Um serviço de autorização Java EE baseado em certificados de atributos X.509

Stefan Neusatz Guilhen

DISSERTAÇÃO APRESENTADA
AO
INSTITUTO DE MATEMÁTICA E ESTATÍSTICA
DA
UNIVERSIDADE DE SÃO PAULO
PARA
OBTENÇÃO DO TÍTULO
DE
MESTRE EM CIÊNCIAS

Programa: **Ciência da Computação**
Orientador: **Prof. Dr. Francisco Carlos da Rocha Reverbel**

São Paulo, agosto de 2008

Um serviço de autorização Java EE baseado em certificados de atributos X.509

Este exemplar corresponde à redação final da
dissertação devidamente corrigida e defendida
por Stefan Neusatz Guilhen
e aprovada pela Comissão Julgadora.

Banca Examinadora :

Prof. Dr. Francisco Carlos da Rocha Reverbel (orientador) – IME-USP

Prof. Dr. Fabio Kon – IME-USP

Prof. Dr. Paulo Sérgio Licciardi Messeder Barreto – Poli-USP

Agradecimentos

Primeiramente, gostaria de agradecer à minha família, pelo apoio e incentivo que me foi dado quando decidi me dedicar a este projeto. Da mesma forma, agradeço à Flavia, cuja presença e apoio foram fundamentais para a conclusão deste trabalho. Sua visão crítica e sugestões contribuíram significativamente para o desenvolvimento do projeto, e seu companheirismo e dedicação me deram a segurança para seguir em frente.

Expresso aqui também a minha gratidão ao meu orientador, o professor Francisco Carlos da Rocha Reverbel, pela confiança que depositou em mim e pelo grande aprendizado que me proporcionou durante todo o curso deste projeto.

Agradeço também aos professores Fabio Kon e Paulo Barreto por terem participado na banca de minha qualificação de mestrado. Suas opiniões e sugestões foram valiosas e contribuíram muito para o aprimoramento deste trabalho.

Finalmente, agradeço ao grupo JBoss, pelo apoio financeiro que possibilitou que eu me dedicasse exclusivamente ao desenvolvimento deste trabalho.

Resumo

O surgimento e a popularização de arquiteturas de software que fornecem suporte à programação distribuída orientada a objetos, como CORBA, .NET e Java EE, gerou uma demanda por infra-estruturas de segurança eficientes, capazes de proteger os recursos dos sistemas de ataques maliciosos. Essa proteção começa pela identificação dos usuários que interagem com os sistemas, processo conhecido como autenticação. Entretanto, a autenticação por si só não é suficiente para garantir a segurança dos recursos, uma vez que a autenticação não determina quais ações os usuários estão autorizados a executar depois de autenticados. Em outras palavras, um mecanismo de autorização, que faz valer as políticas de controle de acesso aos recursos definidas pelos administradores de sistemas, se faz necessário.

Neste trabalho estudamos mecanismos de controle de acesso baseado em papéis e a aplicabilidade dos certificados de atributos X.509 como estrutura de armazenamento desses papéis em um ambiente Java EE. Em particular, estendemos a infra-estrutura de segurança do servidor de aplicações JBoss, de modo que ela passasse a comportar os certificados de atributos X.509. Além disso, analisamos as vantagens e desvantagens do uso de tais certificados e avaliamos o desempenho da extensão desenvolvida em relação a outras alternativas que são oferecidas pelo JBoss para o armazenamento de papéis dos usuários.

Abstract

The popularization of software architectures that provide support for distributed object-oriented programming, like CORBA, .NET, and Java EE, revealed the need for efficient security infrastructures to protect the resources of enterprise systems from malicious attacks. This protection usually begins with the identification of the users that interact with the systems, a process known as authentication. However, authentication alone is not enough to guarantee the protection of the resources, as it cannot determine what actions a particular user is allowed to execute on a given resource. In other words, an authorization mechanism is needed in order to enforce the access control policies as defined by the system administrators.

In this dissertation we studied role-based access control mechanisms and the use of X.509 attribute certificates as data structures that store the users' roles in a Java EE environment. Particularly, we added X.509 attribute certificates support to the existing JBoss application server security infrastructure. Furthermore, we evaluated the pros and cons of using these certificates, and compared the performance of the developed extension to the performance of the existing solutions provided by JBoss to store the user's roles.

Sumário

Lista de Abreviaturas

CA X.509	Certificado de atributos X.509
CCP X.509	Certificado de chave pública X.509
DTD	*Document Type Definition*
EJB	*Enterprise Java Beans*
HTTP	*Hypertext Transfer Protocol*
ICP	Infra-estrutura de chaves públicas
IGP	Infra-estrutura de gerenciamento de privilégios
Java EE	*Java Platform, Enterprise Edition*
JAAS	*Java Authentication and Authorization Service*
JCA	*Java Connector Architecture*
JDBC	*Java Database Connectivity*
JMS	*Java Message Service*
JSF	*Java Server Faces*
JSP	*Java Server Pages*
JSSE	*Java Secure Sockets Extension*
LDAP	*Lightweight Directory Access Protocol*
LCR	Lista de certificados revogados
OCSP	*Online Certificate Status Protocol*
RBAC	*Role-Based Access Control*
SSL	*Secure Sockets Layer*
TLS	*Transport Layer Security*
URL	*Uniform Resource Locator*
XACML	*eXtensible Access Control Markup Language*
XML	*eXtensible Markup Language*

Lista de Figuras

1 Introdução

A rápida adoção de plataformas para desenvolvimento de aplicações distribuídas, como CORBA [38], .NET [8] e Java EE [56], trouxe consigo uma série de novas ameaças e riscos, tais como interrupções de serviço, acesso não autorizado a informações, interceptação e manipulação de dados, personificação, distribuição de vírus, entre outros. Estes riscos elevam a importância que infra-estruturas de segurança têm para que organizações e indivíduos protejam suas informações e processos de ataques maliciosos e acessos indevidos. A fim de proporcionar uma proteção adequada, infra-estruturas de segurança devem prover no mínimo os seguintes serviços:

- identificação: usuários que interagem com um sistema devem possuir uma identidade reconhecida pelo mesmo. As formas mais comuns de representação da identidade de um usuário incluem seu *username*, seu *e-mail*, ou qualquer outro atributo que o identifique unicamente, como seu CPF.

- autenticação: quando um usuário alega ter uma identidade particular é necessário que ele apresente credenciais que possam comprovar essa identidade. Freqüentemente essas credenciais estão na forma de uma senha. Entretanto, existem diversas outras possibilidades para credenciais, como certificados digitais, *smart cards* e biometria.

- controle de acesso: todo sistema seguro deve controlar o acesso a seus recursos. Quando um usuário tenta fazer acesso a um recurso (arquivos, componentes, base de dados, etc) o serviço de segurança deve verificar se o usuário tem ou não permissões suficientes para realizar tal acesso.

- proteção dos dados: a proteção dos dados envolve desde a proteção do meio físico onde os dados são armazenados até o uso de criptografia para garantir a integridade e confidencialidade das informações, tanto em seu armazenamento quanto durante o seu tráfego pela rede.

Para prover os serviços acima, a plataforma Java EE estende a infra-estrutura de segurança da plataforma Java e oferece diversas alternativas para identificar e autenticar os usuários. A

proteção dos dados em transporte é comportada através do uso de protocolos de comunicação seguros, implementados pela *Java Secure Socket Extension* (JSSE) [48]. Já o controle de acesso, ou autorização, é feito de acordo com os papéis associados ao usuário. Cada componente especifica quais papéis têm permissão para acessar seus métodos e cada usuário do sistema é então associado a um conjunto de papéis que irá definir o escopo de seus privilégios.

Embora estabeleçam que o controle de acesso deve ser baseado nos papéis do usuário, as especificações da plataforma Java EE não determinam como o mapeamento entre os papéis definidos pela aplicação e os usuários do mundo real deve ser feito. Dessa forma, os servidores de aplicação são livres para implementar esse mapeamento da maneira que for mais conveniente. De maneira geral, os principais servidores existentes [2,3,21,29] fornecem diversas alternativas, que vão desde o uso de arquivos de propriedades até o uso de serviços de diretório ou bancos de dados para armazenar as identidades dos usuários e seus respectivos papéis. Entretanto, todas as alternativas oferecidas atualmente pelos servidores de aplicação baseiam-se no conceito de que cabe ao próprio servidor localizar os papéis dos usuários durante o processo de autorização. Esse modelo de gerenciamento de privilégios é pouco flexível e obriga as entidades responsáveis por tal gerenciamento a manter um repositório que possa ser consultado pelo servidor.

O padrão X.509 [24] especifica, a partir de sua quarta edição, a infra-estrutura de gerenciamento de privilégios (*privilege management infrastructure*), ou IGP [11]. Essa infra-estrutura tem como elemento central o certificado de atributos X.509 (*X.509 attribute certificate*), ou CA X.509, que associa um conjunto de atributos a uma identidade. O gerenciamento de privilégios nessa infra-estrutura fica a cargo da autoridade de atributos (*attribute authority*), que é responsável por gerar e assinar digitalmente os certificados de atributos. O modelo de gerenciamento de privilégios definido pela IGP é bastante flexível, e a autoridade de atributos tem a opção de escolher a forma de divulgação dos CAs X.509 gerados, podendo tanto fornecer um certificado diretamente para o seu titular quanto simplesmente armazenar esse certificado em um repositório público.

Neste trabalho estudamos mecanismos de controle de acesso baseado em papéis e a aplicabilidade dos certificados de atributos X.509 como estrutura de armazenamento desses papéis em um ambiente Java EE. Em particular, estendemos a infra-estrutura de segurança do servidor de aplicações JBoss [13], de modo que ela passasse a comportar a IGP e os certificados de atributos. Além disso, analisamos as vantagens e desvantagens do uso de tais certificados, e avaliamos o desempenho da extensão desenvolvida em relação a outras alternativas oferecidas pelo JBoss para o armazenamento de papéis dos usuários.

Até onde pudemos averiguar, nossa versão estendida da infra-estrutura de segurança do servidor de aplicações JBoss faz desse servidor o único a oferecer um modelo de gerenciamento de

privilégios flexível através da IGP e dos certificados de atributos X.509. Além da flexibilidade, o uso dos CAs X.509 traz outros benefícios para a plataforma Java EE:

- garantia de integridade dos papéis. Os CAs X.509 são documentos assinados digitalmente pelas autoridades de atributos. A assinatura digital assegura a integridade do conteúdo do certificado e dá garantias ao serviço de autorização do servidor de aplicações de que os privilégios contidos no certificado foram emitidos por uma autoridade de confiança, o que aumenta o grau de segurança oferecido pela infra-estrutura como um todo.
- favorecimento à distribuição do gerenciamento de privilégios. Usuários interagem com diversos sistemas, em contextos diferentes. É pouco provável que uma única autoridade de atributos seja capaz de gerenciar todos os privilégios de um usuário em todos os contextos possíveis. Ao invés disso, a tendência é de que existam diversas autoridades, cada uma gerenciando informações de um usuário referentes a um contexto específico. O uso de certificados de atributos favorece a integração do serviço de autorização com as diferentes autoridades, uma vez que todas essas autoridades usarão uma mesma estrutura para armazenar os privilégios dos usuários.

1.1 Organização do texto

Esta dissertação está organizada da seguinte forma: o Capítulo 2 descreve trabalhos relacionados, o Capítulo 3 apresenta uma visão geral de segurança na plataforma Java EE, o Capítulo 4 descreve com mais detalhes a IGP e os CAs X.509, o Capítulo 5 expõe o projeto desenvolvido como parte do nosso estudo, o Capítulo 6 contém uma avaliação de desempenho da implementação e, finalmente, o Capítulo 7 traz nossas conclusões.

2 Trabalhos relacionados

Este Capítulo apresenta os trabalhos relacionados que foram analisados e que contribuíram para o nosso estudo. Primeiramente apresentamos e avaliamos três projetos que, assim como nosso trabalho, também se valem de algum tipo de certificado para armazenar os atributos dos usuários. Em seguida apresentamos brevemente o serviço de segurança de CORBA, cuja arquitetura baseada em interceptadores influenciou significativamente a arquitetura da infra-estrutura de segurança do JBoss. Por fim, fazemos uma análise comparativa entre os projetos estudados e o projeto desenvolvido neste trabalho.

2.1 Akenti Authorisation Infrastructure

O Akenti [1, 30] é uma infra-estrutura de autorização desenvolvida no *Lawrence Berkeley National Laboratory*, na Califórnia. O principal objetivo do projeto é fornecer uma maneira de expressar e fazer valer uma política de controle de acesso de forma distribuída, evitando os problemas que aparecem quando uma única entidade é responsável por gerenciar e garantir os requisitos de controle de acesso.

Tradicionalmente, o controle de acesso aos recursos vinha sendo feito através do uso de listas de controle de acesso. Essas listas são gerenciadas e controladas por uma única pessoa, que é responsável pela manutenção da política de controle adotada. O problema é que nem sempre é adequado centralizar esse tipo de gerenciamento, pois é possível que mais de uma pessoa tenha autoridade para decidir como o acesso ao recurso deve ser feito. Por exemplo, um equipamento que pode causar danos ambientais se mal operado pode ter as seguintes pessoas envolvidas na definição da política de controle de acesso: o líder do projeto, que utilizará o equipamento e precisa que toda sua equipe tenha acesso total; o comitê de impacto ambiental, que tem que aprovar o projeto que será desenvolvido; uma agencia governamental, que pode ter restrições especiais quanto ao uso do equipamento.

Nesse caso, o uso de listas de controle de acesso é totalmente inadequado, pois o caráter centralizado desse tipo de mecanismo impede que mudanças feitas por uma das pessoas envolvidas possam ser aplicadas imediatamente. Uma requisição tem que ser feita para o controle central, que tem que verificar se a requisição foi originada por uma parte autorizada, checar

se a requisição não foi alterada durante o envio e só então aplicar as mudanças apropriadas. Isso compromete a escalabilidade, principalmente nos casos em que as partes envolvidas na definição da política de controle de acesso estão distribuídas geograficamente.

O Akenti foi desenvolvido para lidar com essa situação, promovendo o gerenciamento distribuído da política de acesso aos recursos. Nele, uma parte envolvida na definição da política pode gerenciar os seus requisitos de controle de acesso independentemente de outras partes. Além disso, cada parte pode modificar seus requisitos a qualquer momento e pode ter certeza de que as mudanças irão entrar em vigor imediatamente. O Akenti realiza decisões de autorização baseado em um conjunto de documentos digitalmente assinados que possuem as instruções de autorização. A autenticação dos usuários é feita exclusivamente através de criptografia de chave pública [59].

2.1.1 Arquitetura e componentes

A Figura 2.1 ilustra a arquitetura e os principais componentes do Akenti.

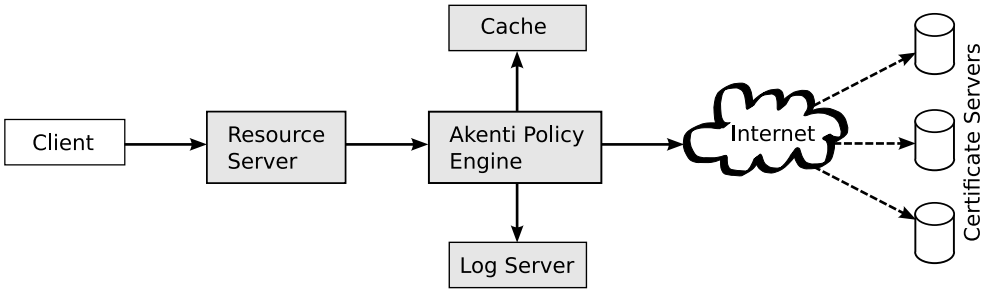


Figura 2.1: Arquitetura do projeto Akenti

No Akenti, os recursos a serem protegidos são configurados como *Resource Servers*. Clientes se autenticam aos *Resource Servers* utilizando certificados de chave pública X.509. As requisições são delegadas para o *Policy Engine*, que decide se o acesso ao recurso deve ou não ser concedido, de acordo com as políticas de controle de acesso definidas para o recurso e com os privilégios do cliente. Tanto as políticas quanto o conjunto de privilégios do cliente são armazenadas em certificados digitais que se encontram distribuídos em diversos *Certificate Servers*. Todas as ações do *Policy Engine* são registradas no *Log Server* para fins de auditoria. Um *cache* pode ser opcionalmente utilizado para evitar que o *Policy Engine* tenha que procurar sempre pelas credenciais de um cliente que já fez requisições anteriormente.

Para implementar o gerenciamento distribuído de políticas de controle de acesso, o Akenti faz uso de um conjunto de certificados digitais, todos definidos pelo próprio Akenti:

- O *Akenti Attribute Certificate*, assim como o definido pelo padrão X.509, é utilizado para armazenar atributos referentes ao seu titular. Um diferencial é que o Akenti permite que sejam especificadas condições para que um atributo seja considerado válido. Por exemplo, pode-se definir que um dado atributo só possa ser considerado se o certificado for apresentado dentro de um determinado horário. Esses certificados são gerados pelas autoridades de atributo utilizando o *Attribute Generator*.

- O *Use-Condition Certificate* é utilizado para definir as condições necessárias para que o acesso a um determinado recurso seja liberado. Há uma grande flexibilidade para se definir as condições de uso, podendo-se exigir que o cliente possua determinado papel, pertença a determinado grupo ou até mesmo origine a chamada a partir de um certo IP. Além das condições, esse certificado define os tipos de ações que podem ser executadas sobre o recurso em questão. São gerados por partes que possuem autoridade para definir condições de uso de um recurso através do *Use-Condition Generator*.

- O *Policy Certificate* é utilizado para armazenar as configurações gerais do *Police Engine*, como, por exemplo, a lista das autoridades certificadoras confiáveis, as URLs dos *Certificate Servers* que contém os certificados de atributos, o tempo de validade das informações no *cache*, entre outras coisas.

Depois de gerados, os certificados de atributos e de condição de uso são distribuídos para os *Certificate Servers*, que podem utilizar desde o sistema de arquivos até serviços de diretório para armazenar esses certificados. Já o *Policy Certificate* é armazenado junto ao *Policy Engine*, que o utiliza para obter as configurações gerais do serviço.

2.1.2 Avaliação crítica

A principal vantagem em se utilizar o Akenti é flexibilidade na escolha do tipo de autorização a ser empregada para proteção dos recursos. Esta pode ser feita de acordo com os papéis do usuário no ambiente da aplicação, com os grupos aos quais o usuário pertence, ou com qualquer outro atributo que possa ser associado a ele. Combinações entre os diversos tipos de atributos também são possíveis (por exemplo, acesso a um recurso só é concedido se o usuário possui o papel *admin* e pertence ao grupo *funcionários*).

Como desvantagens, podemos citar que a única forma de autenticação atualmente comportada pelo projeto é através de certificados de chave pública, dificultando sua utilização em ambientes que aceitam outras formas de autenticação, como usuário e senha, biometria e *smart cards*. Além disso, todos os certificados gerados pelo Akenti, incluindo os certificados

de atributos, são proprietários e não aderem a nenhum padrão. Isso compromete a interoperabilidade pois somente certificados emitidos pelos geradores do Akenti podem ser usados pelo *Policy Engine*. Por fim, o Akenti também é restrito quanto ao modelo de divulgação de privilégios, uma vez que obriga o *Attribute Generator* a manter os certificados gerados em repositórios (*Certificate Servers*).

2.2 PERMIS

O PERMIS (*PrivilEge and Role Management Infraestructure Standards*) [6,42] é uma infraestrutura de autorização baseada nos certificados de atributos X.509 e desenvolvida no Instituto de Segurança da Informação da Universidade de Salford, na Inglaterra.

O PERMIS realiza o controle de acesso dos usuários com base nos papéis dos mesmos. Todos os dados necessários para decisões de autorização, como a especificação dos papéis e os privilégios alocados a cada papel, são descritos por uma política de autorização [5], que é por sua vez armazenada em um CA X.509 para garantir sua integridade. O projeto adotou a XML como linguagem para descrição dessa política porque XML possuía um extenso conjunto de ferramentas de suporte e vinha se consolidado como um padrão na indústria. Algum tempo depois de o PERMIS ter definido o seu DTD, o consórcio *Oasis* [36] iniciou um trabalho para definir a *Extensible Access Control Markup Language*, ou XACML [40], que é uma tentativa de se padronizar uma linguagem para descrever os vários aspectos relacionados ao processo de autorização, como, por exemplo, as entidades envolvidas, os recursos a serem protegidos e as regras que formam a política de acesso. Apesar do PERMIS não aderir à XACML, muitas similaridades existem entre os dois formatos de descrição da política de controle de acesso aos recursos.

Tanto o CA X.509 que contém a política de acesso quanto os CAs X.509 que contém os papéis associados a cada usuário são armazenados em serviços de diretório LDAP distribuídos, sendo esta a única opção que as autoridades de atributos têm para gerenciar os certificados criados.

2.2.1 Arquitetura

A arquitetura do PERMIS é composta por um subsistema de alocação de privilégios, que é responsável por alocar privilégios aos usuários, e um subsistema de verificação de privilégios, que é responsável por autenticar e autorizar usuários. A figura a seguir ilustra o subsistema de alocação de privilégios.

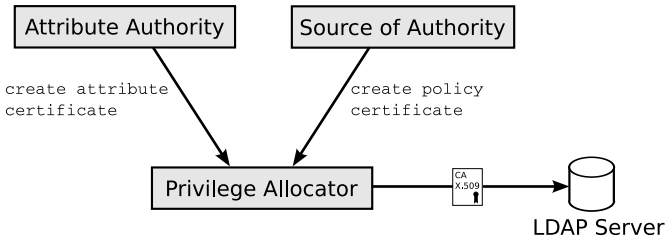


Figura 2.2: Subsistema de alocação de privilégios

Esse subsistema é formado pelo *Privilege Allocator*, componente responsável por gerar e assinar os certificados de atributos X.509 contendo os papéis dos usuários do sistema. Esse mesmo componente é utilizado pela autoridade responsável pela definição da política de autorização, a *Source of Authority* ou SOA, para gerar o certificado que armazena essa política como um atributo. Esse último é um certificado de atributos X.509 padrão com duas características especiais: é um certificado auto-assinado, ou seja, a SOA é ao mesmo tempo a emissora e a titular do certificado, e possui um único atributo, do tipo `pmiXMLPolicy`, cujo valor é o arquivo XML que define a política em questão.

Uma vez gerados, os certificados devem ser armazenados em serviços de diretório LDAP, normalmente acessíveis publicamente para consulta. Os certificados dos usuários devem ser publicados no contexto LDAP de cada usuário, enquanto que o certificado que contém a política de autorização deve ser publicado no contexto da SOA. Como o certificado de atributos é um documento assinado digitalmente, seu conteúdo não pode ser modificado sem violar a assinatura. Por isso, não existe necessidade de se proteger a camada de transporte ao se consultar o serviço de diretório.

O subsistema de verificação de privilégios autentica e autoriza usuários, verificando suas permissões de acesso. A funcionalidade desse subsistema foi dividida em duas partes: a *Access-Control Enforcement Function* ou AEF, responsável pela autenticação dos usuários, e a *Access-Control Decision Function* ou ADF, responsável por aplicar a política de autorização vigente. A comunicação entre as duas partes é feita através da API PERMIS, uma API baseada na API AZN [61] definida pelo *Open Group*. A Figura 2.3 ilustra o subsistema de verificação de privilégios.

A AEF permite que cada aplicação defina a mecanismo de autenticação mais apropriado, independentemente da autorização. Já a ADF é independente de aplicação e irá aplicar o controle de acesso de acordo com a política de autorização definida. A autenticação e a autorização de um usuário ocorrem da seguinte maneira:

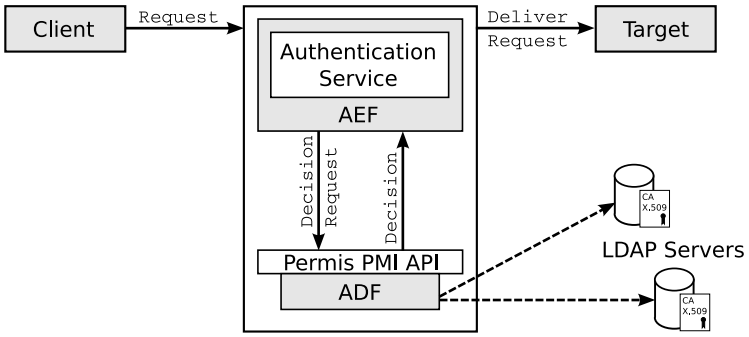


Figura 2.3: Subsistema de verificação de privilégios

- primeiro, AEF autentica o usuário de acordo com o mecanismo de autenticação configurado pela aplicação. Em caso de sucesso, ela invoca o método `GetCreds` da API PERMIS para obter da ADF o conjunto de papéis do usuário autenticado. A ADF localiza os CAs X.509 do usuário, descarta os certificados inválidos, extrai os papéis contidos nos certificados válidos e os devolve para a AEF na forma de um objeto `subject`.

- após a autenticação, a cada vez que o usuário tentar acessar um recurso, a AEF irá invocar o método `Decision` da API, passando o `Subject`, o nome do recurso e ação que o usuário pretende executar no recurso. De posse dessas informações a ADF verifica se os papéis do usuário permitem a ação seja executada e devolve a resposta à AEF.

Além desses métodos, a API fornece ainda o método `Shutdown`, que é utilizado pela AEF para indicar à ADF que a política de autorização em vigor deve ser descartada. Isso pode ocorrer porque a SOA deseja estabelecer uma nova política dinamicamente. Nesse caso, a política mais nova é carregada pela ADF e já entra em vigor imediatamente.

2.2.2 Avaliação crítica

O PERMIS apresenta uma arquitetura sólida e maleável, que separa o processo de autenticação do processo de autorização, oferecendo uma grande flexibilidade na escolha do mecanismo de autenticação mais adequado a cada aplicação. Ao contrário do Akenti, o PERMIS se utiliza de certificados de atributos X.509 padronizados, o que promove a interoperabilidade com autoridades de atributos que também aderem ao padrão. Porém, essa interoperabilidade é prejudicada pelo fato do PERMIS utilizar um atributo proprietário para o armazenamento dos papéis dos usuários dentro dos certificados, ao invés de utilizar o atributo definido pelo padrão X.509 para esse propósito.

Outro aspecto negativo fica por conta da escolha de um único modelo de divulgação de privilégios, que impede as autoridades de atributos de fornecer os certificados diretamente para os seus titulares e obrigando-as a manter repositórios que armazenem esses certificados e estejam acessíveis publicamente para consulta. Um terceiro ponto a ser ressaltado é que o *Privilege Allocator* faz uso de software proprietário para geração dos CAs X.509, o que dificulta o uso de tal ferramenta em ambientes de código aberto como o JBoss.

2.3 SESAME

O SESAME (*Secure European System for Application in a Multi-vendor Environment*) [31, 41] foi um projeto desenvolvido na Europa no final dos anos 80 com o objetivo de desenvolver uma infra-estrutura de segurança para aplicações distribuídas. No SESAME, a autenticação dos usuários é pode ser feita tanto através do protocolo *Kerberos* [35] como através de criptografia de chave pública. A autorização é realizada de acordo com os papéis dos usuários, que são por sua vez armazenados em certificados digitais, chamados de *Privilege Attribute Certificates* (PAC). Tais certificados são na verdade uma uma forma específica de certificado de controle de acesso conforme definido pela *Access Control Framework* do padrão *ISO/ITU-T* [23].

A arquitetura do SESAME é constituída por dois componentes principais: o *Authentication Service*, ou AS, e o *Privilege Attribute Server*, ou PAS. O *Authentication Service* é responsável pela autenticação dos usuários. Já o PAS é o responsável pelo gerenciamento de privilégios dos usuários e pela geração e distribuição dos *Privilege Attribute Certificates*.

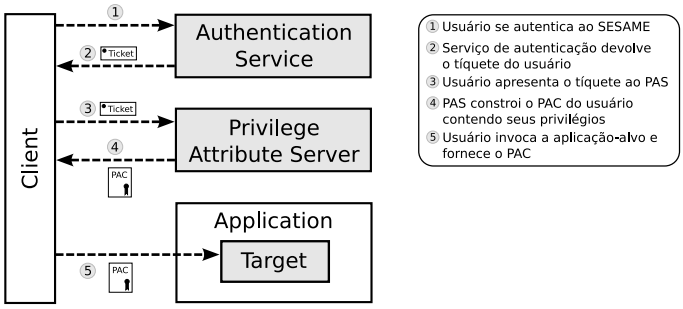


Figura 2.4: Interação entre um cliente e os componentes do SESAME

Uma vez autenticado com sucesso, um usuário obtém do AS um tíquete que comprova sua autenticação. O usuário apresenta então esse tíquete ao PAS para obter uma prova dos seus privilégios na forma de um PAC. Uma vez obtido o PAC, o usuário faz uma requisição a um recurso do sistema alvo e apresenta o seu PAC juntamente com os outros dados da requisição.

O sistema-alvo avalia a política de controle de acesso dos recursos e também as informações contidas no PAC para decidir se o usuário deve ou não ter acesso ao recurso pretendido.

2.3.1 Avaliação

Ao contrário dos dois projetos citados anteriormente, o SESAME não baseia seu modelo de divulgação de privilégios em repositórios disponíveis para consulta. Ao invés disso, a autoridade gerenciadora (PAS) emite os certificados diretamente para os usuários do sistema, e estes são responsáveis por apresentar tais certificados para o serviço de autorização do sistema-alvo. Dessa forma, o mecanismo de autorização do sistema-alvo não necessita de nenhuma consulta a repositórios para obtenção do PAC de um usuário. Entretanto, isso não faz do SESAME um projeto mais flexível, já que o projeto também oferece um único modelo para divulgação de privilégios.

Assim como o Akenti, o SESAME não é muito flexível quanto ao mecanismo de autenticação utilizado. Aplicações que desejam usar o SESAME devem possuir uma infra-estrutura preparada para autenticação utilizando o protocolo *Kerberos* ou se utilizar de mecanismos de autenticação baseados em chaves públicas, uma restrição que nem sempre pode ser satisfeita pelas aplicações.

