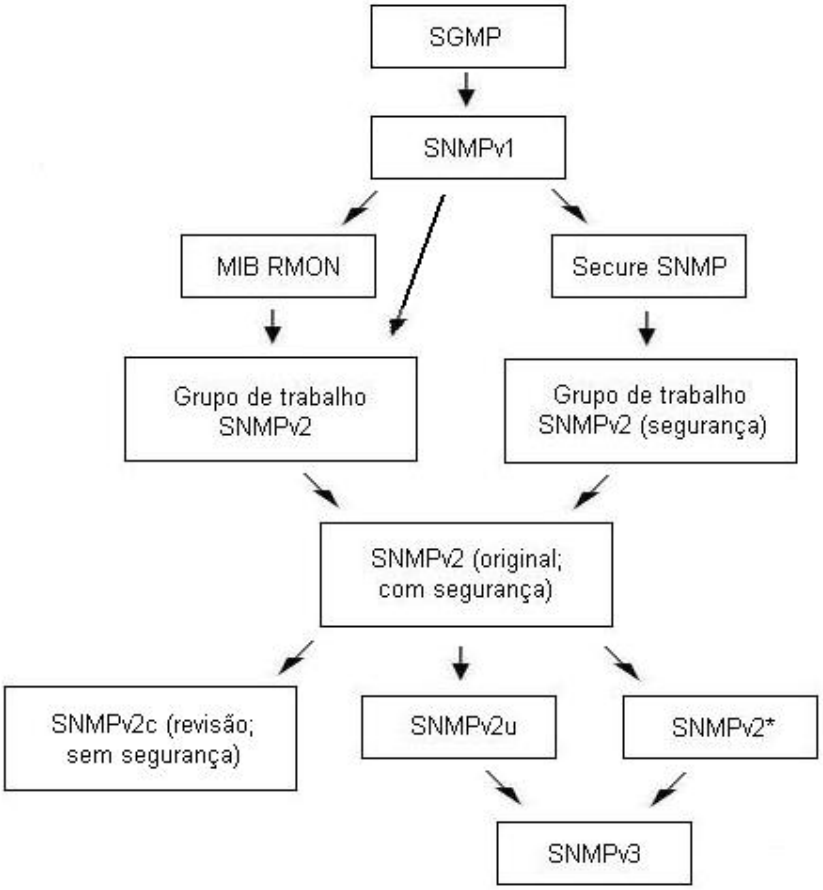


Figura 2.19: Evolução das versões do SNMP

2.4.2 SNMPv3

O grupo de trabalho do SNMPv3 tinha seu principal foco na ausência de segurança do SNMPv2. Porém, a documentação resultante do trabalho deste grupo, e que é comumente denominada SNMPv3, não constitui a especificação completa de um protocolo seguro para gerência de redes. Os documentos definem um conjunto de características de segurança desejáveis e um *framework* para integrá-las às funcionalidades do SNMPv2 ou SNMPv1. O *framework* também é adequado para integração de novas características funcionais, bem como melhorias e alterações na parte de segurança.

Nesta seção, a nomenclatura dos módulos da arquitetura proposta pelo SNMPv3 será apresentada em inglês, para evitar ambigüidades com relação ao protocolo original.

O grupo de trabalho do SNMPv3 produziu cinco documentos, as RFCs 2271 à 2275, em janeiro de 1998. Estas RFCs foram obsoletadas em dezembro de 2002 pelas RFCs 3411 à 3415, que, na ordem numérica, correspondem aos documentos anteriores. A seguir são citadas as atuais RFCs:

- RFC 3411 (*An Architecture for Describing SNMP Management Frameworks*): descreve uma arquitetura de *frameworks* para o SNMP. A arquitetura é modular

para permitir a evolução dos padrões SNMP no futuro, e é baseada numa entidade SNMP, composta por aplicações e por um *engine*. O modelo da entidade SNMP é válido para gerentes e agentes, como será detalhado nesta seção.

- RFC 3412 (*Message Processing and Dispatching for SNMP*): descreve os módulos de processamento e envio/recebimento de mensagens, que fazem parte da *engine* SNMP definida na RFC 3411. São definidos os procedimentos para o processamento adequado de mensagens de múltiplas versões SNMP por um mesmo *engine*. O documento também descreve um dos possível modelos de processamento de mensagens, o *SNMPv3 Message Processing Model*.

- RFC 3413 (*SNMP Applications*): descreve cinco tipos de aplicações de uma entidade SNMP: *Command Generators*, *Command Responders*, *Notification Originators*, *Notification Receivers* e *Proxy Forwarders*.

- RFC 3414 (*User-Based Security Model (USM) for SNMPv3*): descreve o modelo do SNMPv3 para prover segurança à troca de mensagens SNMP, denominado USM. Também é definida uma MIB para gerenciamento remoto de parâmetros de configuração do USM.

- RFC 3415 (*View-Based Access Control Model (VACM) for SNMP*): descreve o modelo do SNMPv3 para prover controle de acesso à informações de gerenciamento SNMP, denominado VACM. Também é definida uma MIB para gerenciamento remoto de parâmetros de configuração do VACM.

No restante dessa seção, são abordados com maior detalhe a entidade SNMP definida na RFC 3411; o modelo USM da RFC 3414; e o modelo VACM da RFC 3415, que foi usado nesse trabalho para comparação com a arquitetura de controle de acesso para Web Services que foi implementada baseada em XACML.

2.4.3 Entidade SNMP

A arquitetura projetada na RFC 3411 (HARRINGTON, 2002) consiste de uma coleção distribuída de entidades SNMP interagindo entre si. Cada entidade pode fazer o papel de um gerente, de um agente, ou mesmo de ambos. A entidade SNMP possui uma *engine* SNMP, que é responsável pelo processamento, envio e recebimento de mensagens, e também pelas funções de segurança, como autenticação, criptografia e controle de acesso. As funções da *engine* são disponibilizados como serviços para as aplicações da entidade SNMP. Em um gerente, as aplicações SNMP tipicamente provêem serviços utilizados por sistemas de alto nível, utilizados por administradores e operadores de rede. Em um agente, as aplicações SNMP são responsáveis pela instrumentação das MIBs, ou seja, pela recuperação e alteração de informações reais do recurso gerenciado pelo agente (habilitar uma interface de transmissão de dados, por exemplo).

A figura 2.20 ilustra os módulos de uma entidade SNMP:

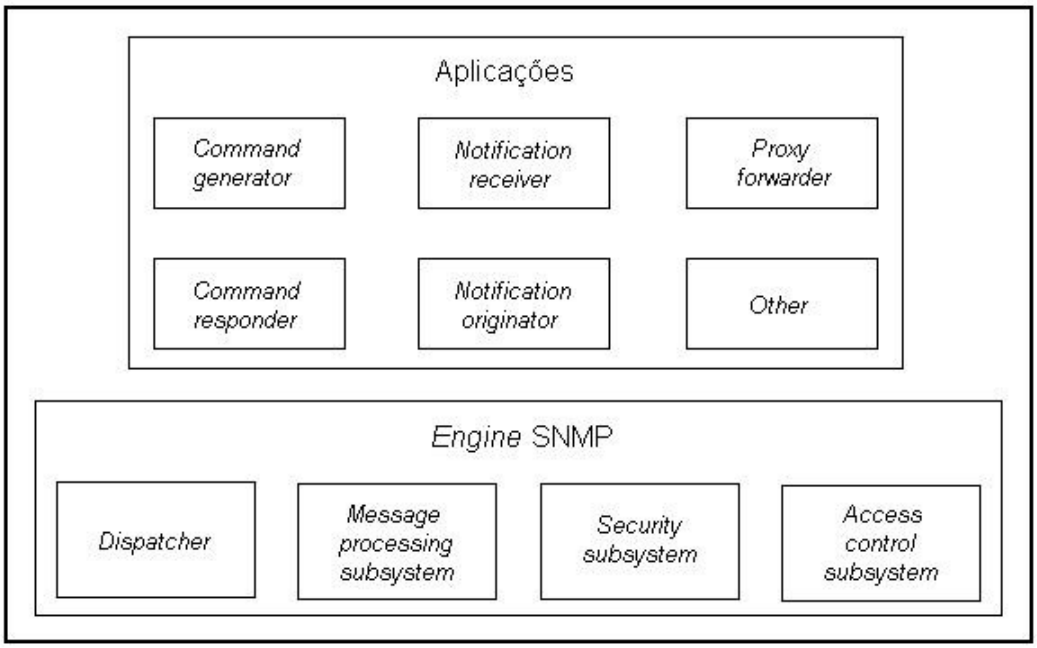


Figura 2.20: Entidade SNMP (definida na documentação SNMPv3)

Os módulos da figura 2.20 são explicados a seguir. O termo *Protocol Data Unit* (PDU) SNMP, usado a seguir, se refere ao conteúdo efetivo (operações, OIDs e valores) contido na mensagem SNMP. A mensagem SNMP em si contém também informações de endereçamento e segurança.

- *Dispatcher*: permite a suporte a múltiplas versões de SNMP em um mesmo *engine*. É responsável pela transmissão PDUs entre aplicações e rede; encaminhamento das PDUs para o *Message Processing subsystem* prepará-las para envio, ou encaminhamento de mensagens para o mesmo sub-sistema para extração de PDUs; e pelo envio recepção das mensagens SNMP na rede.

- *Message Processing subsystem*: prepara mensagens SNMP para envio, e extrai PDUs de mensagens recebidas.

- *Security subsystem*: provê segurança às mensagens, como autenticação, integridade e confidencialidade. Este sub-sistema pode conter múltiplos modelos de segurança; em SNMPv3, comumente é usado o USM.

- *Access Control subsystem*: provê mecanismos de controle de acesso que podem ser utilizados pelas aplicações. Existe apenas em agentes SNMP, não em gerentes. Pode ser invocado por aplicações que recuperam ou alteram objetos, e por aplicações que geram notificações. Em SNMPv3, comumente é usado o VACM.

- *Command Generator*: cria PDUs de requisição (`Get`, `GetNext`, `GetBulk` e/ou `Set`) e processa as respectivas respostas.

- *Command Responder*: recebe PDUs de requisição, realiza a devida operação, usando controle de acesso, e gera a PDU de resposta.

- *Notification Originator*: monitora um sistema quanto a eventos e condições relevantes, e gera mensagens `Trap` ou `Inform` para notificá-los.

- *Notification Receiver*: Escuta mensagens de notificação, gerando respostas no caso de PDUs `Inform`.

- *Proxy Forwarder*: Reencaminhar mensagens SNMP.

2.4.4 *User-Based Security Model* (USM)

O USM (BLUMENTHAL, 2002) é o modelo de segurança atualmente padronizado para o SNMPv3. A especificação do USM aborda as seguintes funcionalides:

- Autenticação: provê integridade e autenticação da origem dos dados.

- *Timeliness*: protege contra atrasos e duplicações na mensagem.

- Privacidade: protege contra alterações no conteúdo das mensagens.

- Formato: define formato de um campo denominado `msgSecurityParameters`, que contém os valores correspondentes às funções de segurança utilizadas.

- Descoberta: define procedimentos para que um *engine* SNMP obtenha informações de outro *engine*.

- Gerenciamento de chaves: define procedimentos para geração, atualização e uso de chaves de segurança.

2.4.5 *View-Based Access Control Model* (VACM)

O VACM (WIJNEN, 2002) é o modelo de controle de acesso padronizado para o SNMPv3. O VACM determina se determinada operação é permitida a um objeto gerenciado em um agente através de uma MIB. O modelo utiliza ele próprio uma MIB para definição das políticas de autorização, permitindo a realização de configuração remota.

O VACM utiliza cinco elementos para decisões de autorização: grupos, nível de segurança, contextos, MIB *views* e política de acesso.

Um grupo de segurança consiste um uma ou mais tuplas `securityModel` + `securityName`. O modelo de segurança é, por exemplo, o USM. O nome de segurança identifica um sujeito que acessa o recurso gerenciado. Através do grupo, um mesmo conjunto de políticas de acesso pode ser definido para vários gerentes. O grupo é identificado pelo `groupName`.

O nível de segurança diz respeito ao uso de privacidade e autenticação. Por exemplo, pode-se querer que a leitura de um objeto por um gerente seja permitida sem autenticação, mas que para escrita, seja necessário se autenticar. Os possíveis valores para o nível de segurança são `noAuthNoPriv`, `authNoPriv` e `authPriv`.

Um contexto é um sub-conjunto de instâncias de uma MIB. Os contextos permitem agregar objetos livremente em grupos com políticas de acesso diversas. O contexto também identifica em qual *engine* SNMP estão as instâncias que referencia.

A MIB *view* é a forma através da qual as políticas de acesso aos objetos são organizadas. Uma MIB *view* é um conjunto parcial de MIBs, composta por famílias de sub-árvores, em que cada sub-árvore pode ser incluída ou excluída da *view*. O conceito de sub-árvore, no SNMPv3, se refere a um nodo em uma MIB e todos seus elementos subordinados. Este mecanismo será explicado em detalhes na seção que descreve a implementação de controle de acesso feita em XACML nesse trabalho, pois essa implementação reproduz o mecanismo de MIB *views* da VACM para permitir uma comparação direta entre os controles de acesso via SNMP e via Web Services.

A política de acesso VACM é definida para cada operação SNMP através de uma função chamada `isAccessAllowed`. Os parâmetro para essa função são aqueles necessários para que a decisão de acesso possa ser tomada. Esses parâmetros são: `securityModel` (por exemplo, USM), `securityName` (que identifica o sujeito que está requisitando a operação), `securityLevel`, tipo de operação (leitura, escrita ou notificação, contexto e instância de objeto.

A combinação dos parâmetros citados acima permite a identificação das variáveis necessárias para a decisão: quem deseja realizar a operação; onde estão as informações desejadas; como deve ser feita a proteção; por quê a operação é requisitada (ou seja, qual o tipo de operação), o que é a informação desejada (tipo do objeto), e qual é a informação (instância de objeto).

A figura 2.21 mostra como a decisão é tomada com base nos parâmetros existentes. A figura cita as quatro tabelas que fazem parte da MIB VACM, `vacmSecurityToGroupTable`, `vacmContextTable`, `vacmAccessTable` e `vacmViewTreeFamilyTable`. Essas tabelas são detalhadas na seção 4.2, que descreve a implementação de controle de acesso via XACML realizada nesse trabalho, baseada em VACM.

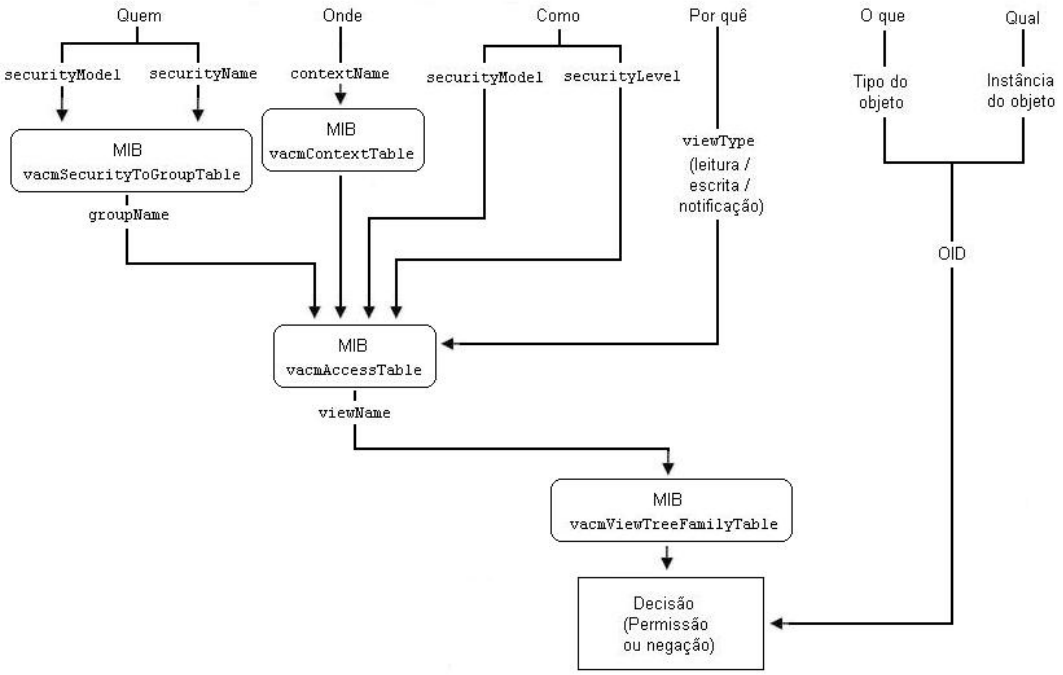


Figura 2.21: Lógica de decisão de acesso em VACM

3 INTEGRAÇÃO ENTRE WS-MANAGEMENT E WS-SECURITY

Neste capítulo, será proposta uma arquitetura para integração de segurança ao gerenciamento de redes baseado em Web Services. Essa arquitetura é baseada na integração dos padrões WS-Security, para segurança, e WS-Management, para gerenciamento.

