Universidade Federal do Amazonas

Instituto de Ciências Exatas

Departamento de Ciência da Computação

Programa de Pós-Graduação em Informática

Flex-LDAP: Middleware LDAP baseado em palavra-chave

Péricles Silva de Oliveira

Manaus / Amazonas

2005

Péricles Silva de Oliveira

Flex-LDAP: Middleware LDAP baseado em palavra-chave

Dissertação apresentada ao Programa de Pós-Graduação em Informática do Departamento de Ciência da Computação da Universidade Federal do Amazonas, como requisito parcial para obtenção do Título de Mestre em Informática. Área de concentração: Redes.

Orientador: Prof. Dr. Edson Nascimento Silva Junior

Péricles Silva de Oliveira

Flex-LDAP: Middleware LDAP baseado em palavra-chave

Dissertação apresentada ao Programa de Pós-Graduação em Informática do Departamento de Ciência da Computação da Universidade Federal do Amazonas, como requisito parcial para obtenção do Título de Mestre em Informática. Área de concentração: Redes.

Banca Examinadora Prof. Dr. Edson Nascimento Silva Junior – Orientador

Departamento de Ciência da Computação – UFAM/PPGI

Prof. Dr. Edjair Mota

Departamento de Ciência da Computação – UFAM/PPGI

Prof. Dr. José Neuman

Departamento de Ciência da Computação – UFCE

Manaus – Amazonas

Abril de 2006

Dedicatória

Aos meus pais Pedro Oliveira e Maria Oliveira pela iniciativa moral e educacional desde os meus primeiros anos de ingresso escolar, que me incentivou a enfrentar e buscar incansavelmente novos conhecimentos.
A minha esposa Edilza Cordeiro pela compreensão e incentivo nos momentos difíceis e ausência do ambiente familiar com nossos filhos. Aos meus filhos Cristhian Oliveira e Luise Oliveira, pelo amor verdadeiro.

Agradecimentos

A Deus pela força invisível mais sempre presente em minha vida.

Ao meu orientador Prof. Dr. Edson Nascimento pelo acompanhamento constante e encorajador nesta caminhada e conquista de minha vida acadêmica.

Aos meus familiares pelo apoio.

Aos verdadeiros amigos.

Aos amigos: Ruan, Arelian, Márcio, Daniel Oliveira e todos que de forma direta ou indiretamente ajudaram-me.

Ao Prof. Dr-Ing. Edjair Mota que confiou em meu potencial, e não mediu esforços para externar o seu apoio incondicional, e que sempre esteve à disposição a qualquer hora para uma simples conversa até decisões importantes neste trabalho.

Ao Prof. Dr. Altigran Soares pelo apoio e orientação da escolha de minha dissertação.

A Universidade Federal do Amazonas pela oportunidade.

Agradeço

Resumo

Em implementações de sistemas constituídos por diretórios é comum a utilização do LDAP (*Lightweight Directory Access Protocol*). Nestes diretórios são armazenados objetos que obedecem à esquemas. As consultas efetuadas sobre os diretórios obedecem a uma especificação de protocolo, onde o cliente efetua consultas representadas através de parâmetros.
Na versão 3 do suíte LDAP não é possível a realização de consultas, sem o conhecimento prévio do esquema.
Nossa proposta é uma especificação de um sistema composto de um suíte LDAP que permita consultas em formato menos rígido, liberando o usuário final da necessidade de conhecer (ou referir-se) previamente o esquema da base para realizar consultas. Em nossa especificação utilizamos uma técnica da área de Recuperação de Informação, denominada de modelo vetorial, incorporada a um servidor LDAP.
Para avaliarmos nossa especificação, foi implementado um servidor que denominamos de Flex-LDAP. Nos experimentos utilizamos dois ambientes: um basea do no servidor LDAP tradicional, e outro utilizando nosso servidor Flex-LDAP.
Os resultados obtidos mostraram significativas vantagens do uso do Flex-LDAP sob o servidor LDAP tradicional. Tanto na questão da flexibilização das consultas, quanto no aspecto do desempenho do sistema.

