# 4
# Estudos de Caso

Este capítulo apresenta a instanciação do modelo proposto no capítulo 3 para dois casos concretos de autenticação e controle de acesso em cenários multi-institucionais. A arquitetura será adaptada levando em consideração as características do serviço e os requisitos da aplicação que faz uso do mesmo.

O primeiro caso trata o serviço de diretório LDAP, um banco de dados que tem como característica a fácil distribuição da base de informação entre servidores, cada um sendo alocado em uma instituição. Esse caso foi o que motivou o trabalho de pesquisa sobre autenticação inter-instituições, de onde foi derivado o modelo proposto, por isso ele será explorado com profundidade.

Para o segundo caso de teste, usamos um serviço muito empregado para a troca de arquivos, o FTP.

## 4.1
## LDAP

O serviço de diretório pode ser visto como um banco de dados especializado que tem como característica marcante o suporte à grande quantidade de pesquisas. Por outro lado, não existe compromisso com desempenho na atualização desses dados. A idéia é oferecer, via rede, um serviço otimizado de busca que proveja um acesso rápido e que trabalhe com informações não muito mutáveis.

Diretórios de propósito geral possuem mecanismos que permitem a criação e adequação de esquemas de informações para necessidades específicas. Geralmente, é provido um conjunto de tipos básicos, assim como nos bancos de dados, que podem ser usados para modelar os tipos de informação que serão armazenados.

No início dos anos 90, duas entidades de padronização, a ISO e a CCITT (futura ITU), publicaram a especificação de um serviço de

diretório de propósito geral, o X500, que provia funções de consultas bem poderosas e foi criado dando alto suporte à distribuição das informações. Mas devido à complexidade da especificação, a implementação do serviço era difícil, o que causou inicialmente um problema de interoperabilidade entre implementações diferentes - ainda que ele houvesse sido proposto como um padrão aberto. A dependência do X500 com os protocolos de rede OSI, que acabaram não tendo a disseminação esperada, também contribuiu para a sua não adoção.

Como o protocolo de acesso ao X500 era complexo, e exigia a pilha OSI, criou-se um outro serviço que oferece um protocolo mais simples, o LDAP (*Lightweight Directory Access Protocol*), para fazer a ponte entre clientes e servidor X500. Com o passar do tempo, percebeu-se que todo o acesso era feito via LDAP, o que levou à decisão de torná-lo um servidor independente, adicionando ao mesmo a capacidade de armazenamento dos dados, seguindo, mas simplificando, as características do X500. A descrição do servidor e de seu protocolo de acesso se transformou em um padrão, descrito nas RFCs [30, 29]. Atualmente, há diversas implementações do padrão, dentre elas o OpenLDAP [23], que inclui o servidor, ferramentas de linha de comando para acesso e manipulação e biblioteca para desenvolvimento de aplicações. A aceitação de um padrão permite desacoplar aplicações e implementações específicas do serviço de diretórios, ou seja, um administrador poderá substituir uma implementação do serviço por outra, sem afetar o funcionamento das aplicações.

O padrão LDAP [10] pode ser decomposto em quatro modelos [12, 2]: modelo de informação, de nome, funcional e de segurança.

O modelo de informação define os tipos de dados suportados e a unidade básica de armazenamento, chamada de *entrada*. Cada entrada é formada de um conjunto de atributos que correspondem a um nome e um valor. Os atributos são criados a partir de tipos básicos como strings e bytes.

Um esquema define a coleção dos atributos e classes de objetos que podem ser usadas na composição das entradas. O padrão impõe que cada entrada seja associada ao menos a uma classe. As classes de objetos especificam quais atributos da entrada são obrigatórios e quais são opcionais. Quando a entrada adota mais de uma classe, o conjunto de atributos que ela pode conter é dado pela união de todos os atributos descritos nas classes. As figura 4.1 e 4.2 mostram, respectivamente, um exemplo da definição de um atributo e de uma classe de objeto, e um exemplo de uma entrada.

Os valores 2.5.4.41 e 1.1.2.2.2 na figura 4.1 são exemplos de identificadores de objetos (OID), usados para distinguir unicamente atributos, classes

Atributo:

```
attributetype( 2.5.4.41 NAME 'name'
DESC 'Nome associado com o objeto'
EQUALITY caseIgnoreMatch
SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 )
```

Classe de Objeto:

```
objectclass ( 1.1.2.2.2 NAME 'person'
DESC 'person'
MUST ( name $ surName )
MAY ( phoneNumber $ address ) )
```

Figura 4.1: Exemplo da definição de um atributo e de uma classe de objeto.

```
dn: uid=brunoos,ou=people,o=puc-rio,dc=br
objectClass: person
name: Bruno
surName: Oliveira Silvestre
phoneNumber: +55 021 3114 1500
address: Rua Marquês de São Vicente - Gávea
```

Figura 4.2: Exemplo de uma entrada LDAP.

e outros elementos. Uma organização pode requisitar o OID junto à IANA (*Internet Assigned Numbers Authority*) – uma autoridade responsável pelo controle dos identificadores – e usá-lo nas suas definições, evitando conflitos com elementos definidos em outras instituições.