A escolha do WS-Security foi feita devido a esse ser o padrão de fato para segurança de Web Services, além de haver previsão de um *framework* de padrões construído tendo por base o WS-Security, sendo alguns desses padrões já existentes (como o WS-Policy), e outros padrões em planejamento (como o WS-Authorization). O uso do WS-Security permitirá que se integrem funcionalidades de autenticação, criptografia e assinatura digital.

Na pilha WS-Security, o padrão previsto para controle de acesso é o WS-Authorization, que está em fase de planejamento. Portanto, para agregar controle de acesso à arquitetura, será necessário o uso de um padrão que não faz parte da pilha WS-Security. Atualmente, o padrão mais completo e utilizado para realização de controle de acesso em XML e Web Services é o XACML, que será utilizado neste trabalho. Como o controle de acesso possui uma série de particularidades, será apresentado em capítulo posterior, como um módulo adicional que pode ser integrado à arquitetura proposta neste capítulo.

O WS-Management foi escolhido, conforme apresentado em detalhes no capítulo anterior, por existir trabalho em andamento visando a convergência entre os dois principais padrões existentes (WS-Management e MUWS), sendo que esse trabalho indica que o padrão convergido será fortemente baseado em WS-Management. Assim, quando houver esse novo padrão, espera-se que a arquitetura aqui proposta possa ser facilmente migrada.

Este capítulo também apresenta a implementação realizada para a arquitetura proposta. É utilizada a API Wiseman como implementação do padrão WS-Management; e o módulo Rampart do servidor de Web Services Axis2 como implementação do padrão WS-Security. Tanto o Wiseman quanto o Axis2 + Rampart são implementados em Java (SIERRA, 2005).

3.1 Arquitetura

A arquitetura proposta nesse trabalho considera uma troca de mensagens WS-Management, entre um gerente e o agente de um recurso gerenciado, e sobre essa troca, a aplicação de funções de segurança às mensagens. E emprego de segurança se dá antes do envio das mensagens à rede, após o processamento que cria a mensagem WS-Management, e após a recepção das mensagens pela rede, antes do processamento que analisa o conteúdo relevante da mensagem WS-Management.

Na definição da arquitetura, foi preciso considerar uma limitação na coexistência entre um servidor Wiseman que processa mensagens WS-Management e um servidor Axis2 que processa mensagens WS-Security. Não é possível integrar o processamento de mensagens do servidor Wiseman ao Axis2, devido a lógicas de endereçamento distintas. O WS-Management define o endereçamento baseado em WS-Addressing, o que não é possível no Axis2. Dessa forma, foram utilizados ambos os servidores na arquitetura, os dois aparecendo no caminho completo de uma mensagem.

A figura 3.1 apresenta a arquitetura proposta:

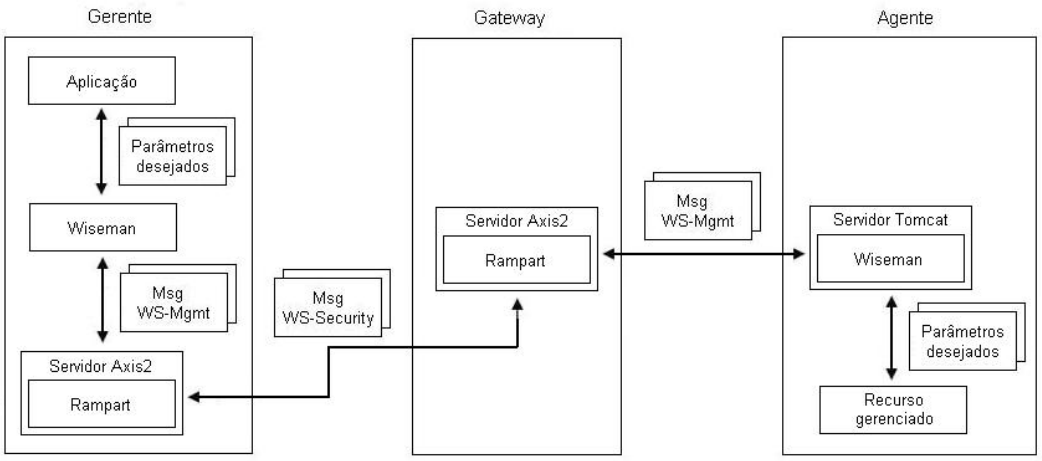


Figura 3.1: Arquitetura para integração WS-Security x WS-Management

Na figura 3.1, é utilizado um *gateway* entre o gerente e o agente, onde fica o servidor Axis2 que faz o processamento de uma mensagem WS-Security para WS-Management, e vice-versa. Isso é uma possibilidade, pelo fato de serem usados dois servidores no lado do agente. Também seria possível não usar o *gateway*, e utilizar o servidor Axis2 no próprio agente.

O caminho realizado por uma troca de mensagens completa é o seguinte, conforme pode ser acompanhado na figura 3.1: um gerente deseja recuperar determinados parâmetros de um agente. O padrão para comunicação de informações de gerência é o WS-Management. Assim, uma aplicação de gerência passa os parâmetros desejados para uma API Wiseman, que constrói uma mensagem WS-Management de requisição desses parâmetros. Em seguida, essa mensagem é passada para o servidor Axis2 no gerente, onde o módulo Rampart será executado e processará a mensagem

WS-Management de forma a introduzir as características de segurança desejada. Então, o servidor Axis2 no gerente envia a mensagem WS-Security resultante.

O *gateway* onde está o servidor Axis2 receptor da mensagem WS-Security decodifica os parâmetros de segurança executando o módulo Rampart. O módulo Rampart, após, chama uma lógica que constrói uma nova mensagem WS-Management com os parâmetros existentes, e troca as informações de endereçamento para que o envio seja para o servidor Tomcat do agente. A mensagem WS-Management, então, é enviada. O servidor Tomcat a recebe e repassa para a API Wiseman, que está rodando sobre o servidor Tomcat. Então, são verificados os parâmetros desejados, e os mesmo são recuperados no recurso gerenciado pelo agente.

A resposta da mensagem segue o caminho inverso à descrição apresentada.

3.2 Implementação

Para a implementação da arquitetura proposta, é necessário escolher um modelo de recurso gerenciável. Como a implementação usada será a mesma para posterior integração de controle de acesso via XACML, na qual será feita análise de desempenho comparando com *gateway* WS x SNMP, é natural que o modelo escolhido seja baseado em uma MIB SNMP, para facilitar a implementação do *gateway* e as comparações.

Foi escolhida a tabela `hrSWRunTable`, que faz parte da MIB *Host Resources* (WALDBUSSER, 2000b). Esta tabela, conhecida como *Running Software Group*, contém uma lista com informações acerca dos softwares executados em determinado host. Cada software representando na MIB ocupa uma instância da entrada `hrSWRunEntry`, cujo OID é .1.3.6.1.2.1.25.4.2.1. As informações disponíveis em cada entrada são exibidas na tabela 3.1.

Tabela 3.1: Tabela `hrSWRunEntry`

Índice	Nome	Descrição
1	`hrSWRunIndex`	Identificador único da entrada corresponde a determinado software
2	`hrSWRunName`	Nome do software
3	`hrSWRunID`	ProductID do software
4	`hrSWRunPath`	Caminho físico onde o software foi inicializado (ex.: um diretório)
5	`hrSWRunParameters`	Parâmetros fornecidos ao software na inicialização
6	`hrSWRunType`	Tipo do software (sistema operacional, aplicação)
7	`hrSWRunStatus`	Estado de execução do software. Pode ser utilizado para interromper a execução

Como o padrão de gerenciamento utilizado é o WS-Management, será implementada uma representação da tabela *Running Software* como recurso de gerenciamento do WS-Management. Na implementação de WS-Management, o Wiseman, um recurso de gerenciamento é modelado como uma simples classe Java. Diferentes entradas correspondentes aos softwares executados são diferentes instâncias desta classe. A classe utilizada foi denonimada `RunningSw`, e possui 7 atributos

representando as informações. Todos os atributos são do tipo String, e os nomes dos atributos são `index`, `name`, `id`, `path`, `parameters`, `type` e `status`.

Nas próximas seções deste capítulo, são mostrados os detalhes de implementação da arquitetura visualizada na figura 3.1, com base na classe `RunningSw`. É apresentado o fluxo de uma mensagem de requisição de informações, e da correspondente mensagem de resposta, consistindo nas implementações realizadas utilizando Wiseman e Axis2 + Rampart no gerente e agente da arquitetura.

3.2.1 Implementação Wiseman no gerente

Uma mensagem WS-Management no Wiseman é abstraída através da classe `Transfer`. A construção de nova instância de uma mensagem é realizada com o uso da classe `TransferMessageValues`. Uma instância dessa classe é inicializada com os parâmetros desejados, e então, se utiliza um método da classe `TransferUtility` para convertê-la em uma mensagem `Transfer`.

A construção de uma mensagem de requisição foi implementada através da classe `WsmanMessageBuilder`, reproduzida na figura 3.2:

```
public class WsmanMessageBuilder {
	private static final String DESTINATION = "http://localhost:8888/runningSw/";
	public static final String RESOURCE_URI =
"http://br/ufrgs/inf/schemas/runningSw.xsd/runningSwService/runningSwResource";

	public static final String buildGet(Selector selector) throws SOAPException,
			JAXBException, DatatypeConfigurationException {
		ObjectFactory objectFactory = new ObjectFactory();
		SelectorType selectorType = objectFactory.createSelectorType();
		Set<SelectorType> selectors = new HashSet<SelectorType>();
		TransferMessageValues tmv = TransferMessageValues.newInstance();

		selectorType.setName(selector.getName());
		selectorType.getContent().add(selector.getValue());
		selectors.add(selectorType);

		tmv.setResourceUri(RESOURCE_URI);
		tmv.setTo(DESTINATION);
		tmv.setSelectorSet(selectors);
		tmv.setTransferMessageActionType(Transfer.GET_ACTION_URI);

		Transfer mgmt = TransferUtility.buildMessage(null, tmv);
		return mgmt.toString();
	}
}
```

Figura 3.2: Código-fonte para construção de uma mensagem WS-Management

O parâmetro `Selector`, passado para o método `buildGet`, consiste em um par nome/valor. O nome e o valor definem qual propriedade do recurso WS-Management será utilizado para selecioná-lo entre o conjunto de recursos `RunningSw` existentes no agente. No caso, é utilizado o nome `index`, pois é o atributo que identifica unicamente um software executando, e o valor do índice desejado. Com o parâmetro Selector, é construído o conjunto `selectors`, utilizado pelo Wiseman para representar os seletores na mensagem WS-Management.

Os seletores são setados na classe TransferMessageValues, assim como atributos de destino e ação (Get, no caso). Em seguida, é construída a instância `mgmt`, que contém

a mensagem em si. O método `mgmt.toString()` retorna a mensagem XML de requisição WS-Management desejada. Esta mensagem XML, então, é utilizada na implementação Axis2 + Rampart, responsável por inserir os parâmetros de segurança necessários, e enviar a mensagem WS-Security resultante para o agente.

3.2.2 Implementação Axis2 + Rampart no gerente

A mensagem XML WS-Management obtida através da API Wiseman é enviada ao agente através do servidor Axis2. Antes do envio, o servidor Axis2 faz os processamentos necessários na mensagens através da chamada de módulos. O módulo Rampart é responsável por inserir parâmetros de segurança conforme o padrão WS-Security.

O Rampart permite o uso de autenticação, via inserção de campos usuário e senha criptografada na mensagem; criptografia dos dados através de XML-Encryption; e assinatura digital via XML-Digital Signature. A utilização destas funcionalidades de segurança via Axis2 + Rampart é feita de forma transparente ao uso de WS-Management, pois o suporte do código é para qualquer mensagem SOAP. É necessário apenas preencher um arquivo de configuração com os parâmetros de segurança desejados, como será detalhado em breve.

O único tratamento necessário a mensagem XML WS-Management, para utilização direta do código do Axis2 + Rampart, é a conversão dessa mensagem para o formato utilizado pela classe Java do Axis2, que é chamada para que o processamento e envio sejam realizados. É necessário converter a mensagem WS-Management para uma instância da classe `org.apache.axiom.soap.SoapEnvelope` do Axis2. Esta classe é a representação de uma mensagem SOAP na modelagem AXIOM (*Axis Object Model*) (AXIOM, 2008) usada pelo Axis2.

Para obter a mensagem no formato desejado, a partir de uma String representando a mensagem XML WS-Management, é utilizado o código exibido na figura 3.3:

```
public static final SOAPEnvelope build(String xmlMessage) {
        Reader r = new StringReader(xmlMessage);
        XMLStreamReader reader = XMLInputFactory.newInstance().createXMLStreamReader(r);
        StAXSOAPModelBuilder soapBuilder = new StAXSOAPModelBuilder(reader);

        return (SOAPEnvelopeImpl) soapBuilder.getDocumentElement();
}
```

Figura 3.3: Conversão de XML em formato String para modelagem AXIOM

A classe `StAXSOAPModelBuilder`, da API Apache AXIOM, que faz parte do pacote de distribuição do Axis2, realiza a conversão necessária de forma transparente para a aplicação. O objeto AXIOM, então, é passado para o método `sendReceive` da `classe org.apache.axis2.client.ServiceClient`. A chamada desse método desencadeia o processamento necessário, incluindo o envio da mensagem SOAP através da rede, e o bloqueio da aplicação esperando pela resposta. A resposta recebida é o retorno do método `sendReceive`. O retorno é no formato `org.apache.axiom.om.OMElement`, que representa o corpo da mensagem SOAP recebida como resposta. Através de métodos simples desta classe, como `getChildElements`, a aplicação pode processar os valores recebidos.

Como mencionado, não há nenhuma codificação extra necessária no gerente. Só é necessário configurar os parâmetros desejados em arquivo específico, normalmente denominado `client.axis2.xml`. Os trechos relativos à segurança deste arquivo de configuração podem ser visualizados na figura 3.4:

```
<parameter name="OutflowSecurity">
        <action>
                <signAllHeaders/>
                <signBody/>
                <encryptAllHeaders/>
                <encryptBody/>
                <items>UsernameToken Encrypt Signature</items>
                <user>client</user>
                <passwordCallbackClass>handler.PWCBHandler</passwordCallbackClass>
                <signaturePropFile>client.properties</signaturePropFile>
                <signatureKeyIdentifier>Thumbprint</signatureKeyIdentifier>
                <encryptionSymAlgorithm>http://www.w3.org/2001/04/xmlenc#tripledes-
cbc</encryptionSymAlgorithm>
                <encryptionKeyIdentifier>DirectReference</encryptionKeyIdentifier>
                <encryptionUser>service</encryptionUser>

                <!-- Remover linha abaixo para signature -->
                <!-- <encryptionPropFile>client.properties</encryptionPropFile> -->
        </action>
</parameter>

<parameter name="InflowSecurity">
        <action>
                <items>UsernameToken Encrypt Signature</items>
                <passwordCallbackClass>handler.PWCBHandler</passwordCallbackClass>
                <signaturePropFile>client.properties</signaturePropFile>

                <!-- Remover linha abaixo para signature -->
                <!-- <decryptionPropFile>client.properties</decryptionPropFile> -->
        </action>
</parameter>
```

Figura 3.4: Arquivo de configuração com parâmetros de segurança para o Rampart

As configurações são distintas para entrada e saída de mensagens, conforme os parâmetros `OutflowSecurity` e `InflowSecurity`. A tag `items` especifica quais facilidades de segurança devem ser utilizadas. Isso permite que as fases de autenticação, criptografia e assinatura digital sejam habilitas e desabilitadas facilmente, o que tornou bastante prática a avaliação de desempenho, apresentada posteriormente, em que foram feitas medidas com o uso individual e combinado dessas três fases. Os demais parâmetros são usadas para indicar quais partes das mensagens devem ser criptografas e assinadas digitalmente; como a senha deve ser obtida na saída e verifica na entrada de mensagens; e quais algoritmos de criptografia e assinatura digital devem ser utilizados, bem como os parâmetros necessários para esses algoritmos.

Após o processamento pelo Rampart, o Axis2 envia a mensagem WS-Security para o agente adequado, conforme parâmetros de transportes setados na classe `ServiceClient` antes da chamada do método `sendReceive`. Ao receber a resposta, os parâmetros de segurança inseridos pelo servidor Axis2 do agente são decodificados, e o corpo da mensagem SOAP WS-Management resultante é setado como retorno do método `sendReceive`.