Abstract

In directories implementation is common the use of LDAP (Lightweight Directory Access Protocol) protocol. In these directories are stored objects that obey the LDAP schema. The queries effected on the directories use protocol specification, where the client effects queries with filters using parameters. In LDAP version 3 not possible queries without the knowledge of the LDAP schema. Our proposal is a LDAP system, named Flex-LDAP, that to be possible keywords-based queries. That LDAP system use Information Retri eval Model (vectorial model) to queries processing.

To evaluate ours LDAP was implemented a server that running on two environments: the first with only LDAP server in version 3 and other with the Flex-LDAP between LDAP client and LDAP Server in version 3. The tests result shows better performance in throughput and flexible queries (only keywords in queries filter).

Sumário

Lista de Figuras

*

Capítulo 1

Introdução

Diretórios implementados baseados em modelos LDAP têm sido disponibilizados em ambientes de rede, na maioria dos casos voltados para consulta a informações de funcionário ou informações profissionais dos funcionários. A forma comum de acesso a essas informações tem sido através do protocolo LDAP, onde existem esquemas padronizados, criados por órgãos como o IETF [3]. Apesar dos esforços de padronização de esquemas para diretórios modelados com LDAP [12], as instituições têm a liberdade de desenvolver seus próprios esquemas.

Outra especificidade é a necessidade de conhecimento prévio do esquema para realização de consultas em diretórios acessados pelo protocolo LDAP. Isto tem motivado a criação de interfaces especializadas para consultas em diretórios; exemplos são os sítios de consultas pela Internet das Universidades de Michigan [24], Notre Dame [27], Havard [26], que criaram formulários complexos com diversos atributos para que o usuário consiga realizar consultas diversificadas. E existem, ainda , casos em que a página disponibiliza a descrição do esquema implementado em seus diretórios, possibilitando que as consultas sejam especificadas manualmente, através do uso associado de atributos, valores desejados e operadores lógicos. Este tipo de consulta não é amigável para usuários menos experientes.

Rumo contrário a de complexidade de interfaces têm seguido as soluções de máquina de busca, um simples campo de texto é usado pelo usuário para digitar livremente a sua consulta [1]. Deixando a tarefa de tratar a consulta para a máquina de busca, que utiliza modelos e técnicas de recuperação da informação (RI). Estas máquinas

de busca se antecipam aos fatos, realizando indexações de documentos na rede de dados, criando informações que ajudem na obtenção de uma melhor resposta para a consulta. O modelo vetorial [6] tem sido utilizado freqüentemente como base no processamento de consultas, onde os resultados obtidos devolvem um conjunto ordenado pelo grau de similaridade entre a consulta do usuário e os documentos HTML indexados pela máquina de busca.

1.1 Motivação e Objetivos

A primeira motivação que se define neste contexto é a necessidade de consultas mais amigáveis, onde um cliente possa realizar buscas de informações, sem a necessidade de especificar atributos e operadores lógicos. Como o protocolo LDAP na versão 3 segue a especificação da IETF, fica imposto a necessidade de conhecer previamente o esquema utilizado na representação dos objetos de uma árvore de diretórios (DIT). A padronização de esquemas é uma forma de normatizar a representação dos objetos em DITs, possibilitando que clientes oriundos de uma instituição realizem consultas em diretórios de outras instituições, utilizando-se de atributos já conhecidos, que estariam em um padrão de esquema publicado por órgão de competência, como o IETF. Um exemplo é o EduPerson [10], onde há um esforço para desenvolvimento de um esquema padronizado para utilização em Universidades. Porém, a possibilidade de construção de esquemas de diretórios pelas instituição abre a possibilidade de não padronização de esquemas.