2.4 CORBA Security

A especificação *CORBA Security Service*, ou CORBAsec [4, 37], foi criada com o objetivo de definir uma arquitetura de segurança para proteção de objetos CORBA de maneira não intrusiva. Os serviços especificados pela CORBAsec incluem autenticação, autorização, auditoria e proteção de mensagens. É importante ressaltar que a especificação apenas define uma arquitetura padronizada de segurança em termos de interfaces bem definidas. Os mecanismos concretos de segurança, ou implementações, são providos separadamente por fornecedores independentes de tecnologias CORBA.

O aspecto não intrusivo da CORBAsec fica por conta dos interceptadores de segurança, que são responsáveis por aplicar as regras de segurança em cada invocação feita para um objeto CORBA. Como esses interceptadores são configurados externamente à aplicação, os objetos CORBA podem ser desenvolvidos independentemente dos mecanismos de segurança que serão utilizados para proteger o acesso a esses objetos, em uma clara separação de responsabilidades. A Figura 2.5 exibe os dois interceptadores definidos por CORBAsec: o *AccessControlInterceptor*, que é utilizado para fazer valer as regras de controle de acesso tanto na aplicação cliente como no servidor, e o *SecureInvocationInterceptor*, que é responsável por estabelecer

uma conexão segura entre a aplicação cliente e o servidor.

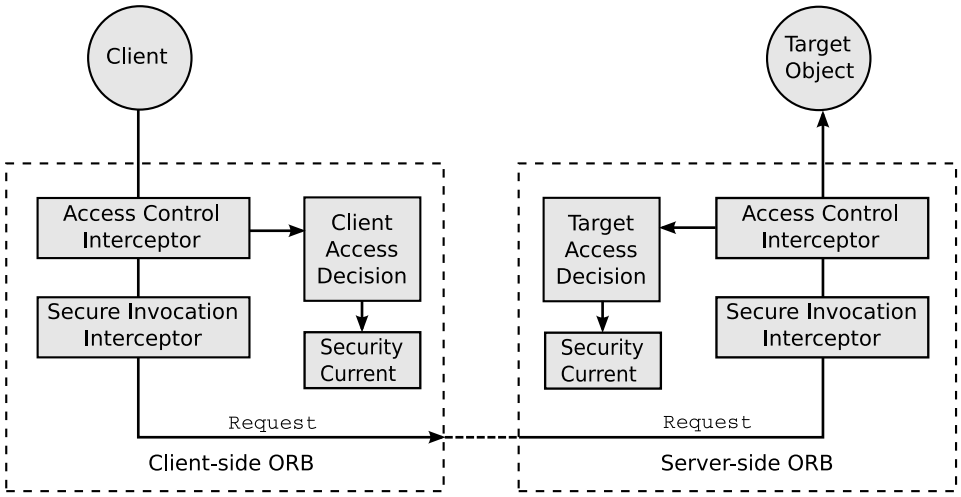


Figura 2.5: Interceptadores de segurança de CORBA

A autenticação fica por conta da interface *PrincipalAuthenticator*, que atua como uma abstração do mecanismo de autenticação sendo utilizado. Como resultado do processo de autenticação o *PrincipalAuthenticator* cria um objeto do tipo *Credentials*, que contém os atributos que descrevem os privilégios do usuário autenticado, e associa esse objeto ao *SecurityCurrent*. Os atributos se dividem em dois grupos: *(i)* atributos de identidade, que contém informações adicionais a respeito da identidade do usuário, e *(ii)* atributos de privilégios, que descrevem as permissões atribuídas ao usuário.

O *SecurityCurrent* é utilizado para manter as informações de segurança associadas ao contexto de execução das requisições feitas pelo cliente. As funções de controle de acesso do cliente e do servidor consultam o *SecurityCurrent* para obter os atributos que foram inseridos no objeto *Credentials* e utilizam os atributos obtidos para decidir se o acesso ao objeto-alvo deve ou não ser concedido com base na política de autorização vigente. É interessante notar que a CORBAsec permite que a aplicação cliente tenha uma política de autorização própria e independente da política utilizada no servidor. Essa política é verificada pelo *AccessControlInterceptor* do lado cliente, que pode simplesmente interromper uma invocação para um objeto-alvo caso o cliente não possua os atributos necessários.

A relevância da CORBAsec para o nosso trabalho se encontra na arquitetura baseada em interceptadores que atuam de forma não intrusiva às aplicações. Como veremos no Capítulo 5, a arquitetura de segurança do JBoss também se baseia em interceptadores de segurança tanto no lado cliente quanto no lado servidor, se assemelhando bastante à arquitetura estabelecida

pela especificação *CORBA Security Service*.

2.5 Comparação com nosso trabalho

Nas avaliações individuais dos projetos apresentados neste capítulo, descrevemos os pontos fortes e também os problemas de cada projeto. Nesta seção, avaliaremos como este trabalho se compara aos projetos avaliados, e como ele lida com os problemas apresentados por esses projetos.

2.5.1 Modelo de divulgação de privilégios

O primeiro ponto a ser comparado é quanto ao modelo de divulgação de privilégios adotada. No Akenti e no PERMIS, autoridades que gerenciam privilégios são obrigadas a armazenar os certificados de atributos que contém esses privilégios em repositórios disponíveis para consulta. Os módulos de autorização de ambos os projetos são capazes apenas de lidar com esse modelo, tendo que fazer uma busca nos repositórios mantidos pelas autoridades para obter os certificados de atributos dos usuários. O SESAME, por outro lado, adota o outro modelo, no qual os certificados de atributos emitidos pelo PAS devem sempre ser entregues ao usuário titular dos privilégios.

Em relação a este trabalho, todos os três projetos se apresentam igualmente rígidos quanto ao modelo de divulgação de privilégios. Nossa extensão da infra-estrutura de segurança do JBoss oferece suporte total à IGP, de forma que as autoridades de atributos podem escolher a forma mais apropriada para o gerenciamento dos certificados gerados. O mecanismo de autorização que desenvolvemos é capaz de extrair os CAs X.509 dos usuários tanto de repositórios quanto diretamente da chamada feita pelo cliente para um recurso protegido. Ele é, portanto, mais flexível que os serviços de autorização dos projetos estudados.

2.5.2 Aderência a padrões

A aderência de um projeto de segurança a algum padrão proposto e aceito na literatura pode afetar significativamente a interoperabilidade de tal projeto com outros sistemas e arquiteturas de segurança. Tanto PERMIS quanto o SESAME utilizam certificados de atributos definidos por padrões bem estabelecidos, enquanto o Akenti utiliza certificados definidos por um formato proprietário, o que prejudica substancialmente sua interoperabilidade com outros sistemas. No PERMIS, a interoperabilidade não é completa, já que o projeto faz uso de um atributo proprietário para o armazenamento de papéis.

Nosso trabalho apresenta aderência total ao padrão X.509. Ao contrário do que ocorre no PERMIS, os papéis dos usuários são armazenados no atributo especificado pelo padrão X.509 para esse fim. Isso reflete positivamente na interoperabilidade de nosso projeto, já que qualquer CA X.509 emitido por uma autoridade de atributos aderente ao padrão X.509 pode ser utilizado por nosso serviço de autorização.

2.5.3 Restrições impostas

O Akenti e o SESAME apresentam restrições quanto ao mecanismo de autenticação a ser utilizado. Apesar de não lidar com autenticação, nosso trabalho faz uso da infra-estrutura de segurança já existente no JBoss, e esta é muito flexível quanto à escolha do mecanismo de autenticação a ser utilizado. Como o mecanismo de autorização implementado é totalmente independente do processo de autenticação, as únicas limitações encontradas quanto à forma de autenticação são as limitações já existentes hoje no servidor de aplicações. Isso significa que nosso trabalho não impôs nenhuma restrição adicional no tocante à autenticação.

3 Segurança na plataforma Java EE

A plataforma Java EE (*Java Enterprise Edition*) especifica um conjunto de componentes e tecnologias para o desenvolvimento de aplicações corporativas distribuídas portáveis e escaláveis. A arquitetura dessa plataforma define camadas lógicas que agrupam os componentes de acordo com as funcionalidades providas. As camadas e componentes normalmente utilizados para representar a arquitetura Java EE são exibidas na Figura 3.1.

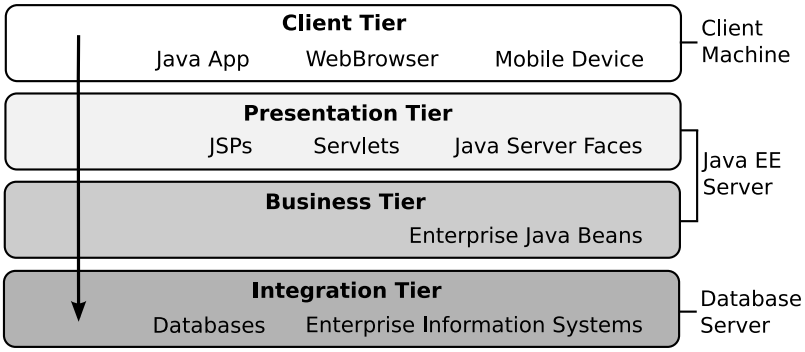


Figura 3.1: Camadas e componentes da plataforma Java EE

- camada cliente: a camada cliente (*client tier*) representa as diversas aplicações que clientes utilizam para interagir com uma aplicação Java EE. Exemplos de tais aplicações incluem aplicações Java tradicionais, navegadores *Web*, *Java applets* [27] e clientes móveis (aplicações Java ME) [28].

- camada de apresentação: a camada de apresentação (*presentation tier*) agrupa os componentes responsáveis pela geração e apresentação de interfaces *Web*. São eles: Servlets [53], *Java Server Pages* (JSP) [58] e *Java Server Faces* (JSF) [57].

- camada de negócios: a camada de negócios (*business tier*) é composta pelos componentes EJB (*Enterprise Java Beans*) [51], responsáveis pela implementação da lógica de negócios em uma aplicação Java EE.

- camada de integração: a camada de integração (*integration tier*) representa a conexão e comunicação com serviços de gerenciamento de dados, como bancos de dados, servidores LDAP e sistemas de gerenciamento de informação. A comunicação entre aplicações Java EE e esses serviços se dá através de tecnologias como JCA [52], JMS [50] e JDBC [55].

Os componentes nessas camadas são executados por contêineres específicos, como o contêiner *Web* e o contêiner EJB. Os contêineres fornecem o ambiente onde os componentes são executados de forma controlada e gerenciável. Em adição ao ambiente de execução, os contêineres fornecem também aos seus componentes outros serviços gerenciados, como segurança, transações e persistência. Neste capítulo apresentamos uma visão geral dos serviços de segurança disponibilizados pela plataforma Java EE para proteção de cada camada, demonstrando com mais detalhes como a proteção dos componentes *Web* e EJB é realizada.

3.1 Aspectos gerais

Com a intenção de constituir uma infra-estrutura de segurança robusta e completa, a plataforma Java EE permite que segurança seja aplicada em todas as camadas da arquitetura, protegendo não apenas os componentes individuais como também a comunicação entre esses componentes. A proteção dos componentes envolve a identificação, autenticação e autorização dos usuários, enquanto que a proteção da comunicação entre os componentes e camadas é obtida através de protocolos seguros, como o SSL - *Secure Socket Layer* [14] ou o TLS - *Transport Layer Security* [10].

Aplicações na camada cliente podem se utilizar das tecnologias fornecidas pela plataforma Java para satisfazer seus requisitos de segurança locais. Ao interagir com as camadas de apresentação e de negócios, um cliente precisa primeiro se autenticar ao servidor, um processo que varia de acordo com o componente sendo chamado. As especificações dos componentes *Web* definem alguns mecanismos padronizados para que o cliente se autentique através de seu navegador. Já a especificação de EJB não define como clientes devem ser autenticados, de forma que cada servidor de aplicações define um mecanismo próprio para esse fim.

Depois de autenticado, o cliente precisa ser autorizado pelo contêiner apropriado. Na plataforma Java EE a autorização é feita de acordo com os papéis que o cliente possui. Cada componente especifica os papéis que podem executar os seus métodos, e a infra-estrutura de segurança do contêiner verifica em tempo de execução se o cliente possui algum dos papéis necessários para que o acesso ao componente seja concedido. Os contêineres Java EE oferecem dois modelos de autorização: autorização declarativa, na qual os papéis e permissões associadas a cada papel são especificados em descritores de implantação externos à aplicação,

e autorização programática, na qual as decisões de autorização são realizadas pelos próprios componentes.

3.2 Segurança dos componentes Web

Nesta seção abordamos as configurações de segurança fornecidas pela plataforma Java EE para proteção de recursos *Web*. Primeiramente discutimos os mecanismos que podem ser configurados em aplicações *Web* para autenticação dos usuários. Em seguida demonstramos como a proteção dos componentes em função dos papéis dos usuário é feita. Por fim, descrevemos como uma aplicação *Web* pode especificar os requisitos de proteção da camada de transporte.

3.2.1 Autenticação de usuários

Para proteger os componentes de aplicações *Web*, o contêiner *Web* oferece mecanismos de autenticação que podem ser configurados para uma aplicação antes da implantação da mesma, através de um arquivo descritor. Nas aplicações *Web*, esse descritor é o arquivo `web.xml`, que é empacotado juntamente com as classes, arquivos JSP e conteúdo estático da aplicação.

Quando um cliente não autenticado tenta acessar um componente *Web* protegido, o contêiner envia uma requisição para o cliente para que este se autentique utilizando o mecanismo configurado no arquivo `web.xml`. As requisições do cliente para os componentes protegidos não serão aceitas pelo contêiner enquanto o usuário não se autenticar com sucesso. A autenticação se dá através de um dos seguintes mecanismos:

- BASIC: é a forma mais básica de autenticação por HTTP, na qual uma janela é exibida pelo navegador para que o cliente insira sua identidade e senha. Os dados são enviados sem nenhum tipo de criptografia, por isso recomenda-se o uso de protocolos seguros em conjunto com esta opção.

- FORM: essa opção permite a personalização da página de *login* que será utilizada pela aplicação para que o cliente se autentique. É possível também especificar a página de erro que a aplicação deve exibir em caso de falha na autenticação. Também não se utiliza de nenhum tipo de criptografia para o envio das informações.

- CLIENT-CERT: método conhecido por autenticação mútua, no qual tanto o cliente quanto o servidor se autenticam um ao outro através de certificados digitais. Essa opção requer a instalação de um certificado no navegador do cliente.

- DIGEST: método no qual a autenticação ocorre através de uma troca de desafios entre o cliente e o servidor. As especificações de Servlets e JSP não obrigam que o contêiner ofereça suporte a esse método, já que nem todos os navegadores trabalham com esse tipo de mecanismo.

A configuração do mecanismo se dá através da seção `<login-config>` do descritor de implantação `web.xml`. A Figura 3.2 demonstra um exemplo de configuração da opção FORM contendo o nome das páginas de *login* e de erro.

```
<web-app>
....
    <login-config>
        <auth-method>FORM</auth-method>
        <form-login-config>
            <form-login-page>/LoginForm.html</form-login-page>
            <form-error-page>/LoginError.html</form-error-page>
        </form-login-config>
    </login-config>
....
</web-app>
```

Figura 3.2: Configuração do método FORM de autenticação

A configuração dos demais métodos é bastante similar, bastando especificar o valor do elemento `auth-method` com o nome do mecanismo desejado. É importante ressaltar que os mecanismos de autenticação configurados apenas definem a forma como o cliente fornece sua identidade e credencial. A validação dessas informações, isto é, a autenticação propriamente dita, é realizada por algum outro mecanismo utilizado pelo contêiner. O servidores de aplicação atuais normalmente autenticam seu clientes através do *Java Authentication and Authorization Service* (JAAS) [47].

3.2.2 Especificação das regras de autorização

Vimos anteriormente que a autorização na plataforma Java EE é feita de acordo com os papéis dos usuários. Em aplicações *Web*, administradores utilizam o arquivo `web.xml` para descrever os requisitos de controle de acesso. Isso envolve duas etapas: *(i)* declaração dos papéis da aplicação e *(ii)* especificação das permissões associadas aos papéis.

Uma aplicação *Web* declara os seus papéis através do elemento `<security-role>`. A declaração de cada papel envolve a definição do nome do papel e uma descrição opcional do mesmo. É recomendável que a descrição seja sempre fornecida a fim de facilitar o gerenciamento e administração da aplicação.

A Figura 3.3 ilustra a declaração de dois papéis de segurança em uma aplicação *Web*. O primeiro (`administrator`), é atribuído aos usuários administradores, enquanto que o segundo (`regular-user`) é atribuído aos demais usuários da aplicação.

```
<web-app>
....
    <security-role>
        <description>Papel atribuído a administradores de sistema</description>
        <role-name>administrator</role-name>
    </security-role>
    <security-role>
        <description>Papel atribuído aos demais usuários</description>
        <role-name>regular-user</role-name>
    </security-role>
....
</web-app>
```

Figura 3.3: Declaração dos papéis em uma aplicação *Web*

Uma vez descritos os papéis da aplicação, é preciso especificar as regras de controle de acesso em si, isto é, definir quais papéis podem acessar cada recurso. Como nas aplicações *Web* cada recurso (JSPs, Servlets, páginas HTML, etc) é acessado através de uma URL, o controle de acesso é feito com base nas URLs da aplicação. Tal controle é descrito através do elemento `<security-constraint>`, conforme exemplificado pela Figura 3.4.

```
<web-app>
....
    <security-constraint>
        <web-resource-collection>
            <web-resource-name>Administration</web-resource-name>
            <description>URLs das páginas de administração do sistema</description>
            <url-pattern>/admin/*</url-pattern>
            <url-pattern>/monitor/*</url-pattern>
            <http-method>POST</http-method>
            <http-method>GET</http-method>
        </web-resource-collection>
        <auth-constraint>
            <description>Acesso restrito a usuários administradores</description>
            <role-name>administrator</role-name>
        </auth-constraint>
    </security-constraint>
....
</web-app>
```

Figura 3.4: Especificação das regras de autorização em uma aplicação *Web*

O elemento `<security-constraint>` é composto de duas seções. A primeira, identificada

pelo elemento `<web-resource-collection>`, descreve quais URLs estão sendo protegidas. A segunda, identificada pelo elemento `<auth-constraint>`, descreve quais papéis devem poder acessar as URLs protegidas. Os papéis especificados na seção `<auth-constraint>` devem constituir um subconjunto dos papéis de segurança já declarados através de elementos `<security-role>`.

O conjunto das URLs sendo protegidas é especificado através de um ou mais elementos `<url-pattern>`, e dos métodos HTTP especificados por elementos `<http-method>`. No exemplo da Figura 3.4, todo acesso por *POST* ou *GET* para as URLs que casam com os padrões `admin/*` e `monitor/*` será controlado, e somente usuários no papel de administrador serão autorizados.

A especificação do método HTTP é opcional, e caso nenhum elemento `<http-method>` esteja presente, a proteção das URLs será realizada para todos os métodos. Por outro lado, se um ou mais métodos forem especificados, apenas os acessos feitos por estes métodos serão protegidos. Isso significa que, por exemplo, se apenas o método *GET* for especificado, o acesso só será controlado caso a requisição do cliente seja feita por este método. Eventuais acessos por *POST* serão liberados.

Dessa forma, o elemento `<security-constraint>` especifica um agrupamento de URLs (que tipicamente contém recursos que participam de funcionalidades relacionadas) e as regras de acesso a esse agrupamento. Aplicações se utilizam de um ou mais elementos `<security-constraint>` para definir as regras de controle de acesso que devem ser aplicadas para cada grupo de URLs.

3.2.3 Proteção da camada de transporte

Freqüentemente, a comunicação entre um cliente e uma aplicação *Web* se dá através da *Internet*. Quando a troca de informações entre este cliente e a aplicação envolver dados sigilosos (como durante a autenticação, onde o cliente fornece sua senha), é recomendável o uso de protocolos de comunicação seguros, como SSL, para garantia de integridade e confidencialidade desses dados.

Os requisitos quanto à segurança da camada de transporte podem ser especificados no arquivo `web.xml` através do elemento `<user-data-constraint>` (Figura 3.5). Este elemento permite a especificação de três diferentes níveis de segurança para a camada de transporte, a saber:

- NONE: é o valor padrão e indica que nenhuma forma de proteção da camada de transporte é necessária.

- INTEGRAL: indica que os dados transmitidos entre o cliente e o servidor não podem ser modificados durante a comunicação.

- CONFIDENTIAL: indica que os dados transmitidos entre o cliente e o servidor devem ser criptografados para não serem inspecionados durante a comunicação.

```
<web-app>
....
<user-data-constraint>
    <description>Confidentiality required</description>
    <transport-guarantee>CONFIDENTIAL</transport-guarantee>
</user-data-constraint>
....
</web-app>
```

Figura 3.5: Especificação da camada de transporte em uma aplicação *Web*

Na prática, o uso de INTEGRAL ou CONFIDENTIAL tem o mesmo resultado, já que normalmente o protocolo de comunicação selecionado (SSL) oferece os dois tipos de proteção. Além disso, a simples especificação de uma dessas opções no arquivo `web.xml` não é suficiente. Algum tipo de configuração do servidor tem de ser feita para que as conexões abertas pelo cliente se utilizem do protocolo HTTPS (HTTP sobre SSL). Para isso é preciso consultar a documentação de segurança do servidor utilizado.

3.3 Segurança dos componentes EJB

Nesta seção abordamos as configurações de segurança que podem ser aplicadas para proteger recursos EJB na plataforma Java EE. Em particular, discutimos as diferenças em relação aos componentes *Web* quanto ao processo de autenticação, e apresentamos as formas para especificar o controle de acesso baseado em papéis para os componentes EJB

3.3.1 Autenticação

A especificação de segurança de EJB não define nenhum tipo de mecanismo de autenticação que deva ser utilizado pelos clientes para fornecer suas identidades e credenciais. Dessa forma, um cliente que deseje chamar um componente EJB diretamente (isto é, sem interagir com uma camada de apresentação *Web*) deve se autenticar ao contêiner EJB utilizando o mecanismo disponibilizado pelo contêiner para este fim.

Contrastando esse cenário com as especificações de segurança dos componentes *Web*, uma questão vem à tona: por que a especificação de EJB não define os mecanismos de autenticação que podem ser utilizados?

Em primeiro lugar, aplicações *Web* interagem com os clientes através dos navegadores *Web*, de forma que o mecanismo de autenticação escolhido deve ser compatível com os navegadores em conformidade com o padrão HTTP. Isto de certa forma já especifica quais mecanismos podem ser utilizados. Aplicações EJB, por outro lado, podem ser acessadas a partir dos mais diversos tipos de aplicação cliente. Não especificar uma forma de autenticação para aplicações EJB permite que cada contêiner forneça várias formas diferentes de autenticação, que são usadas pelas aplicações-cliente de acordo com os seus requisitos e poder de processamento.

Em segundo lugar, aplicações Java EE são normalmente construídas de forma que os clientes interajam apenas com os componentes *Web* (camada de apresentação) e não com os componentes EJB (camada de negócios). Isto significa que tipicamente os clientes se autenticam à camada *Web* e esta chama os componentes EJB para processar as regras de negócios. Isto não significa, porém, que os componentes EJB não são protegidos. Pelo contrário, cada componente pode descrever suas regras de controle de acesso assim como os componentes *Web*. O servidor de aplicações é responsável pela propagação do contexto de segurança do usuário nas chamadas feitas pelos componentes *Web* para os componentes EJB, de maneira que os últimos possam aplicar suas regras de segurança a partir das informações contidas no contexto propagado (Figura 3.6).

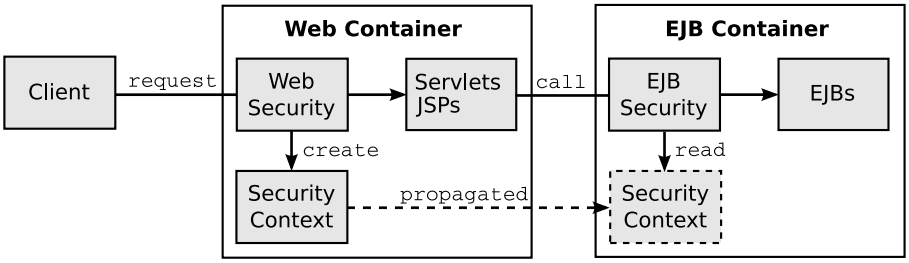


Figura 3.6: Propagação do contexto de segurança entre contêineres

3.3.2 Especificação das regras de controle de acesso

A definição das regras de autorização dos componentes EJB é bastante similar ao que é feito para os componentes *Web*: um arquivo descritor é utilizado para declarar os papéis da aplicação e o conjunto de permissões associado a cada papel. Este arquivo, o `ejb-jar.xml`, é empacotado juntamente com as classes que compõem a aplicação EJB no arquivo `jar` que é

implantado no servidor de aplicações.

A declaração dos papéis de segurança é feita na seção `<assembly-descriptor>` do arquivo `ejb-jar.xml` através do elemento `<security-role>` — o mesmo utilizado para declarar os papéis de segurança em aplicações *Web*. A Figura 3.7 demonstra a declaração de papéis no arquivo `ejb-jar.xml` para uma aplicação fictícia de gerenciamento de salários e benefícios.

```
<assembly-descriptor>
....
    <security-role>
        <description>
            Usuários nesse papel têm permissão para modificar salários e benefícios.
        </description>
        <role-name>gerente_rh</role-name>
    </security-role>
    <security-role>
        <description>
            Usuários nesse papel podem apenas consultar seus salários e benefícios.
        </description>
        <role-name>funcionario</role-name>
    </security-role>
....
</assembly-descriptor>
```

Figura 3.7: Especificação de papéis de segurança no arquivo `ejb-jar.xml`

O acesso aos componentes EJB é feito através dos métodos declarados na interface que é exposta para os clientes. Por isso, as regras de autorização são especificadas para os métodos, descrevendo quais papéis devem ter acesso aos métodos de cada EJB. Isso é feito através do elemento `<method-permission>`.

Cada elemento `<method-permission>` contém um ou mais papéis de segurança, identificados pelo elemento `<role-name>`, e um ou mais métodos de negócio, identificados pelo elemento `<method>`. A interpretação de um `<method-permission>` é a seguinte: todos os papéis informados em `<method-permission>` têm permissão para acessar todos os métodos descritos dentro desse mesmo elemento.

A Figura 3.8 mostra a definição de algumas permissões de método no arquivo `ejb-jar.xml`. O trecho exibido especifica que usuários com papel `gerente_rh` podem executar todos os métodos do EJB `GerenciadorDeBeneficios` e também o método `aumentarSalario()` do EJB `GerenciadorDeSalarios`. Ao mesmo tempo, usuários com o papel `funcionario` podem apenas invocar os métodos `findByPrimaryKey()` e `getBeneficios()` do EJB `GerenciadorDeBeneficios` e o método `findByPrimaryKey()` do `GerenciadorDeSalarios`.

```
<assembly-descriptor>
....
    <method-permission>
        <role-name>gerente_rh</role-name>
        <method>
            <ejb-name>GerenciadorDeBeneficios</ejb-name>
            <method-name>*</method-name>
        </method>
        <method>
            <ejb-name>GerenciadorDeSalarios</ejb-name>
            <method-name>aumentarSalario</method-name>
        <method>
    </method-permission>
    <method-permission>
        <role-name>funcionario</role-name>
        <method>
            <ejb-name>GerenciadorDeBeneficios</ejb-name>
            <method-name>findByPrimaryKey</method-name>
        </method>
        <method>
            <ejb-name>GerenciadorDeBeneficios</ejb-name>
            <method-name>getBeneficios</method-name>
        </method>
        <method>
            <ejb-name>GerenciadorDeSalarios</ejb-name>
            <method-name>findByPrimaryKey</method-name>
        </method>
    </method-permission>
....
</assembly-descriptor>
```

Figura 3.8: Especificação das regras de autorização em uma aplicação EJB

3.3.3 Anotações de segurança

A partir da versão 3.0 da especificação de EJB [54] é possível especificar as regras de controle de acesso diretamente no código-fonte dos componentes, através do uso de anotações[1] (*annotations*). A Figura 3.9 a seguir oferece um exemplo do uso de anotações para definir as regras de autorização em um EJB do tipo *Session*.

A anotação `@RolesAllowed("admin")` especificada para a classe `MySessionBean` indica que o papel de segurança `admin` deve ter permissão para executar todos os métodos da classe que não estiverem anotados. A anotação `@RolesAllowed("gerente")` do método `metodoPara-Gerentes()` sobrescreve esse comportamento, permitindo que somente usuários no papel de `gerente` façam acesso a esse método. A anotação `@PermitAll` do método `metodoParaTodos()`

[1] Trata-se de um recurso introduzido a partir da versão 5.0 de Java que permite a declaração de meta-dados diretamente no código-fonte das classes, eliminando a necessidade de usar arquivos descritores para essa finalidade.

```
@RolesAllowed("admin")
@Stateless public class MySessionBean implements MySession {

    public void metodoParaAdmins() {
        // admins executam este método.
    }

    @RolesAllowed("gerente")
    public void metodoParaGerentes() {
        // gerente executam este método.
    }

    @PermitAll
    public void metodoParaTodos() {
        // todos executam este método.
    }

    @DenyAll
    public void metodoParaNinguem() {
        // ninguém executa este método.
    }
}
```

Figura 3.9: Exemplo de uso das anotações de segurança

também sobrescreve o comportamento padrão, especificando que o acesso a este método deve ser irrestrito. Por fim, a anotação `@DenyAll` do método `metodoParaNinguem()` sobrescreve a configuração padrão para impedir o acesso a todos os papéis de segurança.

É importante ressaltar que esta forma de declaração das regras de acesso é apenas uma alternativa ao uso do descritor `ejb-jar.xml`. Se por um lado o uso de anotações facilita o processo de configuração de segurança, por outro ele é intrusivo ao código e requer que eventuais mudanças nas regras de controle de acesso sejam feitas diretamente no código-fonte, o que exige a recompilação desse código. Já o arquivo descritor apresenta maior complexidade na descrição das regras, mas por outro lado ele não interfere no código-fonte da aplicação, permitindo que regras sejam modificadas sem que a aplicação tenha que ser recompilada.