O modelo de nomes especifica como as entidades são organizadas no diretório, ou seja, a estrutura lógica, e como elas são referenciadas dentro dessa estrutura. O LDAP suporta uma organização em forma de árvore, parecida com a dos sistemas de arquivo do UNIX. Entretanto, diferente dos diretórios do sistema de arquivo, no serviço de diretórios todos os nós da árvore são também entradas, e assim, todos podem, além de possuir filhos, guardar informações.

Cada entrada do serviço de diretórios é identificada unicamente através de um atributo chamado *distinguished name* (DN). Ele é composto pelo caminho da raiz da árvore até o local onde a entrada se encontra concatenado com um identificador da entrada. A figura 4.3 mostra um exemplo de estrutura de nomes.

Uma característica interessante do serviço é que as informações mostradas na figura 4.3 podem estar distribuídas entre vários servidores. Isso

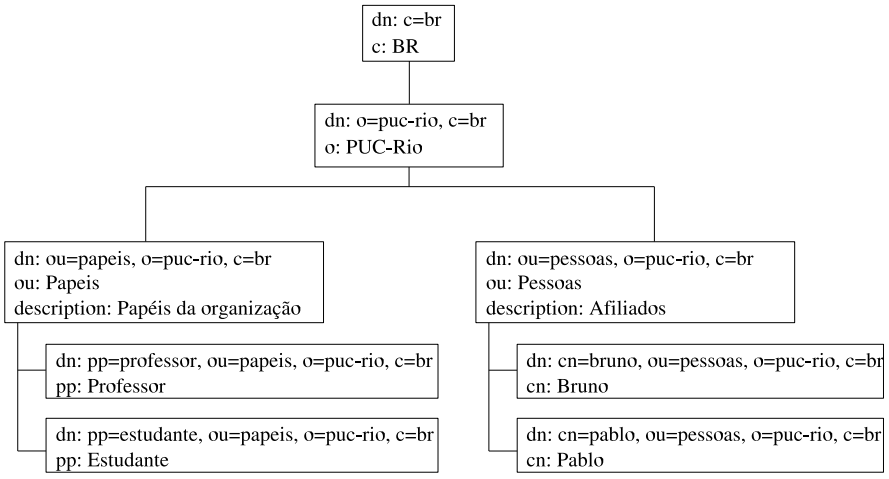


Figura 4.3: Exemplo de uma estrutura do diretório LDAP.

permite que uma instituição escolha a melhor maneira de manter suas informações, administrando-as de maneira centralizada ou delegando a gerência de sub-árvores a diferentes setores. E permite ainda que instituições diferentes colaborem na criação de uma árvore única de informações: como exemplo, podemos ter uma federação de instituições de pesquisa fornecendo um cadastro nacional de pesquisadores.

O modelo funcional define o conjunto de ações que pode ser usado para manipular o diretório, como consultas, alterações, comparações e exclusões.

O modelo de segurança trata dos mecanismos de autenticação que serão empregados para identificar um cliente que deseja acesso à base. Essa identificação pode se dar através de uma simples senha ou empregando métodos mais complexos, como o Kerberos. A especificação da versão 3 (a atual) do LDAP obriga o suporte a apenas um método de autenticação, o DIGEST-MD5, e deixa como opção o uso da arquitetura SASL (seção 4.1.1) como forma de ampliar os mecanismos de autenticação [20, 12].

A autenticação de um usuário só faz sentido associada a um mecanismo de permissões de acesso. Infelizmente, o padrão não define como o controle de acesso deve ser feito, ficando a cargo dos implementadores proverem soluções. O OpenLDAP, assim como alguns outros servidores, permite especificar direitos de usuários sobre classes de entradas ou sobre atributos, com diferentes granularidades, em uma lista de permissões.

Para garantir integridade e confidencialidade de informações transmitidas entre servidor e cliente, o LDAP determina que exista suporte a conexões seguras com o padrão TLS, podendo um cliente iniciar uma sessão segura a qualquer momento.