3.2.3 Implementação Axis2 + Rampart no agente

A mensagem WS-Security é recebida pelo servidor Axis2 existente no agente WS-Management (ou, conforme a figura 3.1, em um *gateway*). Os parâmetros de segurança são decodificados de acordo com os valores do parâmetro `InflowSecurity` do arquivo de configuração, comumente denominado `services.xml`. Este arquivo é idêntico ao arquivo utilizado no gerente, e apresentado na figura 3.4.

Após as fases de decodificação dos parâmetros de segurança (decriptação, checagem de validade da assinatura digital e do usuário/enviados), se tem a mensagem WS-Management original como resultado. A mensagem WS-Management, então, deve ser processada pelo servidor Wiseman, para obtenção e envio da resposta.

Conforme a arquitetura da solução, o servidor Wiseman não está integrado ao servidor Axis2. Dessa forma, a mensagem WS-Management precisa ser enviada ao servidor Wiseman, e a resposta WS-Management que será retornada, fornecida como resposta para o servidor Axis2 inserir parâmetros de segurança e enviar de volta para o gerente. Essas funções foram implementadas em uma classe específica para ser chamada pelo servidor Axis2 no agente. Esta classe foi denominada `WsmanMessageReceiver`, e implementa a interface `MessageReceiver` do Axis2. Na configuração do servidor Axis2, a classe `WsmanMessageReceiver` é setada como responsável pelo processamento das mensagens.

Na classe `WsmanMessageReceiver`, os objetos AXIOM repassados pelo Axis2 são transformados em uma mensagem WS-Management, com uso das classes `TransferUtility` e `TransferMessageValues` do Wiseman. Esta mensagem, então, é enviada ao servidor Wiseman. A mensagem WS-Management de resposta retornada pelo servidor Wiseman é então convertida para uma estrutura AXIOM `SoapEnvelope`, que então é processada para envio ao gerente pelo Axis2 + Rampart.

3.2.4 Implementação Wiseman no agente

A implementação Wiseman no agente é responsável por obter as informações dos recursos gerenciados requisitadas pelo gerente (softwares executando no agente). Para isso, são usadas uma série de facilidades do pacote de distribuição do Wiseman, que dão apoio ao desenvolvimento de agentes WS-Management.

Com base em um XML Schema representando o modelo do recurso gerenciado, que no caso representa um software executando, targets Ant (APACHE, 2009d) do Wiseman geram esqueletos de código para a integração ao servidor Wiseman. Esses esqueletos de código serão chamados na recepção das mensagens, e devem ser preenchidos de forma a obter e persistir os valores requisitados ou setados pelo gerente, seja em arquivo, bases de dados, implementações proprietárias, ou mesmo MIBs SNMP. Entre esses esqueletos de código, a própria classe `RunningSw` é gerada.

O XML Schema (arquivo `runningSw.xsd`) utilizado é apresentado na figura 3.5:

```
<xs:schema targetNamespace="http://schemas.inf.ufrgs.br/runningSw.xsd"
               elementFormDefault="qualified"
               blockDefault="#all"
               xmlns:sw="http://schemas.inf.ufrgs.br/runningSw.xsd"
               xmlns:xs="http://www.w3.org/2001/XMLSchema">
```

```
        <xs:complexType name="RunningSw">
                <xs:sequence>
                        <xs:element name="index" type="xs:string"/>
                        <xs:element name="name" type="xs:string"/>
                        <xs:element name="id" type="xs:string"/>
                        <xs:element name="path" type="xs:string"/>
                        <xs:element name="parameters" type="xs:string"/>
                        <xs:element name="type" type="xs:string"/>
                        <xs:element name="status" type="xs:string"/>
                </xs:sequence>
        </xs:complexType>

        <xs:element name="runningSw" type="sw:RunningSw"/>

</xs:schema>
```

Figura 3.5: XML Schema para recurso `RunningSw`

O código gerado pelas targets Ant do Wiseman incluem classes JAXB para a conversão entre representações em XML e em Java do recurso gerenciado. O principal código gerado é a classe `RunningSwresourceHandler`, que possui métodos que serão executados para as operações efetivas sobre os recursos gerenciados pelo agente, como Get e Put. É nesses métodos que as implementações proprietárias de carga e persistência dos valores dos/nos recursos gerenciados é realizada.

4 CONTROLE DE ACESSO VIA XACML UTILIZANDO MODELO VACM

Neste capítulo, será apresentada a integração de controle de acesso à arquitetura de segurança proposta no capítulo anterior. O protocolo utilizado para controle de acesso é o XACML, devido ao fato de o padrão WS-Authorization, previsto para o *framework* WS-Security, estar em planejamento. A característica modular da arquitetura é preservada, permitindo quaisquer combinações de uso entre controle de acesso, autenticação, criptografia e assinatura digital.

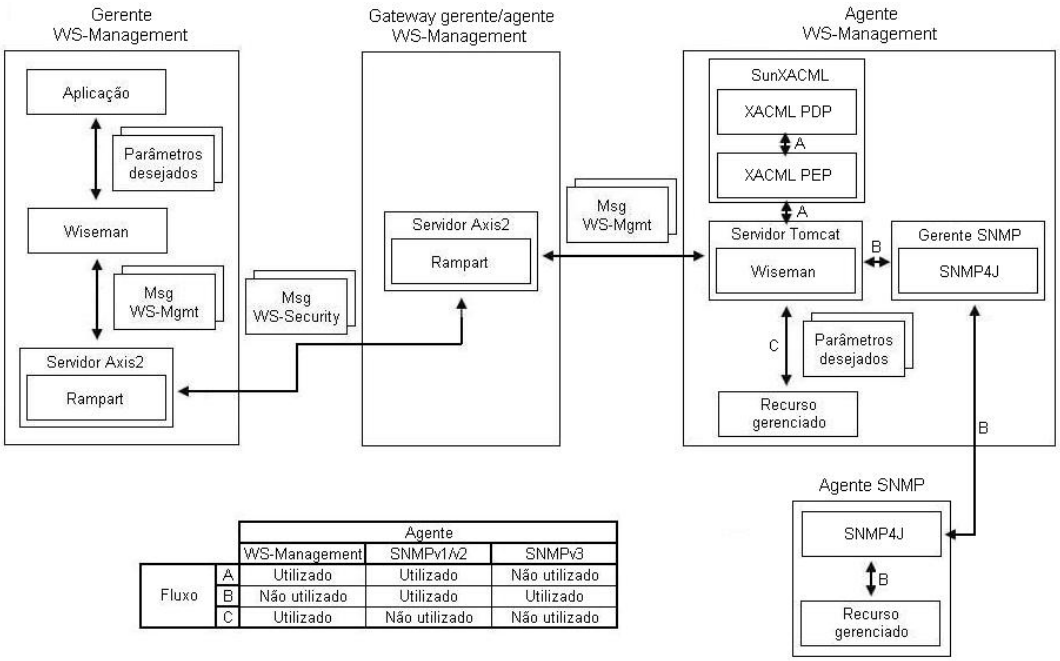
A definição da arquitetura de controle de acesso passa pela definição de uma modelagem para as permissões de acesso às informações dos recursos gerenciados. Já existe uma modelagem tradicional para controle de acesso em gerenciamento de redes, a MIB VACM do SNMPv3. Assim, esse foi o modelo utilizado na implementação do controle de acesso via XACML. Isso facilita que a arquitetura seja planejada para gerenciar redes que contêm tanto agentes WS-Management quanto agentes SNMP, sendo necessária para isso a presença de um *gateway* WS-Management x SNMP.

Além de apresentar a arquitetura completa com integração de controle de acesso e suporte a agentes SNMPv1/v2 e SNMPv3, este capítulo dá maiores detalhes sobre a modelagem VACM, para então explicar como essa modelagem foi implementada com uso do XACML. Também é apresentada a implementação do *gateway* WS-Management x SNMP. O uso do *gateway* permitiu uma comparação de desempenho entre o uso de XACML e da MIB VACM. Esta comparação é apresentada no capítulo que aborda Avaliação de Desempenho.

4.1 Arquitetura

A arquitetura apresentada neste capítulo se sobrepõe à arquitetura WS-Management x WS-Security já detalhada. Do ponto de vista de agentes WS-Management, simplesmente são integrados um PEP e um PDP XACML, dando suporte ao controle de acesso. Porém, a arquitetura também prevê o controle de acesso para agentes SNMPv1/v2 e SNMPv3 através de *gateways* WS-Management x SNMP. Nos agentes SNMPv3, que tem controle de acesso próprio, o *gateway* simplesmente repassa às requisições Get e Set, deixando que o controle de acesso seja realizado no agente. Nos agentes SNMPv1/v2, o controle de acesso não existe, mas a solução faz com que haja esse suporte, pois o controle é realizado pelo próprio *gateway*, que encaminha a requisição apenas caso a mesma seja autorizada pelas políticas XACML/VACM definidas.

A figura 4.1 apresenta a arquitetura de segurança completa. Esta figura referencia a API SNMP em Java utilizada neste trabalho, o SNMP4J (*SNMP for Java*) (SNMP4J, 2008):



		Agente		
		WS-Management	SNMPv1/v2	SNMPv3
Fluxo	A	Utilizado	Utilizado	Não utilizado
	B	Não utilizado	Utilizado	Utilizado
	C	Utilizado	Não utilizado	Não utilizado

Figura 4.1: Arquitetura WS-Security x WS-Management com integração de controle de acesso

A figura 4.1 acrescenta, em relação à arquitetura WS-Security x WS-Management do capítulo anterior, apresentada na figura 3.1, uma implementação SunXACML no agente WS-Management, com um PEP XACML responsável pelo encaminhamento de requisições e aplicação de decisões, e um PDP responsável por determinar a permissão ou negação dos acessos. O PEP XACML é chamado pelo servidor Wiseman do agente, em fluxo identificado como Fluxo A. Este mesmo fluxo engloba a chamada do PDP pelo PEP. O PDP utiliza políticas XACML baseadas no modelo VACM para tomar a decisão de acesso. O Fluxo A é utilizado quando o agente é WS-Management ou SNMPv1/v2. Quando o agente é SNMPv3, o Fluxo A não é utilizado, pois o controle de acesso é realizada pela MIB VACM do agente.

Quando o agente gerenciado não é WS-Management, aparecem também o gerente SNMP e o agente SNMP. O Fluxo B indica a participação destes componentes na arquitetura. Quando o agente é SNMP, o servidor Wiseman do agente atua como *gateway*, convertendo a requisição WS-Management em requisição SNMP. Então, a requisição se dá conforme uma comunicação SNMP normal entre o gerente SNMP, presente no agente WS-Management, e o agente SNMP externo. Quando o agente é WS-Management, o Fluxo C, presente na arquitetura original, é utilizado, ao invés do B, com o agente WS-Management sendo responsável por recuperar/configurar as informações no recurso gerenciado.

A tabela Fluxo x Agente da figura 4.1 resume, para cada agente que pode participar da arquitetura, quais fluxos são e não são utilizados.

4.2 Implementação

A implementação da arquitetura de controle de acesso foi realizada com base na modelagem VACM do SNMPv3. Basicamente, as políticas XACML utilizadas para controle de acesso possuem a mesma estrutura lógica das MIBs VACM. As funções usadas no PDP XACML, que determinam qual política de acesso deve ser usada para determinada requisição, e se essa requisição é permitida, também são semelhantes à VACM. Para permitir a compreensão do trabalho realizado, as MIBs VACM são explicadas em maiores detalhes nesta seção. Em seguida, é apresentado como a modelagem VACM foi implementada através de políticas XACML. Por fim, é descrita a implementação do *gateway* WS-Management x SNMP, que permite a gerência simultânea de agentes SNMP e WS-Management.

4.2.1 Modelo VACM

O modelo VACM é abstraído em uma MIB, denominada `snmpVacmMIB`. A MIB VACM possui 4 tabelas, conforme mencionado anteriormente na figura 2.21. Estas tabelas são: `vacmContextTable`, `vacmSecurityToGroupTable`, `vacmAccessTable` e `vacmViewTreeFamilyTable`.

A tabela `vacmContextTable` lista os nomes dos contextos existentes no agente. Esta tabela serve apenas para leitura e não pode ser configurada via SNMP. O conjunto de contextos existentes é controlado internamente pela entidade SNMP. Um contexto com nome vazio (“”) representa o contexto default.

A tabela `vacmSecurityToGroupTable` armazena nomes de grupos. Os índices desta tabela são `vacmSecurityModel` e `vacmSecurityName`, sendo que cada par destes tem um nome de grupo associado. Este nome de grupo é usado nas políticas de acesso, e permite, portanto, definir uma política única para um conjunto de usuários sob determinado modelo de segurança (sendo que o modelo de segurança default do SNMPv3 é o USM).

A tabela `vacmAccessTable` é a tabela em que as políticas de acesso são configuradas. A definição de cada política é armazenada por um conjunto de propriedades: contexto, grupo, modelo de segurança, e nível de segurança. Para cada conjunto com essas propriedades, são definidas MIB *views* diversas para operações de leitura, escrita e notificação.

A tabela `vacmViewTreeFamilyTable` lista as sub-árvores de objetos que compõem cada MIB *view*. A lógica para representar o conjunto de objetos de uma MIB *view* é detalhada posteriormente nesta seção.

A tabela 4.1 lista os objetos presentes em cada uma dessas tabelas. As entradas que possuem sufixos `StorageType` e `Status` são, respectivamente, dos tipos SMI *StorageType* e *RowStatus*, cujo funcionamento é detalhado em (McCLOGHRIE, 1999).

Tabela 4.1: Tabelas e objetos da MIB VACM

vacmContextTable	
vacmContextName (índice)	Nome do contexto
vacmSecurityToGroupTable	
vacmSecurityModel (índice)	Modelo de segurança (no SNMPv3, o default é o USM)
vacmSecurityName (índice)	Nome de usuário
vacmGroupName	Nome do grupo
vacmSecurityToGroupStorageType	Indica se a entrada é volátil ou permanente
vacmSecurityToGroupStatus	Permite a criação e remoção de entradas
vacmAccessTable	
vacmAccessContextPrefix (índice)	Nome ou prefixo do contexto
vacmAccessSecurityModel (índice)	Modelo de segurança
vacmAccessSecurityLevel (índice)	Nível de segurança (noAuthNoPriv, authNoPriv ou authPriv)
vacmAccessContextMatch (índice)	Indica se vacmAccessContextPrefix deve ser considerado como nome ou prefixo do contexto
vacmAccessReadViewName	Nome da MIB *view* para operações de lectura
vacmAccessWriteViewName	Nome da MIB *view* para operações de escrita
vacmAccessNotifyViewName	Nome da MIB *view* para operações de notificação
vacmAccessStorageType	Indica se a entrada é volátil ou permanente
vacmAccessStatus	Permite a criação e remoção de entradas
vacmViewTreeFamilyTable	
vacmViewTreeFamilyViewName (índice)	Nome da MIB *view*
vacmViewTreeFamilySubtree (índice)	Sub-árvore que, combinada com vacmViewTreeFamilyMask, define uma familia de sub-árvores
vacmViewTreeFamilyMask	Máscara de bits que, combinada com vacmViewTreeFamilySubtree, define uma familia de sub-árvores
vacmViewTreeFamilyType	Indica se essa sub-árvore deve ser incluída ou excluída da MIB *view*
vacmViewTreeFamilyStatus	Permite a criação e remoção de entradas

A tabela vacmViewTreeFamilyTable é indexada pelos objetos vacmViewTreeFamilyViewName e vacmViewTreeFamilySubtree. Podem haver múltiplas entradas associadas a uma MIB *view*, identificada pelo viewName. Nesse caso, a MIB *view* é constituída pela união de todas as sub-árvores. O conceito de sub-árvore utilizado na formação das MIB *views* corresponde a um objeto de uma MIB, e todos os objetos subordinados na hierarquia da MIB. Este conjunto de objetos tem em um OID em comum como prefixo.

A máscara permite definir que alguns identificadores do OID de prefixo podem variar. Por exemplo, uma sub-árvore com OID 1.3.6, e máscara 1.0, definine que para esta sub-árvore, o 2º identificador pode variar. Assim qualquer OID que inicie por 1.X.6, sendo X qualquer valor, faz parte da sub-árvore. O valor default da máscara, que é vazio, corresponde a uma máscara com todos os bits em 1, ou seja, todos os objetos

tendo exatamente o OID da sub-árvore como prefixo. Caso se deseje que algum OID varie, deve-se usar uma máscara com 0 em algum identificador. Quando a máscara tem um número de identificadores menor que o do OID correspondente, assume-se que os valores faltantes são 1 (o que é condizente com o valor default para máscara). Assim, no exemplo apresentado, a máscara 1.0 é tratada como 1.0.1. Se fosse desejado que o 3º identificador também fosse variável, a máscara precisaria valer 1.0.0.