3.4 Implementação das especificações de segurança

Conforme apresentado neste capítulo, a plataforma Java EE especifica modelos para os processos de autenticação (quando aplicável) e autorização utilizados para proteger os componentes *Web* e EJB de acessos maliciosos. Contudo, apesar de especificar os modelos de segurança, a plataforma não especifica os mecanismos concretos que implementam tais modelos. Assim, por exemplo, um cliente que se autentica a uma aplicação *Web* fornece sua

identidade e credenciais de acordo com o modelo configurado pela aplicação. A autenticação em si, isto é, o processo de verificação da validade das credenciais fornecidas, é executada por algum mecanismo específico do servidor de aplicações. O mesmo vale para o processo de autorização, onde um mecanismo específico é responsável por encontrar os papéis associados à identidade do cliente e utilizar esses papéis para realizar o controle de acesso.

O processo de autenticação normalmente envolve consultas a algum repositório de dados confiável para estabelecer a validade das informações enviadas pelo cliente, a menos que a autenticação seja feita através do protocolo de comunicação (SSL com autenticação mútua), pois nesse caso o próprio protocolo é capaz de autenticar o cliente. O serviço de autenticação JAAS vem sendo utilizado largamente por servidores de aplicação para implementar o processo de autenticação devido à flexibilidade oferecida pelo JAAS na escolha e configuração do mecanismo de autenticação concreto a ser utilizado.

O processo de autorização, por sua vez, envolve o mapeamento entre identidades e seus papéis. A implementação do serviço de autorização deve ter a capacidade de estabelecer em tempo de execução os papéis associados a uma identidade. As soluções atualmente implementadas pelos servidores de aplicação são todas baseadas em consultas a repositórios para determinar o mapeamento entre identidades e papéis. Esse tipo de arquitetura é pouco flexível do ponto de vista da autoridade que gerencia esses papéis (e outros atributos dos clientes), pois obriga que tais entidades mantenham esses atributos em repositórios. Este trabalho propõe o uso de certificados de atributos X.509 para associar identidades aos seus papéis e a implementação de um mecanismo de autorização capaz de processar esses certificados em total conformidade com a IGP. Mais especificamente, o serviço de autorização criado deve lidar tanto com certificados armazenados em repositórios como com certificados enviados diretamente pelos clientes, permitindo assim que as autoridades que gerenciam os privilégios dos clientes escolham a forma de divulgar tais privilégios.

No próximo capítulo analisaremos com maiores detalhes a IGP e os certificados de atributos X.509. Demonstraremos que o uso desses certificados como estrutura padronizada para realizar o mapeamento de identidades e papéis em um ambiente Java EE não só é flexível para as autoridades como também oferece maior segurança para o serviço de autorização do servidor de aplicações por conta do uso de assinaturas digitais para proteção das informações contidas nos certificados.

4 A infra-estrutura de gerenciamento de privilégios

Neste capítulo, apresentamos a infra-estrutura de gerenciamento de privilégios, descrevendo as motivações para sua criação e como ela se relaciona com a infra-estrutura de chaves públicas. Detalhamos em seguida o elemento principal dessa infra-estrutura, o certificado de atributos, e discutimos os modelos que podem ser adotados por autoridades de atributos para distribuição desses certificados. Finalmente, descrevemos como esses certificados podem ser utilizados para implementar diferentes modelos de controle de acesso baseados em papéis.

4.1 Histórico e motivação

A infra-estrutura de gerenciamento de privilégios, ou IGP, surgiu a partir da necessidade de um mecanismo forte de autorização que fosse independente do mecanismo de autenticação. O padrão X.509 definiu inicialmente a infra-estrutura de chaves públicas (*public key infrastructure*), ou ICP [20, 62], cujo elemento central é o certificado de chave pública (*public key certificate*), ou simplesmente CCP. O principal objetivo dessa infra-estrutura é definir um modelo forte de autenticação.

O certificado de chave pública é um documento gerado e assinado digitalmente pela autoridade certificadora (*certification authority*), e associa a identidade de um usuário a uma chave pública. A chave privada é mantida em segredo pelo titular do certificado. A utilização mais conhecida de uso desses certificados é pelos mecanismo de autenticação de protocolos seguros de comunicação como SSL e TLS. Nesses protocolos, uma série de desafios é realizada utilizando as chaves pública e privada do servidor (e também do cliente, quando necessário), a fim de se verificar as identidades das partes envolvidas.

Na prática, o uso da ICP revelou a necessidade de se armazenar outros tipos de dados em um certificado, além da chave pública e da identidade do seu titular. Por isso, versões recentes do padrão X.509 definem uma série de extensões que permitem o armazenamento de informações adicionais nos certificados de chave pública. Exemplos das informações normalmente adicionadas incluem os papéis que um usuário desempenha, suas permissões, ou qualquer outro

tipo de informação de autorização. Porém, de acordo com Zhou e Meinel [64], a utilização das extensões dos CCPs para o armazenamento de informações de autorização apresenta efeitos negativos, dentre os quais os que mais se destacam são:

- As informações de autorização normalmente não têm o mesmo tempo de validade da identidade e da chave pública que são armazenadas em um CCP. Em geral, informações de autorização tendem a ter validade mais curta do que a identidade e chave pública. Dessa forma, se tais informações são incluídas nas extensões de um CCP, o tempo durante o qual o CCP deveria ser válido é encurtado, pois modificações das informações de autorização irão requerer que um novo CCP seja emitido e o antigo seja revogado. A seguinte analogia é usada para ajudar na compreensão: um CCP pode ser comparado a um passaporte, que identifica o seu titular e tende a ser valido por um longo período. Informações de autorização e controle de acesso, por sua vez, são como vistos de entrada, que tipicamente são emitidos por autoridades diferentes e não são válidos por um longo período.

- A entidade que emite CCPs provavelmente não tem autoridade suficiente para estabelecer informações de autorização, uma vez que esse tipo de informação é muito dependente do contexto onde se encontram os titulares dos certificados. Como resultado, as autoridades certificadoras acabam ficando mais complexas à medida que precisam consultar outros serviços para descobrir o conjunto de atributos do usuário para o qual ela está gerando um certificado de chaves públicas.

Somando-se a esses fatores, outros problemas com a ICP [18,19] impulsionaram o desenvolvimento de uma abordagem alternativa para lidar com informações de autorização. O resultado desse trabalho foi a criação da infra-estrutura de gerenciamento de privilégios, que tem como objetivo a definição de um modelo forte de autorização, independente do processo de autenticação. Essa infra-estrutura tem como elemento central os certificados de atributos X.509, gerados e assinados digitalmente pelas autoridades de atributos.

Graaf e Carvalho [63] discutem os efeitos positivos da IGP, concluindo que "os certificados de atributos X.509 estabelecem uma separação clara entre o processo de autenticação (identificação) e autorização (controle de acesso)". As autoridades de atributos assumem a responsabilidade pelo gerenciamento dos atributos de autorização dos usuários, removendo das autoridades certificadoras a complexidade resultante da consulta a serviços externos para encontrar os privilégios dos usuários. Os autores apresentam ainda as vantagens decorrentes da separação dos processos de autenticação e autorização:

- "Promove a interoperabilidade à medida que favorece o gerenciamento distribuído de privilégios e atributos". Como os atributos de autorização não são emitidos pelas autoridades certificadoras, é possível distribuir o gerenciamento desses atributos por várias autoridades de atributos, cada uma fornecendo informações de autorização relativas a um contexto diferente. Por exemplo, em um país é possível que cada cidadão tenha um CCP contendo sua identidade assinado por uma única autoridade certificadora central. Os seus privilégios nos mais diversos contextos da vida social podem ser emitidos por diferentes autoridades de atributos. Uma autoridade de atributos pode, por exemplo, emitir um certificado de atributos contendo informações sobre os privilégios de um advogado junto à sua ordem, enquanto outra autoridade totalmente distinta pode emitir um certificado contendo os privilégios do advogado junto ao fórum onde ele trabalha.

- "Promove a separação de jurisdição, uma vez que os certificados de atributos são emitidos por autoridades que realmente possuem as informações de autorização, evitando a delegação de responsabilidades para as autoridades certificadoras."

- "Certificados de atributos podem ter um tempo de vida muito mais curto do que os certificados de chave pública e podem ser revogados separadamente". Isso é uma conseqüência imediata da remoção dos atributos de autorização dos certificados de chave pública. Se os atributos de um usuário mudam, apenas os seus certificados de atributos precisam ser revogados. Não há necessidade de se revogar o certificado de chave pública desse usuário.

Embora tenham sido desenvolvidas para trabalharem em conjunto, as infra-estruturas de chaves públicas e de gerenciamento de privilégios são independentes e podem ser empregadas separadamente. Isso significa que um sistema que autentica usuários através da ICP não precisa necessariamente utilizar a IGP para autorizar esses usuário e vice-versa. A Seção a seguir detalha os certificados de atributos e demonstra como eles podem ser utilizados tanto em conjunto com a ICP quanto com qualquer outro mecanismo de autenticação.

4.2 Certificado de atributos X.509

Os certificados digitais são documentos protegidos por uma assinatura digital. Tal assinatura é gerada aplicando-se uma função de resumo criptográfico (*hash*) sobre o conteúdo do documento, e criptografando o valor resultante dessa função com a chave privada da autoridade certificadora a fim de garantir sua integridade. As funções de resumo criptográfico têm

como principal característica o fato de serem capazes de gerar valores completamente diferentes a partir de uma mínima alteração do documento original. Isso significa que qualquer tentativa de alteração indevida do documento resultará em um valor de *hash* totalmente diferente. Arcabouços de segurança validam os certificados recebidos recalculando o valor de *hash* do conteúdo do certificado, e comparando este valor com o valor obtido decriptografando-se a assinatura digital com a chave pública da autoridade certificadora. Valores diferentes indicam que o conteúdo do documento foi violado e que, portanto, o certificado não deve ser aceito. Esse processo é ilustrado pela Figura 4.1.

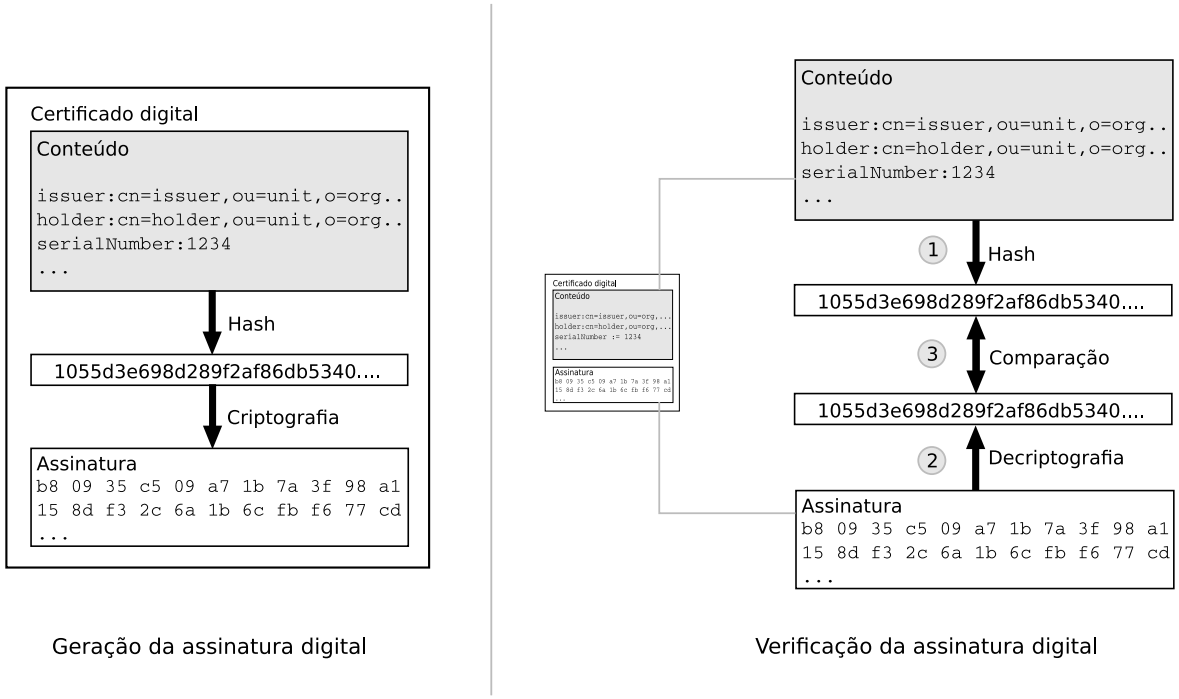


Figura 4.1: Geração e validação de um certificado digital

Para que esse tipo de interação entre autoridade certificadora e arcabouços de segurança seja possível, é preciso que ambos concordem quanto ao algoritmo de assinatura digital utilizado. Por essa razão, certificados digitais geralmente incluem um campo indicando qual algoritmo deve ser utilizado para validar a assinatura digital. Outras informações normalmente disponíveis nos certificados digitais incluem: a identidade da autoridade certificadora, a identidade do titular do certificado, o período de validade do certificado, e o número de série que identifica o certificado unicamente junto à autoridade.

Dito isso, podemos explorar com mais detalhes o conteúdo de um certificado de atribu-

tos X.509. A Figura 4.2 ilustra a estrutura simplificada desses certificados de acordo com a RFC3281 [11].

```
AttributeCertificate ::= SEQUENCE {
     acinfo               AttributeCertificateInfo,
     signatureAlgorithm   AlgorithmIdentifier,
     signatureValue       BIT STRING
}

AttributeCertificateInfo ::= SEQUENCE {
     holder               Holder,
     issuer               AttCertIssuer,
     serialNumber         CertificateSerialNumber,
     attrCertValidityPeriod   AttCertValidityPeriod,
     attributes           SEQUENCE OF Attribute,
     extensions           Extensions OPTIONAL
}
```

Figura 4.2: Certificado de atributos X.509

Apresentamos a seguir uma descrição dos principais campos que compõe um CA X.509:

- holder: o campo `holder`[1] (Figura 4.3) identifica o titular do certificado. Como os certificados de atributos são utilizados por mecanismos de autorização depois que a autenticação do usuário foi efetuada, o campo `holder` deve, de alguma forma, se referenciar à identidade do usuário autenticado (Figura 4.4). Tendo como objetivo a flexibilidade, o padrão X.509 define três opções diferentes para este campo, que são utilizadas de acordo com o mecanismo de autenticação utilizado.

  A opção `baseCertificateID` deve ser utilizada quando a autenticação é baseada em CCPs, e quando a autoridade que emite os CCPs é a mesma autoridade que emite o certificado de atributos, ou seja, quando uma mesma entidade faz o papel de autoridade certificadora e autoridade de atributos. Nesse caso, os campos `serialNumber` e `issuer` do CCP utilizado para autenticar o usuário devem ser idênticos aos campos `serial` e `issuer`, respectivamente, da estrutura `IssuerSerial` contida no campo holder do CA X.509 e exibida na Figura 4.3.

  Quando a opção `entityName` é utilizada, o mecanismo de autenticação pode ou não ser baseada em CCPs; se for, normalmente a autoridade certificadora e a autoridade de

[1] Nos CCPs, o campo `subject` é usado para armazenar a identidade à qual a chave pública está associada. Já nos certificados de atributos, o campo `holder` é utilizado para representar a identidade do titular dos atributos. Essa diferença de nomenclatura pode trazer confusões e o ideal seria que o termo *subject* tivesse sido mantido para esse campo.

```
Holder ::= SEQUENCE {
      baseCertificateID   [0] IssuerSerial OPTIONAL,
                -- the issuer and serial number of
                -- the holder's Public Key Certificate
      entityName          [1] GeneralNames OPTIONAL,
                -- the name of the claimant or role
      objectDigestInfo    [2] ObjectDigestInfo OPTIONAL
                -- used to directly authenticate the holder,
                -- for example, an executable
}
IssuerSerial  ::=  SEQUENCE {
     issuer         GeneralNames,
     serial         CertificateSerialNumber,
     issuerUID      UniqueIdentifier OPTIONAL
}
```

Figura 4.3: Definição do campo `holder` de um CA X.509

atributos são entidades distintas, e o campo `entityName` deve conter o mesmo valor encontrado no campo `subject` do CCP utilizado para autenticar o usuário. Caso contrário, este campo deve conter apenas um *distinguished name* [26] que represente a identidade do titular.

Figura 4.4: Relação entre a IGP e a ICP estabelecida a partir do campo `holder`

A terceira e última opção, `objectDigestInfo` não é necessária para uma implementação dessa especificação e pode não estar presente. A idéia dessa opção é associar o CA X.509 a um objeto colocando um *hash* do objeto no campo holder do certificado. Isso permite a produção de certificados de atributos que contenham privilégios associados a um objeto

executável, como uma classe Java.

Ao menos uma das opções deve estar presente nesse campo. Apesar de ser possível utilizar mais de uma opção em um CA X.509, a especificação não recomenda essa prática porque não existe uma regra que determine qual opção deve ser considerada principal e quais devem ser consideradas secundárias, o que pode causar problemas na interpretação do conteúdo desse campo.

- issuer: o campo issuer contém um *distinguished name* que representa a identidade da autoridade de atributos que emitiu o certificado.

- serialNumber: o par `issuer/serialNumber` deve formar uma combinação única para todo certificado de atributos gerado em conformidade com a especificação, mesmo que o prazo de validade desses certificados seja curto. As autoridades de atributos devem garantir que o `serialNumber` seja um inteiro positivo, mas não é necessário que esses números sejam seqüenciais ou mesmo crescentes.

- signature: contém o identificador do algoritmo usado para validar a assinatura do certificado de atributos. Deve especificar um dos algoritmos definidos em [43].

- attrCertValidityPeriod: o campo de validade do certificado especifica o período de tempo pelo qual a autoridade de atributos garante que a associação entre o titular e seus atributos será válida. A estrutura exibida na Figura 4.5 é utilizada para armazenar esse período.

```
AttCertValidityPeriod  ::= SEQUENCE {
     notBeforeTime  GeneralizedTime,
     notAfterTime   GeneralizedTime
}
```

Figura 4.5: Campo `attrCertValidityPeriod` de um CA X.509

`GeneralizedTime` é um tipo do padrão ASN.1 [25] usado para representação de tempo. Para certificados de atributos, os valores de `GeneralizedTime` devem ser expressos de acordo como o *Greenwich Mean Time* (GMT) e devem incluir minutos e segundos.

- attributes: a grande razão da existência dos certificados de atributos X.509. É formado por uma seqüência de `Attribute`s, que contém os atributos associados à identidade referenciada pelo campo `holder`. A Figura 4.6 exibe a definição do elemento `Attribute`.

```
Attribute ::= SEQUENCE {
      type      AttributeType,
      values    SET OF AttributeValue
        -- at least one value is required
}
AttributeType ::= OBJECT IDENTIFIER
AttributeValue ::= ANY DEFINED BY AttributeType
```

Figura 4.6: Campo `attributes` de um CA X.509

Um CA X.509 válido deve conter ao menos um atributo, e cada tipo de atributo pode aparecer uma única vez (ou seja, é proibida a presença de dois ou mais elementos `Attribute` com mesmo `AttributeType`). Como os valores dos atributos podem variar desde uma simples `String` até estruturas mais complexas, a especificação permite que qualquer estrutura possa ser utilizada como `AttributeValue`. Alguns tipos de atributos-padrão são definidos, entre eles o atributo `role`, que define a sintaxe que deve ser utilizada para a inserção dos papéis de um usuário em um atributo do CA X.509.

- extensions: as extensões são normalmente utilizadas para armazenar informações adicionais a respeito do certificado ou da autoridade de atributos. Autoridades geralmente incluem informações a respeito de repositórios ou serviços que podem ser utilizados por verificadores de certificados para obter informações quanto à revogação do certificado sendo validado.

Conforme discutido no início dessa Seção, a associação entre a IGP e a ICP se dá através do campo `holder` do CA X.509. Isso significa que um validador de certificados deve ter acesso ao CCP utilizado na autenticação para poder verificar que a associação entre os dois certificados é válida. Quando o mecanismo de autenticação não utiliza CCPs, a validação desse campo tem de ser feita de forma um pouco diferente. Como é pouco provável que os usuários se autentiquem utilizando um *distinguished name* como identidade, o validador pode, por exemplo, checar se um fragmento (por exemplo, o valor da propriedade `cn`) do *distinguished name* contido no campo `holder` corresponde à identidade do usuário.

4.3 Modelos de divulgação de certificados

Um serviço de autorização flexível deve permitir que a autoridade de atributos escolha a melhor forma de divulgar os certificados de atributos gerados. Existem dois modelos de publicação que as autoridades podem utilizar: o modelo *pull* (Figura 4.7), no qual os certificados são publicados em repositórios acessíveis para consulta por parte de servidores que desejam

"puxar" essas informações, e o modelo *push* (Figura 4.8), no qual os certificados são fornecidos diretamente para o seu titular, que assume a responsabilidade de reuni-los e depois "empurrá-los" para o servidor ao fazer acesso a um recurso protegido. Esse último modelo só é possível graças a assinatura digital do certificado de atributos, que permite ao serviço de autorização do servidor verificar que as informações sendo fornecidas pelo cliente são de fato verdadeiras e não foram alteradas.

No modelo *pull*, o serviço de autorização do servidor fica responsável pela busca dos certificados em repositórios, o que aumenta a complexidade do serviço e também implica que a autoridade de atributos deve manter um repositório disponível para consulta. Essas características fazem com que esse modelo seja mais adequado para ambientes nos quais os privilégios dos clientes são definidos e atribuídos no próprio domínio do servidor, em geral sendo emitidos por autoridades localizadas também no mesmo domínio.

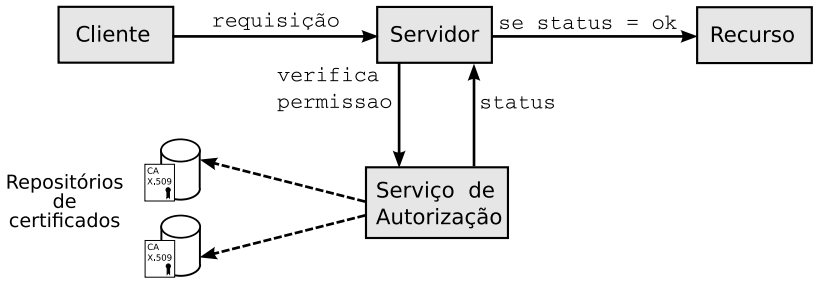


Figura 4.7: Modelo *pull* de publicação de certificados

Já o modelo *push* transfere para o cliente a responsabilidade de fornecer os seus certificados de atributos X.509 para o servidor, reduzindo a complexidade do serviço de autorização. Esse modelo é mais adequado quando os privilégios dos clientes são definidos e atribuídos em um domínio diferente do servidor, geralmente sendo emitidos por autoridades totalmente independentes do ambiente servidor. Embora esse modelo remova do servidor a responsabilidade de encontrar os certificados dos clientes, ele exige maiores cuidados com a validação dos certificados.

Vários fatores podem colaborar para que um certificado seja considerado inválido, mas os dois principais são: o certificado expirou (ou seja, seu prazo de validade venceu) ou foi revogado, seja porque o titular não é mais considerado um usuário aprovado do sistema ou porque seu conjunto de privilégios foi alterado. O primeiro caso é fácil de se verificar já que o certificado contém o prazo de validade como um de seus atributos. Já com respeito à revogação, a solução usual adotada é de fazer com que a autoridade emissora do certificado publique listas de certificados revogados (*certificate revocation list*), ou LCRs, contendo os números de série

dos certificados que não devem mais ser considerados válidos. Entretanto, existem problemas conhecidos [44] relacionados às LCRs, sendo o principal deles o fato de que as listas de revogação são publicadas periodicamente, de forma que a revogação de privilégios de um certo usuário não pode ser aplicada imediatamente. Por conta disto, outros mecanismos [22, 32] também foram propostos, entre eles o *online certificate status prococol* (OCSP) [33], que é um protocolo que permite a verificação em tempo real da validade de um certificado junto à entidade certificadora.

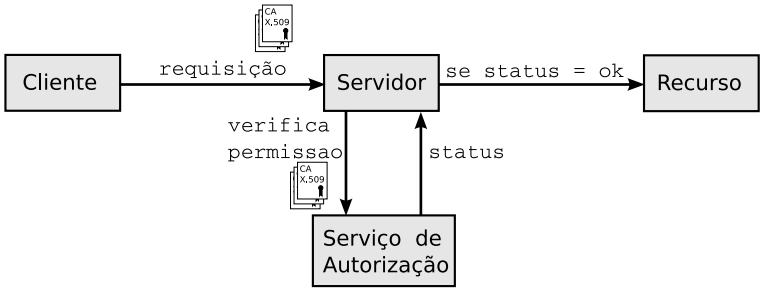


Figura 4.8: Modelo *push* de publicação de certificados

O modelo *pull*, por outro lado, pode lidar com a revogação de certificados de uma maneira muito mais simples. Como os certificados não são distribuídos para os seus titulares e são armazenados em repositórios mantidos pelas autoridades de atributos, a revogação de um certificado pode ser diretamente tratada através da simples remoção do certificado do repositório. Isso implica que somente certificados válidos ficam armazenados e o serviço de autorização não é obrigado a checar se os certificados obtidos foram ou não revogados. Crampton e Khambhammettu [9] discutem com mais detalhes os prós e contras desses dois modelos e argumentam que embora o modelo *pull* precise fazer uma busca para obter as informações de autorização, ele é tão eficiente quanto o modelo *push*, já que este também precisa fazer uma consulta para estabelecer a validade do certificado por conta da possibilidade de revogação.

4.4 Suporte ao controle de acesso baseado em papéis

Controle de acesso baseado em papéis (*Role-Based Access Control* ou RBAC) [12] é uma forma de controle de acesso onde as decisões de autorização são tomadas com base nos papéis que os usuários desempenham em um determinado domínio. Por atribuir permissões a papéis e não a indivíduos, RBAC pode simplificar de maneira significativa o gerenciamento de controle de acesso para um número grande de usuários, já que o número de papéis que os usuários podem desempenhar é normalmente bem menor que o número de usuários em si.

Existem atualmente quatro modelos de RBAC [45], representados na Figura 4.9. O modelo básico, ou $RBAC_0$, define formalmente os conceitos de papéis, usuários, permissões, e sessões. Durante uma sessão, um usuário pode desempenhar um ou mais papéis, que geralmente representam posições profissionais tais como contador, diretor e gerente. O administrador do sistema atribui a cada papel um conjunto de permissões, que representam as ações que os papéis podem executar nos recursos protegidos, e depois atribui um ou mais papéis a cada usuário real. Quando um usuário faz acesso a um recurso, o serviço de autorização deve utilizar os papéis desse usuário para determinar o conjunto de privilégios do mesmo e então decidir se o acesso deve ou não ser permitido de acordo com a política de controle de acesso vigente.

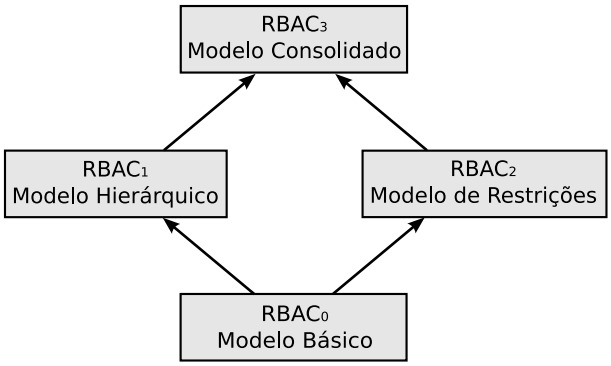


Figura 4.9: Relação entre os modelos RBAC

O modelo hierárquico, ou $RBAC_1$, é uma extensão mais sofisticada do modelo básico que permite que um papel estenda outros papéis e herde o seu conjunto de permissões. Assim, por exemplo, um papel gerente pode estender o papel funcionário para indicar que todos os privilégios alocados a um funcionário também se aplicam a um gerente, mesmo que isso não esteja explicitamente descrito. Esse modelo simplifica o gerenciamento de privilégios porque permite que privilégios comuns a vários papéis sejam associados a um papel-pai, eliminando a duplicação desse tipo de informação.

O modelo de restrições, ou $RBAC_2$, é uma outra extensão do modelo básico, embora a numeração possa sugerir que ele seja uma extensão do modelo hierárquico. O $RBAC_2$ permite a implantação de restrições que ajudam a definir uma política de segurança de alto nível. O exemplo mais comum de restrição é declarar que certos papéis são mutuamente exclusivos, estabelecendo que um dado usuário não pode ser associado a mais de um desses papéis. Outro tipo comum de restrição é a de cardinalidade, que permite limitar tanto o número de pessoas que podem estar associadas a um papel quanto o número de papéis que uma pessoa pode ter.

O modelo consolidado, ou $RBAC_3$, combina os modelos hierárquico e de restrições em um

único modelo. O $RBAC_3$ permite que restrições sejam aplicadas à própria hierarquia de papéis, como, por exemplo, limitar o número de super (ou sub) papéis que um dado papel pode ter, ou impedir que dois ou mais papéis descendam de um papel em comum. Esse tipo de restrição permite que o gerenciamento da hierarquia de papéis seja delegado para uma autoridade externa de uma forma controlada, preservando a política de segurança estabelecida.

A Figura 4.10 ilustra como os conceitos definidos pelos quatro modelos de RBAC se relacionam.

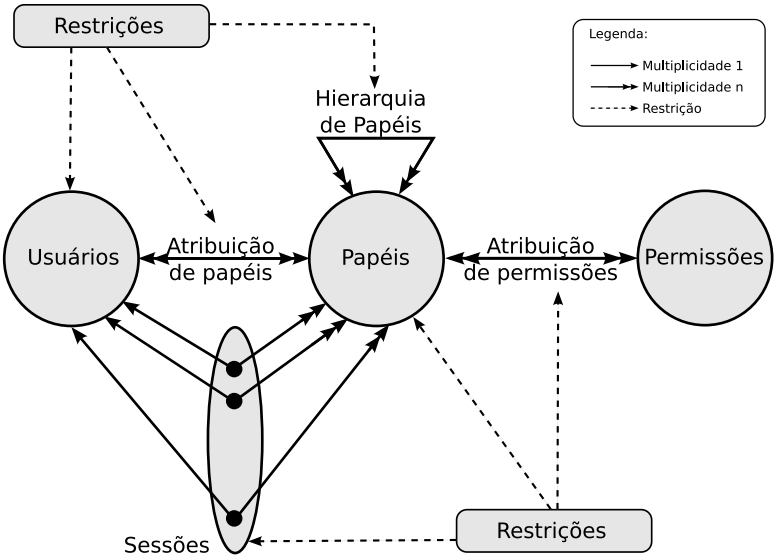


Figura 4.10: Relações entre os conceitos definidos pelos modelos RBAC

É importante salientar que o mecanismo de autorização proposto pelo RBAC especifica formalmente os modelos que podem ser seguidos, mas não especifica como a política de controle de acesso deve ser descrita, como os privilégios devem ser associados aos papéis, nem como os papéis devem ser associados aos usuários. Implementações que seguem os modelos descritos são livres para definir esses aspectos da forma que for mais conveniente.