### 4.1.1 SASL – Simple Authentication and Security Layer

SASL [30, 29] é um arcabouço para desenvolvimento de métodos autenticação em protocolos orientados a conexão (por exemplo, o TCP), permitindo um desacoplamento do processo de identificação do usuário do resto do sistema. Ele foi desenvolvido para trabalhar com a abordagem de desafios, isto é, o servidor emite desafios aos clientes que devem respondê-los de forma apropriada. Se todas as respostas forem aceitas, o cliente é tido como autenticado e o controle é passado para o servidor.

Os desafios e suas respectivas respostas dependem do método de autenticação selecionado. Assim sendo, no início da execução do protocolo de negociação o cliente deve informar ao servidor qual método será usado. Caso o servidor não suporte o método proposto, ele gera um erro e o processo de autenticação é interrompendo. Perceba que as partes do SASL acopladas no cliente e no servidor é que conduzem a comunicação, assim, quando o cliente seleciona um método, a sua parte do SASL também deve suportar o método escolhido, caso contrário não será possível responder a nenhum desafio.

Uma flexibilidade interessante do padrão SASL é o *proxy* de autenticação. Após o cliente ter as suas credenciais aceitas pelo SASL, ele poderá fornecer uma identificação, diferente da apresentada na fase de autenticação, que será usada para referenciá-lo posteriormente. Por exemplo, suponha que dois usuários do sistema, Pedro e Silvana, acumulem a função de administradores e que o método de autenticação se dê através de *login* e senha. Ao invés de ambos compartilharem uma mesma conta com direitos plenos, eles utilizam as suas credenciais pessoais para autenticação no sistema, e após serem aceitos, usam o mecanismo *proxy* para assumir a identificação de administrador. Isto permite a ambos a condição de administrador, ao mesmo tempo em que o sistema continua sendo capaz de distinguir um usuário do outro. Em nossa implementação, utilizamos esse mecanismo para implementar o mapeamento entre papéis externos e papéis locais discutidos na seção 3.2.

Vale destacar que o mecanismo de *proxy* não é uma parte obrigatória para a autenticação do usuário, mas apenas uma facilidade oferecida pela arquitetura SASL.

### 4.1.2 Arquitetura para o LDAP

O LDAP, devido à padronização e à sua facilidade em distribuir ramos da base de informações por diversos servidores localizados geograficamente distribuídos, vem sendo apontado como uma boa solução para a troca de informações entre organizações. Contudo, a falta de um mecanismo padrão para autenticação de membros externos acaba levando à adoção das estratégias de conta de acesso compartilhada e replicação de cadastro, como exposto na seção 3.1.

Desenvolvemos uma aplicação web, baseada na arquitetura proposta, que tem como objetivo disponibilizar a usuários autenticados informações recolhidas em serviços LDAP localizados em diferentes instituições. O usuário deve prover para a aplicação um bilhete, que será utilizado para autenticação nos diversos servidores. A figura 4.4 mostra o cenário montado para este estudo de caso.

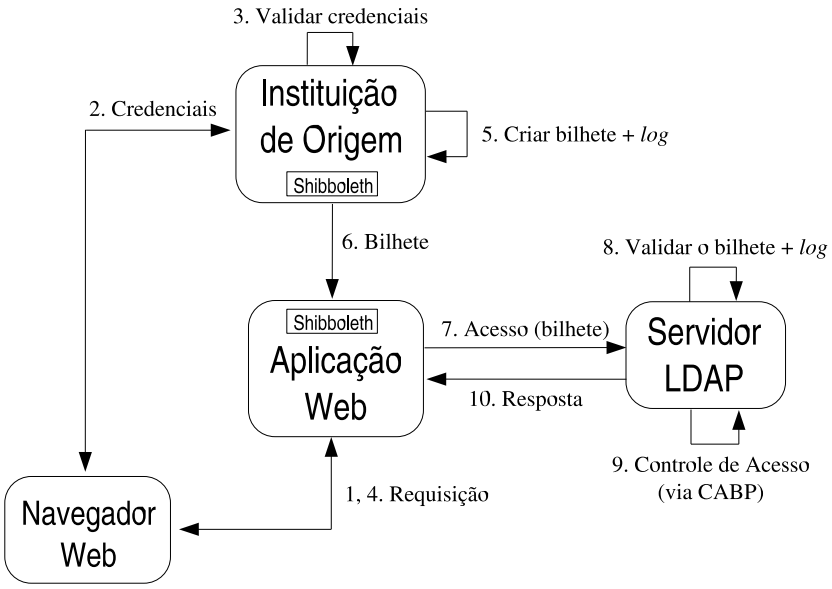


Figura 4.4: Aplicação de busca sobre informações.