O parâmetro `type` permite indicar que a entrada deve ser incluída ou excluída. Portanto, a combinação de diversas entradas, através dos campos sub-árvore, máscara e tipo, permite definir uma coleção arbitrária de objetos, com uma quantidade mínima de configurações. O conjunto de entradas que formam uma MIB *view* também é denominada família de sub-árvores.

4.2.2 Implementação do modelo VACM em XACML

Neste trabalho, a modelagem de controle de acesso utilizada foi a VACM. Esta modelagem foi implementada em XML, para possibilitar o controle de acesso no contexto de gerenciamento via Web Services, no qual o padrão utilizado foi o WS-Management.

Na implementação de exemplo de recurso gerenciado para a arquitetura de segurança, foi utilizada a tabela `hrSWRunEntry` da MIB *Host Resources*. Esta tabela foi abstraída como um recurso WS-Management denominado `RunningSw`. De forma análoga, as tabelas VACM foram modeladas como recursos WS-Management. Na implementação realizada, nem todas as tabelas VACM foram implementadas; focou-se nas tabelas `vacmAccessTable` e `vacmViewTreeFamilyTable`, que contêm as informações mais relevantes da MIB VACM para controle de acesso: políticas de acesso para cada grupo de usuário, em função de conjuntos de objetos, na `vacmAccessTable`, e conteúdo dos conjuntos de objetos, na `vacmViewTreeFamilyTable`.

Os recursos WS-Management representando estas duas tabelas foram denominados `AccessEntry` e `TreeFamilyView`. Assim como no caso do recurso `RunningSw`, estes recursos são compostos de propriedades correspondentes aos objetos das tabelas VACM, sendo todos do tipo String. As propriedades do recurso `accessEntry` são `ipAddress`, `groupName`, `contextPrefix`, `securityModel`, `securityLevel`, `contextMatch`, `readViewName`, `writeViewName` e `notifyViewName`. As propriedades do recurso `treeFamilyView` são `ipAddress`, `viewName`, `subtree`, `mask` e `type`.

Entre as propriedades dos recursos WS-Management que representam as políticas de acesso, nota-se uma propriedade que não faz parte da MIB VACM: a propriedade `ipAddress`. Esta propriedade foi adicionada pois foi considerado, na implementação, que um determinado agente WS-Management pode agir como *gateway* de diversos agentes SNMP. Através do campo `ipAddress`, é possível modelar as políticas de acesso desses diversos agentes SNMP como recursos de um mesmo agente WS-Management.

Algumas propriedades da MIB VACM, por sua vez, não estão representadas na modelagem dos recursos WS-Management. As propriedades removidas são as do tipo

RowStatus, utilizado para criação e remoção de entradas, pelo fato dessas operações serem realizadas diretamente através das operações *Create* e *Delete* do WS-Management; e a propriedade do tipo *StorageType*, que foi desconsiderada por não ser relevante para o estudo realizado.

A implementação em Java suportando as operações WS-Management sobre os recursos `AccessEntry` e `TreeFamilyView` no agente WS-Management foi obtida da forma já apresentada para o recurso `RunningSw`, através de código gerado por ferramentas do Wiseman a partir de arquivos XML Schema *accessEntry.xsd* e *treeFamilyView.xsd* representando os recursos desejados.

As políticas de acesso representadas via recursos WS-Management são armazenadas no agente em arquivos XML. Exemplos de arquivos XML para `AccessEntry` e `TreeFamilyView` são apresentados, respectivamente, nas figuras 4.2 e 4.3:

```
<accessEntries xmlns:ns2="http://schemas.inf.ufrgs.br/accessEntry.xsd">
    <entry>
        <ns2:ipAddress>190.88.8.80</ns2:ipAddress>
        <ns2:groupName>initial</ns2:groupName>
        <ns2:contextPrefix>pref</ns2:contextPrefix>
        <ns2:securityModel>USM</ns2:securityModel>
        <ns2:securityLevel>authNoPriv</ns2:securityLevel>
        <ns2:contextMatch>exact</ns2:contextMatch>
        <ns2:readViewName>toreadview</ns2:readViewName>
        <ns2:writeViewName>towriteview</ns2:writeViewName>
        <ns2:notifyViewName>tonotifyview</ns2:notifyViewName>
    </entry>
</accessEntries>
```

Figura 4.2: XML representando recurso `AccessEntry`

```
<treeFamilyViews xmlns:ns2="http://schemas.inf.ufrgs.br/treeFamilyView.xsd">
    <view>
        <ns2:ipAddress>190.88.8.80</ns2:ipAddress>
        <ns2:viewName>towrite</ns2:viewName>
        <ns2:subtree>80.23.34.4.5</ns2:subtree>
        <ns2:mask>1.1.1.1.1.1.0.1.1.0</ns2:mask>
        <ns2:type>included</ns2:type>
    </view>
    <view>
        <ns2:ipAddress>190.88.8.81</ns2:ipAddress>
        <ns2:viewName>running-sw-read-view</ns2:viewName>
        <ns2:subtree>1.3.6.1.2.1.25.4.2.1</ns2:subtree>
        <ns2:mask>1</ns2:mask>
        <ns2:type>included</ns2:type>
    </view>
</treeFamilyViews>
```

Figura 4.3: XML representando recursos `TreeFamilyView`

Quando as operações WS-Management *Create*, *Get*, *Put* e *Delete* são executadas, os XML que armazenam os recursos WS-Management são alterados. Estes recursos são uma representação direta da modelagem VACM no formato suportado pelo WS-Management, e são utilizados para armazenar essas informações tanto para agentes WS-Management quanto SNMPv1/v2 e SNMPv3. Porém, para aplicação das políticas de acesso em agentes WS-Management e SNMPv1/v2, é necessário que sejam definidas políticas XACML correspondentes aos recursos que representam o modelo VACM.

As políticas de acesso XACML são armazenadas no agente, e são função dos recursos `AccessEntry` e `TreeFamilyView` existentes. Assim, sempre que uma operação sobre os recursos do agente é executada, as políticas XACML são alteradas em paralelo, mantendo-as assim atualizadas.

As políticas XACML são armazenadas de forma que, cada para cada `AccessEntry`, existam 3 arquivos XACML representando as políticas existentes para as MIB *views* daquela entrada, representadas em recursos `TreeFamilyView`. Há a necessidade de 3 arquivos pois para cada operação realizada sobre os recursos gerenciados, deve existir apenas uma política de acesso válida. Como são três operações no contexto da VACM, *Read*, *Write* e *Notify*, são necessárias 3 políticas XACML, uma para cada operação, que no XACML, são tratadas como `Actions`.

Nas políticas XACML, a determinação do agente, identificado pelo endereço IP, é realizada através do nome do arquivo, que contém essa informação. O nome do grupo associado a política, por sua vez, é representado como um `Subject` do XACML. O conjunto de objetos sobre o qual uma política se aplica é determinado através de `Rules` XACML. Para cada entrada de `TreeFamilyView` existente para o conjunto IP do agente + grupo + operação, é adicionada uma `Rule` no arquivo XML que representa aquela política.

A figura 4.4 exibe um arquivo representado uma política XACML. Esta política corresponde à segunda *view* representada no XML da figura 4.3, para o grupo `initial`.

```
<Policy PolicyId="initial-READ" RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-
combining-algorithm:deny-overrides">
  <Target>
    <Subjects>
      <Subject>
        <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
          <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">initial</AttributeValue>
          <SubjectAttributeDesignator AttributeId="group-name"
DataType="http://www.w3.org/2001/XMLSchema#string"/>
        </SubjectMatch>
      </Subject>
    </Subjects>
    <Resources>
      <AnyResource/>
    </Resources>
    <Actions>
      <Action>
        <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
          <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">READ</AttributeValue>
          <ActionAttributeDesignator
AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
DataType="http://www.w3.org/2001/XMLSchema#string"/>
        </ActionMatch>
      </Action>
    </Actions>
  </Target>
  <Rule RuleId="1.3.6.1.2.1.25.4.2.1" Effect="Permit">
    <Target>
      <Subjects>
        <AnySubject/>
      </Subjects>
      <Resources>
        <AnyResource/>
      </Resources>
      <Actions>
        <AnyAction/>
      </Actions>
```

```
    </Target>
    <Condition FunctionId="oid-in-subtree-family">
      <Apply FunctionId="oidAttribute-one-and-only">
        <ResourceAttributeDesignator
AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id" DataType="oidType"/>
      </Apply>
      <AttributeValue DataType="oidType">1.3.6.1.2.1.25.4.2.1</AttributeValue>
      <AttributeValue DataType="treeMaskType">1</AttributeValue>
    </Condition>
  </Rule>
</Policy>
```

Figura 4.4: Exemplo de política XACML para o modelo VACM

Pode-se notar que o nome utilizado para a regra foi o nome do OID correspondente à propriedade `subtree`. Esta determinação é coerente, pois a política XACML representa o conjunto de entradas de uma MIB *view*, e para uma mesma MIB *view*, cada entrada precisa ter um OID identificando uma `subtree` diferente.

Na figura 4.4, também nota-se que a `Rule` XACML é composta por uma condição que se baseia na função `oid-in-subtree-family`. Esta função implementa a lógica VACM que determina se um certo OID faz parte de uma sub-árvore, identificada por um OID e uma máscara (para os quais, respectivamente, foram implementados os tipos XACML `oidType` e `treeMaskType`). A propriedade `type` da entrada `TreeFamilyView`, que indica se a sub-árvore deve ser incluída ou excluída na MIB *view*, é representada através do `Effect` associado à regra, que pode valer `Permit` ou `Deny`.

A figura 4.5 apresenta o código Java utilizado para implementar a função `oid-in-subtree-family`.

```
public class OidAttribute extends AttributeValue {
       . . .
       private int[] oid;
       . . .
       public boolean isInSubtreeFamily(OidAttribute oidAttribute, TreeMaskAttribute
maskAttribute) {
              if (this.size() >= oidAttribute.size()) {
                     int[] sizedMask =
maskAttribute.getSizedOid(oidAttribute.getOid().length);

                     for (int i = 0; i < oidAttribute.getOid().length; i++) {
                            if ((sizedMask[i] == 1) && (oid[i] !=
oidAttribute.getOid()[i])) {
                                   return false;
                            }
                     }
                     return true;
              }
              return false;
       }
       . . .
}
```

Figura 4.5: Código implementando a função XACML `oid-in-subtree-family`

As classes `OidAttribute` e `TreeMaskAttribute` extendem a classe `AttributeValue` do SunXACML. Para implementar novos tipos de funções XACML no SunXACML, é preciso extender esta classe. No código das próprias classes, é definido que, em políticas XACML, os tipos referentes a estas classe são representados por `OidType` e `TreeMaskType`, conforme aparecem na figura 4.5. De forma similar,

foi implementada a classe `IsInSubtreeFamilyFunction`, que extende a classe `FunctionBase` do SunXACML. O método `evaluate` desta classe é chamado pelo SunXACML quando o mesmo avalia a política da figura 4.4, com os parâmetros correspondentes à requisição XACML sendo avaliada. É a partir do método evaluate que é chamado o método `isInSubtreeFamily`, que aparece na figura 4.5.

O método `isInSubtreeFamily` é chamado para o OID existente na requisição avaliada, e os parâmetros do método são o OID e a máscara da política contra a qual o PDP avaliará a requisição. Caso o OID requisitado tenha menos identificadores que o OID da política, o método imediatamente retorna falso. Senão, a máscara é ajustada para o mesmo tamanho do OID da política (através do método `getSizedMask`, que adicionada quantos 1's forem necessários), e então, se percorre o OID da política, e caso se encontre um identificador diferente do OID requisitado, sendo que nesse indicador a máscara possui o valor 1, é retornado falso. Caso nenhum identificador se encontre nestas condições, o OID requisitado está na sub-árvore indicada e o método retorna verdadeiro. Nesse caso, o efeito `Permit` ou `Deny` da regra é aplicado, o que é consistente com a inclusão ou exclusão da sub-árvore da MIB *view*. Caso o método retorne falso, o resultado XACML é `NotApplicable`, caso em que a operação não é permitida.

Para que a avaliação de uma requisição XACML seja realizada, os parâmetros necessários (endereço IP do agente, OID, ação e nome de grupo) são obtidos da requisição e passados para o método `VacmRequest.permit`, listado na figura 4.6:

```
public static final ResponseCtx getResponse(String ipAddress, String groupName, String
                oid, String action) {
        SimplePDP pdp = new SimplePDP(new String[0], ipAddress);

        return pdp.evaluate(groupName, oid, action);
}

public static final boolean permit(String ipAddress, String groupName, String oid,
                String action) {
        ResponseCtx response = getResponse(groupName, oid, action);

        if (response.getResults() != null) {
                for (Object r : response.getResults()) {
                        Result result = (Result) r;

                        if (result.getDecision() == 0) {
                                return true;
                        }
                }
        }
        return false;
}
```

Figura 4.6: Chamada de PDP para avaliação de requisição XACML

A inicialização da instância da classe `SimplePDP` constrói um PDP com as políticas existentes para o IP indicado. Então, o método `evaluate` do PDP é chamado. Este método constrói uma representação em Java de uma requisição XACML, com os parâmetros indicados, através de métodos auxiliares do SunXACML, e avalia essa requisição em relação às políticas do PDP. O resultado é armazenado em um objeto `ResponseCtx`, que contém um objeto `Result` com a decisão efetiva de autorização para a requisição.

4.2.3 Implementação do *gateway* WS-Management x SNMP

A arquitetura de controle de acesso projetada prevê a gerência de uma rede heterogênea, em que há a presença de agentes WS-Management, SNMPv1/v2 e SNMPv3. Quando o agente é SNMP, é necessário que um *gateway* realize a tradução entre mensagens WS-Management e mensagens SNMP. No caso de agentes SNMPv1/v2, o próprio *gateway* é responsável pelo controle de acesso, realizando-o via XACML, já que não há tal controle no SNMPv1/v2. Já nos agentes SNMPv3, o controle de acesso é realizado pelo próprio agente SNMP.

O *gateway* foi implementado para a tabela `hrSWRunEntry` da MIB *Host Resources*, correspondendo ao recurso `RunningSw` do WS-Management, que foi a tabela SNMP de referência utilizada ao longo deste trabalho. Assim, foi realizado um mapeamento direto entre os parâmetros do recurso `RunningSw` e os OIDs da tabela `hrSWRunEntry`, na realização de mensagens *Get* e *Set*. Quando uma mensagem WS-Management *Get* é enviada para o agente, os OIDs correspondentes ao índice de `RunningSw` desejado são obtidos, e uma mensagem *Get* SNMP para esses OIDs é construída e enviada para o agente SNMP em questão. O procedimento é similar para mensagens *Set*.

A API SNMP4J foi utilizada para construção e envio das mensagens SNMP, bem como para o processamento das mensagens de resposta do agente.

5 AVALIAÇÃO DE DESEMPENHO

Neste capítulo, é apresentada a avaliação de desempenho, realizada com o intuito de verificar o desempenho das implementações realizadas. A avaliação de desempenho apresentada foi realizada através de conjuntos de testes isolados, um conjunto para a integração entre WS-Security e WS-Management, e outro para a integração de XACML à arquitetura anterior, sendo que neste segundo caso, também são realizadas comparações entre o uso de agentes WS-Management e agentes SNMP.

A primeira implementação avaliada, WS-Security x WS-Management, é um sub-conjunto da segunda (emprego de XACML). Deste modo, a forma de uso de computadores para avaliação e a configuração dos mesmos é apresentado em conjunto para os dois conjuntos de testes.

A figura 4.1, que representa a arquitetura completa, que é composta de 4 entidades: gerente WS-Management, *gateway* gerente/agente WS-Management, agente WS-Management e agente SNMP. As entidades intermediárias, o *gateway* gerente/agente WS-Management e o agente WS-Management, contêm servidores Axis2 + Rampart e Wiseman, respectivamente. Como mencionado na apresentação da arquitetura, essas duas entidades podem ser executadas no mesmo sistema, que conteria os dois servidores. Esta foi a forma utilizada nesta avaliação de desempenho. Assim, na apresentação dos computadores utilizados, serão apresentadas 3 entidades, cada uma executando em um computador diferente: gerente WS-Management, agente WS-Management (contendo os dois servidores necessários), e agente SNMP. O gerente e agente WS-Management são utilizados nos dois conjuntos de testes apresentados. O agente SNMP, por sua vez, é utilizado apenas na 2ª avaliação, em que é comparado o desempenho de um agente WS-Management com agentes SNMPv2 e SNMPv3.