Chadwick et al. [7] analisa como a IGP oferece suporte a RBAC, concluindo que:

- o modelo simples, ou $RBAC_0$, é comportado diretamente pela IGP através de dois tipos de certificados de atributos X.509: os certificados de especificação de papéis, que possuem os privilégios de cada papel como atributos (por exemplo, papel gerente pode aprovar créditos) e os certificados de associação de papéis, que possuem os papéis de cada usuário como atributo.

- o modelo hierárquico também é contemplado pela IGP fazendo com que as autoridades

de atributo coloquem tanto papéis quanto permissões nos certificados de especificação de papéis. Dessa forma, um papel pode ter outros papéis como atributo, herdando os privilégios desses papéis em adição aos privilégios diretamente atribuídos a ele em seu certificado.

- o modelo restrito é contemplado de forma limitada pela IGP. Extensões definidas pela RFC 3281 comportam algumas restrições, como, por exemplo, especificar sobre quais alvos os atributos de um CA X.509 podem ser aplicados. Outros tipos de restrições, como exclusão mútua de papéis, exigem mecanismos externos para verificá-las.

Com isso, aplicações que são baseadas nos modelos simples e hierárquico encontram nos certificados de atributos X.509 estruturas ideais para implementar seus mecanismos de controle de acesso. Já aquelas baseadas no modelo restrito podem necessitar de estruturas adicionais para oferecer suporte aos tipos de restrições desejadas. Como exemplo do primeiro caso, podemos citar o objeto de estudo desse projeto, o mecanismo de autorização de servidores de aplicação Java EE. As especificações de segurança da plataforma Java EE, descritas no Capítulo 3, se baseiam no modelo simples, não exigindo suporte a hierarquias de papéis ou restrições, o que torna o uso dos certificados de atributos certamente adequado para implementar as exigências de tais especificações.

5 A implementação do serviço de autorização baseado em certificados de atributos X.509

Neste trabalho desenvolvemos um mecanismo de autorização para um servidor de aplicações Java EE baseado em certificados de atributos X.509. Tal mecanismo foi integrado à infra-estrutura de segurança do JBoss, estendendo o conjunto de funcionalidades já oferecidas por essa infra-estrutura e oferecendo um modelo de autorização baseado na IGP.

A escolha do JBoss para a implementação desse mecanismo se deu por duas razões principais: em primeiro lugar, trata-se de um servidor de aplicações de código aberto já bem estabelecido na comunidade Java, o que permite que as contribuições resultantes deste trabalho beneficiem um grande número de desenvolvedores. Em segundo lugar, já possuíamos uma certa experiência de desenvolvimento neste servidor, o que facilita a implementação e integração de nosso trabalho.

Além do JBoss, outro software de código aberto foi largamente utilizado neste trabalho. Trata-se do projeto Bouncycastle [60], uma iniciativa que implementa diversas *APIs* relacionadas à segurança e criptografia. Em particular, o projeto Bouncycastle oferece implementações das estruturas ASN.1 que formam os certificados do padrão X.509, uma biblioteca para codificação e decodificação dessas estruturas, geradores de certificados X.509 de atributos e de chaves públicas, geradores de listas de certificados revogados e processadores de requisições e respostas OCSP, além de uma biblioteca para criptografia em Java.

Como nosso trabalho estende a infra-estrutura de segurança do JBoss, é interessante apresentarmos esta infra-estrutura antes de detalharmos a implementação do nosso projeto.

5.1 Infra-estrutura de segurança do JBoss

Esta seção apresenta a infra-estrutura de segurança do JBoss. Primeiramente, uma visão geral da arquitetura e dos principais componentes dessa infra-estrutura é apresentada. Em seguida, demonstramos como o JBoss faz uso da JAAS para autenticar e autorizar os usuários. Por fim, descrevemos brevemente como aplicações-cliente que acessam diretamente a camada de negócios (EJBs) podem se autenticar junto ao servidor.

É importante destacar que a arquitetura e os componentes exibidos nesta seção são referentes à versão 4.x do JBoss. A partir da versão 5.0 (que no presente momento se encontra em estágio *beta* de lançamento) algumas mudanças foram feitas na arquitetura de segurança do servidor, tornando-a mais flexível e coesa. Nossa escolha pela implementação do serviço de autorização na versão 4.x (mais especificamente a versão 4.0.2) se justifica pela estabilidade desta versão em comparação à versão 5.0 do JBoss, que ainda era muito incipiente quando do início de nosso trabalho.

5.1.1 Visão geral da arquitetura

No JBoss, a infra-estrutura responsável pela implementação dos serviços de segurança e pela aplicação das regras de autenticação e autorização especificadas pela plataforma Java EE é a *JBoss Security Extension*, ou JBoss SX. O projeto JBoss SX estabelece um arcabouço de segurança definido em termos de algumas interfaces-chave, cuja implementação se utiliza da JAAS para fornecer as funcionalidades exigidas. A Figura 5.1 exibe as principais classes e interfaces envolvidas nos processos de autenticação e autorização.

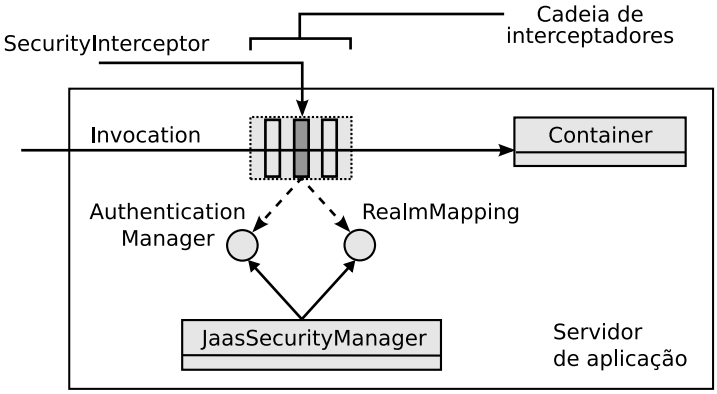


Figura 5.1: Componentes da infra-estrutura de segurança do JBoss

De maneira geral, quando um cliente faz uma chamada para um componente do servidor, essa chamada passa por uma cadeia de interceptadores que está associada ao contêiner que gerencia o componente requisitado. Cada interceptador é responsável pela execução de algum tipo de serviço antes da chamada ser encaminhada ao contêiner. Entre os interceptadores normalmente configurados encontramos o `SecurityInterceptor`, que é o componente central da infra-estrutura de segurança do JBoss. Cabe a este interceptador verificar se o cliente que originou a chamada foi autenticado e se este possui os papéis necessários para que o acesso ao componente seja concedido.

O `SecurityInterceptor` interage com duas interfaces para autenticar e autorizar clientes. A primeira, `AuthenticationManager`, é responsável pela autenticação em si. Ela define o método `isValid(Principal identity, Object credential)`, que verifica se a credencial fornecida é ou não um comprovante válido da identidade do cliente. A segunda interface, `RealmMapping`, é responsável pelo mapeamento dos papéis da aplicação para os clientes do mundo real. Ela define o método `doesUserHaveRole(Principal identity, Set roles)`, utilizado para verificar se o cliente possui algum dos papéis necessários para que o seu acesso seja autorizado.

Há ainda um segundo interceptador envolvido com segurança: o `SecurityProxyInterceptor`. Este interceptador permite que regras de controle de acesso adicionais àquelas especificadas pela plataforma Java EE sejam executadas através de um `SecurityProxy`, uma classe que implementa as novas regras e que é chamada pelo interceptador para a execução dessas regras. Uma aplicação pode, por exemplo, implementar um `SecurityProxy` que especifica que o acesso a um determinado componente seja apenas concedido em determinados horários do dia. Este é um exemplo de regra que não pode ser expressa através dos descritores de implantação definidos pela plataforma Java EE.

A classe `JaasSecurityManager` fornece uma implementação baseada na JAAS para as interfaces `RealmMapping` e `AuthenticationManager`. O modelo de autenticação de JAAS se baseia no conceito de módulos configuráveis de autenticação. Cada módulo JAAS implementa um mecanismo próprio de autenticação e aplicações podem escolher os mecanismos a serem utilizados através da especificação de um ou mais módulos JAAS em um arquivo de configuração. Por serem configuráveis, os módulos JAAS permitem que aplicações autentiquem seus clientes de forma transparente e independente do mecanismo concreto de autenticação, que pode ser substituindo no arquivo de configuração sem nenhuma necessidade de recompilação da aplicação.

5.1.2 Processos de autenticação e autorização

O processo de autenticação começa quando o `SecurityInterceptor` intercepta uma chamada de um cliente não-autenticado para um componente. As chamadas são representadas por objetos que contém informações a respeito do componente, como o nome e parâmetros do método a ser executado, e também informações a respeito do cliente, como sua identidade e credenciais. O `SecurityInterceptor` dá início à autenticação chamando o método `isValid` da classe `JaasSecurityManager` passando os dados de segurança obtidos da chamada. O processo de autenticação JAAS, ilustrado pela Figura 5.2, ocorre da seguinte forma:

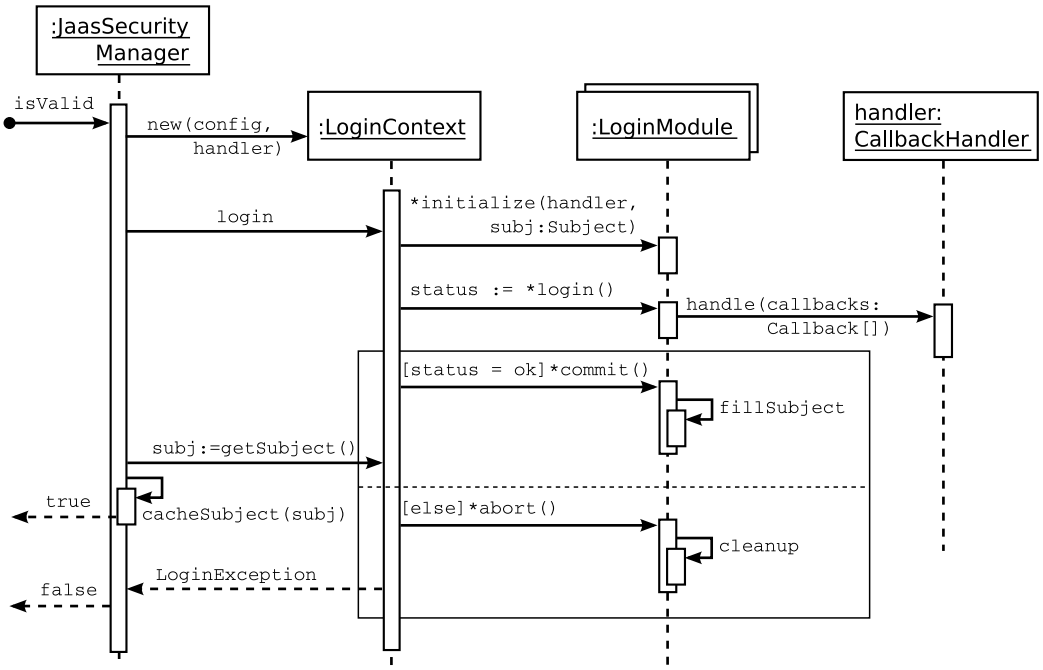


Figura 5.2: Processo de autenticação implementado por `JaasSecurityManager`

1. O método `isValid` cria uma instância de `LoginContext`, fornecendo o nome da configuração JAAS a ser utilizada e um objeto `CallbackHandler`. Esse último é o responsável pela interação com o servidor para obter as informações necessárias para que os módulos autentiquem o cliente. No caso do JBoss, o `CallbackHandler` é instanciado e preenchido pelo método `isValid` com as informações de segurança obtidas pelo `SecurityInterceptor`.

2. Ao ser criado, o `LoginContext` usa o nome da configuração para determinar todos os módulos a serem utilizados para autenticar o cliente. Os módulos, que são implementações da interface `LoginModule`, são então instanciados e inicializados pelo `LoginContext`. Durante a inicialização, cada módulo recebe, entre outras coisas, uma referência para o `CallbackHandler` e uma referência para um objeto `Subject` que representa o cliente sendo autenticado.

3. O método `isValid` executa o método `login` do objeto `LoginContext`, fazendo com que cada um dos módulos carregados execute o seu próprio método `login` para autenticar o cliente. Para obter as informações do cliente que foram extraídas da chamada, um módulo deve fornecer os objetos `Callback` apropriados para o método `handle` do `Call-`

`backHandler`. Por exemplo, para obter a identidade do cliente, um `Callback` do tipo `NameCallback` deve ser passado para o `CallbackHandler`. Já para obter as credenciais do mesmo, um `ObjectCallback` deve ser utilizado. Cada `Callback` é preenchido pelo método `handle` de acordo com o seu tipo.

4. Uma vez que o módulo obtém os objetos `Callback` preenchidos do `CallbackHandler`, as informações extraídas são utilizadas para realizar a autenticação. Quando todos os módulos terminam de executar o método `login`, o `LoginContext` avalia o resultado do processo de autenticação. Em caso de sucesso, o método `commit` de cada `LoginModule` é invocado para que o módulo associe a identidade autenticada e os papéis do cliente ao `Subject` recebido na inicialização. Em caso de fracasso, o método `abort` de cada módulo é executado, dando uma oportunidade para os módulos limparem qualquer estado salvo anteriormente.

Ao final do processo de autenticação, o `JaasSecurityManager` armazena o `Subject` resultante em um *cache* em caso de sucesso e devolve o resultado para o `SecurityInterceptor`. Se a autenticação tiver falhado, o interceptador lança uma exceção e impede que a chamada chegue ao contêiner. Caso contrário, o `SecurityInterceptor` dá início ao processo de autorização, obtendo junto ao descritor de implantação os papéis que têm permissão para executar a operação no componente e chamando o método `doesUserHaveRole` do `JaasSecurityManager` para determinar se o cliente possui algum desses papéis (Figura 5.3).

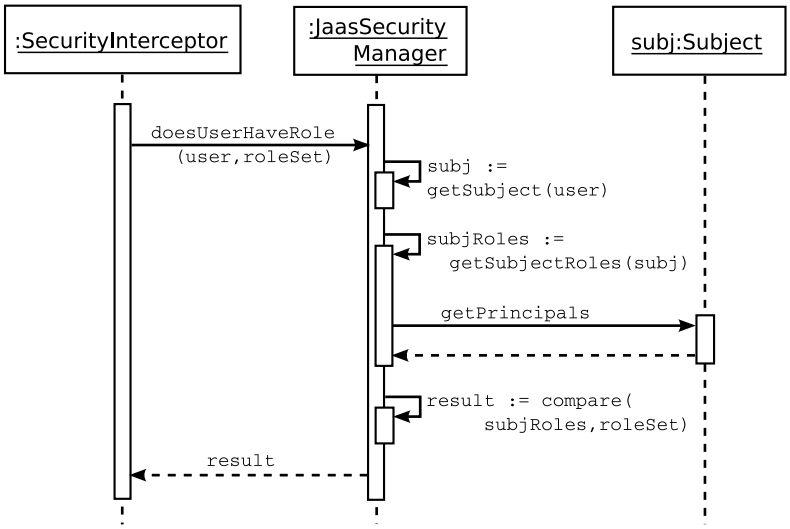


Figura 5.3: Processo de autorização no JBoss SX

O `JaasSecurityManager` primeiro obtém o `Subject` que representa esse cliente e depois procura pelo grupo `Roles` dentro desse `Subject` através de uma chamada ao método `getPrincipals`. Esse grupo contém os papéis dos usuários e é inserido no `Subject` pelos `LoginModule`s como parte da execução método `commit` do processo de autenticação. De posse do grupo, o `JaasSecurityManager` obtém os papéis atribuídos ao cliente e compara esse conjunto ao conjunto de papéis fornecidos pelo `SecurityInterceptor`. Uma intersecção não-vazia dos dois grupos indica que o cliente possui ao menos um dos papéis exigidos para que o acesso ao componente seja autorizado.

5.1.3 Módulos de autenticação do JBoss SX

O JBoss oferece implementações de diversos `LoginModule`s. A peculiaridade é que todos os módulos do JBoss, além de realizarem autenticação, são também capazes de associar papéis às identidades dos clientes. Isso é feito no método `commit` de cada módulo, onde o conjunto de papéis atribuídos ao cliente autenticado é obtido de forma específica pelo módulo e associado ao `Subject` juntamente com a identidade do cliente.

Aplicações configuram os módulos de autenticação no arquivo `login-config.xml`. Cada módulo é configurado com o nome da classe que o implementa, um atributo `flag` indicando a importância do módulo no processo global de autenticação, e um conjunto de propriedades ou opções que são usadas para ajustar o comportamento do módulo. A Figura 5.4 exibe um exemplo de configuração.

```
<application-policy name="testapp">
    <authentication>
        <login-module code="org.jboss.security.auth.spi.UsersRolesLoginModule"
                      flag="required">
            <module-option name="password-stacking">useFirstPass</module-option>
            <module-option name="usersProperties">users.props</module-option>
            <module-option name="rolesProperties">roles.props</module-option>
        </login-module>
    </authentication>
</application-policy>
```

Figura 5.4: Exemplo de configuração de um módulo

O exemplo ilustra a configuração de segurança para a aplicação com o nome de `testapp` e define um único módulo de segurança, o `UsersRolesLoginModule`. O atributo `flag` com valor `required` indica que o módulo precisa ter sucesso na autenticação para que o processo todo seja considerado um sucesso. Além disso, as propriedades `password-stacking`, `usersProperties` e `rolesProperties` são configuradas para o módulo.

Os principais módulos de autenticação oferecidos atualmente pelo JBoss SX para configura-

ção da infra-estrutura de segurança das aplicações implantadas são:

- `UsersRolesLoginModule`: realiza autenticação do usuário comparando as credenciais fornecidas com as configuradas em um arquivo de propriedades. A propriedade `usersProperties` é utilizada para especificar o arquivo que contém as identidades e as senhas dos clientes. Outro arquivo de propriedades, identificado pela propriedade `rolesProperties`, é utilizado para obter os papéis do cliente autenticado.

- `DatabaseServerLoginModule`: busca pelas credenciais e papéis do usuário em um banco de dados. As propriedades `principalsQuery` e `rolesQuery` são utilizadas para indicar ao módulo quais são as *queries SQL* que devem ser executadas para localizar as credenciais e os papéis do cliente, respectivamente.

- `LdapLoginModule`: autentica o usuário em um serviço de diretório LDAP. Diversas propriedades são definidas para estabelecer conexão com o servidor LDAP, construir o *distinguished name* do usuário a partir da sua identidade e indicar o contexto que deve ser usado para procurar pelos papéis do cliente. Como papéis não são facilmente armazenados em serviços de diretório por não existirem atributos específicos para isso, esse módulo é geralmente configurado em conjunto com algum outro módulo responsável por obter os papéis dos clientes.

- `BaseCertLoginModule`: esse módulo assume que a autenticação foi feita pela camada de transporte e apenas disponibiliza o CCP do cliente para os demais módulos configurados. Esse módulo associa um conjunto vazio de papéis ao `Subject` (isto é, ele não tem capacidade para estabelecer a associação entre um cliente e seus papéis). Por isso, é necessário que outros módulos sejam usados em conjunto com o `BaseCertLoginModule` para obter os papéis atribuídos aos clientes.

Além dos módulos pré-disponibilizados, aplicações podem implementar seus próprios módulos de autenticação JAAS, bastando para isso respeitar o contrato imposto pela infra-estrutura de segurança do JBoss, que exige que os papéis do cliente sejam associados ao `Subject` quando da execução do método `commit`.

5.1.4 Autenticação de clientes EJB

Conforme discutido no Capítulo 3, clientes se autenticam à camada *Web* através do mecanismo de autenticação configurado pela aplicação *Web*. Já clientes que interagem diretamente com a camada de negócios (EJB) precisam se autenticar utilizando um mecanismo próprio

do servidor para esse fim. No caso do JBoss, a autenticação de clientes de componentes EJB também é realizada com ajuda do JAAS.

O JBoss oferece um módulo de autenticação especial, batizado de `ClientLoginModule`, que é utilizado por aplicações que são clientes de componentes EJB para fornecer ao servidor identidade e as credenciais dos. De forma similar ao processo que acontece no servidor, a aplicação-cliente realiza a autenticação invocando o método `login` da classe `LoginContext` passando o `CallbackHandler` que será utilizado pelo `ClientLoginModule` para obter a identidade e as credenciais dos clientes. A diferença é que este módulo em si não executa nenhum tipo de autenticação. Ele apenas armazena as informações obtidas do `CallbackHandler` em variáveis `ThreadLocal`, de acordo com o padrão *Thread-Specific Storage* [46], para que elas sejam posteriormente inseridas no contexto de segurança de cada chamada que o cliente fizer para um componente EJB do servidor, onde a autenticação será realizada. Retomaremos esse assunto na Seção 5.4.2, que detalha o modelo *push* de propagação de certificados de atributos.

5.2 Visão geral do serviço de autorização baseado em certificados de atributos X.509

A implementação do serviço de autorização baseado em certificados de atributos X.509 pode ser divida nas seguintes partes:

1. Implementação de um gerador de certificados de atributos: a escassez de ferramentas para geração de certificados de atributos X.509 nos levou a implementar nossa própria solução e o resultado foi a criação de um arcabouço extensível para geração de certificados de atributos e listas de certificados revogados. Esse arcabouço é apresentado na Seção 5.3.

2. Extensão da infra-estrutura de segurança do JBoss: a integração de nosso serviço de autorização com a infra-estrutura de segurança do JBoss se dá através de dois módulos JAAS que são capazes de extrair os papéis dos clientes de certificados de atributos X.509. A Seção 5.4 discute os detalhes da solução implementada.

3. Implementação de verificadores de revogação: como nosso trabalho oferece suporte ao modelo *push* de propagação de certificados, a implementação de mecanismos capazes de verificar a revogação dos certificados apresentados pelos clientes se faz necessária. A Seção 5.5 apresenta os mecanismos desenvolvidos para lidar com essa situação.

4. Mapeamento de papéis: como os certificados de atributos podem ser gerados por autoridades remotas ao servidor de aplicações, é provável que os papéis definidos atribuídos

pelas autoridades aos clientes não correspondam diretamente aos papéis que são definidos pela aplicação implantada no servidor. Em outras palavras, é preciso estabelecer um mapeamento entre esses papéis, assunto tratado com maiores detalhes na Seção 5.6.

5.3 O gerador de certificados de atributos X.509

A primeira etapa do trabalho foi o desenvolvimento de uma ferramenta para geração dos certificados de atributos X.509. A princípio, o *Privilege Allocator* do PERMIS seria utilizado para esse fim, mas infelizmente essa ferramenta faz uso de um componente externo para codificar e decodificar os certificados gerados e tal componente não possui licença aberta para uso. Somando-se a isso, a dificuldade de se encontrar soluções alternativas nos levou a desenvolver o nosso próprio gerador, batizado de ACGen.

O resultado desta parte do trabalho foi a criação de um arcabouço extensível para geração e armazenamento de certificados de atributos X.509 e listas de certificados revogados. O ACGen organiza funcionalidades importantes em módulos intercambiáveis que implementam interfaces bem definidas, permitindo assim que implementações alternativas dos módulos sejam criadas e configuradas para a ferramenta.

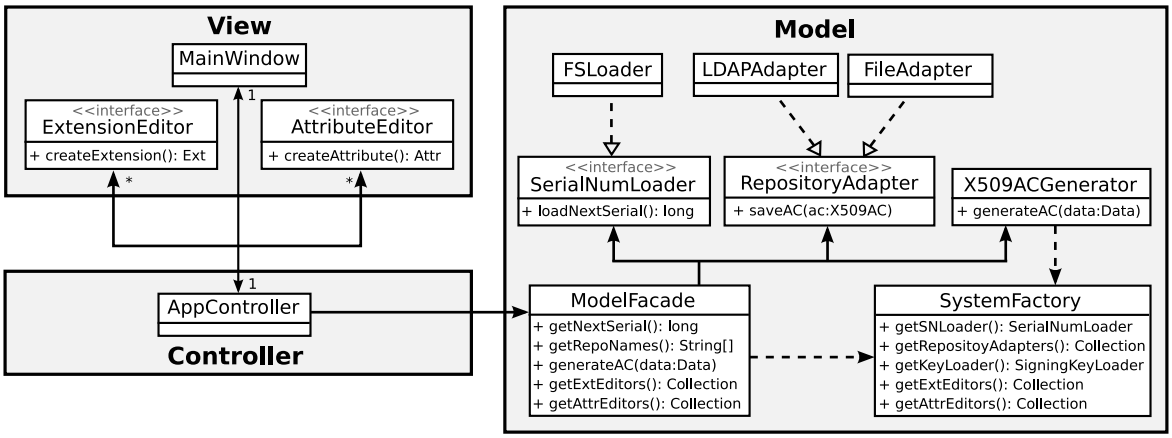


Figura 5.5: Arquitetura do arcabouço ACGen

A Figura 5.5 exibe as principais classes e interfaces do ACGen, organizadas conforme a arquitetura *MVC* [15]. A camada de apresentação é composta por três componentes: a janela principal, com a qual o usuário interage para gerar e armazenar os certificados de atributos, e os editores de atributos e de extensões, que são editores específicos para cada tipo de atributo ou extensão do certificado sendo gerado. Tanto a janela principal quanto os editores são

gerenciados por um controlador central, o `AppController`, que é responsável pelo gerenciamento da comunicação entre a camada de apresentação e o modelo. Essa comunicação é feita através de uma fachada, a `ModelFacade`, responsável por intermediar as requisições feitas pelo controlador para os componentes do modelo.

Os módulos extensíveis do ACGen podem ser identificados pelas interfaces exibidas na Figura 5.5. São eles:

- `AttributeEditor`: define a interface que deve ser implementada por editores de atributo. Como os atributos têm sintaxes diferentes, é preciso que exista um editor especializado para cada tipo de atributo. Cada implementação oferece uma interface gráfica com os campos apropriados para a geração de um tipo específico de atributo.

- `ExtensionEditor`: assim como ocorre com os atributos, cada extensão de um certificado de atributos também possui uma sintaxe própria e requer que um editor especializado seja responsável pela sua criação. Implementações dessa interface oferecem um editor gráfico com os campos apropriados para a geração de cada tipo de extensão.

- `SerialNumLoader`: define a interface que deve ser implementada por módulos responsáveis pelo gerenciamento do número de série dos certificados gerados. Uma implementação deve ser capaz de fornecer um número de série único para cada certificado de atributos através do método `loadNextSerial`.

- `RepositoryAdapter`. Define a interface que deve ser implementada por módulos responsáveis pelo armazenamento dos certificados de atributos gerados.

As implementações dos módulos acima citados são configuradas em um arquivo XML, o `acgen-config.xml`, que é lido em tempo de execução pela `SystemFactory` para instanciar os módulos do arcabouço. Um exemplo de configuração, onde as implementações dos módulos oferecidas por padrão pelo ACGen são utilizadas, pode ser encontrado na Figura 5.6.

A primeira seção do arquivo de configuração, `<issuer-credentials>`, descreve a localização e senha da *keystore* que contém a chave privada da autoridade de atributos que será utilizada para assinar os certificados gerados. Em seguida, a seção `<repository-modules>` ilustra os dois módulos que o ACGen oferece para o armazenamento dos certificados de atributos: o `FileAdapter`, que armazena o certificado gerado em um arquivo, e o `LDAPAdapter`, que armazena o certificado em um servidor LDAP. Note que, além do nome da classe, um módulo também pode especificar um conjunto de propriedades. No caso do `LDAPAdapter`, as propriedades especificadas indicam ao módulo como estabelecer uma conexão com o servidor LDAP para armazenar os certificados de atributos.

```
<acgen>
    <issuer-credentials>
        <keystore-file>keystore.p12</keystore-file>
        <keystore-password>examplepass</keystore-password>
    </issuer-credentials>
    <repository-modules>
        <module name="File System" class="acgen.model.repository.FileAdapter"/>
        <module name="LDAP" class="acgen.model.repository.LDAPAdapter">
            <!-- ldap server context factory and URL -->
            <module-property name="java.naming.factory.initial"
                             value="com.sun.jndi.ldap.LdapCtxFactory"/>
            <module-property name="java.naming.provider.url" value="ldap://localhost:389"/>
            <!-- ldap authentication information -->
            <module-property name="java.naming.security.authentication" value="simple"/>
            <module-property name="java.naming.security.principal"
                             value="cn=Manager,dc=example,dc=com"/>
            <module-property name="java.naming.security.credentials" value="secret"/>
        </module>
    </repository-modules>
    <attribute-editors>
        <module name="Role Attribute" class="acgen.view.editor.RolesAttributeEditor"/>
    </attribute-editors>
    <extension-editors>
        <module name="CRL Extension" class="acgen.view.editor.CRLExtEditor"/>
        <module name="OCSP Extension" class="acgen.view.editor.OCSPExtEditor"/>
    </extension-editors>
    <serialnumloader>
        <module name="File SN" class="acgen.model.FSLoader"/>
    </serialnumloader>
</acgen>
```

Figura 5.6: Arquivo de configuração do ACGen

Os editores de atributos são especificados na seção `<attribute-editors>`. O ACGen oferece uma implementação padrão (`RolesAttributeEditor`) de um editor de atributos que é utilizada para armazenar os papéis atribuídos ao titular do certificado. Similarmente, a seção `<extension-editor>` especifica os editores de extensão. A ferramenta oferece duas implementações de editores de extensões, ambas utilizadas para ajudar no processo de consulta de revogação dos certificados: o `CRLExtEditor`, que é utilizado para gerar uma extensão contendo a localização da lista de certificado revogados mantida pela autoridade de atributos, e o `OCSPExtEditor`, que é utilizado para gerar uma extensão contendo a localização de um serviço de consulta OCSP. Finalmente, a seção `<serialnumloader>` especifica o módulo responsável pelo fornecimento do número de série dos certificados. A implementação padrão incluída no ACGen armazena o último número utilizado em um arquivo e atualiza esse número sempre que um novo certificado é gerado.