#### Criação do Bilhete

O Shibboleth desempenha a função de criador dos bilhetes que serão usados pela aplicação. Como apresentado na seção 2.2, ele é capaz de autenticar e transferir atributos dos usuários de forma segura. Neste contexto, o atributo transmitido para a aplicação é o bilhete.

A implementação do Shibboleth localizado na instituição de origem emprega o conceito de fontes de dados, que é o modo como as informações dos usuários são obtidas para que os atributos sejam criados. Além das fontes já disponíveis, como por exemplo, para consulta a bancos de dados ou a serviços de diretórios, o Shibboleth possibilita que novas fontes de dados sejam desenvolvidas e acopladas, permitindo estender a obtenção de informações para outros meios.

Desenvolvemos uma nova fonte de dados que pesquisa o papel que o usuário desempenha em uma base LDAP da organização, cria o identificador, ajusta o tempo de validade do bilhete, executa a assinatura digital e realiza o *log* para auditoria.

Voltando à figura 4.4, podemos agora acompanhar a seqüência de passos. (1) Quando o usuário acessa a aplicação de busca, (2,3) ele é conduzido, através do Shibboleth, até sua instituição de origem para que a autenticação seja realizada. Após ser identificado, (4) o usuário é novamente redirecionado para a aplicação de busca, o que faz o Shibboleth que protege essa aplicação entrar em contato com a instituição de origem solicitando os atributos. Através do identificador do usuário, (5) o Shibboleth origem pesquisa seus dados na base LDAP, monta o bilhete e o envia na forma de atributo, que será repassado para a aplicação web. (6) Esta recebe o bilhete e (7) o usa para acessar as bases LDAP das instituições apropriadas e extrair as informações. (8,9,10) O servidor LDAP verifica o bilhete, aplica as regras de controle de acesso baseadas no papel e devolve o resultado para o cliente.

Utilizamos o Shibboleth como meio de transferência do bilhete entre a origem e a aplicação devido às suas característica que garantem a entrega segura dos dados. Outra forma (menos segura) de obter o bilhete é fornecer as credenciais do usuário à aplicação, que ficaria incumbida de entrar em contato com a instituição para retirar o bilhete. Dessa forma, a aplicação deveria ser de inteira confiança, pois de posse das credenciais ela pode recuperar, além do bilhete, informações indevidas sobre o usuário. O Shibboleth impede que a isso aconteça, atuando como intermediário e filtrando os dados que são distribuídos para cada aplicação web.

### Autenticação

Em nossa implementação, utilizamos o OpenLDAP que, como mencionado antes, dá suporte à arquitetura SASL. Para isso, o OpenLDAP utiliza a biblioteca Cyrus-SASL [9].

O Cyrus-SASL contempla o uso de *plugins* para facilitar a adição de novos métodos de autenticação, bem como funções de *callback,* que permitem aos *plugins* se comunicarem de uma forma padronizada com o programa usuário da biblioteca. Assim, desenvolvemos um *plugin* que possibilita ao servidor OpenLDAP e aos clientes do serviço usarem o bilhete como credencial de acesso. Esse *plugin* emprega os seguintes desafios:

- **password**: deve ser respondido com o bilhete.
- **username**: deve ser respondido com a identificação que será utilizada para o mecanismo de *proxy* de autenticação (discutido na seção 4.1.2).

A aplicação cliente, em nosso caso a aplicação web de busca, informa ao servidor OpenLDAP que a autenticação se dará pelo uso do bilhete. O servidor, através do *plugin*, responde lançando o desafio *password*, onde a aplicação envia o bilhete obtido como atributo do usuário. O *plugin* usa a identificação da instituição, extraída do bilhete, para recuperar a chave que é utilizada para a conferência da assinatura digital. Na atual implementação, as chaves são armazenadas em arquivos e o administrador fica responsável por gerenciar a inclusão, exclusão e atualização das chaves referentes a cada uma das organizações da federação. Uma outra abordagem seria usar uma estrutura de distribuição de chaves [1], automatizando o processo.

Além da assinatura, o *plugin* também confere se o prazo de validade não expirou. Caso as verificações sejam bem sucedidas o usuário é tido como autenticado e é identificado pelo papel que consta no bilhete (papel externo). No entanto, o servidor lança mais um desafio, o *username*, cuja resposta será usada com o mecanismo de *proxy* para alterar a identidade do usuário. Como esse mecanismo não é obrigatório, o cliente pode responder ao desafio com uma string vazia, indicando que não deseja utilizá-lo. Com isso, ele continuará sendo identificado pelo servidor através do papel externo, ou seja, o controle de acesso será realizado levando em consideração o papel externo. A figura 4.5 mostra o protocolo de interação entre a aplicação e o servidor LDAP.