As configurações de *hardware* e *software* utilizadas são apresentadas na tabela 5.1. Em cada item, são citados qual a configuração utilizada para cada uma das 3 entidades executadas; onde só existe um valor, significa que a configuração é a mesma nos 3 computadores utilizados. Em alguns casos, o item não se aplica a todos os 3 computadores. As implementações WS-Management e WS-Security só existem no gerente e agente WS-Management; a implementação XACML, apenas no agente WS-Management; e a implementação SNMP, apenas no agente SNMP e no agente WS-Management (a partir do qual são trocadas mensagens SNMP com o agente SNMP).

Tabela 5.1: Configuração dos computadores

CPU	Pentium 4 2.4GHz (gerente WS-Management e agente SNMP) Intel Core 2 Duo 2GHz (agente WS-Management)
Memória RAM	2 GB
Sistema Operacional	Windows Server 2003 (gerente WS-Management e agente SNMP) Windows XP SP2 (agente WS-Management)
JVM	Sun JDK 1.6.0_03
Implementação WS-Security	Axis 1.2 com módulo Rampart 1.2
Implementação WS-Management	Wiseman 1.0
Implementação XACML	SunXACML 1.2
Implementação SNMP	SNMP4J 1.9.1f

5.1 Avaliação de desempenho WS-Security

A primeira parte da arquitetura proposta neste trabalho é focada na integração dos padrões WS-Management e WS-Security, de modo a realizar uma troca de mensagens de gerenciamento via Web Services com segurança. Aqui, a avaliação de desempenho foi realizada com o objetivo de observar o impacto da segurança no desempenho da troca de mensagens. Foram realizadas medidas de dois aspectos: tempo de resposta e tráfego na rede. Estes aspectos foram medidos pois são os habitualmente considerados em pesquisas focadas no gerenciamento de redes via Web Services, como visto na seção de trabalhos relacionados.

A arquitetura incorpora 3 características de segurança: autenticação, criptografia e assinatura digital. A implementação dessas 3 características é realizada através de fases do módulo Rampart no Axis2. Essas fases são independentes uma da outra, e podem ser facilmente habilitadas/desabilitadas individualmente na implementação, através da alteração de arquivos de configuração. Assim, o impacto no desempenho pode ser avaliado tanto para cada fase aplicada isoladamente na arquitetura, quanto para a utilização conjunta das mesmas.

Na avaliação de desempenho realizada, foram feitas 30 medidas para 5 combinações das funcionalidades do WS-Security. Estes 5 cenários serão denominados conforme abaixo:

- Cenário ALL OFF: sem uso de WS-Security.
- Cenário AUTH ON: uso apenas de autenticação através de campos com nome de usuário e senha criptografada.
- Cenário ENCR ON: uso apenas de criptografia.
- Cenário SIGN ON: uso apenas de assinatura digital.
- Cenário ALL ON: uso de autenticação, criptografia e assinatura digital.

Os valores medidos para tempo de resposta e tráfego na rede, para cada um dos 5 cenários, são apresentados nas próximas seções.

5.1.1 Avaliação do tempo de resposta

Na troca completa de mensagens de requisição e resposta, foram realizadas 7 medidas de tempo parciais. Assim, temos as seguintes 8 medidas realizadas:

- T_T: Tempo total de processamento da troca de mensagens, medido desde antes do envio da requisição WS-Management pelo gerente, até o término do processamento da resposta à essa requisição pelo gerente.

- T_{TG}: Tempo total de processamento no gerente WS-Management. Este tempo constitui a soma entre o tempo de processamento da mensagem de requisição, com o tempo posterior de processamento da resposta (ou seja, não constitui uma medida contínua, mas a soma de duas medidas).

- T_{SOG}: Tempo de processamento, no gerente WS-Management, no envio da mensagem de requisição, relativo à transformação da mensagem WS-Management em mensagem WS-Security. "SO" é a abreviação de *Security Out*.

- T_{SIG}: Tempo de processamento, no gerente WS-Management, na recepção da mensagem de resposta, relativo à transformação da mensagem WS-Security em mensagem WS-Management. "SI" é a abreviação de *Security In*.

- T_{TA}: Tempo total de processamento no agente WS-Management. Este tempo constitui a recepção e processamento da requisição, realização da operação desejada, e construção e envio da resposta.

- T_{SOA}: Tempo de processamento, no agente WS-Management, no envio da mensagem de resposta, relativo à transformação da mensagem WS-Management em mensagem WS-Security.

- T_{SIA}: Tempo de processamento, no agente WS-Management, na recepção da mensagem de requisição, relativo à transformação da mensagem WS-Security em mensagem WS-Management.

- T_R: Tempo de rede, na comunicação entre gerente e agente.

É interessante destacar algumas relações entre os tempos apresentados:

- $T_T = T_{TG} + T_{TA} + T_R$: o tempo total é a soma dos tempos de processamento no agente e no gerente com o tempo de rede.

- Para X = A ou X = G, $T_{TX} > T_{SOX} + T_{SIX}$: no agente e no gerente, o tempo total de processamento inclui os tempos de *Security Out* e *Security In*. Portanto, o tempo de processamento não relacionado a segurança pode ser obtido por $T_{TX} - T_{SOX} - T_{SIX}$.

Foram realizadas 31 medidas para uma troca completa de mensagens na implementação da arquitetura, para cada um dos 5 cenários avaliados. A primeira

medida foi descartada, já que o tempo de inicialização de classes Java, não-existente nas medidas subsequentes, introduziria um erro significativo nas médias obtidas. O tempo médio, desvio padrão e intervalo de confiança de 95% das outras 30 medidas são apresentados na tabela 5.2. Os valores são dados em milisegundos.

Tabela 5.2: Tempos de resposta na solução WS-Management x WS-Security

		Gerente			Agente			
	T_T	T_{TG}	T_{SOG}	T_{SIG}	T_{TA}	T_{SOA}	T_{SIA}	T_R
ALL ON								
Tempo médio	961,97	567,27	46,93	106,20	392,20	24,57	84,27	2,50
Desvio padrão	194,00	134,19	14,26	69,25	76,22	17,70	52,96	10,94
Intervalo de confiança	69,42	48,02	5,10	24,78	27,27	6,33	18,95	3,92
AUTH ON								
Tempo médio	787,50	468,27	9,40	20,90	313,63	5,33	10,40	5,60
Desvio padrão	170,17	106,91	7,82	7,57	74,93	7,67	14,96	12,02
Intervalo de confiança	60,89	38,25	2,80	2,71	26,81	2,75	5,35	4,30
ENCR ON								
Tempo médio	883,87	528,23	17,23	69,83	347,83	12,07	57,77	7,80
Desvio padrão	200,03	119,52	9,51	21,32	88,58	17,32	32,90	9,98
Intervalo de confiança	71,58	42,77	3,40	7,63	31,70	6,20	11,77	3,57
SIGN ON								
Tempo médio	868,77	513,47	34,30	33,30	348,37	19,27	25,57	6,93
Desvio padrão	159,79	113,25	12,02	19,15	63,61	15,20	17,22	12,08
Intervalo de confiança	57,18	40,53	4,30	6,85	22,76	5,44	6,16	4,32
ALL OFF								
Tempo médio	753,13	444,87	1,07	5,27	291,70	3,10	0,50	16,57
Desvio padrão	171,72	103,96	4,06	7,58	78,42	6,31	2,74	25,12
Intervalo de confiança	61,45	37,20	1,45	2,71	28,06	2,26	0,98	8,99

Pode-se perceber que, em relação ao tempo total de troca de uma mensagem, o tempo utilizado no processamento da adição e interpretação de informações de segurança não é o mais significativo da operação. Mesmo quando todas as funcionalidades de segurança são empregadas (cenário ALL ON), este tempo (T_{SOG} + T_{SIG} + T_{SOA} + T_{SIA}) corresponde a 27,23% do total (T_T). A partir desta observação dos resultados, foi realizada uma análise do código da solução para determinar o motivo deste tempo de processamento.

A razão para o alto tempo total de processamento, quando comparado com o tempo usado pelas fases de segurança do Rampart, é o uso da API JAXB pelo Wiseman. A principal função desta API é converter instâncias em memória de classes Java para Strings representando estas classes em XML, e vice-versa. Estas conversões são realizadas via JAXB através de métodos de um objeto do tipo `JAXBContext`. A inicialização deste objeto é demorada, pois são carregados alguns arquivos de configuração com diretrizes para os métodos que mapeiam classes e pacotes Java para

XML. A inicialização desta classe é chamada 3 vezes ao longo do processamento medido em T_T, devido à forma como a API Wiseman é implementada (as classes do Wiseman usadas para a construção das mensagens WS-Management exige que essas 3 chamadas à inicialização de `JAXBContext` sejam realizadas).

O fato de o tempo total de processamento não relacionado ao uso de segurança ser grande faz com que as variações neste processamento se sobressaiam nas medidas efetuadas. Mesmo assim, é possível notar a influência do processamento relacionado à segurança no tempo total. O cenário com autenticação (AUTH ON) tem tempo total 4,56% superior ao cenário sem segurança (ALL OFF). Os cenários com criptografia (ENCR ON) e assinatura digital (SIGN ON) apresentam maior impacto, com tempos totais, respectivamente, 17,36% e 15,35% superiores ao cenário ALL OFF. Por fim, o tempo total do cenário com todas as funcionalidades de segurança (ALL ON), supera o do cenário ALL OFF em 27,73%.

Para melhor visualizar as diferenças no tocante à segurança entre os cinco cenários avaliados, os tempos de processamento relacionados à segurança (T_{SOG}, T_{SIG}, T_{SOA} e T_{SIA}) são exibidos no gráfico de barras da figura 5.1.

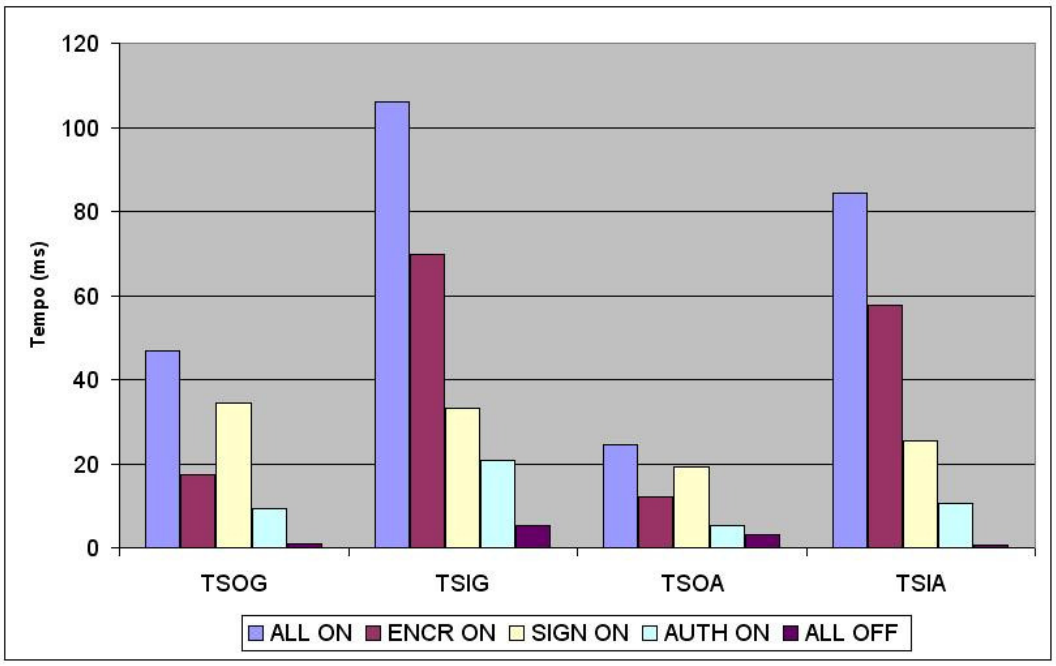


Figura 5.1: Tempos de processamento relacionados a segurança na solução WS-Security x WS-Management

O gráfico da figura 5.1 mostra que há tempo de processamento maior que zero no cenário ALL OFF. Neste cenário, esse tempo é composto apenas do processamento necessário para executar a chamada ao módulo Rampart, que executa um trecho de código que carregam e percorrem as fases configuradas (nesse caso, o código não executa nenhuma fase na prática, conforme definido nos arquivos de configuração).

Há uma diferença relevante nos cenários ENCR ON e SIGN ON, que são as funcionalidades de segurança com maior peso de processamento. No cenário ENCR

ON, os tempos de processamento de interpretação de mensagens seguras (*Security In*) são mais altos que o tempo de introdução de informações de segurança nas mensagens (*Security Out*). No cenário SIGN ON, no entanto, esses tempos são similares. Isso é umr reflexo direto da natureza dos procedimentos de criptografia e assinatura digital. A decriptação tende a ser mais trabalhosa que a encriptação. Já a construção de uma assinatura digital tem complexidade próxima à interpretação da mesma assinatura.

5.1.2 Avaliação do tráfego na rede

Foram realizadas medidas de tráfego na rede para os 5 cenários apresentados na avaliação de tempo de processamento (ALL ON, ENCR ON, SIGN ON, AUTH ON e ALL OFF). Foi medido o tamanho total, em bytes, das mensagens WS-Security de requisição e resposta entre o gerente e o agente. Também foi verificado o tamanho em bytes utilizado nas mensagens para cada protocolo (Ethernet, TCP e IP) e para os dados em si (*payload*).

As medidas realizadas são apresentadas na tabela 5.3. As medidas relativas às mensagens de requisição do gerente para o agente são apresentadas nas colunas indicadas pela expressão G => A (Gerente => Agente). As medidas de mensagens de resposta do agente para o gerente aparecem nas colunas A => G (Agente => Gerente):

Tabela 5.3: Tráfego na rede para a solução WS-Management x WS-Security

	ALL ON		AUTH ON		ENCR ON		SIGN ON		ALL OFF	
Bytes	**G=>A**	**A=>G**	**G=>A**	**A=>G**	**G=>A**	**A=>G**	**G=>A**	**A=>G**	**G=>A**	**A=>G**
Ethernet	224	210	56	56	168	140	98	126	56	70
IP	320	300	80	80	240	200	140	180	80	100
TCP	320	300	80	80	240	200	140	180	80	100
Dados	11503	11203	2143	2779	7220	7242	5492	5795	1515	1915
Total	**12367**	**12013**	**2359**	**2995**	**7868**	**7782**	**5870**	**6281**	**1731**	**2185**

Podemos perceber que a adição de funcionalidades de segurança traz um impacto alto para o tráfego na rede. O único caso em que o impacto é pouco significativo é no uso de autenticação, em que a mensagem de requisição aumenta 1,36 vezes, e a mensagem de resposta, 1,37 vezes. Nos outros cenários, a diferença de tamanho total é alta: no cenário SIGN ON, a mensagem de requisição é 3,39 vezes maior, e na de resposta, 2,87 vezes. Com uso exclusivo de criptografia, a requisição é 4,55 vezes maior, e a resposta, 3,56 vezes. Por fim, no cenário ALL ON, com todos os aspectos de segurança avaliados, a mensagem de requisição aumenta 7,14 vezes, e a de resposta, 5,50 vezes.

Para melhor visualização da diferença entre os tamanhos de mensagens de cada cenário, os tamanhos totais medidos são apresentrados no gráfico de barras da figura 5.2:

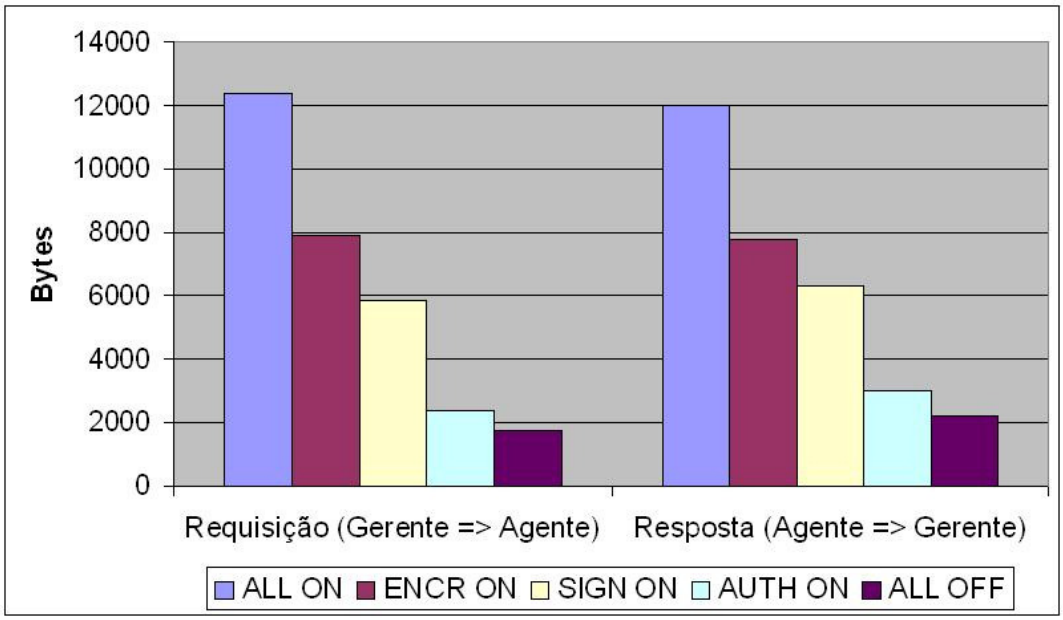


Figura 5.2: Tráfego na rede para a solução WS-Security x WS-Management

5.2 Avaliação de desempenho XACML x SNMP

A segunda parte da avaliação de desempenho foca na integração de controle de acesso à solução WS-Management x WS-Security existente. A arquitetura completa deste trabalho, com esta integração, que pode ser visualizada na figura 4.1, permite o uso de controle de acesso via XACML, para introduzir controle de acesso quando há agentes SNMPv1/v2 na rede, ou a MIB VACM diretamente, no caso de agente SNMPv3.