É importante destacar que o ACGen oferece implementações-padrão dos módulos por uma comodidade apenas. Desenvolvedores são livres para criar seus próprios módulos e adicioná-los aos módulos já existentes ou simplesmente substituí-los de acordo com suas necessidades.

Outro aspecto que vale frisar é que o ACGen também é capaz de gerar listas de certificados revogados, oferecendo em sua interface gráfica uma área para o gerenciamento dos números dos certificados de atributos que devem ser incluídos nas LCRs geradas. O fato de não existirem ferramentas de código aberto com características similares faz do ACGen uma importante contribuição deste trabalho.

5.4 A extensão da infra-estrutura de segurança do JBoss

A arquitetura do JBoss SX, baseada nos módulos de autenticação do JAAS, nos levou a dividir a implementação do serviço de autorização em dois módulos JAAS, um implementando o modelo *pull* e outro o modelo *push* para obter os certificados de atributos dos clientes. Dois motivos principais nos levaram a essa decisão:

- Separação de responsabilidades: cada modelo exige configurações diferentes para encontrar os certificados do cliente e misturar os dois módulos em um só acabaria por criar um módulo inchado, com muitas responsabilidades.
- Maior flexibilidade no uso dos módulos: aplicações podem escolher qual modelo de distribuição de certificados é o mais adequado de acordo com os seus requisitos, podendo escolher apenas um dos modelos ou combinar a utilização dos dois.

Por conta disso, esta seção foi dividida em duas partes. A primeira apresenta como foi adicionado ao JBoss SX o suporte ao modelo *pull*, no qual os certificados de atributos são obtidos de um repositório. A segunda detalha a implementação do suporte ao modelo *push*, no qual os certificados de atributos acompanham as requisições enviadas pelo cliente.

5.4.1 Suporte ao modelo pull de distribuição de certificados

Com o objetivo de facilitar a compreensão da solução desenvolvida, a apresentação da integração do modelo *pull* será feita em três etapas. Na primeira, apresentamos uma visão geral da arquitetura, utilizando um diagrama de classes para auxiliar no entendimento. Na segunda, demonstramos como as classes discutidas interagem para obter os certificados de atributos dos clientes. Por fim, exibimos exemplos de configuração da solução desenvolvida.

5.4.1.1 Visão arquitetural

O componente central da implementação do modelo *pull* é o `X509ACPullLoginModule`, um módulo JAAS capaz de se conectar a um repositório externo para obter os certificados de

atributos X.509. A arquitetura da solução, exibida na Figura 5.7, foi idealizada de forma que o `X509ACPullLoginModule` pudesse interagir de forma transparente e configurável com diversos tipos de repositórios. Para atingir esse objetivo, a interação do módulo com um repositório se dá através de uma interface bem definida, batizada de `X509ACRepositoryAdapter`. Como o próprio nome sugere, essa interface e suas implementações seguem a arquitetura definida pelo padrão *Adapter* [15], que permite que uma classe interaja com implementações diferentes de uma mesma funcionalidade através de uma interface única. A implementação concreta de `X509ACRepositoryAdapter` é configurada como uma propriedade do `X509ACPullLoginModule` e instanciada em tempo de execução pela `X509ACSystemFactory`.

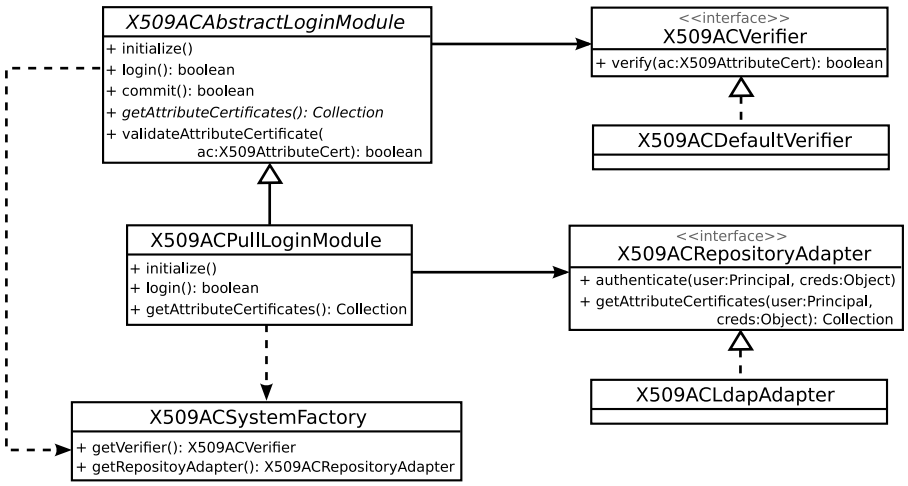


Figura 5.7: Diagrama de classes do modelo *pull*

A superclasse abstrata `X509ACAbstractLoginModule` implementa as funcionalidades que são comuns aos dois modelos de obtenção de certificados. Em particular, essa classe implementa as funcionalidades relacionadas à manipulação dos certificados de atributos X.509, como validação, extração de papéis, e associação dos papéis extraídos ao `Subject` ao final do processo de autenticação JAAS, conforme exigido pelo JBoss SX. O processo de validação dos certificados de atributos é delegado para um verificador externo, que implementa a interface `X509ACVerifier` e também é configurado como uma propriedade do módulo concreto (neste caso, o `X509ACPullLoginModule`).

Além de definir os módulos e as interfaces com as quais eles se relacionam, também desenvolvemos implementações-padrão para estas interfaces. O `X509ACLdapAdapter` fornece uma implementação de `X509ACRepositoryAdapter` que se comunica com um serviço de diretórios LDAP para obter os certificados de atributos X.509 dos clientes. Esta será a implementação

utilizada pelo `X509ACPullLoginModule` caso nenhum outro adaptador seja configurado explicitamente. Já a classe `X509ACDefaultVerifier` implementa um verificador que executa um conjunto mínimo de validações para que um certificado de atributos seja considerado válido. Mais especificamente, esta classe verifica a assinatura digital, o prazo de validade e o titular do certificado de atributos. As duas primeiras verificações são executadas para comprovar a integridade das informações contidas no certificado, enquanto que a última verifica se o certificado realmente pertence ao usuário que está sendo autenticado pelo `JaasSecurityManager`.

5.4.1.2 Obtenção dos certificados de atributos dos clientes

A interação entre as classes em tempo de execução pode ser melhor compreendida através do diagrama de colaboração exibido pela Figura 5.8. Podemos dividir esta interação em três momentos distintos, de acordo com a etapa do processo de autenticação JAAS sendo executada: a etapa de inicialização, onde o `X509ACPullLoginModule` obtém as referências para as classes que serão utilizadas, a etapa de autenticação, onde o módulo utiliza o `X509ACRepositoryAdapter` para autenticar o cliente, e a etapa de confirmação, onde o módulo obtém junto ao repositório os certificados de atributos do cliente e associa os papéis contidos nesses certificados ao `Subject` resultante.

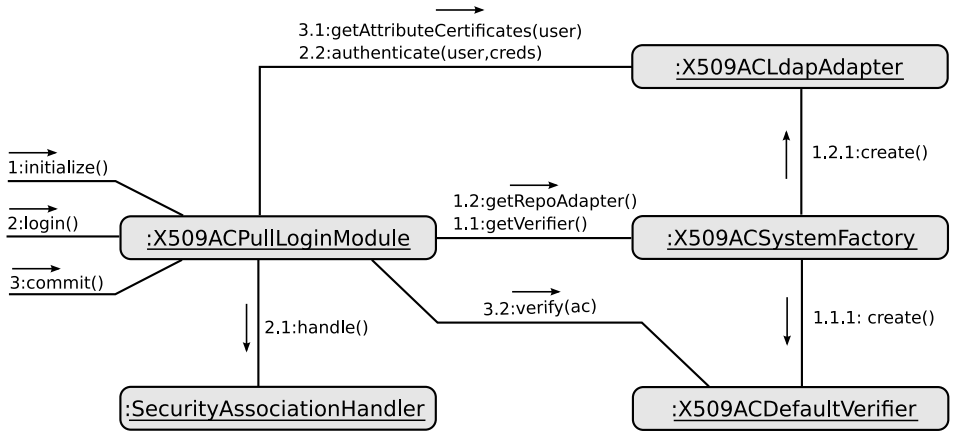


Figura 5.8: Colaboração entre as classes no modelo *pull*

A etapa de inicialização ocorre de acordo com os seguintes passos:

1. O ambiente de execução JAAS chama o método `initialize` de cada `LoginModule`, fornecendo, entre outras coisas, uma referência para `Subject`, o `CallbackHandler` a ser utilizado, e um mapa contendo as propriedades do módulo.

2. A implementação do método `initialize` do `X509ACPullLoginModule` primeiro chama o mesmo método na superclasse `X509ACAbstractLoginModule`. Este, por sua vez, obtém do mapa de opções o nome do verificador de certificados a ser utilizado e interage com a `X509ACSystemFactory` para obter uma referência para este verificador. A execução volta então para o método `initialize` do `X509ACPullLoginModule`, que procura no mapa de propriedades pelo nome do adaptador a ser utilizado para acessar o repositório. De posse do nome, o módulo chama o método `getRepositoryAdapter` da fábrica para obter uma referência para o adaptador concreto.

Uma vez que todos os módulos tenham sido inicializados, o processo de autenticação JAAS chama o método `login` em cada módulo para que a autenticação do cliente seja efetuada. No `X509ACPushLoginModule`, a autenticação do cliente envolve os seguintes passos:

1. Primeiro o módulo utiliza o `CallbackHandler` fornecido a ele durante a inicialização para obter a identidade e as credenciais do cliente. Isso é feito através do método `handle`, que preenche objetos do tipo `Callback` com as informações necessárias.

2. Depois de obter a identidade e as credenciais junto ao `CallbackHandler`, o módulo chama o método `authenticate` do `X509ACRepositoryAdapter` e fornece as informações obtidas para que o adaptador possa autenticar o cliente. A implementação do adaptador atua junto ao repositório para validar a identidade do cliente e devolve o resultado da autenticação para o `X509ACPullLoginModule`.

É importante observar que o processo de autenticação descrito pressupõe que o repositório contém informações suficientes para autenticar o usuário. Ou seja, o `X509ACPullLoginModule` pressupõe que o repositório que ele consulta para obter os certificados de atributos X.509 também possui as informações necessárias para autenticar os clientes. Na prática, entretanto, nem sempre esse é o caso, já que o repositório mantido pela autoridade de atributos pode conter apenas os certificados de atributos X.509 dos clientes, não dispondo assim de nenhum tipo de informação que possibilite a autenticação desses clientes. Quando isto acontece, nenhum cliente será autenticado com sucesso pelo módulo, e essa falha na autenticação implica também que nenhum cliente poderá fazer acesso aos componentes protegidos do servidor.

A solução para este problema consiste na configuração de outro módulo JAAS que seja capaz de autenticar os cliente em conjunto com o `X509ACPullLoginModule`. A presença de outro módulo que se responsabilize pela autenticação faz com o método `login` do `X509ACPullLoginModule` não seja executado. Nesta situação, nosso módulo atua apenas como um mecanismo

para obter os certificados de atributos X.509 dos clientes junto ao repositório representado pelo adaptador utilizado.

Quando a fase de autenticação termina com sucesso, dá-se início à fase de confirmação, onde o método `commit` de cada módulo é invocado pelo JAAS. Esta etapa se desenrola da seguinte maneira:

1. O método `commit` da superclasse `X509ACAbstractLoginModule` é executado, e este, por sua vez, chama o método abstrato `getAttributeCertificates` para obter os certificados de atributos X.509 do cliente.

2. O método `getAttributeCertificates` implementado pelo `X509ACPullLoginModule` é executado e chama o método de mesmo nome do `X509ACRepositoryAdapter` fornecendo a identidade do cliente.

3. A implementação do `X509ACRepositoryAdapter` utiliza a identidade do cliente para localizar os certificados do mesmo no repositório. Os certificados encontrados são devolvidos para o `X509ACPullLoginModule`.

4. De posse dos certificados, o método `commit` prossegue para a validação dos mesmos. A validação é feita passando cada um dos certificados obtidos para o método `verify` do `X509ACVerifier` configurado. Ao final do processo, os certificados considerados inválidos são descartados.

5. O método `commit` completa sua execução extraindo os papéis dos certificados válidos e associando esses papéis ao objeto `Subject` fornecido na etapa de inicialização.

Ao final do processo, o `JaasSecurityManager` armazena o `Subject` preenchido no *cache* e indica para o `SecurityInterceptor` que a autenticação foi concluída com sucesso. Como vimos na Seção 5.1, o processo de autorização do JBoss SX utiliza os papéis associados ao `Subject` mantido pelo `JaasSecurityManager` para decidir se o acesso do cliente ao componente desejado deve ou não ser concedido.

5.4.1.3 Exemplos de configuração

No JBoss, toda aplicação Java EE protegida é associada ao nome de uma configuração de segurança. Tal configuração é especificada no arquivo `login-config.xml` do servidor e define quais módulos JAAS devem ser utilizados para autenticar os clientes da aplicação, bem como o conjunto de propriedades associado a cada um dos módulos.

A Figura 5.9 a seguir apresenta um exemplo de configuração de segurança que utiliza o `X509ACPullLoginModule` no JBoss, demonstrando algumas das propriedades que são normalmente utilizadas para configurar o funcionamento do módulo.

```
<application-policy name="testapp">
    <authentication>
        <login-module code="org.jboss.security.auth.spi.X509ACPullLoginModule" flag="required">
            <!-- attribute certificate verifier settings -->
            <module-option name="verifier">
                    org.jboss.security.auth.certs.X509ACDefaultVerifier
            </module-option>
            <module-option name="securityDomain">java:/jaas/exampledomain</module-option>
            <!-- repository adapter settings -->
            <module-option name="repositoryAdapter">
                    org.jboss.security.auth.spi.X509ACLdapAdapter
            </module-option>
            <module-option name="java.naming.factory.initial">
                    com.sun.jndi.ldap.LdapCtxFactory
            </module-option>
            <module-option name="java.naming.provider.url">ldap://localhost:389</module-option>
            <module-option name="java.naming.security.authentication">simple</module-option>
            <module-option name="acHolderDNPrefix">cn=</module-option>
            <module-option name="acHolderDNSuffix">, o=example, dc=com</module-option>
        </login-module>
    </authentication>
</application-policy>
```

Figura 5.9: Exemplo de configuração do `X509ACPullLoginModule`

No exemplo, uma configuração de nome `testapp` especifica que o `X509ACPullLoginModule` deve ser utilizado para autenticar os clientes. O módulo em si é declarado através do elemento `<login-module>`, enquanto que as propriedades são especificadas através de elementos `<module-option>`. As propriedades do exemplo se encontram divididas em dois grupos principais: propriedades de configuração do verificador de certificados, e propriedades de configuração do repositório.

No primeiro grupo encontramos duas propriedades: `verifier` e `securityDomain`. A primeira especifica o nome da classe do verificador a ser instanciado, enquanto a segunda especifica a localização no *JNDI* [49] do `SecurityDomain` do servidor. O `SecurityDomain` é um componente do JBoss SX que é configurado com os certificados de chave pública das autoridades nas quais o servidor confia e é disponibilizado para outros componentes através do *JNDI*. Durante a validação de um certificado de atributos, o verificador utiliza o `SecurityDomain` para obter o CCP da autoridade de atributos que gerou o certificado a fim de validar a assinatura digital desse certificado.

No segundo grupo encontramos a propriedade `repositoryAdapter`, que especifica qual implementação de `X509ACRepositoryAdapter` deve ser utilizada pelo módulo para se comunicar com o repositório. Em seguida, temos as propriedades que configuram a conexão com o repo-

sitório. No caso particular do `X509ACLdapAdapter`, o acesso ao servidor LDAP é feito através de *JNDI*, logo as propriedades que definem como a conexão deve ser feita são as propriedades definidas pelo *JNDI* para este fim. Elas são facilmente identificadas no exemplo, pois são todas aquelas que começam com `java.naming`. Além das propriedades definidas pelo `JNDI`, o `X509ACLdapAdapter` ainda faz uso de outras duas propriedades especiais: `acHolderDNPrefix` e `acHolderDNSuffix`. Essas propriedades são utilizadas juntamente com a identidade do cliente fornecida pelo `X509ACPullLoginModule` para construir o *distinguished name* que identifica o contexto do cliente na árvore de diretórios LDAP. Dessa forma, quando o módulo solicita ao adaptador os certificados de atributos do cliente, o nome formado pela aplicação dessas propriedades à identidade do cliente é utilizado para encontrar o contexto do cliente no servidor `LDAP` e então extrair os certificados armazenados nesse contexto.

É interessante notar que, no exemplo, o `X509ACPullLoginModule` é o único módulo especificado para a configuração `testapp`. Isto significa que a configuração conta com o módulo para autenticar os clientes, ou seja, assume que o `X509ACLdapAdapter` é capaz de executar a autenticação junto ao servidor `LDAP`. No entanto, é possível que a autoridade de atributos mantenha os certificados em um servidor `LDAP` que não possui as informações necessárias para autenticar os clientes. Nesse caso, é preciso que outro módulo assuma a responsabilidade pelo processo de autenticação.

```
<application-policy name="testapp">
    <authentication>
        <login-module code="org.jboss.security.auth.spi.UsersRolesLoginModule"
                      flag="required">
            <module-option name="password-stacking">useFirstPass</module-option>
            <module-option name="usersProperties">users.props</module-option>
            <module-option name="rolesProperties">roles.props</module-option>
        </login-module>
        <login-module code="org.jboss.security.auth.spi.X509ACPullLoginModule" flag="required">
            <module-option name="password-stacking">useFirstPass</module-option>
            <!-- attribute certificate verifier settings -->
            <module-option name="verifier">
                    org.jboss.security.auth.certs.X509ACDefaultVerifier
            </module-option>
            .....
        </login-module>
    </authentication>
</application-policy>
```

Figura 5.10: Exemplo de configuração conjunta do `X509ACPullLoginModule`

A Figura 5.10 ilustra um exemplo onde o `X509ACPullLoginModule` é configurado em conjunto com o módulo `UsersRolesLoginModule` do JBoss. O primeiro aspecto a se notar no exemplo é a ordem dos módulos. Essa ordem é importante porque ela é respeitada pelo JAAS no processo de autenticação. O segundo aspecto é quanto ao uso da propriedade `password-`

`stacking` com valor `useFirstPass` em ambos os módulos. Um módulo com essa propriedade não executa a autenticação do cliente se algum outro módulo com a mesma propriedade já o fez anteriormente com sucesso e funciona apenas como um módulo de obtenção de papéis. É exatamente isso que acontece no exemplo ilustrado, onde o `X509ACPullLoginModule` funciona apenas como um mecanismo para obter os certificados de atributos dos clientes. A autenticação é realizada pelo `UsersRolesLoginModule`, que é executado antes pelo JAAS.

5.4.2 Suporte ao modelo push de distribuição de certificados

A implementação do suporte ao modelo *push* é um pouco mais complexa que aquela desenvolvida para lidar com o modelo *pull*, já que no modelo *push* os certificados de atributos precisam ser transportados do cliente para o servidor de aplicações, o que exige alterações na infra-estrutura de segurança para acomodar a passagem desses certificados. Em nosso trabalho, a implementação do modelo *push* contempla apenas clientes de aplicações EJB, não estando disponível para clientes *Web*. As principais razões para esta limitação são: *(i)* a complexidade em se adicionar suporte a certificados de atributos X.509 aos navegadores *Web*, e *(ii)* a dificuldade em se implementar a passagem dos certificados de atributos através do protocolo HTTP

Para que um cliente *Web* pudesse se utilizar do modelo *push*, mecanismos teriam que ser adicionados aos navegadores *Web* para possibilitar aos clientes a configuração dos seus certificados de atributos X.509 de forma que estes fossem transmitidos ao servidor quando necessário. Entretanto, a própria transmissão é um problema por si só, já que a passagem de certificados de atributos não é contemplada por nenhum protocolo *Web* hoje existente. Julgamos que a resolução destes problemas foge do escopo deste trabalho e por isso decidimos implementar o modelo *push* apenas para clientes de aplicações EJB, já que neste caso podemos alterar mais facilmente os mecanismos de transporte do JBoss para acomodar a passagem dos certificados de atributos junto com as demais informações de segurança dos clientes.

5.4.2.1 Propagação do contexto de segurança

A propagação das informações de segurança do cliente para o servidor, ilustrada na Figura 5.11, é feita em duas etapas distintas. Na primeira, o cliente fornece sua identidade e credenciais ao `ClientLoginModule`. Na segunda, o cliente faz uma chamada a um EJB, e as informações fornecidas anteriormente são incluídas na chamada para que o servidor possa autenticar e autorizar o cliente. De forma mais detalhada, o processo todo se desenrola como se segue:

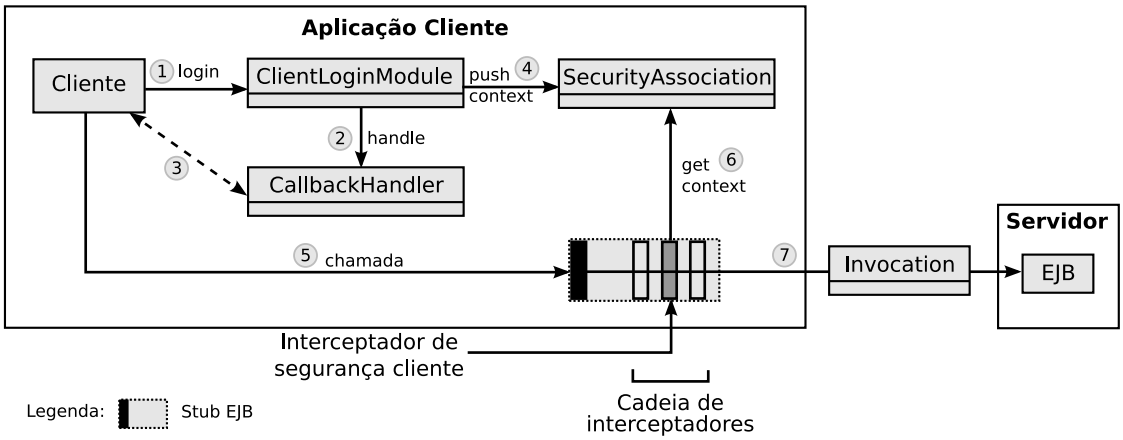


Figura 5.11: Propagação do contexto de segurança do cliente para o servidor

1. A aplicação cliente inicia um login JAAS e fornece o `CallbackHandler` que deve ser utilizado pelo `ClientLoginModule` para obter as informações do cliente.

2. O `ClientLoginModule` chama o método `handle` do `CallbackHandler` passando os objetos `Callback` que serão preenchidos pelo `CallbackHandler` com a identidade e as credenciais do cliente.

3. O `CallbackHandler` interage com o cliente e, após obter as informações necessárias, preenche os objetos `Callback` fornecidos pelo `ClientLoginModule`.

4. O `ClientLoginModule` extrai as informações dos `Callback`s e armazena essas informações em variáveis `ThreadLocal` na classe `SecurityAssociation`.

5. O cliente obtém no *JNDI* um `Stub`, que atua como um representante local de um EJB, e utiliza esse `Stub` para fazer uma chamada ao EJB.

6. O `Stub` cria um objeto do tipo `Invocation` contendo, entre outras coisas, o nome e os parâmetros do método a ser chamado no EJB implantado no servidor. Esse objeto passa então por uma cadeia de interceptadores no próprio `Stub`. Um desses interceptadores, o `SecurityInterceptor`, obtém a da classe `SecurityAssociation` a identidade e as credenciais do cliente.

7. O `SecurityInterceptor` insere as informações obtidas no contexto de segurança do objeto `Invocation`, que é então despachado para o servidor.

Ao chegar no servidor, o objeto `Invocation` passa pela cadeia de interceptadores do contêiner EJB. O interceptador de segurança do contêiner extrai o contexto de segurança desse objeto e utiliza as informações do contexto para autenticar e autorizar o cliente, conforme apresentado na seção 5.1.

Para permitir a passagem dos certificados de atributos do cliente para servidor como parte do contexto de segurança, algumas alterações nas classes apresentadas acima foram necessárias. Em primeiro lugar, implementamos o `X509ACCallback`, um novo tipo de `Callback` que é preenchido pelo `CallbackHandler` com os certificados de atributos do cliente. Alteramos então o `ClientLoginModule` para que este se utilizasse do novo `Callback` para obter os certificados de atributos junto ao `CallbackHandler` da aplicação-cliente. É importante destacar que o processo todo só funciona se o `CallbackHandler` utilizado pela aplicação-cliente for capaz de tratar o `X509ACCallback`.

Em seguida, tivemos que alterar a classe `SecurityAssociation`, adicionando uma nova variável `ThreadLocal` para o armazenamento dos certificados obtidos pelo `ClientLoginModule` através do `X509ACCallback`. Com isso, concluímos o processo de obtenção e armazenamento local dos certificados de atributos do cliente. O próximo passo consistiu em inserir esses certificados no contexto de segurança da invocação. Para isso, adicionamos um campo no contexto de segurança do objeto `Invocation`, e alteramos o `SecurityInterceptor` para que este preenchesse esse novo campo com os certificados de atributos obtidos na classe `SecurityAssociation`.

Todas essas pequenas alterações permitiram assim a propagação dos certificados de atributos como parte do contexto de segurança da invocação sem causar mudanças significativas no processo de propagação do contexto de segurança.

5.4.2.2 Visão arquitetural

A arquitetura do modelo *push* no servidor, exibida na Figura 5.12 foi idealizada de forma que a obtenção dos certificados de atributos transmitidos pelo cliente fosse feita da forma mais limpa e simples possível pelo novo módulo, o `X509ACPushLoginModule`.

Para isso, algumas alterações no lado servidor foram necessárias. Em primeiro lugar, foi preciso alterar o `SecurityInterceptor` do servidor para este extraísse os certificados de atributos do objeto `Invocation` e fornecesse esses objetos para o `JaasSecurityManager` quando da chamada ao método `isValid`. Por isso, adicionamos também um parâmetro adicional neste método para permitir ao `SecurityInterceptor` passar os certificados de atributos juntamente com a identidade e as credenciais do cliente.

Quando o método `isValid` é executado, o `JaasSecurityManager` cria uma instância de

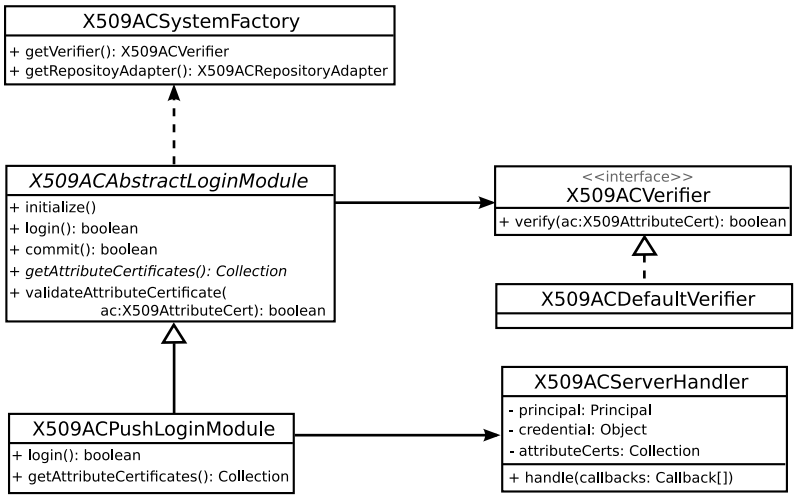


Figura 5.12: Diagrama de classes do modelo *push*

`CallbackHandler` e preenche essa instância com as informações de segurança fornecidas como parâmetros para o método. Para permitir a inclusão dos certificados de atributos, criamos um novo *handler*, chamado de `X509ACServerHandler`, que mantém os certificados de atributos e os fornece para o `X509ACPushLoginModule` através de um `X509ACCallback`.

Com isso, tudo que o `X509ACPushLoginModule` precisa fazer para obter os certificados de atributos do cliente é chamar o método `handle` da classe `X509ACServerHandler` passando um objeto do tipo `X509ACCallback`, da mesma forma que os demais módulos fazem para obter a identidade e credenciais fornecidas pelo cliente.

5.4.2.3 Visão de processos

Assim como no modelo *pull*, a interação entre o `X509ACPushLoginModule` e as demais classes da arquitetura se divide em três momentos: inicialização do módulo, autenticação do cliente, e obtenção dos papéis do cliente nos certificados de atributos X.509. O diagrama de colaboração da Figura 5.13 exibe as mensagens trocadas entre as classes nesses três momentos.

A etapa de inicialização é totalmente delegada para a implementação do método `initialize` da superclasse `X509ACAbstractLoginModule`, que se comunica com a `X509ACSystemFactory` para obter uma referência para o verificador de certificados a ser utilizado. Assim como no modelo *pull*, a configuração do verificador concreto é feita através da propriedade `verifier` do `X509ACPushLoginModule`.

A autenticação, que começa com uma chamada ao método `login`, não é comportada pelo `X509ACPushLoginModule`. A implementação do método `login` neste módulo lança uma exceção

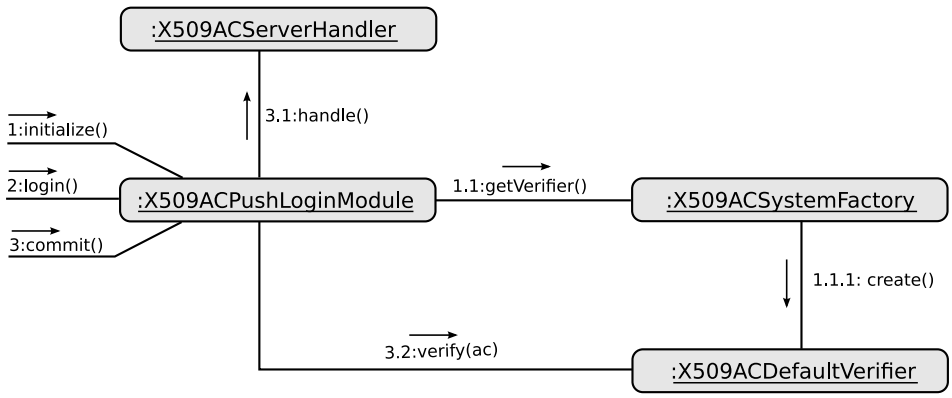


Figura 5.13: Colaboração entre as classes no modelo *push*

caso o usuário não tenha sido autenticado por outro módulo do JBoss. Isso acontece porque o `X509ACPushLoginModule` funciona apenas como um módulo de obtenção de papéis a partir dos certificados de atributos fornecidos pelo cliente e armazenados no `X509ACServerHandler`. A decisão de obrigar o uso de outro módulo para autenticar o cliente foi tomada porque o `X509ACPushLoginModule` não interage com nenhum repositório que contenha as informações necessárias para o processo de autenticação.