Caso o cliente forneça uma outra identificação como resposta ao desafio *username*, e esta seja aceita pelo servidor LDAP, ele passará a ser reconhecido por essa nova identificação. Esse mecanismo é utilizado para a ativação de papéis internos e é discutido na seção seguinte.

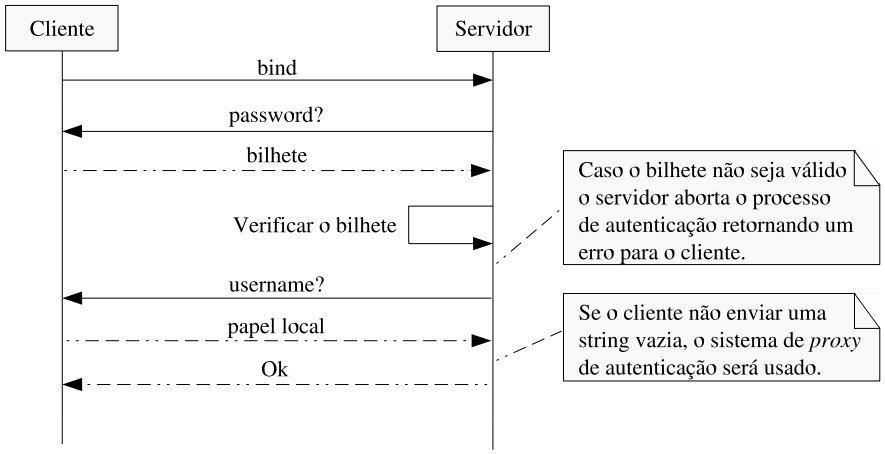


Figura 4.5: Protocolo de autenticação através do bilhete.

### Ativação dos Papéis e CABP

Na seção 3.2, discutimos a possibilidade dos papéis externos serem mapeados para um conjunto de papéis locais com o intuito de facilitar o gerenciamento dos direitos de acesso. O mecanismo de *proxy* disponibilizado pelo SASL pode ser usado para esse fim, ou seja, o cliente apresenta o bilhete como credencial para o acesso e, ao ser desafiado com *username*, ele informa (ativa) um dos papéis internos da instituição em vez de uma string vazia.

Na realidade, o SASL é apenas uma forma de identificar o cliente, sendo o OpenLDAP responsável por definir se o cliente pode ou não assumir a identidade proposta pelo *proxy*, impedindo que algum usuário tente fazer uso de um papel que não lhe cabe. O controle se dá da seguinte forma: o OpenLDAP recebe as duas identificações, o papel contido no bilhete e o identificador do *proxy*, e as mapeia para entradas da base, digamos $E_{Pa}$ e $E_{Pr}$ (o OpenLDAP faz o mapeamento das identificações para DNs seguindo regras estabelecidas em um arquivo de configuração). Então ele verifica, através de consulta aos atributos de $E_{Pa}$ e $E_{Pr}$, se existe autorização para um usuário autenticado como $E_{Pa}$ assumir o papel descrito por $E_{Pr}$. São os atributos *saslAuthzTo* e *saslAuthzFrom* que efetivamente indicam essa permissão; se existe um atributo *saslAuthzTo* localizado em $E_{Pa}$ apontando para $E_{Pr}$ ou um atributo *saslAuthzFrom* em $E_{Pr}$ apontado para $E_{Pa}$ então $E_{Pr}$ pode assumir a identidade $E_{Pa}$. Para exemplificar, suponha um diretório com as entradas (i), (ii) e (iii) como mostradas na figura 4.6. Se um usuário foi mapeado para a entrada (i) ele poderá efetuar a mudança de identidade para (ii) ou (iii). Porém, caso fosse possível efetuar a autenticação como as entradas (ii) ou (iii) não seria possível assumir nenhuma outra identidade.

O conjunto de papéis locais pode ser pré-definido e conhecido pelas ou-

```
(i)
dn: cn=pesquisador,o=rnp,ou=roles,o=puc-rio,dc=br
saslAuthzTo: cn=professor,ou=roles,o=puc-rio,dc=br

(ii)
dn: cn=estudante,ou=roles,o=puc-rio,dc=br
saslAuthzFrom: cn=pesquisador,o=rnp,ou=roles,o=puc-rio,dc=br

(iii)
dn: cn=professor,ou=roles,o=puc-rio,dc=br
```

Figura 4.6: Exemplo dos atributos *saslAuthzTo* e *saslAuthzFrom*.

tras organizações. Contudo, seguindo a abordagem da descoberta dinâmica do mapeamento, onde o usuário recupera do serviço quais papéis ele pode ativar, foi criado um esquema *role* contendo um atributo, *roleMapping*, onde são armazenados os nomes dos papéis locais que a entrada (papel externo) pode assumir (o esquema é mostrado na figura 4.7).