O objetivo da avaliação realizada é verificar o desempenho da integração de agentes SNMP no tocante à controle de acesso, comparando cenários com agentes SNMPv2 (com uso de XACML ou sem controle de acesso) e SNMPv3 (com a MIB VACM), e também com ausência de *gateway*, para permitir verificar qual o impacto da arquitetura com uma solução usando apenas gerente e agente SNMP.

As combinações de características citadas resultaram em 5 cenários de avaliação:

- Cenário 1: agente SNMPv2, sem *gateway*.
- Cenário 2: agente SNMPv2, com *gateway*, sem uso de XACML.
- Cenário 3: agente SNMPv2, com *gateway*, com uso de XACML.
- Cenário 4: agente SNMPv3, sem *gateway*, com MIB VACM.
- Cenário 5: agente SNMPv3, com *gateway*, com MIB VACM.

Nos cenários 1 e 4, sem *gateway*, não faz sentido falarmos em XACML, por não termos a presença de gerente WS-Management (foi testada a comunicação direta entre gerente e agente SNMP). No cenário 5, também não se utiliza XACML, pois o controle de acesso é realizado diretamente pelo agente SNMPv3, através da MIB VACM. Por sua vez, a MIB VACM não aparece nos cenários de 1 a 3, pois não é suportada pelo agente SNMPv2.

No cenário 3, em que foi usado XACML com agente SNMPv2, foram realizadas medidas com diferentes números de regras XACML armazenadas no *gateway*, variando de 1 a 32, em potências de 2. O objetivo desta variação foi verificar o impacto do volume de regras existentes no tempo de processamento. Para permitir uma comparação direta, do cenário 3 com o cenário 5, no cenário 5 também foram feitos testes variando o número de entradas da MIB VACM de 1 a 32, em potências de 2. No cenário 4, foi realizada apenas medida com 32 entradas da MIB VACM, para não gerar um volume muito grande de medidas a apresentar, já que esse cenário foi utilizado apenas para comparação entre a presença e ausência de *gateway*, o que não está relacionado à introdução de segurança, que é o foco principal deste trabalho

Os valores medidos para tempo de resposta e tráfego na rede, para cada um dos 5 cenários, são apresentados nas próximas seções.

5.2.1 Avaliação do tempo de resposta

Na troca completa de mensagens, foram realizadas 4 medidas parciais: tempo de processamento no gerente WS-Management (T_G), no *gateway* (T_{GW}), no agente SNMP (T_A), e na rede (T_R). O tempo total é referenciado como T_T. Para os cenários 1 e 4, onde não há gerente WS-Management, T_G não faz sentido. O tempo de processamento do gerente SNMP, para esses cenários, aparece em T_{GW}, já que para os demais cenários, que utilizam a arquitetura completa, é nesta parcial que o mesmo tempo de processamento do gerente SNMP aparece, sendo que nesses cenários o gerente SNMP é chamado pelo *gateway*.

Foram realizadas 31 medidas para uma troca completa de mensagens, para cada um dos 5 cenários avaliados. A primeira medida foi descartada, para eliminar o impacto do tempo de inicialização de classes Java, e o tempo médio, desvio padrão e intervalo de confiança de 95% das outras 30 medidas são apresentados na tabela 5.4. Os valores são dados em milisegundos. Apesar de termos 5 cenários, como nos cenários 3 e 5 houve medidas com 6 variações de entradas de controle de acesso (XACML ou MIB VACM), tivemos a realização de 15 baterias de 31 medidas.

Tabela 5.4: Tempos de resposta para avaliação de uso de controle de acesso

Cenários		T_T	T_G	T_{GW}	T_A	T_R
1	Tempo Médio	37,93	NA	6,33	26,50	5,10
	Desvio Padrão	15,16	NA	3,90	8,07	3,64
	Intervalo de Confiança	8,55	NA	2,33	2,59	1,73
2	Tempo Médio	833,87	512,50	263,40	26,47	31,50
	Desvio Padrão	163,37	103,60	73,91	6,82	13,23
	Intervalo de Confiança	58,46	37,07	26,45	2,44	4,74
3 (1 regra XACML)	Tempo Médio	836,97	487,57	288,50	25,50	35,40
	Desvio Padrão	149,30	97,01	68,12	7,14	10,78
	Intervalo de Confiança	53,42	34,71	24,37	2,55	3,86
3 (2 regras XACML)	Tempo Médio	831,27	495,43	282,03	23,47	30,33
	Desvio Padrão	152,39	98,97	69,31	7,60	15,93
	Intervalo de Confiança	54,53	35,42	24,80	2,72	5,70
3 (4 regras XACML)	Tempo Médio	832,30	504,73	270,10	24,47	33,00
	Desvio Padrão	138,88	112,61	57,90	7,53	14,42
	Intervalo de Confiança	49,70	40,30	20,72	2,70	5,16
3 (8 regras XACML)	Tempo Médio	859,90	506,07	291,87	26,03	35,93
	Desvio Padrão	161,09	97,25	77,50	6,85	12,15
	Intervalo de Confiança	57,64	34,80	27,73	2,45	4,35
3 (16 regras XACML)	Tempo Médio	865,67	509,47	292,70	27,53	35,97
	Desvio Padrão	146,72	106,33	59,09	6,41	12,64
	Intervalo de Confiança	52,50	38,05	21,15	2,20	4,52
3 (32 regras XACML)	Tempo Médio	922,93	505,00	356,43	25,90	35,60
	Desvio Padrão	125,33	97,11	65,98	7,13	11,56
	Intervalo de Confiança	44,85	34,75	23,61	2,55	4,14
4 (32 entradas VACM)	Tempo Médio	156,27	NA	2,23	148,43	5,60
	Desvio Padrão	27,66	NA	1,61	26,76	3,71
	Intervalo de Confiança	9,90	NA	0,81	9,58	1,76
5 (1 entrada VACM)	Tempo Médio	903,63	466,63	270,27	126,23	40,50
	Desvio Padrão	165,91	115,39	66,26	8,50	10,63
	Intervalo de Confiança	59,37	41,29	23,71	3,04	3,80
5 (2 entradas VACM)	Tempo Médio	926,43	501,07	254,87	129,47	41,03
	Desvio Padrão	158,61	103,45	62,45	9,57	10,49
	Intervalo de Confiança	58,12	37,02	22,97	3,43	3,75
5 (4 entradas VACM)	Tempo Médio	915,10	487,10	257,63	130,43	39,93
	Desvio Padrão	140,87	93,81	57,01	12,46	11,47
	Intervalo de Confiança	50,41	33,57	20,40	4,46	4,11
5 (8 entradas VACM)	Tempo Médio	921,33	490,50	257,07	133,80	39,97
	Desvio Padrão	151,76	112,99	69,99	10,88	10,10
	Intervalo de Confiança	54,30	40,43	25,04	3,89	3,61
5 (16 entradas VACM)	Tempo Médio	921,37	482,30	256,60	141,47	41,00

	Desvio Padrão	153,21	100,20	73,83	17,24	8,07
	Intervalo de Confiança	54,82	35,86	26,42	6,17	2,89
	Tempo Médio	939,60	493,30	259,87	141,47	44,97
5 (32 entradas VACM)	Desvio Padrão	208,52	107,29	161,84	13,92	12,16
	Intervalo de Confiança	74,62	38,39	57,91	4,98	4,35

Para melhor visualizar as diferenças entre os cenários avaliados, os tempos de processamento totais são exibidos no gráfico de barras da figura 5.3.

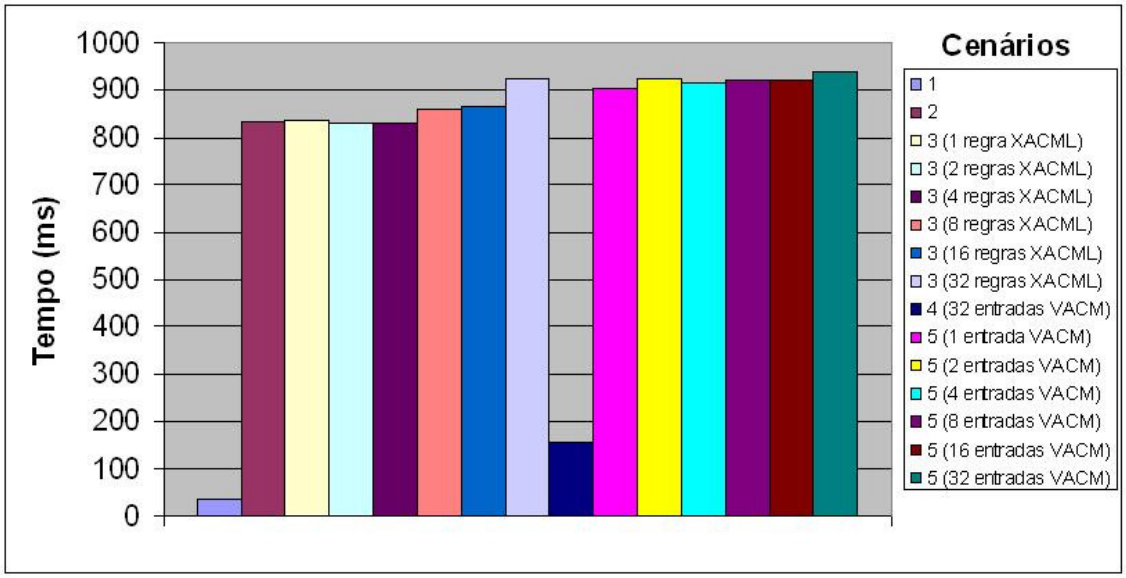


Figura 5.3: Tempos de processamento na solução com controle de acesso

O cenário 1, em que não há uso de *gateways* nem de controle de acesso, tem tempo total de processamento de 37,93 ms. O cenário 4, em que se passa a usar controle de acesso via MIB VACM do SNMPv3, tem um tempo 4,12 vezes maior. Este impacto é natural, e ocorre com a introdução de características de segurança em qualquer sistema. A maior parte deste acréscimo de tempo está localizada no agente SNMPv3, que é onde as operações de controle de acesso são executadas.

Os cenários 2, 3 e 5, que utilizam a arquitetura completa, com *gateways* Web Services x SNMP, tem um tempo consideravelmente maior. No menor tempo desses cenários, que é o cenário 3 com 2 regras XACML, a duração é 5,32 vezes maior que a do cenário 4. No maior tempo, o cenário 5 com 32 entradas VACM, a duração é 6,01 vezes maior. Esta alta diferença de tempo não torna a arquitetura proposta impeditiva, pois conforme exposto na explicação da tabela 5.2, a API JAXB do Wiseman é a principal causa deste tempo, e melhorias no uso da API poderiam facilmente diminuir esse tempo por uma razão de 3 vezes.

Comparando as medidas do cenário 2 com as do cenário 3, em que é introduzido o uso de XACML, é interessante notar que, havendo poucas regras XACML no agente WS-Management, o tempo total tem pouca variação, inclusive diminuindo quando

haviam 2 e 4 regras XACML. Isso ocorre devido ao impacto alto do processamento JAXB nas medidas totais, que acabam encobrindo a pequena variação de tempo decorrente do uso de XACML. Mesmo assim, ao aumentar, no cenário 3, o número de regras para 8, 16 e 32, pode-se notar um impacto razoável no tempo total, chegando a 10,68% para 32 regras, com relação ao tempo do cenário 2. Esta variação aparece em T_{GW}, que é onde está localizado o agente WS-Management, e onde as regras XACML são buscadas e avaliadas.

Da mesma forma que o impacto do uso de XACML pode ser verificado comparando os cenários 2 e 3, o impacto do uso da MIB VACM, através da introdução de um agente SNMPv3, pode ser verificado comparando os cenários 2 e 5. Ao contrário do XACML, este cenário traz impacto visível já com apenas uma entrada VACM, impacto este de 8,37% em relação ao tempo total do cenário 2. O tempo total aumenta conforme o número de entradas VACM aumenta para 32, chegando a 12,68% em relação ao cenário 2. Esta variação aparece em T_A, onde as políticas de acesso VACM são determinadas e aplicadas.

Pode-se visualizar também uma comparação entre os cenários 3 e 5, onde o desempenho do controle de acesso via Web Services (XACML) é comparado com o controle via SNMPv3 (MIB VACM). Nas comparações diretas entre as medidas em que o número de regras XACML é idêntico ao número de entradas VACM, o tempo total de processamento usando Web Services é sempre menor. Apesar disso, é preciso ressaltar que, com a variação nos números utilizados (1 a 32), o crescimento dos tempos medidos para o cenário 3 conforme se aumentam as regras é maior que o do cenário 5. Comparando-se as medidas do cenário 3 com 1 e 32 regras XACML, temos um aumento de 10,27%. Comparando-se o cenário 5 com 1 e 32 entradas VACM, o aumento é de 3,98%. Isso pode indicar um desempenho pior da solução com XACML caso haja um grande número (centenas, por exemplo) de regras XACML presentes.

5.2.2 Avaliação do tráfego na rede

A utilização ou não de controle de acesso, seja via XACML ou via MIB VACM, não tem impacto no tamanho dos pacotes trafegados na rede, seja de requisição ou de resposta, pois o uso de controle de acesso não altera as mensagens trocadas, apenas o processamento nos agentes WS-Management e SNMPv3. Assim, no tocante aos cenários utilizados para verificar o tempo de processamento das soluções com controle de acesso, há variações de tráfego na rede apenas devido ao uso de versões diferentes de SNMP, e ao uso ou não de *gateways* WS-Management x SNMP. Estas variações aparecem nas comunicações gerente-agente nos 3 diferentes protocolos utilizados: WS-Management, SNMPv2 e SNMPv3.

Foi medido o tamanho total, em bytes, das mensagens de requisição e resposta entre o gerente e o agente, para os 3 protocolos utilizados. Também foi verificado o tamanho em bytes utilizado nas mensagens para cada protocolo (Ethernet, TCP / UDP e IP) e para os dados em si (*payload*). O protocolo de transporte é TCP para WS-Management e UDP para SNMPv2 e SNMPv3.

As medidas realizadas são apresentadas na tabela 5.5. As medidas relativas às mensagens de requisição do gerente para o agente são apresentadas nas colunas indicadas pela expressão G => A (Gerente => Agente). As medidas de mensagens de resposta do agente para o gerente aparecem nas colunas A => G (Agente => Gerente):

Tabela 5.5: Tráfego na rede para avaliação de uso de controle de acesso

	WS-Management		SNMPv2		SNMPv3	
Bytes	**G=>A**	**A=>G**	**G=>A**	**A=>G**	**G=>A**	**A=>G**
Ethernet	56	70	14	14	14	14
IP	80	100	20	20	20	20
TCP / UDP	80	100	8	8	8	8
Dados	1515	1915	132	173	217	257
Total	**1731**	**2185**	**174**	**215**	**259**	**299**

Percebe-se o alto impacto que o uso de WS-Management traz ao tráfego na rede. Comparando com o SNMPv3, o impacto na mensagem de requisição é de 6,68 vezes, e na mensagem de resposta, de 7,31 vezes. Já a introdução de segurança no SNMPv3, trouxe um aumento razoável em relação ao uso de SNMPv2: 48,85% de aumento no tráfego para a requisição, e 39,07% para a resposta.

Para melhor visualização da diferença entre os tamanhos de mensagens para cada protocolo, os tamanhos totais medidos são apresentrados no gráfico de barras da figura 5.4.