É importante destacar que, embora o uso deste módulo obrigue a configuração de um módulo adicional para tratar a autenticação dos clientes, a arquitetura da solução proposta é muito flexível, pois permite combinar o modelo *push* de distribuição de certificados com qualquer um dos mecanismos de autenticação fornecidos pelo JBoss. Além disso, essa solução permite que o `X509ACPushLoginModule` se especialize numa única função, que é a de obter os papéis do cliente a partir dos certificados de atributos transmitidos por esse cliente como parte da chamada a um componente protegido.

A terceira e última etapa, de confirmação, começa com o método `commit` de cada módulo sendo executado pelo JAAS. O processo que se desenrola é bastante similar ao que apresentamos no modelo *pull*:

1. O método `commit` da superclasse `X509ACAbstractLoginModule` chama o método abstrato `getAttributeCertificates` para obter os certificados de atributos do cliente.

2. O método `getAttributeCertificates` implementado pelo `X509ACPushLoginModule` é executado e chama o `X509ACServerHandler` passando uma instância de `X509ACCallback` como parâmetro do método `handle`.

3. O `X509ACServerHandler` preenche o `X509ACCallback` com os certificados de atributos

do cliente que foram passados a ele pelo `JaasSecurityManager`.

4. O método `getAttributeCertificates` extrai os certificados do `X509ACCallback` e os devolve para o método `commit` da superclasse.

5. De posse dos certificados, o método `commit` prossegue para a validação dos mesmos, que consiste em passar cada um desses certificados ao método `verify` do `X509ACVerifier` configurado. Ao final do processo, os certificados considerados inválidos são descartados.

6. O método `commit` completa sua execução extraindo os papéis dos certificados válidos e associando esses papéis ao objeto `Subject` fornecido na etapa de inicialização.

Assim, ao final do processo de autenticação JAAS, o objeto `Subject` resultante contém os papéis que foram extraídos dos certificados de atributos válidos fornecidos pelo cliente. Quando o `SecurityInterceptor` chama o método `doesUserHaveRole` do `JaasSecurityManager` para autorizar o cliente, esses mesmos papéis são utilizados para decidir se o acesso do cliente ao componente chamado deve ser concedido ou não.

5.4.2.4 Exemplo de configuração

A configuração do `X509ACPushLoginModule`, apesar de exigir a presença de outro módulo do JBoss para autenticar o cliente, é mais simples do que a configuração exigida pelo `X509ACPullLoginModule`, já que no modelo *push* não é necessário configurar o acesso a algum tipo de repositório. Um exemplo de configuração pode ser encontrado na Figura 5.14 a seguir.

```
<application-policy name="testapp">
    <authentication>
        <login-module code="org.jboss.security.auth.spi.UsersRolesLoginModule" flag="required">
            <module-option name="password-stacking">useFirstPass</module-option>
            <module-option name="usersProperties">users.props</module-option>
            <module-option name="rolesProperties">roles.props</module-option>
        </login-module>
        <login-module code="org.jboss.security.auth.spi.X509ACPushLoginModule" flag="required">
            <module-option name="password-stacking">useFirstPass</module-option>
            <!-- attribute certificate verifier settings -->
            <module-option name="verifier">
                    org.jboss.security.auth.certs.X509ACDefaultVerifier
            </module-option>
            <module-option name="securityDomain">java:/jaas/exampledomain</module-option>
        </login-module>
    </authentication>
</application-policy>
```

Figura 5.14: Exemplo de configuração do `X509ACPushLoginModule`

O primeiro aspecto a se notar no exemplo é a presença do `UsersRolesLoginModule` em conjunto com o `X509ACPushLoginModule`. Como o `X509ACPushLoginModule` não realiza a autenticação dos clientes e atua apenas como um módulo para obtenção de papéis, outro módulo do JBoss precisa ser configurado para tratar o processo de autenticação.

Quanto à configuração do `X509ACPushLoginModule` em si, podemos observar que esta é muito simples, bastando apenas especificar a implementação do verificador de certificados e a localização do `SecurityDomain` que será utilizado pelo verificador para obter o CCP da autoridade de atributos. Assim como no caso do modelo *pull*, essa configuração é feita através das propriedades `verifier` e `securityDomain`.

5.5 Implementação dos verificadores de revogação

A validação dos certificados de atributos apresentados pelos clientes no modelo *push* não pode se limitar à verificação da assinatura digital e de outros campos desses certificados. É preciso também garantir que os certificados apresentados não foram revogados pela autoridade de atributos após sua emissão. Isso é necessário porque, ao contrário do que ocorre no modelo *pull*, no modelo *push* a autoridade de atributos não tem a posse dos certificados que ela emite. Por isso, essas autoridades normalmente prevêem algum mecanismo para que aplicações obtenham os números dos certificados que foram revogados e não devem ser aceitos.

O mecanismo mais comumente utilizado pelas autoridades é a lista de certificados revogados, um documento assinado digitalmente que contém os números de série dos certificados revogados. A autoridade, ao gerar um certificado, adiciona uma extensão que contém a localização (normalmente uma URL) da lista de certificados revogados que ela mantém. Uma aplicação, ao validar um certificado, obtém a lista de certificados revogados a partir da localização contida na extensão apropriada e verifica se o número de série do certificado se encontra nesta lista.

O problema dessas listas é que elas são geradas periodicamente pela autoridade e, portanto, a revogação de um certificado não é efetivada imediatamente. Somente quando a lista for atualizada pela autoridade é que as aplicações tomarão conhecimento da revogação do certificado. Por isso, outros mecanismos foram propostos. A alternativa mais utilizada atualmente é o protocolo de consulta dinâmica OCSP. Através desse protocolo, aplicações fazem requisições em tempo real para um serviço que gerencia a revogação dos certificados. A localização desse serviço é incluída pela autoridade em uma extensão específica do certificado de atributos.

Tendo em mente a diversidade de mecanismos que podem ser utilizados para verificar se um certificado foi ou não revogado, desenvolvemos uma solução extensível que permite que

cada módulo JAAS especifique quais mecanismos devem ser executados durante a validação dos certificados de atributos. A arquitetura dessa solução, baseada no padrão *Chain of Responsibility* [15], é exibida na Figura 5.15.

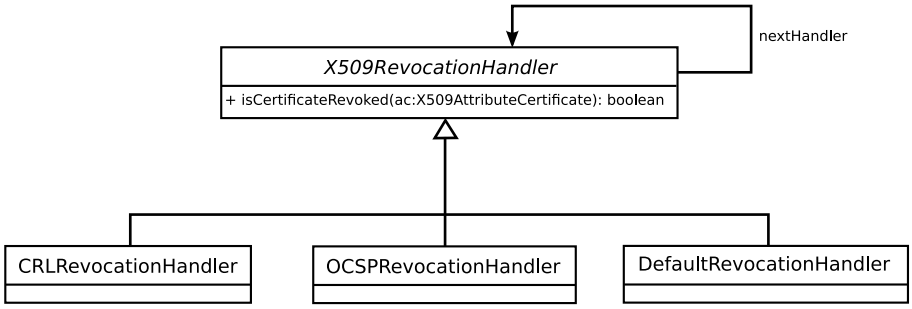


Figura 5.15: Arquitetura dos verificadores de revogação

A idéia consiste basicamente em tomar o certificado de atributos que se deseja validar e passá-lo por uma cadeia de verificadores de revogação. Cada elemento nessa cadeia se utiliza de um mecanismo específico para obter informações quanto à revogação do certificado. Caso um elemento tenha sucesso em determinar essas informações, o fluxo pela cadeia é interrompido e o resultado é devolvido. Caso contrário, o elemento passa o certificado para o próximo membro da cadeia. O último verificador na cadeia é um verificador-padrão que decide qual deve ser o veredito final caso nenhum dos verificadores anteriores tenham conseguido determinar se o certificado foi ou não revogado.

Todos os verificadores implementam a interface `X509RevocationHandler`, que define o método `isCertificateRevoked`. Como o nome sugere, um verificador concreto devolve `true` caso o certificado esteja revogado, e `false` caso o certificado seja válido. Como parte do trabalho, desenvolvemos três implementações de `X509RevocationHandler`, a saber:

- `CRLRevocationHandler`: procura no certificado de atributos pela extensão que contém a localização da lista de certificados revogados mantida pela autoridade de atributos. Caso tal extensão seja encontrada, o verificador obtém a lista de certificados revogados e verifica se o número de série do certificado sendo validado faz parte ou não da lista. Caso a extensão não seja localizada ou o verificador encontre algum problema ao tentar obter a LCR, o certificado de atributos é passado adiante para o próximo verificador da cadeia.

- `OCSPRevocationHandler`: procura no certificado de atributos pela extensão que contém a localização do serviço de consulta OCSP que deve ser utilizado. Caso tal extensão seja

encontrada, o verificador envia uma requisição OCSP ao serviço para determinar se o certificado foi revogado ou não. Caso contrário, o certificado é passado para o próximo verificador da cadeia.

- `DefaultRevocationHandler`: este verificador é sempre adicionado ao final das cadeias de verificadores de revogação pela `X509ACSystemFactory`. Ele define qual resultado deve ser devolvido pela cadeia caso nenhum dos verificadores configurados para o módulo seja capaz de determinar se o certificado de atributos foi ou não revogado após sua emissão.

5.5.1 Configuração da cadeia de verificadores

A configuração dos verificadores da cadeia, bem como a ordem em que eles devem ser executados, é feita através de um conjunto de propriedades do `X509ACPushLoginModule`.

```
<application-policy name="testapp">
    <authentication>
        <login-module code="org.jboss.security.auth.spi.UsersRolesLoginModule" flag="required">
            ....
        </login-module>
        <login-module code="org.jboss.security.auth.spi.X509ACPushLoginModule" flag="required">
            ....
            <!-- revocation handler chain -->
            <module-option name="revocationHandler1">
                    org.jboss.security.auth.certs.OCSPRevocationHandler
            </module-option>
            <module-option name="revocationHandler2">
                    org.jboss.security.auth.certs.CRLRevocationHandler
            </module-option>
            <module-option name="defaultHandlerBehavior">accept</module-option>
        </login-module>
    </authentication>
</application-policy>
```

Figura 5.16: Configuração do `X509ACPushLoginModule` com verificadores de revogação

A Figura 5.16 ilustra a configuração de uma cadeia contendo o `OCSPRevocationHandler` como primeiro verificador, e o `CRLRevocationHandler` como segundo verificador. Cada verificador é configurado através de uma propriedade `revocationHandler` seguida da posição desse verificador na cadeia. A propriedade `defaultHandlerBehavior` define qual deve ser o comportamento do `DefaultRevocationHandler` quando nenhum dos verificadores configurados conseguir determinar o estado dos certificados quanto à revogação.

5.5.2 Execução dos verificadores de revogação

A cadeia de interceptadores é carregada pela `X509ACSystemFactory` na fase de inicialização, ou seja, quando o método `initialize` da superclasse `X509ACAbstractLoginModule` é

chamado. A `X509ACSystemFactory` instancia os verificadores de revogação e constrói a cadeia de acordo com a ordem definida no arquivo de configuração, adicionando o `DefaultRevocationHandler` ao final da cadeia. Quando o `DefaultRevocationHandler` é criado, o valor da propriedade `defaultHandlerBehavior` é utilizado pela fábrica para definir qual deve ser o comportamento desse verificador. O valor `accept` indica que todo certificado de atributos que chegar nesse verificador deve ser considerado válido. Por outro lado, o valor `reject` indica que esses certificados de atributos devem ser descartados.

A execução dos verificadores de revogação se dá logo após os certificados de atributos terem sido validados pelo `X509ACVerifier` e antes de o `X509ACAbstractLoginModule` extrair os papéis do usuário. Cada certificado de atributos validado pelo `X509ACVerifier` é passado para o primeiro membro da cadeia de verificadores de revogação. Esse primeiro verificador tenta determinar o estado do certificado quanto à revogação e, em caso de fracasso, repassa o certificado ao próximo verificador. O processo continua até que um verificador seja capaz de dar um parecer, ou até que o certificado chegue no `DefaultRevocationHandler`, que decide se o certificado deve ser aceito ou não com base no valor definido para a propriedade `defaultHandlerBehavior` na configuração do módulo.

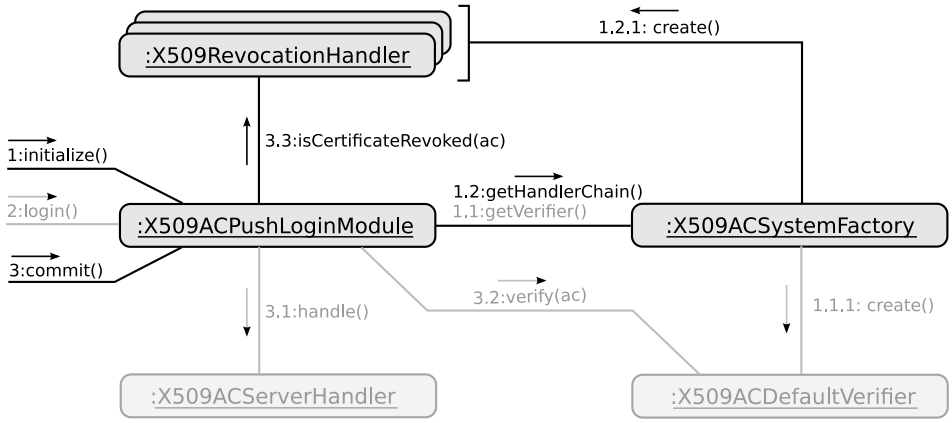


Figura 5.17: Colaboração entre as classes e os verificadores de revogação

O diagrama de colaboração da Figura 5.17 ilustra as interações entre as classes no modelo *push*, destacando as comunicações que ocorrem quando uma cadeia de verificadores é configurada para o módulo.

É importante destacar que, embora os verificadores de revogação sejam mais utilizados pelo `X509ACPushLoginModule`, a arquitetura desenvolvida permite também sua configuração por parte do `X509ACPullLoginModule`, o que pode ser bastante útil de acordo com os requisitos de segurança das aplicações implantadas no servidor. Isto é possível porque tanto o processo de

criação da cadeia de verificadores quanto o processo de validação dos certificados de atributos são realizados pela superclasse `X509ACAbstractLoginModule`, que é comum a ambos os módulos. Assim, todo comportamento implementado pela superclasse está disponível para ambos os modelos de propagação de certificados através das mesmas propriedades de configuração.

5.6 Mapeamento de papéis

Autoridades de atributos gerenciam os privilégios dos usuários de forma totalmente independente das aplicações que fazem uso desses privilégios. Em um ambiente Java EE, isso significa que os papéis dos usuários atribuídos pelas autoridades de atributos podem não corresponder exatamente ao conjunto de papéis definido por uma aplicação específica. Por conta disso, desenvolvemos um mecanismo simples que permite especificar como os papéis atribuídos pelas autoridades se relacionam com os papéis definidos pela aplicação.

A parte central desse mecanismo é um arquivo de configuração XML que é utilizado para descrever como os papéis emitidos por uma certa autoridade devem ser traduzidos para os papéis definidos pela aplicação. Em tempo de execução, os mapeamentos descritos pelo arquivo são aplicados aos papéis extraídos dos certificados de atributos do cliente antes que esses papéis sejam associados ao `Subject` pelo `X509ACAbstractLoginModule`.

A Figura 5.18 exibe um exemplo do arquivo de configuração de mapeamentos de papéis.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mappings SYSTEM "role-mappings.dtd">