Então, a ativação dos papéis se dá da seguinte forma: quando o usuário se autentica via bilhete, mas não utiliza o mecanismo de *proxy*, as únicas informações que podem ser acessadas são os nomes dos papéis locais que ele tem direito de ativar (atributos *roleMapping*). Assim, o programa cliente realiza uma pesquisa que recupera essas informações, e em seguida, realiza uma segunda autenticação usando o bilhete e responde ao desafio *username* com um dos papéis internos recuperados – antes de realizar essa segunda autenticação, o programa cliente pode interagir com o usuário para que este selecione o papel interno adequado. Então, o usuário passará a ser identificado pelo servidor pelo papel interno, e não mais pelo papel externo.

O atributo *roleState* foi introduzido para simplificar a gerência do mapeamento. Esse atributo indica se o papel poderá ou não ser usado para a autenticação através do bilhete, tendo como valores permitidos *enable* e *disable*. Esse mecanismo possibilita ao administrador do diretório suspender o uso de um papel de forma fácil, sem a necessidade de excluí-lo.

A aplicação web, o *plugin* e os demais elementos discutidos foram usados no projeto do Grupo de Trabalho de Diretórios para a Educação, financiado pela Rede Nacional de Pesquisa (RNP) em 2003/2004. Essa aplicação foi utilizada para extrair informações sobre vídeos das bases LDAP, onde o controle era realizado de acordo com o papel que o pesquisador desempenhava em sua instituição de origem.

O apêndice A mostra a aplicação web desenvolvida seguindo o modelo

```
attributetype ( 8.1.1.1.1
  NAME 'role'
  DESC 'Papel interno'
  EQUALITY caseIgnoreMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 )

attributetype ( 8.1.1.1.2
  NAME 'roleState'
  DESC 'Estado atual do papel'
  EQUALITY caseIgnoreMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.15
  SINGLE-VALUE )

objectclass ( 8.1.1.2.1
  NAME 'roleMapping'
  DESC 'Mapeamento entre os papéis'
  SUP top AUXILIARY
  MUST ( role $ roleState ) )
```

Figura 4.7: Esquema utilizado no mapeamento de papéis.



proposto para o estudo de caso desta seção. O objetivo da aplicação é realizar buscas sobre informações de vídeos em servidores LDAP e exibí-las aos usuários cadastrados em outros servidores da federação de instituição. Utilizamos a linguagem Lua [14, 15] juntamente com a ferramenta CGILua [16], destinada à criação de aplicações web, e a biblioteca LuaLDAP [17], para acesso aos servidores LDAP. Houve a necessidade de estender a versão do LuaLDAP empregada (versão 1.0a) pois esta não oferece suporte ao uso de autenticação SASL. Adicionamos um conjunto de rotinas que usam a biblioteca Cyrus-SASL para gerar uma interface de acesso ao LDAP através da autenticação SASL.

## 4.2 FTP – File Transport Protocol

A motivação para a escolha do serviço de FTP [24] para implementação de um estudo de caso foi a sua grande difusão, sendo um dos principais meios de trocas de arquivos entre diferentes organizações.

Uma prática muito comum é replicar os arquivos em diversos servidores FTP espalhados geograficamente (*mirrors*) para maximizar a distribuição dos dados. Por exemplo, suponha um curso sendo ministrado à distância por um conjunto de instituições de ensino, onde os arquivos de exercícios

são distribuídos via FTP. Pode-se ter servidores em diferentes localidades, e instituições, para agilizar a distribuição. Assumindo que o conteúdo é acessível apenas pelos participantes do curso, há a necessidade de garantir o controle de acesso aos servidores. A estratégia de instanciação do modelo para a segurança em servidores FTP está na figura 4.8.