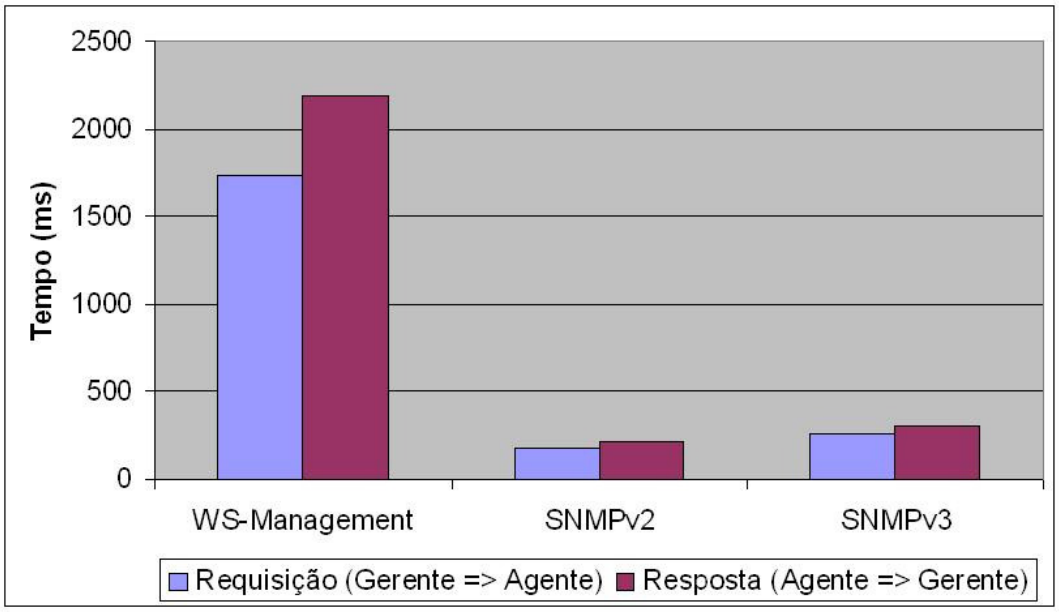


Figura 5.4: Tráfego na rede para a solução com controle de acesso

Os dois conjuntos de testes avaliados mostraram que o impacto da segurança no gerenciamento de redes via Web Services é alto. Isto é esperado e comum em quaisquer sistemas em que se introduz segurança. Em geral, a degradação de desempenho acaba sendo admitida em aplicações nas quais confidencialidade, integridade e controle de acesso são requisitos de funcionamento.

6 CONCLUSÕES E TRABALHOS FUTUROS

As redes de computadores têm presença maciça na sociedade atual. Grande parte dos serviços oferecidos entre as pessoas e empresas depende das redes de computadores, de forma mais acentuada com o surgimento e massificação do uso da Internet. Problemas de funcionamento nas redes de computadores podem causar inúmeros transtornos, com graves impactos financeiros ou à vida humana, por exemplo. Desta forma, há a necessidade de um gerenciamento de alta qualidade nas redes de computadores. A pesquisa e desenvolvimento de tecnologias na área de gerência de redes é preponderante para a obtenção da qualidade necessária.

O protocolo padrão para gerência de redes, o SNMP, possui uma série de limitações diante das redes de computadores atuais, como escalabilidade, segurança e composição de serviços. Por isso, a comunidade industrial e científica vem pesquisando padrões alternativos para o gerenciamento de redes, dentre os quais surgiu com força a tecnologia de Web Services. Esta tecnologia possui uma série de características que a tornam uma opção interessante para o gerenciamento de redes, dentre as quais se destacam o transporte baseado em HTTP e HTTPS, o uso de XML para troca de dados, e o modelo de desenvolvimento orientado a serviços. Em princípio, a maior preocupação com relação aos Web Services era relacionada com desempenho, pelo fato de a tecnologia ser baseada em XML, que é um protocolo verboso. Assim, os estudos iniciais focaram em desempenho, como por exemplo, (NEISSE, 2004) e (PRAS, 2004). Esses estudos comprovaram a viabilidade de uso dos Web Services, mostrando que o desempenho não limitava o uso da tecnologia, especialmente em testes feitos com troca de alto volume de informações de gerenciamento.

Com a concordância na viabilidade do uso de Web Services, houve desenvolvimento de pesquisas mais específicas na área acadêmica. O foco das pesquisas se voltou para análises de desempenho e aplicabilidade em áreas específicas de gerência de redes, como a gerência por delegação (FIOREZE, 2005) e notificações / *traps* (LIMA, 2006). O desempenho dos Web Services nas áreas específicas também foi satisfatório. Como as características dos Web Services possibilitam o desenvolvimento de arquiteturas otimizadas e de mais fácil utilização do que o SNMP, a idéia do uso de Web Services se consolidou. Isso propiciou o desenvolvimento, pela indústria, de padrões específicos para o gerenciamento de redes, como o WS-Management e o MUWS.

Uma das áreas mais relevantes para a gerência de redes é a segurança. A garantia de confidencialidade e não-violação de informações de gerenciamento é vital para

aplicações reais, especialmente quando o tráfego se dá via Internet, o que é uma tendência crescente. A segurança é uma das lacunas principais do padrão SNMP; por outro lado, a pilha de protocolos dos Web Services inclui padrões relacionados à segurança. O padrão WS-Security possibilita que as mensagens XML sejam trocadas com garantia de confidencialidade, integridade e autenticação, inclusive em pontos intermediários. Dessa forma, a aplicação de segurança para a gerência de redes baseada em Web Services é possibilitada pelo WS-Security. Diante da ausência de estudos quanto à aplicação de segurança ao gerenciamento de redes via Web Services, este trabalho tem por objetivo preencher esta lacuna e motivar o desenvolvimento de outros trabalhos nesta abrangente área. Este trabalho propõe uma arquitetura que integra segurança ao gerenciamento de redes via Web Services. Esta arquitetura utiliza o padrão WS-Security, que é o padrão de consenso na indústria para segurança de Web Services, e o WS-Management, que é o padrão de gerência de redes via Web Services em que se vê maior tendência de utilização futura, dentre os dois existentes (WS-Management e MUWS).

A integração dos padrões na arquitetura é direta, com a aplicação de segurança via WS-Security antes do envio de uma mensagem WS-Management de requisição pelo gerente, e a verificação das informações de segurança no recebimento da mensagem pelo agente WS-Management. A mesma aplicação de segurança se dá no envio da resposta pelo agente e recepção da mesma pelo gerente. A aplicação de WS-Security na arquitetura proposta possibilita que se tenha confidencialidade na troca de mensagens (via XML-Encryption), bem como integridade (via XML-Digital Signature) e autenticação.

Uma funcionalidade de segurança muito importante e não contemplada ainda pelo WS-Security é o controle de acesso. Esta funcionalidade foi também integrada à arquitetura proposta, através do uso do padrão XACML, que implementa controle de acesso através de requisições, respostas e políticas de acesso representadas via XML. Esta integração foi realizada através da definição de políticas de acesso XACML para os recursos gerenciados através de um agente WS-Management. Quando o controle de acesso via XACML é utilizado, e uma requisição chega ao agente, é determinada a política de acesso correspondente à requisição, e então, o acesso é permitido ou negado.

A arquitetura proposta utiliza a implementação Wiseman do padrão WS-Management, e a implentação Axis2 + Rampart para o padrão WS-Security. Para o XACML, a implementação SunXACML é utilizada. A arquitetura da implementação Axis2 + Rampart facilita com que as diferentes opções de segurança existentes (confidencialidade, integridade e autenticação) seja habilitada e desabilitada individualmente com praticidade, o que foi útil para a avaliação de desempenho realizada.

No tocante ao planejamento da arquitetura e no desenvolvimento de sua implementação, não houveram dificuldades consideráveis, devido à natureza robusta e flexível da arquitetura dos Web Services. Assim, realizado o trabalho de implementação, foram encaminhados testes de desempenho, que são de alta relevância na área de segurança, que geralmente tem alto impacto no desempenho.

A avaliação de desempenho deste trabalho foi dividida em 2 etapas. Primeiro, foi avaliado o desempenho da integração entre WS-Security e WS-Management. Num segundo momento, foi avaliado o desempenho da integração de XACML à arquitetura.

Na primeira avaliação realizada, foi medido o impacto da aplicação de WS-Security em uma comunicação de gerenciamento via WS-Management. O impacto foi verificado em termos de tempo de processamento e tráfego na rede. Para fins de comparação foram avaliados 5 cenários: um sem uso de segurança, 3 cenários com aplicação individual de autenticação, confidencialidade e integridade, e um cenário combinando estas 3 funcionalidades.

Em relação ao tempo de processamento, os testes mostraram um impacto muito alto do uso da API JAXB, utilizada pelo Wiseman para transformação entre objetos Java e mensagens XML. Devido à forma como a API Wiseman é utilizada para construção de mensagens, alguns procedimentos demorados do JAXB são realizados várias vezes. Dessa forma, é interessante avaliar o impacto de segurança tanto em termos relativos quanto absolutos. Em termos relativos, o uso de autenticação aumentou o tempo de processamento em cerca de 5%, criptografia e integridade, por volta de 15%, e o uso de todas as possibilidades de segurança, 27%. Caso o uso de JAXB fosse otimizado, esse impacto relativo da segurança seria bem maior. Acreditamos que uma melhoria simples no uso do JAXB diminuiria o tempo total não-relacionado à segurança em 3 vezes. Com isso, o impacto de todos os aspectos de segurança segurança passaria de 27% para quase 100%. Em termos absolutos, o uso de autenticação trouxe um acréscimo de cerca de 45 ms à comunicação. A integridade adicionou 110 ms, e a criptografia, 150 ms. A aplicação simultânea das 3 características de segurança adicionou 260 ms à comunicação.

No tocante a tráfego na rede na primeira avaliação, notou-se o alto impacto da aplicação de segurança. Foi verificada a alteração do tamanho das mensagens de requisição e resposta. O impacto na mensagem de requisição, que nos testes era menor que a mensagem de resposta, foi maior em todos os cenários, com exceção do cenário com autenticação. Neste cenário, as mensagens de requisição e resposta cresceram, respectivamente, 36 e 37%. Com integridade, o aumento foi de 3,39 e 2,87 vezes. Com criptografia, 4,55 e 3,56 vezes. Por fim, com todos os parâmetros juntos, as mensagens aumentaram 7,14 e 5,50 vezes.

Na segunda avaliação de desempenho, foi avaliado o impacto do uso de XACML. Para essa avaliação, foram integrados agentes SNMPv2 e SNMPv3 à arquitetura, gerenciáveis através de um *gateway* WS / SNMP presente no agente WS-Management. O uso de XACML permitiu que houvesse controle de acesso para o agente SNMPv2, que não possui essa funcionalidade. O controle de acesso foi feito no próprio *gateway*, e quando o acesso era permitido, então o *gateway* era acionado para repassar a requisição ao agente SNMP. Assim, foi possível comparar essa solução com o uso de controle de acesso nativo do SNMPv3 (MIB VACM). As medidas foram realizadas para tempo de processamento e tráfego na rede.

No tempo de processamento, observou-se que o uso de XACML em si trouxe pouco impacto à comunicação. Isso foi concluído com os tempos medidos em cenários em que se aplicou XACML, mas haviam poucas regras XACML para o agente SNMPv2 no *gateway*. Nesses casos, a variação de tempo foi nula (aqui, os testes foram prejudicados pelo impacto do JAXB no tempo total de processamento). Porém, quando o número de regras foi passado para 32, a variação foi de cerca de 10%. Já o uso de agente SNMPv3 com a MIB VACM, mesmo com poucas entradas VACM (1, no caso),

já trouxe um impacto de 8%. Com 32 entradas, o impacto foi de 12%. Portanto, a solução com XACML teve desempenho superior. Porém, a curva de crescimento em relação ao número de regras XACML ou entradas da MIB VACM indica que, para um número maior de regras ou entradas (centenas, por exemplo), a degradação do desempenho do XACML pode ser pior.

Em relação à tráfego na rede, o uso de controle de acesso não traz impacto, pois o conteúdo das mensagens não muda. Assim, foi feita apenas a medida das mensagens WS-Management, SNMPv2 e SNMPv3 utilizadas. Comparando o SNMPv3 com SNMPv2, as mensagens de requisição e resposta aumentaram, respectivamente, 48 e 39%. O WS-Management, em relação ao SNMPv2, acarretou um aumento de 6,68 e 7,31 vezes no tamanho das mensagens.

A avaliação de desempenho realizada comprova o que se esperava com relação à aplicação de segurança aos cenários de gerenciamento via Web Services: o impacto é consideravelmente alto, tanto em tempo de processamento quanto no tráfego na rede. Esta degradação de desempenho causada pelo uso de segurança é tradicional nos mais diversos sistemas, e não foi diferente para os Web Services. Esse impacto se dá pois os procedimentos de segurança, como encriptação e decriptação de dados, e construção e interpretação de assinaturas digitais, são custosos. A complexidade dos algoritmos aplicados é necessária para que a segurança seja efetiva, e esta complexidade reflete diretamente em processamento mais demorado e mensagens maiores.

Devido ao seu impacto no desempenho, como em qualquer sistema, a apliação se segurança ao gerenciamento de redes baseado em Web Services deve ser avaliada conforme sua necessidade. Em aplicações financeiras, por exemplo, não há alternativas senão abrir mão do desempenho para que seja possível ter a segurança desejada. Porém, este trabalho demonstra que a aplicação da idéia em si, de agregar segurança à gerência de redes via Web Services, é viável e facilmente obtida. Isso traz mais uma comprovação, com a área de segurança plenamente atendida, de que a tecnologia de Web Services é adequada para a área de gerenciamento de redes.

Por ser um primeiro estudo com foco em segurança para gerência de redes baseada em Web Services, a realização deste trabalho permitiu a identificação de uma série de trabalhos futuros para trazer novas informações à esta área, conforme relatado a seguir.

Em relação aos aspectos de desempenho considerados, poderiam ser feitos testes visando avaliar o consumo de CPU e o uso de memória. Em relação ao segundo conjunto de testes, seria interessante avaliar cenários com maior número de regras XACML e entradas da MIB VACM, visando verificar se realmente há degradação do XACML em comparação ao SNMPv3, conforme os testes levam a crer, e quão alta é essa degradação.

É viável alterar o uso da API JAXB pelo Wiseman, e aplicar esta alteração na arquitetura proposta. Conforme mencionado anteriormente, um procedimento custoso de inicialização de valores do JAXB é executado 3 vezes no processamento de uma troca de mensagens de requisição e resposta. Esta melhoria só é possível alterando a API Wiseman. Além disso, investigações mais detalhadas podem revelar outras melhorias. Por exemplo, os testes deste trabalho foram sempre realizadas com uma troca de mensagens. Na troca de diversas mensagens sucessivas, a inicialização custosa que

foi mencionada poderia ser realizada apenas na primeira vez, reduzindo muito o tempo total de processamento.

Há trabalho em andamento visando um padrão convergente para gerência de redes via Web Services, fortemente baseado no WS-Management, mas visando substituir o uso de WS-Management ou MUWS. Com a conclusão deste novo padrão, a arquitetura pode ser alterada para utilizá-lo.

Devido a detalhes de forma de endereçamento do Wiseman e do Axis2, ambos foram utilizados separadamente, com as mensagens seguras recebidas pelo servidor Axis2 sendo interpretadas para mensagens WS-Management e encaminhadas para o servidor Wiseman, que é totalmente independente do servidor Axis2 (podendo inclusive estarem instalados em locais físicos distintos). Poderia se implementar a chamada das funções oferecidas pela implementação do servidor Wiseman diretamente no código executado pelo servidor Axis2. Estas funções são responsáveis por interpretar as requisições WS-Management e gerar as respostas conforme o padrão especifica.

No primeiro conjunto de testes realizados (WS-Security x WS-Management), poderia ser feita a mesma comparação com SNMPv3 realizada no conjunto de testes de controle de acesso. Ou seja, utilizar WS-Security para agregar confidencialidade, integridade e autenticação a um servidor SNMPv2. Obviamente, presume-se que, nesse caso, não seria um problema a comunicação insegura entre o *gateway* WS / SNMP e o agente SNMPv2, que poderia ser isolada. Como o SNMPv3 também tem as funcionalidades de segurança mencionadas, seria possível comparar as soluções.

Por fim, poderiam ser variados os algoritmos de criptografia e assinatura digital utilizados, bem como utilizadas diferentes formas de autenticação, para avaliar as diferenças de desempenho das diversas combinações.

REFERÊNCIAS

ALEXANDER, J. et al. **Web Services Enumeration (WS-Enumeration)**. 2006. W3C Member Submission. Disponível em http://www.w3.org/Submission/WS-Enumeration. Acesso em janeiro de 2009.