<mappings default-mapping="passthrough">
    <mapping authority-dn="CN=Test Auth, OU=Test Unit, O=Test ORG, ST=NY, C=US">
        <role-mapping from-role="RoleA" to-role="RoleB"/>
        <role-mapping from-role="RoleC" to-role="RoleM"/>
        <role-exclusion role-name="RoleW"/>
    </mapping>
    <mapping authority-dn="CN=Other Auth, OU=Other Unit, O=Other Org, ST=RJ, C=BR">
        <role-mapping from-role="RoleX" to-role="RoleK"/>
        <role-mapping from-role="RoleY" to-role="RoleZ"/>
    </mapping>
    <mapping authority-dn="CN=Example Auth, OU=Example Unit, O=Example Org, ST=SP, C=BR">
        <role-mapping from-role="RoleX" to-role="RoleC"/>
        <role-mapping from-role="RoleY" to-role="RoleB"/>
        <role-exclusion role-name="RoleA"/>
        <role-exclusion role-name="RoleD"/>
    </mapping>
</mappings>
```

Figura 5.18: Exemplo de arquivo de especificação de mapeamentos de papéis

O administrador do servidor especifica os mapeamentos aplicáveis a cada autoridade de atributos através do elemento `<mapping>`, que leva como parâmetro o *distinguished name* da

autoridade. Existem dois tipos de mapeamentos que podem ser aplicados: os mapeamentos tradicionais e as exclusões de papéis. Os primeiros, especificados pelo elemento `<role-mapping>`, são utilizados para mapear os papéis emitidos pela autoridade de atributos para os papéis definidos pela aplicação. Já as exclusões de papéis, especificadas pelo elemento `<role-exclusion>`, são utilizadas para descrever quais papéis emitidos pela autoridade de atributos devem ser ignorados pela aplicação. Papéis não especificados em nenhum desses elementos são utilizados normalmente pela infra-estrutura de segurança.

A motivação para a especificação de exclusões de papéis reside na possibilidade de tanto a autoridade de atributos quanto a aplicação Java EE definirem um papel com mesmo nome, porém com significados distintos. Por exemplo, uma certa autoridade pode utilizar um papel `admin` para indicar que o titular do certificado ocupa um cargo administrativo na companhia, como gerente, ou diretor. Em contrapartida, uma aplicação específica da companhia pode utilizar o papel `admin` para representar os usuários que têm poderes para administrar a aplicação. Claramente, os dois papéis, apesar de terem o mesmo nome, não possuem o mesmo significado. Nesse caso, existem duas opções para resolver o conflito: mapear o papel `admin` emitido pela autoridade para algum outro papel da aplicação, ou ignorar esse papel através de uma `role-exclusion` caso não haja um mapeamento adequado para ele.

Como podemos observar, o mecanismo de mapeamento de papéis atua como uma espécie de filtro para os papéis extraídos de certificados de atributos emitidos pelas autoridades descritas no arquivo de configuração. Porém, isso levanta uma questão: o que fazer com os papéis emitidos pelas demais autoridades, não listadas no arquivo de mapeamentos? A resposta se encontra no atributo `default-mapping`, especificado no elemento-raiz `mappings`. O valor `passthrough` para esse atributo indica que nenhum tipo de filtro deve ser aplicado aos papéis emitidos por autoridades não especificadas no arquivo de mapeamentos, enquanto o valor `reject` tem o efeito inverso, e indica que todos os papéis nessas condições devem ser excluídos pela infra-estrutura de segurança.

5.6.1 Integração do mecanismo de mapeamento aos módulos

A integração do mecanismo de mapeamento de papéis aos módulos desenvolvidos começa pela especificação do arquivo de mapeamentos, que é feita através da propriedade `mappingConfig` na configuração dos módulos. Em tempo de execução, o método `initialize` da classe `X509ACAbstractLoginModule` obtém o nome do arquivo de mapeamentos a partir dessa propriedade e interage com a `X509ACSystemFactory` para conseguir uma referência para a classe `RoleMapper`. Essa classe é instanciada com as informações contidas no arquivo de mapeamentos e é responsável por aplicar essas regras aos papéis extraídos dos certificados de atributos.

O diagrama de colaboração da Figura 5.19 ilustra as comunicações entre as classes para uma configuração completa do módulo *push*, isto é, uma configuração que contém uma cadeia de verificadores de revogação e também um arquivo de mapeamento de papéis. Este diagrama amplia aquele da Figura 5.17 para destacar as interações que ocorrem por conta do uso do mecanismo de mapeamento de papéis.

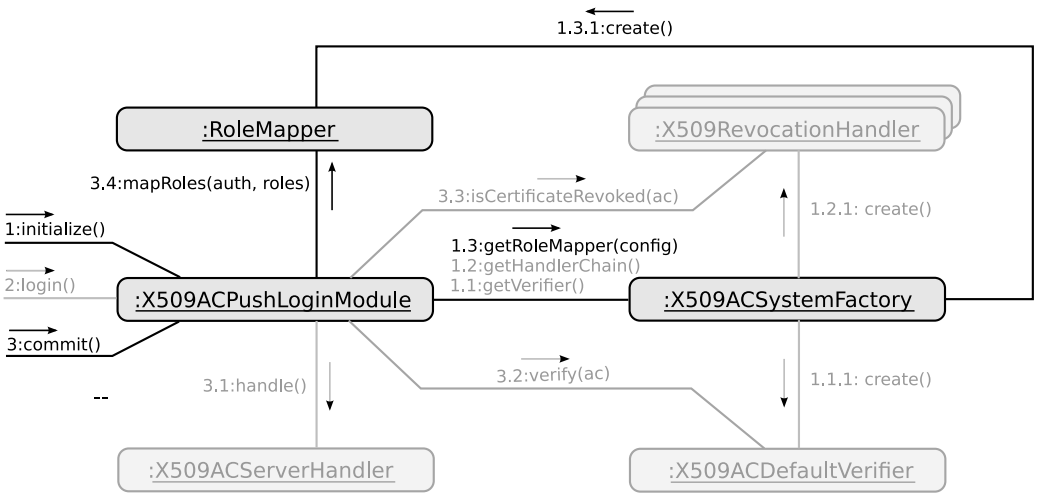


Figura 5.19: Diagrama de colaboração completo do modelo *push*

Através do diagrama pode-se observar que o mapeamento de papéis ocorre após a validação dos certificados de atributos obtidos pelo módulo concreto. Os papéis extraídos de cada certificado válido, bem como o nome da autoridade de atributos que gerou esse certificado, são passados para a instância de `RoleMapper` criada pela `X509ACSystemFactory`. Esse objeto utiliza o nome da autoridade de atributos para obter o conjunto de mapeamentos a serem executados e então aplica esses mapeamentos aos papéis fornecidos. O conjunto de papéis resultante é então devolvido ao módulo para que este faça a associação dos papéis mapeados ao objeto `Subject` que será armazenado pelo `JaasSecurityManager` ao final do processo.

Embora o diagrama apresente apenas as colaborações entre as classes no modelo *push*, o mecanismo de mapeamentos também está disponível para o modelo *pull*, sendo configurado e executado pelo `X509ACPullLoginModule` exatamente da mesma forma. Mais uma vez, isso se deve ao fato de que tanto o processo de criação do `RoleMapper` quanto o processo de execução dos mapeamentos são realizados pela superclasse `X509ACAbstractLoginModule`, que disponibiliza o mecanismo de mapeamento de papéis para ambos os módulos concretos.

6 Resultados Experimentais

Este capítulo apresenta os resultados dos experimentos que conduzimos a fim de comparar o desempenho dos módulos desenvolvidos em relação aos demais módulos oferecidos pelo JBoss. Avaliamos também o custo associado às operações que manipulam os certificados de atributos X.509, como a validação da assinatura digital, a verificação de revogação e o mapeamento de papéis. Esse esforço, além de oferecer uma análise comparativa dos módulos, nos revelou onde se encontram os pontos que mais penalizam o desempenho de uma infra-estrutura de segurança baseada em certificados de atributos, nos permitindo dessa forma fazer uma avaliação mais completa dos prós e contras de nossa abordagem.

Antes de analisarmos os resultados obtidos, convém detalhar o ambiente e a metodologia dos testes. Na Seção 6.1 a seguir descrevemos o ferramental utilizado, incluindo informações a respeito do *hardware*, nome e versão do sistema operacional, versão da máquina virtual Java e também as configurações realizadas no JBoss para a execução dos testes. Já na Seção 6.2 descrevemos a metodologia dos testes, ou seja, o processo utilizado para obter as medidas de desempenho que são discutidas nas seções subseqüentes deste capítulo.

6.1 Ambiente de testes

Os testes foram executados em uma rede local *Fast Ethernet*, formada por computadores equipados com processador *Intel Centrino Duo* de 2.0GHz, 2Gb de memória *RAM* e sistema operacional *Linux*, versão 2.6.23, distribuição *Fedora Core 7*. Os diversos serviços envolvidos na execução dos testes foram distribuídos em computadores diferentes, conforme ilustrado na Figura 6.1.

Um dos computadores abriga o servidor de aplicações JBoss que contém a aplicação de testes, o servidor de banco de dados e o serviço de diretórios LDAP. Um segundo computador na rede abriga o serviço de consulta OCSP e as listas de certificados revogados. A aplicação cliente, que mede o tempo gasto nas chamadas realizadas à aplicação de testes, pode ser executada tanto junto ao servidor de aplicações (modo local) quanto em um outro computador na rede (modo remoto).

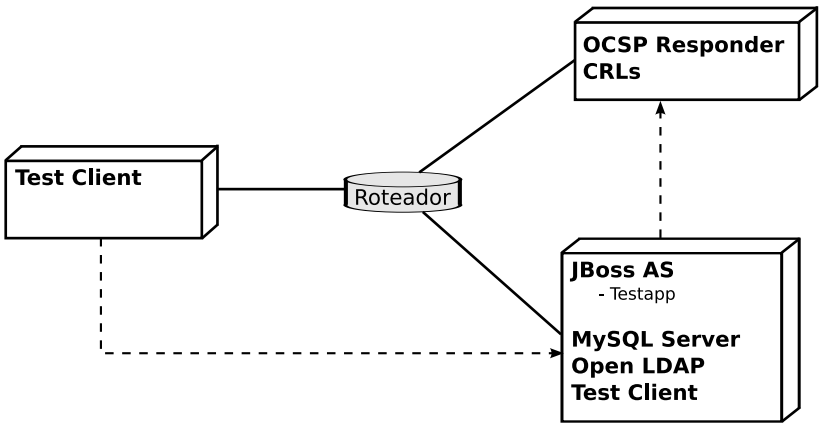


Figura 6.1: Diagrama de implantação dos serviços utilizados nos testes

A máquina virtual Java utilizada foi a *Sun Java SE Development Kit* (*JDK*) [16], versão 1.5.0_13-b05. Em relação ao servidor de aplicações, utilizamos a versão 4.0.2 (*SVN Tag JBoss_4_0_2*) do JBoss, juntamente com nossa extensão da infra-estrutura de segurança. Nos testes envolvendo banco de dados utilizamos o *MySQL* [34], versão 5.0.45. Já nos testes envolvendo serviços de diretórios LDAP utilizamos o *OpenLDAP* [39], versão 2.3.34.

A preparação do servidor de aplicações consistiu basicamente na configuração de cada módulo JAAS sendo testado. Criamos um componente EJB do tipo sessão que contém um único método vazio (ou seja, um método que não realiza nenhum processamento), declaramos no descritor de implantação que esse método deve ser protegido, e medimos, para cada módulo JAAS configurado, o tempo gasto nas chamadas que um usuário que faz para esse método. A idéia de se utilizar um método vazio é evitar que eventuais custos de processamento do EJB influenciem no resultado das medições. Os módulos utilizados nos testes foram os seguintes:

- `X509ACPullLoginModule` e `X509ACPushLoginModule`: os módulos que implementamos como parte de nosso trabalho.

- `UsersRolesLoginModule`: módulo do JBoss que se utiliza de dois arquivos de propriedades para autenticar a autorizar os usuários. Um dos arquivos contém as senhas associadas a cada identidade, enquanto que o outro arquivo contém os papéis associados a cada identidade.

- `DatabaseServerLoginModule`: módulo do JBoss que busca a senha e os papéis dos usuários em um banco de dados. É configurado com duas consultas *SQL*, uma descrevendo como obter a senha do usuário, e outra descrevendo como obter os papéis do mesmo.

- `LdapLoginModule`: módulo do JBoss que delega a autenticação dos usuários para um serviço de diretórios LDAP. Um conjunto de propriedades indica ao módulo como autenticar o usuário e como obter os papéis desse usuário na árvore de diretórios.

A fim de avaliar o desempenho de cada módulo em uma situação de uso real, criamos mil usuários e vinte papéis fictícios e atribuímos aleatoriamente de um a cinco papéis a cada usuário. Os repositórios utilizados por cada módulo foram então preenchidos com os usuários gerados. Dessa forma, tanto os arquivos de propriedades usados pelo `UsersRolesLoginModule`, quanto o servidor de banco de dados e o serviço de diretórios LDAP tiveram uma carga inicial de mil usuários associados a um média de três papéis cada.

Nos testes comparativos entre os módulos, utilizamos as configurações padrão para cada módulo. Isto significa que as configurações tanto do `X509ACPullLoginModule` quanto do `X509ACPushLoginModule`especificam apenas o prefixo e o sufixo usados na construção dos *distinguished names* e o nome do `SecurityDomain` que deve ser utilizado pelo validador padrão. Já no caso dos módulos do JBoss, seguimos os exemplos da documentação de segurança do servidor para configurar os módulos da forma mais simples possível. É preciso destacar também que, em todos os experimentos envolvendo o `X509ACPullLoginModule` e o `X509ACPushLoginModule`, o cliente possuía apenas um único certificado de atributos X.509.

6.2 Metodologia

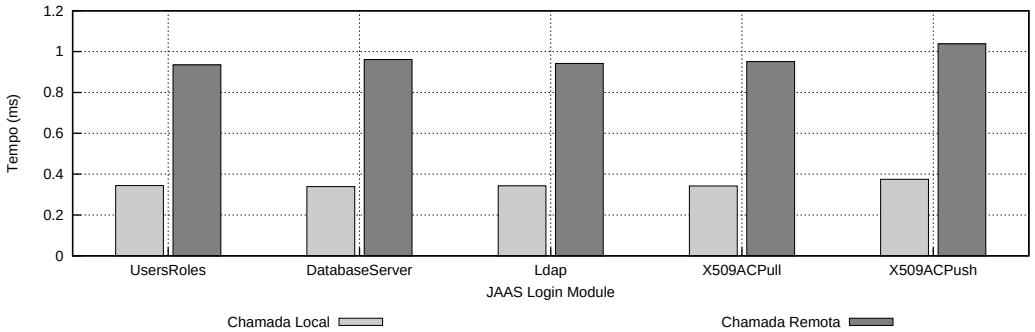
A execução de cada experimento pode ser dividida em três fases distintas. Na primeira, configuramos o módulo sendo avaliado para proteger o componente EJB de testes, iniciamos o servidor de aplicações com essa configuração e implantamos o componente de testes. Na segunda, usamos a aplicação de testes cliente para fazer cinco mil chamadas ao componente EJB com o objetivo de estabilizar o sistema. Na terceira e última fase, usamos a aplicação cliente para fazer vinte mil chamadas ao componente e medimos o tempo gasto em cada chamada. Os resultados obtidos serviram como base para o cálculo de uma série de estatísticas a respeito do desempenho do módulo, como média, desvio padrão, intervalo de confiança, valor mínimo e valor máximo dos tempos medidos.

Após a coleta de estatísticas referentes à execução de um módulo específico, o servidor de aplicações é desligado, um novo módulo é configurado e o processo se repete. O fato de o servidor de aplicações ter de ser reiniciado após a execução de uma bateria de testes garante as mesmas condições iniciais para todos os experimentos.

6.3 Estudo comparativo dos módulos

O primeiro estudo realizado visa comparar o desempenho dos módulos que implementamos como parte de nosso trabalho com o dos módulos já oferecidos pelo JBoss. Cada módulo foi avaliado em duas situações: primeiro com o *cache* de segurança do JBoss habilitado, depois com o *cache* de segurança desabilitado.

A Figura 6.2 exibe os resultados dos experimentos com o *cache* de segurança habilitado. Em particular, a Figura 6.2a apresenta a duração média das chamadas locais e remotas ao componente EJB de testes para cada módulo testado. Já a tabela da Figura 6.2b apresenta as estatísticas calculadas a partir dos tempos coletados: a média, o desvio padrão, o mínimo, o máximo e o intervalo de confiança (I.C.) de 95%.



(a) Durações médias das chamadas ao EJB de testes.

		UsersRoles	DatabaseServer	Ldap	X509ACPull	X509ACPush
Chamadas Locais	**Média**	0,344	0,339	0,343	0,342	0,375
	Desvio Padrão	0,636	0,360	0,411	0,570	0,492
	Mínimo	0,309	0,302	0,307	0,307	0,340
	Máximo	64,972	25,332	25,329	72,025	37,576
	I.C. (95%)	[0,335; 0,353]	[0,334; 0,344]	[0,337; 0,349]	[0,334; 0,350]	[0,368; 0,382]
Chamadas Remotas	**Média**	0,935	0,961	0,942	0,951	1,038
	Desvio Padrão	1,518	0,452	0,367	0,463	0,391
	Mínimo	0,555	0,577	0,592	0,576	0,668
	Máximo	206,056	28,446	26,862	35,148	33,162
	I.C. (95%)	[0,914; 0,956]	[0,955; 0,967]	[0,937; 0,947]	[0,945; 0,957]	[1,033; 1,043]

(b) Durações das chamadas com *cache* de segurança habilitado. Todos os tempos são expressos em milissegundos.

Figura 6.2: Resultados das chamadas com o *cache* de segurança habilitado

É importante esclarecer que o `X509ACPushLoginModule` testado foi configurado em conjunto com outro módulo, já que ele não é capaz de autenticar clientes. Para isso, implementamos um módulo auxiliar que apenas devolve `true` no método `login`. Por se tratar de uma implementação muito simples, o tempo gasto pelo módulo auxiliar pouco impacta na medição do tempo

gasto pelo `X509ACPushLoginModule` para obter os papéis do cliente a partir do certificado de atributos enviado ao servidor.

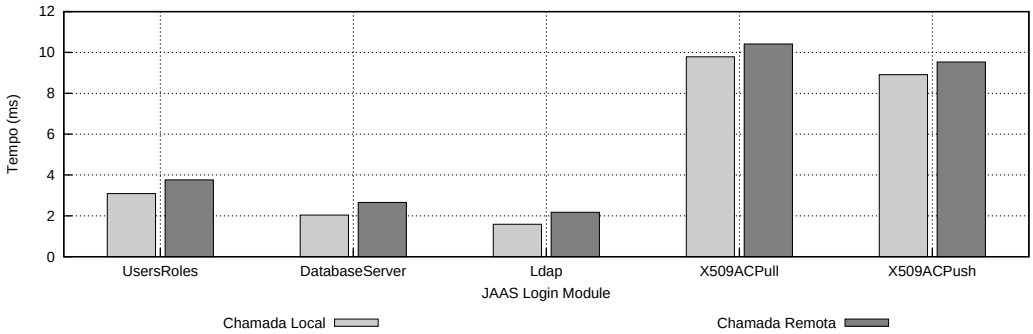
Feita essa observação, podemos avaliar os resultados obtidos com o *cache* de segurança habilitado. Em primeiro lugar, nota-se que o tempo gasto pelas chamadas remotas é maior do que o tempo gasto pelas chamadas locais. No entanto, apesar do acréscimo de tempo ser percentualmente alto, em termos absolutos ele é bastante pequeno, pois representa uma diferença de aproximadamente $0,6$ milissegundos apenas. Um segundo aspecto a ser notado refere-se à equivalência dos tempos gastos por todos os módulos nessa situação, tanto nas chamadas locais quanto remotas. Esse comportamento é esperado, já que os módulos em si são executados apenas na primeira chamada feita pelo cliente e o `Subject` resultante é armazenado em *cache* pelo `JaasSecurityManager`. Todas as chamadas subseqüentes resultam em um acesso ao *cache* para obtenção do `Subject`, de forma que os módulos não são executados novamente enquanto o *cache* for válido.

Outro ponto a ser observado é quanto ao valor máximo obtido em cada experimento, que destoa consideravelmente do valor médio. Isso ocorre devido às coletas de lixo executadas pela máquina virtual Java, que em nossos experimentos ocorriam aproximadamente uma vez a cada dez mil chamadas feitas ao EJB de testes. Por se tratarem de valores isolados, tais medições não tiveram influência significativa sobre os valores médios que obtivemos.

Os resultados apresentados pelos experimentos com o *cache* de segurança habilitado nos permitem concluir que o uso de tal *cache* de certa forma iguala o desempenho dos módulos, já que a maior parte das requisições para um componente protegido resulta em uma consulta ao *cache* para obter o `Subject` do cliente. Somente na primeira chamada, ou quando o *cache* expira, é que o módulo JAAS configurado para a aplicação é executado. Assim, o tempo gasto pelo módulo nas raras ocasiões em que ele é executado praticamente não influencia o tempo médio das chamadas ao componente protegido.

Tendo como objetivo medir o desempenho real de cada módulo, desabilitamos o *cache* de segurança do servidor e refizemos os experimentos. A Figura 6.3 exibe os resultados dos experimentos com o *cache* de segurança desabilitado. Mais especificamente, a Figura 6.3a exibe a duração média das chamadas ao componente EJB de testes, enquanto a tabela da Figura 6.3b exibe as estatísticas calculadas para cada módulo.

Analisando a Figura 6.3a, é possível notar o desempenho superior do `LdapLoginModule` em relação aos demais módulos, o que comprova o alto desempenho de serviços de diretórios na busca de informações. O `DatabaseServerLoginModule` apresentou um desempenho ligeiramente inferior ao primeiro, enquanto o `UsersRolesLoginModule` se apresentou cerca de 95% mais lento do que o `LdapLoginModule` no modo local e 75% mais lento no modo remoto. Em



(a) Durações médias das chamadas ao EJB de testes.

		UsersRoles	DatabaseServer	Ldap	X509ACPull	X509ACPush
Chamadas Locais	**Média**	3,091	2,039	1,585	9,788	8,909
	Desvio Padrão	1,616	1,680	1,813	3,638	2,887
	Mínimo	2,963	1,926	1,395	9,510	8,613
	Máximo	217,279	217,472	221,072	307,641	272,759
	I.C. (95%)	[3,069; 3,113]	[2,016; 2,062]	[1,560; 1,610]	[9,738; 9,838]	[8,869; 8,949]
Chamadas Remotas	**Média**	3,761	2,660	2,174	10,411	9,529
	Desvio Padrão	1,568	1,666	1,674	3,381	2,982
	Mínimo	3,223	2,291	1,685	9,916	8,928
	Máximo	210,028	219,944	220,529	298,631	278,490
	I.C. (95%)	[3,739; 3,783]	[2,637; 2,683]	[2,151; 2,197]	[10,364; 10,458]	[9,488; 9,570]

(b) Durações das chamadas com cache de segurança desabilitado. Todos os tempos são expressos em milissegundos.

Figura 6.3: Resultados das chamadas com o *cache* de segurança desabilitado

relação aos módulos que desenvolvemos, podemos observar que estes foram significativamente mais lentos que os demais módulos do JBoss com o *cache* desabilitado. Essa diferença de desempenho nos fornece uma primeira noção do custo envolvido na manipulação dos certificados de atributos X.509 para a obtenção dos papéis dos usuários.

Uma comparação entre o `LdapLoginModule` e o `X509ACPullLoginModule` pode nos auxiliar na avaliação desse custo. O `X509ACPullLoginModule` em sua configuração padrão autentica os clientes junto a um serviço de diretórios LDAP exatamente da mesma forma que o `LdapLoginModule`. A diferença entre os módulos reside então na maneira de obtenção dos papéis associados a cada cliente. Enquanto o `LdapLoginModule` precisa fazer uma busca no serviço de diretórios para encontrar os papéis associados a um usuário, o `X509ACPullLoginModule` obtém um ou mais certificados de atributos diretamente no contexto LDAP do cliente. Os certificados obtidos são validados e os papéis do usuário extraídos. Essa manipulação dos certificados de atributos é então a responsável pela diferença de desempenho de aproximadamente $8,2$ milissegundos entre os módulos, conforme podemos observar na Figura 6.3b. Em termos percentuais, o `X509ACPullLoginModule` foi 520% mais lento que o `LdapLoginModule` no modo

local e 375% mais lento no modo remoto.

A forma como esse acréscimo de aproximadamente 8,2 milissegundos por chamada afeta o desempenho das aplicações Java EE depende do número e da freqüência de chamadas que cada aplicação faz aos seus componentes protegidos, bem como da complexidade dos métodos implementados por esses componentes. Não sofrerão uma redução significativa de desempenho as aplicações que fizerem poucas chamadas a componentes protegidos e/ou invocarem métodos cuja execução consuma tempo muito superior ao gasto com a manipulação dos certificados de atributos. Por outro lado, haverá uma sensível redução no desempenho das aplicações com alta freqüência de chamadas a métodos de baixa complexidade dos componentes protegidos. São essas as aplicações que mais se beneficiam do *cache* de segurança.

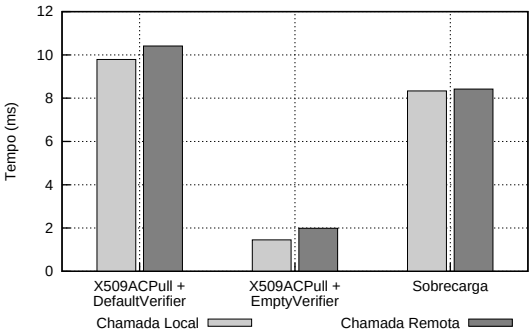
6.4 Estimativa da sobrecarga imposta pelo verificador padrão

Na seção anterior comparamos os módulos de segurança desenvolvidos neste trabalho com os módulos já oferecidos pelo JBoss, e avaliamos o impacto da manipulação dos certificados de atributos X.509 no desempenho dos módulos criados. Esta seção apresenta um estudo mais detalhado desse impacto, identificando o custo associado à validação do certificado de atributos do cliente.

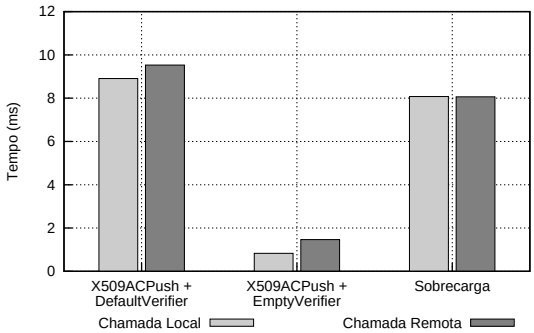
O experimento conduzido consistiu na substituição do verificador de certificados de atributos padrão por um verificador vazio, isto é um verificador que não realiza nenhum tipo de validação e aceita todos os certificados como sendo válidos. Configuramos este novo verificador junto ao `X509ACPullLoginModule` e ao `X509ACPushLoginModule` e repetimos os experimentos com esses módulos, comparando o desempenho obtido com o verificador vazio em relação ao desempenho medido anteriormente com o verificador padrão.

A Figura 6.4 apresenta os resultados dos experimentos realizados com o verificador vazio. Mais especificamente, as Figuras 6.4a e 6.4b exibem a sobrecarga imposta pelo uso do verificador padrão nos modelos *pull* e *push*, respectivamente, enquanto a tabela da Figura 6.4c apresenta as estatísticas calculadas a partir dos valores medidos.

Como podemos observar, as validações feitas pelo verificador padrão são responsáveis por uma sobrecarga superior a 8,0 milissegundos no desempenho de ambos os módulos. No caso do `X509ACPushLoginModule`, isso representa 90% do tempo total gasto pelo módulo no modo local, e 85% no modo remoto. Já no caso do `X509ACPullLoginModule`, essa sobrecarga representa 85% do tempo total medido no experimento no modo local, e 80% no modo remoto. Comparando esses resultados com os resultados discutidos na seção anterior, podemos concluir que a maior parte da sobrecarga imposta pela manipulação dos certificados de atributos X.509



(a) Sobrecarga imposta pelo uso do verificador padrão no modelo *pull*.



(b) Sobrecarga imposta pelo uso do verificador padrão no modelo *push*.

		X509ACPull + EmptyVerifier	**X509ACPull + DefaultVerifier**	**X509ACPush + EmptyVerifier**	**X509ACPush + DefaultVerifier**
Chamadas Locais	**Média**	1,454	9,788	0,830	8,909
	Desvio Padrão	2,726	3,638	1,683	2,887
	Mínimo	1,331	9,510	0,764	8,613
	Máximo	294,494	307,641	215,834	272,759
	I.C. (95%)	[1,416; 1,492]	[9,738; 9,838]	[0,807; 0,853]	[8,869; 8,949]
Chamadas Remotas	**Média**	1,991	10,411	1,485	9,529
	Desvio Padrão	2,566	3,381	1,592	2,982
	Mínimo	1,557	9,916	1,165	8,928
	Máximo	264,976	298,631	213,902	278,490
	I.C. (95%)	[1,955; 2,027]	[10,364; 10,458]	[1,463; 1,507]	[9,488; 9,570]

(c) Durações das chamadas com diferentes configurações do verificador utilizado. Todos os tempos são expressos em milissegundos.

Figura 6.4: Resultados das chamadas utilizando um verificador vazio

reside na validação desses certificados. Esse processo é particularmente custoso devido à verificação da assinatura digital, que envolve a aplicação de funções criptográficas para assegurar que o conteúdo dos certificados não foi alterado.

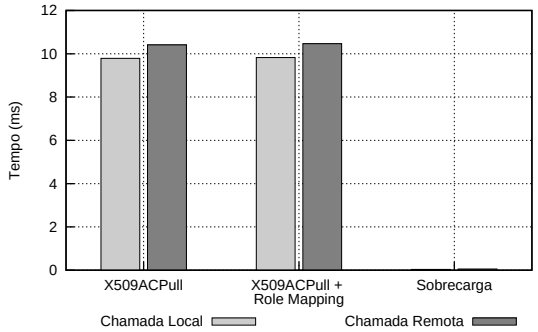
É importante destacar que nos experimentos realizados o cliente possuía apenas um único certificado de atributos X.509. Como os certificados são validados um a um pelo verificador padrão, a sobrecarga imposta por essa validação aumenta de acordo com o número de certificados do cliente. Em outras palavras, o verificador padrão gasta em torno de $8,0$ milissegundos para cada certificado de atributos do cliente.

Concluindo a discussão sobre este experimento, podemos dizer que, embora a maior parte do tempo gasto na manipulação dos certificados de atributos seja devido à validação desses certificados, consideramos que a validação é vital para o funcionamento de nossa solução de forma segura. A sobrecarga imposta nesse processo é inerente a qualquer solução baseada em certificados digitais, já que a verificação da assinatura digital é o que garante a integridade das informações contidas nesses certificados. Assim, a não realização da validação constituiria

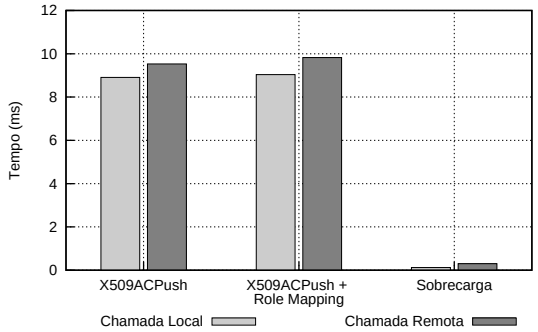
uma grave falha de segurança e anularia todos os benefícios trazidos pelos certificados de atributos como estrutura de armazenamento de papéis, como garantia de origem e integridade das informações.

6.5 Estimativa da sobrecarga imposta pelo mapeamento de papéis

Nesta seção avaliamos o custo resultante do uso de nosso mecanismo de mapeamento de papéis. Para isso, criamos um arquivo de mapeamentos que atua como uma função identidade, mapeando cada papel para ele mesmo, e adicionamos esse arquivo à configuração dos nossos módulos. Além disso, voltamos a utilizar o verificador padrão na condução dos experimentos.



(a) Sobrecarga imposta pelo uso do mecanismo de mapeamento no modelo *pull*.



(b) Sobrecarga imposta pelo uso do mecanismo de mapeamento no modelo *push*.

		X509ACPull	X509ACPull + Role Mapping	X509ACPush	X509ACPush + Role Mapping
Chamadas Locais	**Média**	9,788	9,827	8,909	9,035
	Desvio Padrão	3,638	4,087	2,887	2,902
	Mínimo	9,510	9,587	8,613	8,728
	Máximo	307,641	312,370	272,759	273,044
	I.C. (95%)	[9,738; 9,838]	[9,770; 9,884]	[8,869; 8,949]	[8,995; 9,075]
Chamadas Remotas	**Média**	10,411	10,465	9,529	9,828
	Desvio Padrão	3,381	3,842	2,982	2,845
	Mínimo	9,916	9,970	8,928	9,305
	Máximo	298,631	303,609	278,490	271,979
	I.C. (95%)	[10,364; 10,458]	[10,412; 10,518]	[9,488; 9,570]	[9,789; 9,867]

(c) Durações das chamadas. Todos os tempos são expressos em milissegundos.

Figura 6.5: Resultados das chamadas utilizando o mecanismo de mapeamento de papéis

A Figura 6.5 apresenta os resultados dos experimentos realizados com o mecanismo de mapeamento ativado. Em particular, as Figuras 6.5a e 6.5b comparam a execução dos módulos *pull* e *push* com e sem o mecanismo de mapeamento de papéis, ilustrando a sobrecarga imposta pelo uso desse mecanismo. As estatísticas calculadas a partir dos tempos medidos podem ser

encontradas na tabela da Figura 6.4c.

Analisando os resultados apresentados pela tabela da Figura 6.4c, podemos observar que o uso do mecanismo de mapeamento de papéis não teve influência significativa no desempenho de nossos módulos, tanto no modo local quanto no modo remoto. Um fator que certamente contribuiu para isso foi o fato do arquivo de configuração de mapeamentos ser lido uma única vez pela `X509ACSystemFactory`. Ou seja, todos os custos associados à leitura do arquivo de mapeamentos o ocorrem somente na primeira chamada feita a qualquer módulo que utilize esse arquivo para descrever os mapeamentos de papéis. A `X509ACSystemFactory` utiliza as informações lidas para instanciar os objetos `RoleMapper` e mantém os objetos criados em *cache*.

Outro fator que contribuiu para o bom desempenho do mecanismo foi o fato do processo de mapeamento não apresentar um custo de desempenho maior do que o processo de extração dos papéis dos certificados de atributos. Essa afirmação pode ser comprovada da seguinte forma: seja n o número de papéis contidos dos certificados de atributos do cliente. Como os papéis são obtidos um a um, podemos afirmar que o custo de extração de todos papéis é $O(n)$, uma vez que a extração em si consiste apenas na leitura de um campo do certificado de atributos. Quando os papéis são processados pelo `RoleMapper`, este se utiliza de um `HashMap` para realizar o mapeamento em si, de forma que o mapeamento de cada papel tem um custo $O(1)$. Isso significa que o custo total do mapeamento dos n papéis também é $O(n)$. Com isso, podemos afirmar que o algoritmo de mapeamento de papéis tem a mesma ordem do algoritmo de extração desses papéis. A maior parte do tempo gasto na manipulação dos certificados de atributos continua residindo na validação da assinatura digital, conforme discutimos na seção anterior.

Consideramos os resultados desse experimento bastante satisfatórios, pois os números obtidos encorajam a utilização de nosso mecanismo de mapeamento de papéis. Esse mecanismo adiciona grande flexibilidade à administração das aplicações, uma vez que ele permite um controle mais fino sobre o significado dos papéis emitidos pelas autoridades de atributos com as quais o servidor de aplicações interage. Nenhum dos projetos que estudamos oferece um mecanismo de mapeamento de papéis como o que implementamos. Até onde sabemos, tal mecanismo é uma contribuição de nosso projeto.

6.6 Estimativa da sobrecarga imposta pelos verificadores de revogação

A maioria das infra-estruturas de autorização baseadas em certificados digitais oferece apenas o modelo *pull* de propagação de certificados para evitar a validação desses certificados

quanto à revogação, por se tratar de um processo complexo e custoso em termos de desempenho. No entanto, nossa extensão da infra-estrutura de segurança do JBoss oferece suporte também ao modelo *push*, disponibilizando dois mecanismos para verificar se os certificados de atributos fornecidos pelo cliente foram ou não revogados pela autoridade de atributos após sua emissão.

Nesta seção, dividida em duas partes, estudamos a sobrecarga imposta por esses mecanismos no tempo de execução dos módulos. Na primeira parte, avaliamos o desempenho dos módulos quando configurados com o verificador de revogação baseado no protocolo OCSP. Já na segunda parte, estudamos o comportamento dos módulos quando configurados para utilizar o verificador baseado em listas de certificados revogados. Em particular, avaliamos a variação do tempo de execução de cada módulo em função do tamanho da lista de certificados revogados que é obtida junto à autoridade de atributos.

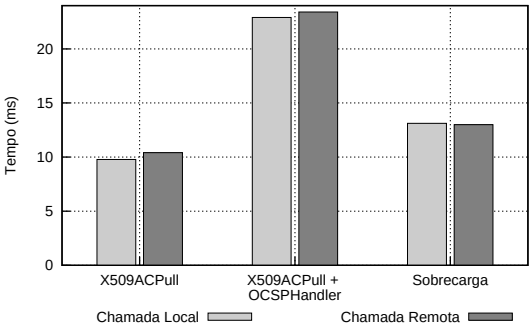
6.6.1 OCSPRevocationHandler

O `OCSPRevocationHandler` é o verificador de revogação que se utiliza do protocolo OCSP para determinar se um certificado foi ou não revogado. Com o objetivo de avaliar o impacto que esse mecanismo causa no desempenho dos módulos, desenvolvemos um serviço capaz de tratar requisições OCSP via HTTP e o implantamos em um servidor JBoss em outro computador da rede. Em seguida, configuramos cada um dos módulos para utilizar esse verificador e refizemos os experimentos, medindo o tempo total gasto pelo módulos nessa nova configuração.

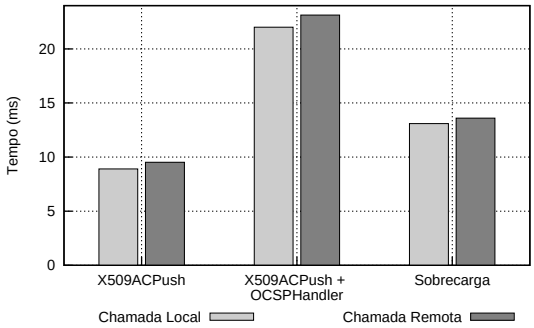
A Figura 6.6 apresenta os resultados dos experimentos realizados com o verificador de revogação OCSP. Mais precisamente, as Figuras 6.6a e 6.6b ilustram, respectivamente, o desempenho do `X509ACPullLoginModule` e do `X509ACPushLoginModule` quando configurados para utilizar o `OCSPRevocationHandler`, demonstrando a sobrecarga imposta por esse verificador. Já a tabela da Figura 6.6c apresenta as estatísticas calculadas a partir dos tempos medidos no experimento.

Conforme podemos observar na Figura 6.6c, a sobrecarga imposta pelo uso do `OCSPRevocationHandler` aos dois módulos foi superior a $13,0$ milissegundos em todos os casos. Essa sobrecarga corresponde a um acréscimo de aproximadamente 130% ao tempo de execução do `X509ACPullLoginModule` e de 145% ao tempo de execução do `X509ACPushLoginModule`, tanto no modo local quanto no modo remoto.

Para entendermos melhor a sobrecarga imposta pelo `OCSPRevocationHandler`, precisamos analisar com mais detalhes o processo de verificação de revogação por OCSP. Quando o `OCSPRevocationHandler` valida um certificado, ele cria uma requisição OCSP contendo o número de série do certificado e assina digitalmente a requisição com o CCP da autoridade de atri-



(a) Sobrecarga imposta pelo uso do verificador de revogação OCSP no modelo *pull*.



(b) Sobrecarga imposta pelo uso do verificador de revogação OCSP no modelo *push*.

		X509ACPull	**X509ACPull + OCSPHandler**	**X509ACPush**	**X509ACPush + OCSPHandler**
Chamadas Locais	**Média**	9,788	22,905	8,909	22,001
	Desvio Padrão	3,638	5,208	2,887	4,181
	Mínimo	9,510	22,286	8,613	21,323
	Máximo	307,641	325,216	272,759	282,508