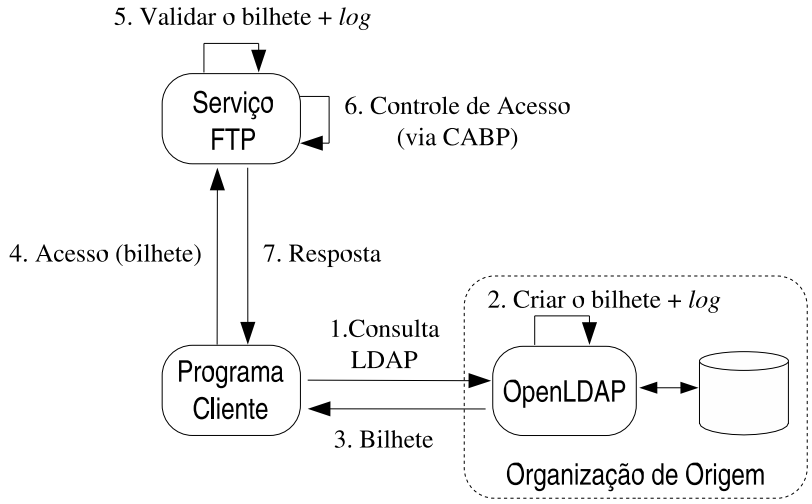


Figura 4.8: Arquitetura para o FTP.

### 4.2.1 Criação do Bilhete

Adotamos o serviço LDAP para criação dos bilhetes como forma de demonstrar o uso de tecnologias variadas na instanciação da arquitetura. O OpenLDAP foi desenvolvido seguindo uma abordagem em duas camadas, uma para o processamento do padrão LDAP e outra para o armazenamento das informações. Para isso, o OpenLDAP emprega uma interface padronizada para o acoplamento de módulos que fazem a ponte com os repositórios de dados. Por exemplo, a versão 2.1.25 (usada no caso de teste) nos permite escolher, dentre outros, o banco de dados de Berkeley, um banco de dados relacional, *shell* ou um outro serviço LDAP. O módulo *shell* possibilita selecionar programas que serão executados para tratar cada uma das requisições feitas à base, como por exemplo, consulta, exclusão, alteração, comparação, etc. A saída desses programas é usada para elaborar a resposta aos clientes.

As operações de interesse para a geração do bilhete são as de autenticação e de consulta – todas as demais são descartadas. Desenvolvemos um conjunto de rotinas em linguagem Lua para atender a essas requisições. Ao

```
attributetype ( 1.2.6.1.4.1.15996.2.1.1
  NAME 'ticket'
  DESC 'Bilhete para a autenticação'
  EQUALITY caseIgnoreMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.15
  SINGLE-VALUE )

objectclass ( 1.2.6.1.4.1.15996.2.2.1
  NAME 'ticketObject'
  DESC 'Classe para abrigar o bilhete.'
  SUP top STRUCTURAL
  MUST (cn $ ticket) )
```

Figura 4.9: Definição do atributo e classe de objeto para o LDAP.

solicitar acesso à base, o usuário deve se autenticar informando as suas credenciais, que são passadas para a rotina de tratamento da autenticação. Na atual implementação, a rotina suporta apenas autenticação via *login* e senha, e a validação é feita através de um arquivo de senhas. Contudo, outros meios de verificação podem ser empregados, como por exemplo, consulta a outra base LDAP. Aceitas as credenciais, o usuário é então mapeado para uma entrada da base, que possa identificá-lo posteriormente.

O bilhete é obtido através de uma consulta aos atributos da entrada para a qual se foi mapeado. A definição da classe de objeto e do atributo usados para especificar o bilhete está na figura 4.9. Quando o usuário realiza a consulta, a requisição e os seus parâmetros são repassados à rotina Lua que gera o bilhete dinamicamente e o retorna ao OpenLDAP, que por sua vez o envia para o usuário.

Um dos parâmetros recebidos é a entrada, de onde é possível determinar para quem o bilhete deve ser criado. A rotina pesquisa em alguma fonte de dados interna qual o papel que o usuário desempenha e gera o bilhete. Exemplificando, suponha que o usuário *eraldo* efetue a autenticação no servidor da PUC-Rio e seja mapeado para a entrada "cn=eraldo,cn=ticket,o=puc-rio,c=br", criada a partir da sua identificação de *login*. Ao consultar os atributos dessa entrada, a rotina de tratamento extrai o identificador *eraldo*, utiliza-o para descobrir o papel desempenhado, cria o bilhete e realiza o *log* das informações.

### 4.2.2 Autenticação no Servidor FTP

A especificação original do serviço FTP define apenas o uso de *login* e senha, sem nenhuma proteção, como forma de autenticação. O suporte ao novo método de autenticação para o servidor FTP seguiu as recomendações da RFC 2228 [11], que define meios para oferecer outros mecanismos de autenticação dos usuários, dando suporte à confidencialidade e integridade da comunicação.

A extensão do protocolo acrescentou comandos e redefiniu o fluxo no processo de identificação dos clientes, mas manteve a compatibilidade, classificando essas mudanças como opcionais. Os novos comandos utilizados foram AUTH e ADATA, juntamente com USER, da especificação original. A figura 4.10 ilustra a abordagem adotada para interação entre cliente e servidor.