ALEXANDER, J. et al. **Web Services Transfer (WS-Transfer)**. 2006. W3C Member Submission. Disponível em http://www.w3.org/Submission/WS-Transfer. Acesso em janeiro de 2009.

APACHE SOFTWARE FOUNDATION. **Apache License, Version 2.0**. 2004. Disponível em http://www.apache.org/licenses/LICENSE-2.0. Acesso em janeiro de 2009.

APACHE SOFTWARE FOUNDATION. **Apache Tomcat**. Disponível em http://tomcat.apache.org. Acesso em janeiro de 2009.

APACHE SOFTWARE FOUNDATION. **Apache WSS4J**. Disponível em http://ws.apache.org/wss4j.

APACHE SOFTWARE FOUNDATION. **Rampart: WS-Security module for Axis2**. Disponível em http://ws.apache.org/axis2/modules/rampart/1_3/security-module.html. Acesso em janeiro de 2009.

APACHE SOFTWARE FOUNDATION. **The Apache Ant Project**. Disponível em http://ant.apache.org. Acesso em janeiro de 2009.

ARORA, A. et al. **Web Services for Management (WS-Management) Specification**. 2008. DMTF Specification. Disponível em http://www.dmtf.org/standards/published_documents/DSP0226_1.0.0.pdf. Acesso em janeiro de 2009.

AXIOM. **Apache AXIOM**. Disponível em http://ws.apache.org/commons/axiom. Acesso em janeiro de 2009.

AXIS2. **Apache Axis2/Java**. Disponível em http://ws.apache.org/axis2. Acesso em janeiro de 2009.

BALLINGER, K. et al. **Web Services Metadata Exchange 1.1 (WS-MetadataExchange)**. 2008. W3C Member Submission. Disponível em http://www.w3.org/Submission/WS-MetadataExchange. Acesso em janeiro de 2009.

BALLINGER, K. et al. **Basic Profile Version 1.1**. 2006. WS-I Final Material. Disponível em http://www.ws-i.org/Profiles/BasicProfile-1.1.html. Acesso em janeiro de 2009.

BELLWOOD, T. et al. **UDDI Version 3.0.2**. 2004. OASIS UDDI Spec Technical Committee Draft. Disponível em http://uddi.org/pubs/uddi_v3.htm. Acesso em janeiro de 2009.

BERGLUND, A. et al. **XML Path Language (XPath) 2.0**. 2007. W3C Recommendation. Disponível em http://www.w3.org/TR/xpath20. Acesso em janeiro de 2009.

BERNERS-LEE, T.; FIELDING, R.; MASINTER, L. **Uniform Resource Identifier (URI)**: Generic Syntax: IETF RFC 3986. IETF, 2005.

BIRON, P. V.; MALHOTRA, A. **XML Schema Part 2**: Datatypes Second Edition. 2004, W3C Recommendation. Disponível em http://www.w3.org/TR/xmlschema-2. Acesso em janeiro de 2009.

BLUMENTHAL, U.; WIJNEN, B. **User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMP)**: IETF RFC 3414. IETF, 2002.

BOX, D. et al. **Web Services Addressing (WS-Addressing)**. 2004. W3C Member Submission. Disponível em http://www.w3.org/Submission/ws-addressing. Acesso em janeiro de 2009.

BOX, D. et al. **Web Services Eventing (WS-Eventing)**. 2006. W3C Member Submission. Disponível em http://www.w3.org/Submission/WS-Eventing. Acesso em janeiro de 2009.

BRAY, T. et al. **Extensible Markup Language (XML) 1.1 (2nd ed.)**. 2006. W3C Recommendation. Disponível em http://www.w3.org/TR/xml11. Acesso em janeiro de 2009.

BRAY, T. et ak. **Namespaces in XML 1.1 (end ed.)**. 2006. W3C Recommendation. Disponível em http://www.w3.org/TR/xml-names11. Acesso em janeiro de 2009.

BULLARD, V.; VAMBENEPE, W. **Web Services Distributed Management**: Management Using Web Services (MUWS 1.1) Part 1. 2006. OASIS Standard. Disponível em http://docs.oasis-open.org/wsdm/wsdm-muws1-1.1-spec-os-01.htm. Acesso em janeiro de 2009.

CANTOR, S. et al. **Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0**. 2005. OASIS Standard. Disponível em http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf. Acesso em janeiro de 2009.

CASE, J. et al. **A Simple Network Management Protocol**: IETF RFC 1067. IETF, 1988.

CASE, J. et al. **A Simple Network Management Protocol (SNMP)**: IETF RFC 1098. IETF, 1989.

CASE, J. et al. **A Simple Network Management Protocol (SNMP)**: IETF RFC 1157. IETF, 1990.

CASE, J. et al. **Introduction to version 2 of the Internet-standard Network Management Framework**: IETF RFC 1441. IETF, 1993.

CHAPPELL, D.; LIU. L. **Web Services Brokered Notification 1.3 (WS-BrokeredNotification)**. 2006. OASIS Standard. Disponível em http://docs.oasis-open.org/wsn/wsn-ws_brokered_notification-1.3-spec-os.pdf. Acesso em janeiro de 2009.

CHINNICI, R. et al. **Web Services Description Language (WSDL) Version 2.0 Part 1**: Core Language. 2007, W3C Recommendation. Disponível em http://www.w3.org/TR/wsdl20. Acesso em janeiro de 2009.

CLINE, K. et al. **Toward Converging Web Service Standards for Resources, Events, and Management**. 2006. HP, IBM, Intel e Microsoft White Paper. Disponível em http://msdn.microsoft.com/en-us/library/aa480724.aspx. Acesso em janeiro de 2009.

COOPER, D. et al. **Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile**: IETF RFC 5280. IETF, 2008.

CURBERA, F. et ak. Unraveling the Web Services Web: An Introduction to SOAP, WSDL, and UDDI. **IEEE Internet Computing**, Los Alamitos, CA, v. 6, p. 2, p. 86-93, Mar/Apr 2002.

DAVIN, J. et al. **A Simple Gateway Monitoring Protocol**: IETF RFC 1028. IETF, 1987.

DAVIS, D. et al. **Web Services Reliable Messaging (WS-ReliableMessaging) Version 1.2**. 2008. OASIS Committee Specification 02. Disponível em http://docs.oasis-open.org/ws-rx/wsrm/v1.2/wsrm.html. Acesso em janeiro de 2009.

DIERKS, T.; RESCORLA, E. **The Transport Layer Security (TLS) Protocol Version 1.2**: IETF RFC 5246. IETF, 2008.

EASTLAKE, D.; REAGLE, J. **XML Encryption Syntax and Processing**. 2002. W3C Recommendation. Disponível em http://www.w3.org/TR/xmlenc-core. Acesso em janeiro de 2009.

EASTLAKE, D. et al, T. **XML Signature Syntax and Processing (2nd ed.)**. 2008. W3C Recommendation. Disponível em http://www.w3.org/TR/xmldsig-core. Acesso em janeiro de 2009.

FIOREZE, T.; GRANVILLE, L. Z.; ALMEIDA, M. J. B.; TAROUCO, L. M. R. **Comparing SNMP with Web Services in a Management by Delegation Environment**. In: IFIP/IEEE International Symposium on Integrated Network Management, 9., 2005, Nice, France. **Integrated Network Management IX**: managing new networked worlds. Piscataway: IEEE, 2005. 14p.

FORD, W. et al. **XML Key Management Specification (XKMS)**. 2001. W3C Note. Disponível em http://www.w3.org/TR/xkms. Acesso em janeiro de 2009.

FREIER, A.; KARLTON, P.; KOCHER, P. **The SSL Protocol Version 3.0**. 1996. IETF Internet Draft. Disponível em http://tools.ietf.org/html/draft-ietf-tls-ssl-version3-00. Acesso em janeiro de 2009.

GRAHAM, G. et al. **Web Services Notification (WS-Notification)**. 2004. IBM, Sonic, SAP, HP, Akamai e Tibco Specification draft. Disponível em http://www.ibm.com/developerworks/library/ws-resource/ws-notification.pdf. Acesso em janeiro de 2009.

GRAHAM, G.; TREADWELL, J. **Web Services Resource Properties 1.2 (WS-ResourceProperties)**. 2004. OASIS Working Draft 04. Disponível em http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-04.pdf. Acesso em janeiro de 2009.

GRAHAM, G. et al. **Web Services Resource 1.2 (WS-Resource)**. 2006. OASIS Standard. Disponível em http://docs.oasis-open.org/wsrf/wsrf-ws_resource-1.2-spec-os.pdf. Acesso em janeiro de 2009.

GRAHAM, G.; HULL, D.; MURRAY, B. **Web Services Base Notification 1.3 (WS-BaseNotification)**. 2006, OASIS Standard. Disponível em http://docs.oasis-open.org/wsn/wsn-ws_base_notification-1.3-spec-os.pdf. Acesso em janeiro de 2009.

GUDGIN, M. et al. **SOAP Version 1.2 Part 1**: Messaging Framework (2nd ed.). 2007. W3C Recommendation. Disponível em http://www.w3.org/TR/soap12-part1. Acesso em janeiro de 2009.

HARRINGTON, D.; PRESUHN, R.; WIJNEN, B. **An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks**: IETF RFC 3411. IETF, 2002.

LEVI, D.; SCHÖNWALDER, J. **Definitions of Managed Objects for the Delegation of Management Scripts**: IETF RFC 3165. IETF, 2001.

LIMA, W. Q.; ALVES, R. S.; VIANNA, R. L.; ALMEIDA, M. J. B.; TAROUCO, L. M. R.; GRANVILLE, L. Z. Evaluating the Performance of SNMP and Web Services Notifications. In: IFIP/IEEE Network Operations and Management Symposium, NOMS, 10., 2006, Vancouver, Canada. **Proceedings**, New York: IEEE, 2006. p. 546-556.

MAGUIRE, T.; SNELLING, D.; BANKS, T. **Web Services Service Group 1.2 (WS-ServiceGroup)**. 2006. OASIS Standard. Disponível em http://docs.oasis-open.org/wsrf/wsrf-ws_service_group-1.2-spec-os.pdf. Acesso em janeiro de 2009.

MCCLOGHRIE, K.; ROSE, M. **Management Information Base for Network Management of TCP/IP-based internets**: MIB-II: IETF RFC 1213. IETF, 1991.

MCCLOGHRIE, K.; PERKINS, D.; SCHÖNWALDER, J. **Textual Conventions for SMIv2**: IETF RFC 2579. IETF, 1999.

MCMANUS, E. **JSR-262 and WS-Management**. Disponível em http://weblogs.java.net/blog/emcmanus/archive/2006/11/jsr262_and_wsma.html. Acesso em janeiro de 2009.

NADALIN, A. et al. **Web Services Security**: SOAP Message Security 1.1. 2006. OASIS Standard. Disponível em http://docs.oasis-open.org/wss/v1.1/wss-v1.1-errata-os-SOAPMessageSecurity.pdf. Acesso em janeiro de 2009.

NADALIN, A. et al. **WS-SecurityPolicy 1.2**. 2006. OASIS Committee Draft 01. Disponível em http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200512/ws-securitypolicy-1.2-spec-cd-01.pdf. Acesso em janeiro de 2009.

NADALIN, A. et al. **WS-Trust 1.3**. 2007. OASIS Standard. Disponível em http://docs.oasis-open.org/ws-sx/ws-trust/v1.3/ws-trust.html. Acesso em janeiro de 2009.

NADALIN, A. et al. **WS-SecureConversation 1.4**. 2008. OASIS Committee Specification 01. Disponível em http://docs.oasis-open.org/ws-sx/ws-secureconversation/v1.4/ws-secureconversation.html. Acesso em janeiro de 2009.

NEISSE, R.; VIANNA, R. L.; GRANVILLE, L. Z.; ALMEIDA, M. J. B.; TAROUCO, L. M. R. Implementation and Bandwidth Consumption Evaluation of SNMP and Web Services Gateways. In: IEEE/IFIP NOMS Network Operations and Management Symposium, NOMS, 9., 2004, Seoul, Korea. **Proceedings**, New York: IEEE, 2004. p. 715-728.

NEUMAN, C. et al. **The Kerberos Network Authentication Service (V5)**: IETF RFC 4120. IETF, 2005.

NIELSEN, H. F.; RUELLAN, H. **SOAP 1.2 Attachment Feature**. 2004. W3C Working Group Note. Disponível em http://www.w3.org/TR/soap12-af. Acesso em janeiro de 2009.

PRAS. A. et al. Comparing the Performance of SNMP and Web Services-Based Management. **IEEE Transactions on Network and Service Management**, v. 1, n. 2, p. 1(2)-11, Dec 2004.

RISSANEN, E. **eXtensible Access Control Markup Language (XACML) Version 2.0**. 2008. OASIS Standard. Disponível em http://www.oasis-open.org/committees/download.php/26986/access_control-xacml-2.0-core-spec-os-errata.doc. Acesso em janeiro de 2009.

SCHÖNWALDER, J. et al. **SNMP Traffic Analysis: Approaches, Tools, and First Result**. International Symposium on Integrated Network Management Proceedings, p. 324-332, May 2007.

SIERRA, K.; BATES, B. **Head First Java**. 2nd ed. Sebastopol, USA: O'Reilly.

SNMP4J. **The SNMP API for Java**. Disponível em http://www.snmp4j.org. Acesso em janeiro de 2009.

SOURCEFORGE. **Net-SNMP**. Disponível em http://www.net-snmp.org. Acesso em janeiro de 2009.

SOURCEFORGE. **Sun's XACML Implementation**. Disponível em http://sunxacml.sourceforge.net. Acesso em janeiro de 2009.

SRINIVASAN, L.; BANKS, T. **Web Services Resource Lifetime 1.2 (WS-ResourceLifetime)**. 2006. OASIS Standard. Disponível em http://docs.oasis-open.org/wsrf/wsrf-ws_resource_lifetime-1.2-spec-os.pdf. Acesso em janeiro de 2009.

STALLINGS, W. **SNMP, SNMPv2, SNMPv3, and RMON 1 and 2**. 3rd ed. Reading, USA: Addison Wesley, 1999.

STALLINGS, W. **Cryptography and Network Security**: Principles and Practices. 4th ed. Upper Saddle River, USA: Prentice Hall, 2005.

SUN MICROSYSTEMS. **Web Services Connector for Java Management Extensions (JMX) Agents**. 2008. JSR262 Specification Public Review. Disponível via http://jcp.org/en/jsr/detail?id=262. Acesso em janeiro de 2009.

SUN MICROSYSTEMS. **Wiseman: A Java implementation of WS-Management**. Disponível em https://wiseman.dev.java.net. Acesso em janeiro de 2009.

THOMPSON, H. S. et al. **XML Schema Part 1**: Structures Second Edition. 2004. W3C Recommendation. Disponível em http://www.w3.org/TR/xmlschema-1. Acesso em janeiro de 2009.

TUECKE, S.; LIU, L.; MEDER, S. **Web Services Base Faults 1.2 (WS-BaseFaults)**. 2004. OASIS Working Draft. Disponível em http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-BaseFaults-1.2-draft-02.pdf. Acesso em janeiro de 2009.

VAMBENEPE, W.; GRAHAM, S.; NIBLETT, P. **Web Services Topics 1.3 (WS-Topics)**. 2006. OASIS Standard. Disponível em http://docs.oasis-open.org/wsn/wsn-ws_topics-1.3-spec-os.pdf. Acesso em janeiro de 2009.

VEDAMUTHU, A. S. et al. **Web Services Policy 1.5 – Framework**. 2007. W3C Recommendation. Disponível em http://www.w3.org/TR/ws-policy. Acesso em janeiro de 2009.

WALDBUSSER, S. **Remote Network Monitoring Management Information Base**: IETF RFC 2819. IETF, 2000.

WALDBUSSER, S.; GRILLO, P. **Host Resources MIB**: IETF RFC 2790. IETF, 2000.

WIJNEN, B.; PRESUHN, R.; MCCLOGHRIE, K. **View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP)**: IETF RFC 3415. IETF, 2002.

WILSON, K.; SEDUKHIN, I. **Web Services Distributed Management**: Management of Web Services (WSDM-MOWS) 1.1. 2006. OASIS Standard. Disponível em http://www.oasis-open.org/committees/download.php/20574/wsdm-mows-1.1-spec-os-01.pdf. Acesso em janeiro de 2009.

XRML. **eXtensible rights Markup Language (XrML) 2.0 Specification Part I**: Primer. 2001. Disponível em http://www.eduworks.com/Documents/Workshops/EdMedia2003/Docs/XrML2_0/xrml2part1.pdf. Acesso em janeiro de 2009.