	I.C. (95%)	[9,738; 9,838]	[22,833; 22,977]	[8,869; 8,949]	[21,943; 22,059]
Chamadas Remotas	**Média**	10,411	23,407	9,529	23,125
	Desvio Padrão	3,381	5,160	2,982	4,426
	Mínimo	9,916	22,655	8,928	22,324
	Máximo	298,631	333,730	278,490	289,126
	I.C. (95%)	[10,364; 10,458]	[23,335; 23,479]	[9,488; 9,570]	[23,064; 23,186]

(c) Durações das chamadas. Todos os tempos são expressos em milissegundos.

Figura 6.6: Resultados das chamadas com o verificador de revogação OCSP

butos. A requisição criada é então enviada ao serviço de consulta, que valida a assinatura da requisição com a chave privada da autoridade de atributos, extrai o número de série do certificado sendo consultado e constrói uma resposta OCSP indicando se o certificado foi ou não revogado. A resposta é assinada digitalmente e enviada de volta ao verificador de revogação, que valida a assinatura da resposta e extrai o resultado da consulta.

Analisando o processo descrito, observamos que tanto as requisições quanto as respostas no protocolo OCSP são assinadas digitalmente, o que exige que tanto o verificador de revogação quanto o serviço de consulta validem essas assinaturas em tempo de execução. Além disso, temos o custo associado à comunicação entre os processos, que envolve a passagem das requisições e respostas OCSP via HTTP. Por fim, temos o custo relativo ao tempo gasto pelo serviço de consulta para determinar se o certificado foi ou não validado. Juntas, essas três etapas (criação e validação dos pares requisição/resposta, comunicação entre os processos e complexidade do serviço de consulta) são responsáveis pelo tempo total gasto pelo `OCSPRevocationHandler` para verificar um certificado quanto à revogação.

Em nosso experimento, implementamos um serviço de consulta muito simples, que sempre responde que o certificado consultado é válido (ou seja, não se encontra revogado), de forma que o custo total do experimento se concentra basicamente na construção e validação das requisições e respostas OCSP e na comunicação entre os processos. Por isso, implementações mais complexas do serviço de consulta podem fazer com que a sobrecarga total imposta pelo mecanismo seja maior do que o valor medido neste experimento.

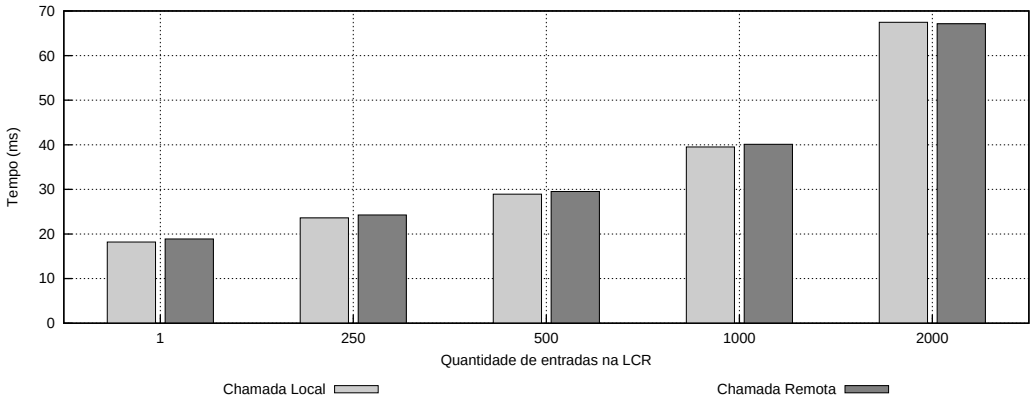
É importante também ressaltar novamente que este experimento foi feito com um único certificado de atributos associado ao cliente de testes. Atualmente, o `OCSPRevocationHandler` é capaz de tratar apenas um certificado por vez, gastando cerca de $13,0$ milissegundos para validar cada certificado. Como conseqüência, clientes com mais de um certificado de atributos devem esperar um aumento do tempo de resposta proporcional ao número de certificados envolvidos.

A conclusão a que chegamos com os resultados obtidos neste experimento é de que, embora a verificação realizada pelo `OCSPRevocationHandler` imponha uma sobrecarga considerável ao tempo de execução dos módulos, o desempenho como um todo ainda pode ser considerado satisfatório. No entanto, como vimos, um número maior de certificados pode resultar em um consumo de tempo inaceitável por parte do `OCSPRevocationHandler` em situações onde o *cache* de segurança não pode ser utilizado.

6.6.2 CRLRevocationHandler

A segunda parte deste experimento consistiu no teste de desempenho do `CRLRevocationHandler`, que verifica se um certificado de atributos foi revogado consultando uma lista de certificados revogados mantida pela autoridade de atributos. Para a execução do experimento, geramos LCRs de diversos tamanhos e configuramos o `X509ACPushLoginModule` de maneira a utilizar o `CRLRevocationHandler` para obter as listas geradas. Todas as listas foram armazenadas no mesmo computador que contém a aplicação de consulta OCSP e foram acessadas pelo `CRLRevocationHandler` via HTTP.

Os resultados dos experimentos realizados nos modelos *pull* e *push* com o `CRLRevocationHandler` podem ser visualizados nas Figuras 6.7 e 6.8, respectivamente. Mais especificamente, as Figuras 6.7a e 6.8a exibem o desempenho dos módulos de acordo com o tamanho da lista de certificados revogados obtida pelo `CRLRevocationHandler`. Já as tabelas das Figuras 6.7b e 6.8b apresentam as estatísticas calculadas. Novamente, os tempos medidos se referem ao cenário onde o cliente de testes possui apenas um certificado de atributos. O uso de um número maior de certificados acarretará em um consumo de tempo maior, já que o `CRLRevocationHandler` realiza a verificação de cada certificado separadamente.



(a) Duração média das chamadas com o `CRLRevocationHandler` no modelo *pull*.

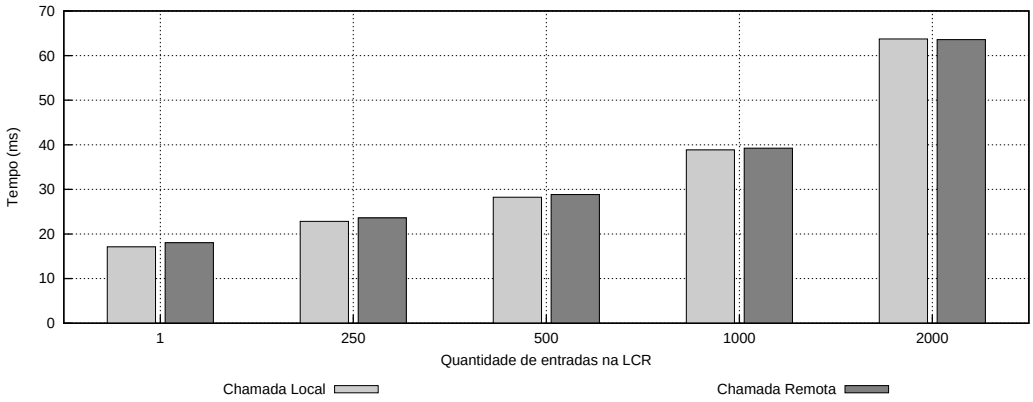
		X509ACPull+ CRLRevocationHandler				
		1 Entrada	**250 Entradas**	**500 Entradas**	**1000 Entradas**	**2000 Entradas**
Chamadas Locais	**Média**	18,199	23,619	28,918	39,519	67,467
	Desvio Padrão	4,725	5,404	6,240	8,056	12,872
	Mínimo	17,865	23,273	28,514	38,976	61,064
	Máximo	323,941	333,201	352,832	376,917	452,710
	I.C. (95%)	[18,134; 18,264]	[23,544; 23,694]	[28,832; 29,004]	[39,407; 39,631]	[67,289; 67,645]
Chamadas Remotas	**Média**	18,878	24,281	29,536	40,103	67,139
	Desvio Padrão	4,603	5,341	6,203	7,962	12,033
	Mínimo	18,305	23,738	28,991	39,384	60,554
	Máximo	311,672	343,433	306,998	366,861	460,032
	I.C. (95%)	[18,814; 18,942]	[24,207; 24,355]	[29,450; 29,622]	[39,993; 40,213]	[66,972; 67,305]

(b) Durações das chamadas. Todos os tempos são expressos em milissegundos.

Figura 6.7: Resultados das chamadas com o `CRLRevocationHandler` no modelo *pull*

As listas de certificados revogados crescem à medida que a autoridade de atributos revoga mais certificados. Os tempos apresentados nas Figuras 6.7 e 6.8 demonstram que o tamanho da lista influencia diretamente o desempenho de cada módulo. Existem ao menos duas razões para isso: primeiro, quanto maior a lista, maior o tempo que o `CRLRevocationHandler` leva para obtê-la. Segundo, como a LCR é assinada digitalmente pela autoridade de atributos, o verificador precisa validar essa assinatura ao obter essa LCR para garantir a integridade de seu conteúdo. Entretanto, quanto maior é a lista, maior é o tempo gasto pelo verificador para calcular o valor de *hash* que é utilizado para validar a assinatura digital.

Através dos valores apresentados nas tabelas das Figuras 6.7b e 6.8b podemos analisar a variação de desempenho dos módulos em função do tamanho da lista de certificados revogados. Uma lista mínima, com uma única entrada, causou um impacto de aproximadamente $8,0$ milissegundos no tempo de execução de ambos os módulos, o que corresponde a um incremento percentual próximo a 85%. O aumento do número de entradas na lista para 250 eleva o tempo



(a) Duração média das chamadas com o `CRLRevocationHandler` no modelo *push*.

		X509ACPush+ CRLRevocationHandler				
		1 Entrada	**250 Entradas**	**500 Entradas**	**1000 Entradas**	**2000 Entradas**
Chamadas Locais	**Média**	17,133	22,835	28,249	38,850	63,711
	Desvio Padrão	3,866	4,381	4,626	5,895	7,748
	Mínimo	16,715	22,500	27,772	38,326	60,543
	Máximo	284,275	277,278	261,969	285,017	317,008
	I.C. (95%)	[17,059; 17,167]	[22,774; 22,896]	[28,185; 28,313]	[38,768; 38,932]	[63,604; 63,818]
Chamadas Remotas	**Média**	18,046	23,627	28,836	39,236	63,574
	Desvio Padrão	4,015	4,385	4,909	5,666	7,831
	Mínimo	17,564	22,990	28,280	38,628	60,436
	Máximo	302,962	284,708	287,750	272,455	312,460
	I.C. (95%)	[17,990; 18,102]	[23,566; 23,688]	[28,768; 28,904]	[39,157; 39,315]	[63,465; 63,682]

(b) Durações das chamadas. Todos os tempos são expressos em milissegundos.

Figura 6.8: Resultados das chamadas com o `CRLRevocationHandler` no modelo *push*

de execução em cerca de $5,5$ milissegundos. Um novo aumento de entradas para 500 eleva o tempo de execução obtido com a lista mínima em $11,0$ milissegundos. Ou seja, ao dobrarmos o tamanho da lista, dobramos também o acréscimo causado ao tempo de execução dos módulos. O caráter linear da variação do tempo de execução em função do tamanho das listas se confirma com as duas últimas medidas: a cada duplicação do número de entradas podemos observar uma duplicação correspondente do tempo em relação àquele medido no experimento com a lista de tamanho 1.

Essa variação do tempo de execução dos módulos em razão do tamanho das listas utilizadas nos permite projetar qual seria o tempo gasto quando listas de certificados maiores fossem utilizadas, uma vez que em nossos experimentos pudemos observar que o tempo de execução aumenta cerca de $5,5$ milissegundos para cada acréscimo de 250 entradas feito à lista de certificados revogados. Dessa forma, ao aumentarmos, por exemplo, o número de entradas de 2000 para 3000, esperamos um aumento proporcional de aproximadamente $22,0$ milissegundos no

tempo de execução dos módulos. É interessante notar também que o `CRLRevocationHandler` só apresentou desempenho superior ao `OCSPRevocationHandler` quando listas de tamanhos inferiores a 250 foram utilizadas.

Ainda em relação a este experimento, é preciso destacar que a implementação atual do `CRLRevocationHandler` não faz uso de nenhum tipo de *cache* para manter as listas de certificados revogados obtidas, o que implica que cada vez que um certificado é verificado a lista correspondente tem de ser obtida novamente junto ao repositório mantido pela autoridade de atributos. Após a avaliação dos resultados, consideramos este um ponto a ser melhorado em nossa implementação e acreditamos que o uso de um *cache* para as LCRs obtidas pode melhorar consideravelmente o desempenho desse verificador.

6.7 Avaliação dos resultados

A condução de todos os experimentos relatados nos permitiu não apenas comparar os módulos desenvolvidos com as soluções já existentes no JBoss, mas também identificar pontos que podem ser melhorados a fim de tornar nossa implementação mais eficiente. Mais especificamente, identificamos oportunidades de melhorias nos mecanismos de verificação de revogação que reduziriam o impacto causado por esses mecanismos no desempenho dos módulos.

No caso do `OCSPRevocationHandler`, verificamos que cenários que envolvem a verificação de mais de um certificado de atributos podem ser muito custosos em termos de desempenho por conta da incapacidade do verificador de processar múltiplos certificados de uma vez só. Por conta disso, planejamos modificar a implementação desse verificador de forma que ele seja capaz de lidar com múltiplos certificados, executando uma única requisição para o serviço de consulta OCSP contendo o número de todos os certificados envolvidos. É evidente que essa solução pressupõe que os certificados podem ser tratados pelo mesmo serviço de consulta, o que nem sempre é verdade, já que autoridades diferentes podem especificar serviços de consulta de revogação distintos. Em outras palavras, mesmo com esse aprimoramento, o verificador pode se ver obrigado a tratar cada certificado separadamente em situações onde cada certificado precisa ser validado por um serviço de consulta OCSP distinto. Isso nos leva à conclusão de que em ambientes que envolvem muitas autoridades de atributos o protocolo OCSP pode não ser a melhor opção para verificar a revogação dos certificados de atributos.

Já com relação ao `CRLRevocationHandler`, pudemos verificar que a implementação de um *cache* para as listas de certificados revogados reduziria consideravelmente o custo associado à execução desse verificador. Outro fator que contribuiria para uma maior eficiência do verificador seria a implementação de um mecanismo que fosse capaz de obter as listas de forma

assíncrona, mantendo o *cache* local sempre atualizado e evitando que o verificador tivesse que obter essas listas em tempo de execução. Planejamos não só incluir ambos os serviços em nossos trabalhos futuros, como também refazer os experimentos para analisar o impacto desses serviços no tempo de execução do verificador.

Independentemente dos custos associados à verificação de revogação dos certificados de atributos, devemos ressaltar que normalmente o *cache* de segurança do servidor de aplicações se encontra habilitado em sistemas em produção. Administradores normalmente optam pelo uso desse *cache* não apenas por motivos de desempenho, mas também de economia de recursos. Módulos de segurança que se conectam a bancos de dados ou serviços de diretórios podem afetar drasticamente a execução das aplicações se todos os clientes tiverem de ser reautenticados a cada chamada feita para as aplicações, já que recursos valiosos, como conexões, são utilizados freqüentemente por esses módulos.

Como vimos em nosso primeiro experimento, o uso do *cache* de segurança iguala o desempenho dos módulos e elimina qualquer tipo de sobrecarga imposta pela manipulação dos certificados de atributos, já que os processos de validação, mapeamento de papéis e verificação de revogação são executados apenas na primeira chamada feita por um cliente para um componente protegido ou quando o prazo de validade do *cache* expira. Por isso, podemos dizer que, mesmo com a configuração de verificadores de revogação, os módulos desenvolvidos podem perfeitamente ser utilizados para proteção de aplicações Java EE em produção sem apresentar desempenho inferior às demais soluções já oferecidas pelo JBoss.

7 Considerações finais

Até onde pudemos averiguar, nossa extensão da infra-estrutura de segurança do JBoss faz deste servidor de aplicações Java EE o único a oferecer suporte à IGP e aos certificados de atributos X.509, comportando tanto o modelo *pull* quanto o modelo *push* de propagação de certificados. Neste capítulo, apresentamos nossas considerações a respeito do projeto desenvolvido. Mais especificamente, discutimos as contribuições mais importantes que resultaram da execução deste projeto e apresentamos nossos planos quanto a trabalhos futuros.

7.1 Principais contribuições

Consideramos que as principais contribuições deste trabalho foram a integração da infra-estrutura de gerenciamento de privilégios do padrão X.509 à plataforma Java EE, um esforço inédito que resultou em uma infra-estrutura de segurança mais robusta e flexível para aplicações Java EE, e a execução de experimentos que nos permitiram avaliar o custo real associado à manipulação dos certificados de atributos X.509 dentro dessa infra-estrutura. Até o presente momento, o trabalho desenvolvido resultou na seguinte publicação:

- *Um Serviço de Autorização Java EE Baseado em Certificados de Atributos X.509* [17]: artigo publicado no *VII Brazilian Symposium on Information and Computer Systems Security (SBSeg 2007)*. Descreve a integração dos certificados de atributos X.509 à infra-estrutura de segurança do JBoss, com ênfase na arquitetura flexível que permite a configuração de vários mecanismos, como o validador dos certificados e os verificadores de revogação. Neste artigo não abordamos os resultados experimentais.

Encontra-se em curso a preparação de um segundo artigo contendo os resultados de nossos experimentos, analisando a sobrecarga imposta pelos diversos mecanismos que manipulam os certificados de atributos X.509 em nossa implementação.

As seções a seguir apresentam em maiores detalhes as contribuições resultantes de nosso trabalho.

7.1.1 Integração da IGP à plataforma Java EE

A integração da infra-estrutura de gerenciamento de privilégios do padrão X.509 à plataforma Java EE teve como motivação principal o estabelecimento de uma infra-estrutura de segurança flexível, capaz de autorizar o acesso de usuários a partir dos papéis extraídos de certificados de atributos X.509. Tais certificados podem ser extraídos de repositórios ou fornecidos pelos próprios usuários. Atualmente, os servidores de aplicações oferecem formas limitadas de associação entre usuários e seus papéis, que obrigam as autoridades que gerenciam os privilégios dos usuários a manterem essas associações em repositórios disponíveis para consulta.

Ao integrar a IGP e os certificados de atributos X.509 à infra-estrutura de segurança do JBoss, nós fomos capazes de remover as limitações existentes e oferecer uma flexibilidade muito maior às autoridades que gerenciam os privilégios desses usuários. Como a integridade das informações contidas nos certificados é protegida pela assinatura digital, as autoridades de atributos podem escolher se mantém os certificados gerados em repositórios disponíveis para consulta, ou se entregam tais certificados diretamente a seus titulares. No primeiro caso, nossa extensão da infra-estrutura de segurança é configurada para buscar pelos certificados no repositório mantido pela autoridade de atributos. No segundo, ela é configurada para obter os certificados diretamente do contexto de segurança da chamada feita pelo usuário a um componente protegido. Em comparação com outros projetos de segurança estudados, nosso trabalho é o único a oferecer suporte tanto ao modelo *pull* quando ao modelo *push* de propagação de certificados.

No entanto, as contribuições da IGP à plataforma Java EE não se limitam à flexibilidade da divulgação de certificados que é oferecida às autoridades de atributos. A assinatura digital dos certificados de atributos que contém os papéis dos usuários oferece garantias de integridade e origem, permitindo ao serviço de segurança do servidor de aplicações verificar que os papéis foram atribuídos ao usuário por uma autoridade de confiança e que tais atribuições não foram alteradas após a emissão do certificado. Essas características, além de fundamentais para a implementação do suporte ao modelo *push*, trazem também vantagens no caso do modelo *pull*, uma vez que a segurança das informações contidas no certificado não está sujeita à segurança do repositório onde esse certificado se encontra armazenado. Em outras palavras, a integridade das informações dos certificados de atributos não depende das políticas de segurança utilizadas pelo repositório.

É importante destacar que, no presente momento, um trabalho para inclusão de nosso projeto à distribuição padrão do JBoss já foi iniciado. A integração dos módulos JAAS baseados nos certificados de atributos X.509 despertou interesse entre desenvolvedores e usuários do JBoss, tanto pela flexibilidade de gerenciamento que nossa solução oferece às autoridades de

atributos quanto pelas garantias adicionais de segurança fornecidas pela assinatura digital dos certificados.

7.1.2 Avaliação do desempenho dos módulos implementados

No Capítulo 6, discutimos os experimentos que realizamos a fim de determinar os custos de desempenho associados ao uso de certificados de atributos X.509 para o armazenamento dos papéis dos usuários. Em particular, os experimentos com o verificador padrão, com o mecanismo de mapeamento de papéis e com os verificadores de revogação forneceram importantes informações a respeito da sobrecarga real imposta pelas diferentes operações que envolvem a manipulação dos certificados de atributos.

Os resultados obtidos nos experimentos com os verificadores de revogação foram particularmente importantes, pois nos permitiram não apenas avaliar o desempenho de cada mecanismo isoladamente, mas também identificar formas de melhorar esse desempenho. Além disso, pudemos comparar os dois mecanismos e observar que a verificação de revogação por LCRs sem o uso de *cache* só apresenta desempenho superior ao protocolo OCSP para listas com poucas entradas. Até onde pudemos averiguar, resultados dessa natureza não foram publicados na literatura.

7.2 Trabalhos futuros

Esta seção apresenta nossas idéias para pesquisas e trabalhos futuros.

Suporte ao modelo push via HTTP. Como vimos no Capítulo 5, o modelo *push* de propagação de certificados está disponível apenas para clientes de aplicações EJB. A fim de disponibilizar esse modelo também a clientes de aplicações *Web*, pretendemos estudar a viabilidade de implementar algum mecanismo que permita aos navegadores *Web* armazenar os certificados de atributos dos clientes e apresentá-los ao interagir com uma aplicação protegida. Teremos também de estudar maneiras de acomodar a passagem desses certificados do navegador para o servidor de aplicações através do protocolo HTTP.

Aprimoramento dos verificadores de revogação. Os experimentos realizados no Capítulo 6 com os verificadores de revogação nos revelaram oportunidades para aprimorar o desempenho desses verificadores. Mais especificamente, pretendemos adicionar ao `OCSPRevocationHandler` a capacidade de tratar múltiplos certificados de atributos com uma única requisição OCSP.

Além disso, tencionamos adicionar ao `CRLRevocationHandler` um *cache* para manter as listas de certificados revogados obtidas junto às autoridades de atributos. Um mecanismo de obtenção assíncrona dessas listas também deverá ser implementado.

A validação da cadeia de confiança da autoridade de atributos é um outro ponto passível de aprimoramento. A implementação atual do processo de validação concentra-se exclusivamente na verificação do certificado de atributos, assumindo que o CCP da autoridade de atributos contido na *keystore* do servidor é sempre válido. Uma implementação mais robusta deve se ocupar também da validação de tal CCP e toda a sua cadeia de confiança. A revogação de um dos certificados da cadeia de confiança deve resultar na revogação de todos os certificados por ele assinados.

Testes de interoperabilidade. Além dos testes de desempenho dos módulos desenvolvidos, consideramos que seria interessante testar a interoperabilidade de nossa implementação com certificados de atributos X.509 gerados por outras ferramentas. Em particular, pretendemos alterar o *Privilege Allocator* do PERMIS para que este se utilize do atributo definido pelo padrão X.509 para o armazenamento dos papéis dos clientes. Em seguida, utilizaremos nossa versão alterada do *Privilege Allocator* para gerar os certificados de atributos dos clientes, e testaremos o funcionamento dos módulos implementados com os certificados gerados pelo PERMIS.

Integração do serviço com outras aplicações. A arquitetura de nosso projeto foi desenhada de forma que os módulos JAAS desenvolvidos tivessem a menor complexidade possível, servindo apenas como ponto de integração entre o serviço de segurança do JBoss e as classes que são realmente responsáveis pela manipulação dos certificados de atributos dos clientes. Em outras palavras, toda a funcionalidade relacionada aos certificados de atributos X.509 pode facilmente ser separada em um sub-sistema totalmente independente do servidor de aplicações, oferecendo a outras aplicações os mecanismos de validação, extração e mapeamento de papéis, verificação de revogação e acesso a repositórios de certificados.

Com o objetivo de facilitar o uso de nossos mecanismos por outras aplicações, tencionamos criar um projeto separado do servidor de aplicações que disponibilize as funcionalidades desenvolvidas através de fachadas bem definidas. Assim, ao invés de um serviço fortemente vinculado ao servidor de aplicações, teremos um projeto que poderá ser utilizado por outras aplicações que desejarem fazer uso dos certificados de atributos X.509 para implementar seus processos de autorização.

Referências

[1] Akenti Project Home Page. `http://dsd.lbl.gov/Akenti/`. Último acesso em 10 de janeiro de 2008.

[2] Apache Geronimo. `http://geronimo.apache.org`. Último acesso em 10 de janeiro de 2008.

[3] BEA Weblogic. `http://www.bea.com`. Último acesso em 10 de janeiro de 2008.

[4] Konstantin Beznosov and Yi Deng. A Framework for Implementing Role-Based Access Control Using CORBA Security Service. In *ACM Workshop on Role-Based Access Control*, pages 19–30, 1999.

[5] David W. Chadwick and Alexander Otenko. RBAC Policies in XML for X.509 Based Privilege Management. In *Proceedings of 17th IFIP International Conference on Information Security - SEC2002*, pages 39–54, 2002.

[6] David W. Chadwick and Alexander Otenko. The PERMIS X.509 role based privilege management infrastructure. In *Proceedings of the 7th ACM Symposium on Access Control Models And Technologies - SACMAT*, pages 135–140, 2002.

[7] David W. Chadwick, Alexander Otenko, and Edward Ball. Role-Based Access Control With X.509 Attribute Certificates. *IEEE Internet Computing*, 7(2):62–69, 2003.

[8] David Chappell. *Understanding .NET, Second Edition*. Addison-Wesley, 2006.

[9] Jason Crampton and Hemanth Khambhammettu. Authorization and Certificates: Are We Pushing When We Should Be Pulling? In *Proceedings of IASTED International Conference on Communication, Network and Information Security*, pages 62–66, 2003.

[10] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.1. RFC 4346 (Proposed Standard), April 2006.

[11] S. Farrel and R. Housley. An Internet Attribute Certificate Profile for Authorization. RFC 3281 (Proposed Standard), April 2002.

[12] David F. Ferraiolo, Ravi S. Sandhu, Serban I. Gavrila, D. Richard Kuhn, and Ramaswamy Chandramouli. Proposed NIST standard for role-based access control. *ACM Transactions of Information and System Security*, 4(3):224–274, 2001.

[13] Marc Fleury and Francisco Reverbel. The JBoss Extensible Server. In *Middleware 2003 — ACM/IFIP/USENIX International Middleware Conference*, volume 2672 of *LNCS*, pages 344–373. Springer-Verlag, 2003. `http://www.ime.usp.br/~reverbel/papers/jboss-mw2003.pdf`.

[14] Alan O. Freier, Philip Karlton, and Paul C. Kosher. The SSL Protocol Version 3.0. http://wp.netscape.com/eng/ssl3/draft302.txt, 1996. Último acesso em 10 de janeiro de 2008.

[15] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns - Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional Computing Series. Addison-Wesley, 1st edition, 1995.

[16] James Gosling, Bill Joy, Guy Steele, and Gilad Bracha. *The Java Language Specification, Third Edition*. Prentice Hall PTR, 2005.

[17] Stefan N. Guilhen and Francisco Reverbel. Um Serviço de Autorização Java EE Baseado em Certificados de Atributos X.509. In *Proceedings of the 7th Brazilian Symposium on Information and Computer Systems Security - SBSeg*, pages 262–275, 2007.

[18] Peter Gutmann. PKI Technology Survey and Blueprint. `http://www.cs.auckland.ac.nz/~pgut001/pubs/pkitech.pdf`. Último acesso em 10 de janeiro de 2008.

[19] Peter Gutmann. PKI: It's Not Dead, Just Resting. *IEEE Computer*, 35(8):41–49, 2002.

[20] R. Housley, W. Polk, W. Ford, and D. Solo. An Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 3280 (Proposed Standard), April 2002.

[21] IBM Websphere. `http://www-306.ibm.com/software/websphere`. Último acesso em 10 de janeiro de 2008.

[22] John Iliadis, Stefanos Gritzalis, Diomidis Spinellis, Danny De Cock, Bart Preneel, and Dimitris Gritzalis. Towards a framework for evaluating certificate status information mechanisms. *Computer Communications*, 26(16):1839–1850, 2003.

[23] ITU-T. *ITU-T Recommendation X.812 ISO/IEC 10181-3. Security Frameworks for Open Systems: Access Control Framework*, November 1995.

[24] ITU-T. *ITU-T Recommendation X.509 ISO/IEC 9594-8. The Directory: Public Key and Attribute Certificate Frameworks*, May 2001.

[25] ITU-T. *ITU-T Recommendation X.680 ISO/IEC 8824-1. Abstract Syntax Notation One (ASN.1): Specification of basic notation*, July 2002.

[26] ITU-T. *ITU-T Recommendation X.500 ISO/IEC 9594-1. The Directory: Overview of concepts, models and services*, August 2005.

[27] Java Applets. `http://java.sun.com/applets/`. Último acesso em 10 de janeiro de 2008.

[28] Java Platform, Micro Edition (Java ME). `http://java.sun.com/javame/index.jsp`. Último acesso em 10 de janeiro de 2008.

[29] JBoss Application Server. `http://www.jboss.com`. Último acesso em 10 de janeiro de 2008.

[30] William Johnston, Srilekha Mudumbai, and Mary Thompson. Authorization and Attribute Certificates for Widely Distributed Access Control. In *Proceedings of 7th Workshop on Enabling Technologies, Infrastructure for Collaborative Enterprises - WETICE*, pages 340–345, 1998.

[31] Per Kaijser, Tom Parker, and Denis Pinkas. SESAME: The solution to security for open distributed systems. *Computer Communications*, 17(7):501–518, 1994.

[32] Silvio Micali. NOVOMODO: Scalable Certificate Validation And Simplified PKI Management. In *Proceedings of First Anual PKI Research Workshop*, pages 15–26, 2002.

[33] M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams. X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP. RFC 2560 (Proposed Standard), June 1999.

[34] MySQL. `http://www.mysql.org`. Último acesso em 10 de janeiro de 2008.

[35] C. Neuman, T. Yu, S. Hartman, and K. Raeburn. The Kerberos Network Authentication Service (V5). RFC 4120 (Proposed Standard), July 2005. Updated by RFCs 4537, 5021.

[36] OASIS Official Web Site. `http://www.oasis-open.org/home/index.php`. Último acesso em 10 de janeiro de 2008.

[37] Object Management Group. *Security Service Specification, version 1.8*, March 2002.

[38] Object Management Group. *Common Object Request Broker Architecture: Core Specification, version 3.0.3*, March 2004.

[39] OpenLdap. `http://www.openldap.org`. Último acesso em 10 de janeiro de 2008.

[40] Organization for the Advancement of Structured Information Standards (OASIS). *Extensible Access Control Markup Language (XACML) Version 2.0*, February 2005.

[41] T. A. Parker. A secure European system for applications in a multi-vendor environment (the SESAME project). In *Proceedings of the 14th American National Security Conference*, 1991.

[42] Permis Project Home Page. `http://sec.isi.salford.ac.uk/permis/`. Último acesso em 10 de janeiro de 2008.

[43] W. Polk, R. Housley, and L. Bassham. Algorithms and Identifiers for the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 3279 (Proposed Standard), April 2002.

[44] Ronald L. Rivest. Can We Eliminate Certificate Revocation Lists? In *Proceedings of Financial Cryptography*, pages 178–183, 1998.

[45] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-Based Access Control Models. *IEEE Computer*, 29(2):38–47, 1996.

[46] Douglas Schmidt, Michael Stal, Hans Rohnert, and Frank Buschmann. *Pattern-Oriented Software Architecture Volume 2 - Patterns for Concurrent and Networked Objects*. Wiley Series In Software Design Patterns. Wiley, 1st edition, 2000.

[47] Sun Microsystems. Java Authentication and Authorization Service (JAAS) Reference Guide for the J2SE Development Kit 5.0. `http://java.sun.com/j2se/1.5.0/docs/guide/security/jaas/JAASRefGuide.html`. Último acesso em 10 de janeiro de 2008.

[48] Sun Microsystems. Java Secure Socket Extension (JSSE) Reference Guide for the Java 2 Platform Standard Edition 5. `http://java.sun.com/j2se/1.5.0/docs/guide/security/jsse/JSSERefGuide.html`. Último acesso em 10 de janeiro de 2008.

[49] Sun Microsystems. *Java Naming and Directory Interface Specification, v1.2*, July 1999.

[50] Sun Microsystems. *Java Message Service Specification, v1.1*, April 2002.

[51] Sun Microsystems. *Enterprise Java Beans Specification, v2.1*, November 2003.

[52] Sun Microsystems. *J2EE Connector Architecture Specification, v1.5*, November 2003.

[53] Sun Microsystems. *Java Servlet Specification, v2.4*, November 2003.

[54] Sun Microsystems. *Enterprise Java Beans Specification, v3.0*, May 2006.

[55] Sun Microsystems. *Java Database Connectivity Specification, v4.0*, December 2006.

[56] Sun Microsystems. *Java Platform Enterprise Edition Specification, v1.5*, May 2006.

[57] Sun Microsystems. *Java Server Faces Specification, v1.2*, May 2006.

[58] Sun Microsystems. *Java Server Pages Specification, v2.1*, May 2006.

[59] Routo Terada. *Segurança de Dados - Criptografia em Redes de Computador*. Editora Edgar Blücher, 2000.

[60] The Legion of the Bouncy Castle. `http://www.bouncycastle.org`. Último acesso em 10 de janeiro de 2008.

[61] The Open Group. *Authorization (AZN) API*. Open Group Tecnical Standard, 2000.

[62] John R. Vacca. *Public Key Infrastructure: Building Trusted Applications and Web Services*. Auerbach, 2004.

[63] Jeroen van de Graaf and Osvaldo Carvalho. Reflecting on X.509 and LDAP, or How separating identity and attributes could simplify a PKI. In *Proceedings of IV Workshop em Segurança de Sistemas Computacionais - WSEC*, pages 37–48, 2004.

[64] Wei Zhou and Christoph Meinel. Implement Role-Based Access Control with Attribute Certificates. Technical report, Institut fur Telematik, Universitat Trier, Forschungsbericht 03-5, 2003.