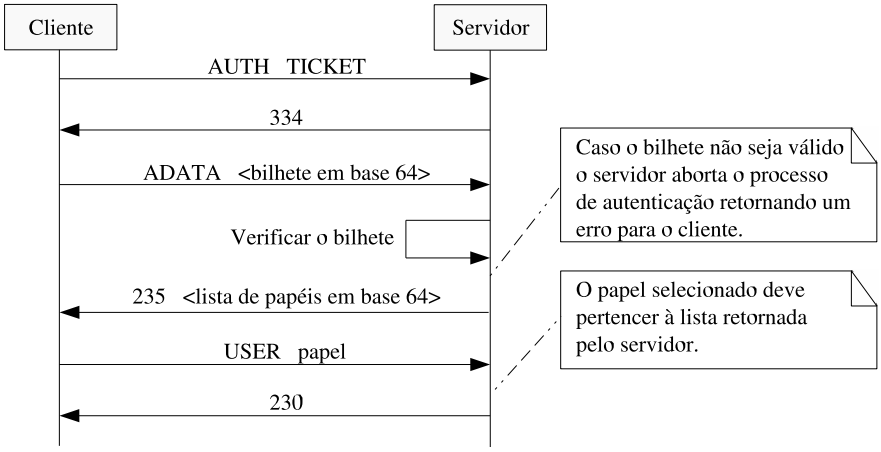


Figura 4.10: Autenticação no serviço FTP.

O cliente, usando o comando AUTH, inicia o processo indicando qual o método de autenticação que será utilizado (usamos "TICKET" para identificar o mecanismo via bilhete). Em seguida, o servidor responde com o código 334, sinalizando que a requisição foi aceita e que o bilhete deve ser transmitido. O cliente codifica o bilhete em base 64 e o envia para o servidor, através do comando ADATA. Na implementação, as chaves são lidas de arquivos, mas, como discutido na seção 4.1.2, poderia ser utilizada uma infra-estrutura de distribuição de chaves.

Tendo verificado o bilhete, o servidor cria uma lista com os papéis internos que foram relacionados com o papel extraído do bilhete e a envia de volta ao cliente. O cliente deve selecionar um elemento da lista e usar o comando USER para ativar o papel e acessar o serviço FTP.

O servidor FTP empregado neste caso de teste, o vsftpd [5], implementa algumas das novas especificações para suportar a autenticação via SSL/TLS. Estendemos suas funcionalidades para fazer uso do mecanismo de autenticação via bilhete, adicionando rotinas de verificação, criação da lista de papéis e alteração no fluxo de identificação dos usuários.

Desenvolvemos o cliente em Lua utilizando a biblioteca LuaSocket [22] que fornece funções de comunicação via rede e algumas facilidades já implementadas para acesso a serviços HTTP e FTP. Incluímos uma nova rotina de autenticação da biblioteca para que ela executasse todos os passos para acessar o servidor FTP usando a abordagem proposta. A rotina recebe o bilhete e uma função de *callback* que é executada tendo como parâmetro a lista de papéis recebida do servidor, e deve retornar um deles. Como mostra a figura 4.8, o programa cliente entra em contato com o servidor OpenLDAP a fim de obter o bilhete; apresenta as credenciais e realiza a busca, que retorna o bilhete. Então, inicia o processo de autenticação com o servidor vsftpd. Ao receber a lista de possíveis papéis, estes são apresentados ao usuário, que determina qual será usado.



### 4.2.3 Controle de Acesso

O controle é feito através das permissões no sistema de arquivos, da forma tradicional. Os papéis que o cliente recebe no ato da autenticação se referem às contas válidas na máquina que se está acessando, ou seja, os papéis são cadastrados no servidor como sendo usuários locais.

A ativação de um papel faz com que o servidor FTP atenda a todas as requisições do cliente com privilégios referentes ao usuário local, e assim, todos os direitos de acesso aos arquivos e diretórios dependem das permissões atribuídas pelo administrador a essa conta.

Como citado na seção 3.4, o modelo não possui um mecanismo de proteção contra o roubo do bilhete, que deve ser oferecido por outros meios. Neste estudo de caso para o serviço FTP, não tratamos esse problema. Uma solução seria estabelecer uma comunicação segura antes dos desafios serem lançados; o cliente, após selecionar o método de autenticação via bilhete, iniciaria um procedimento de troca de chaves com o servidor. Somente depois de estabelecida uma sessão segura é que o servidor enviaria o primeiro desafio.