Outra motivação vem da necessidade de construção de interfaces complexas (com diversos atributos) para que usuários efetuem suas consultas, onde são disponibilizados campos com diversos atributos e operadores lógicos, como pode ser visto em [7] [27] [24].

Encontramos, assim, a necessidade de uma solução mais amigável, onde um cliente LDAP pode realizar consultas utilizando filtros sem especificação dos atributos existentes na DIT.

O objetivo desta dissertação é propor um ambiente onde consultas LDAP possam ser especificadas de maneira mais amigável, sem a necessidade de conhe cimento prévio

do esquema da base de dados. Para tal, realizamos uma incorporação do modelo vetorial [1] em um servidor LDAP, disponibilizando um *middleware* [30] LDAP para o processamento de consultas livres, utilizando-se apenas palavras-chave, as quais passamos a denominar de consultas flexíveis. Com isto, será disponibilizada uma forma de construção de consultas em bases LDAP mais amigável e menos complexa, pois não haverá a necessidade de associações prévias de valores a atributos e conjunções de consultas através de operadores lógicos. Permitindo, ainda, que bases LDAP não necessitem ser refeitas, padronizadas.

Além disso, os sistemas que já utilizam o LDAP poderão incorporar o *middleware* proposto, possibilitando consultas flexíveis a sua base de dados.

1.2 Trabalhos Relacionados

A forma de consulta proposta neste trabalho é conhecida como consulta baseada em palavra-chave. A utilização de palavra-chave para composição de consultas é um recurso comum em sistemas de recuperação da informação [1]. Entretanto, as pesquisas atuais voltadas especificamente a LDAP têm concentrado esforços quanto a padronização de esquemas [10], [3] e testes de performance das distribuições existentes de servidores LDAP [25]. Analisando o LDAP como linguagem de consulta podemos citar trabalhos relacionados no contexto de banco de dados estruturados; entretanto, somente recentemente têm aparecido propostas com a utilização de palavra-chave [5] [28]. O sistema DataSpot descrito em [9] propõe o uso de *plain language* e navegações para explorar um *hyperbase*, onde são publicados conteúdos de um banco de dados na Web. Em [5] é introduzido um sistema que permite consultas a banco de dados utilizando palavra-chave. Seu trabalho, entretanto, foca sobre banco de dados relacional e não provê ordenação do conjunto de respostas retornadas do sistema. Diferentemente, o trabalho descrito em [28] propõe a extensão do XML-QL [29], para consultas em XML baseada em palavra-chave.

Nossa abordagem é distinta destas, pois estaremos integrando a um serviço LDAP a solução de consultas baseadas em palavra-chave. Com isto, os clientes LDAP existentes poderão realizar consultas utilizando palavra-chave, sem a necessidade de

alteração de código já implementados, toda a lógica de indexação e processamento de consultas estará integrada no serviço LDAP. Em situações onde já existem servidores LDAP operando, será necessária apenas a instalação de nossa proposta (Flex-LDAP - *middleware LDAP*), que receberá as consultas providas de um cliente LDAP.

1.3 Organização do Trabalho

No capítulo 1 descrevemos o contexto de nosso trabalho, assim como as motivações e objetivos desse trabalho. E na seção 1.2 realizamos uma revisão de trabalhos relacionados.

No capítulo 2 mostramos uns conceitos básicos que estão relacionados com o nosso trabalho, como a descrição do protocolo LDAP em detalhes. E conceituamos alguns modelos clássicos de recuperação de informação, com ênfase no modelo vetorial.

No capítulo 3 explanamos nossa proposta, denominada Flex-LDAP.

Detalhes do experimento realizado e resultados são descritos no Capítulo 4.

E finalmente, no Capítulo 5 descrevemos nossas conclusões e trabalhos futuros.

Capítulo 2

Conceitos Básicos

Neste capítulo descrevemos os conceitos necessários para o entendimento do protocolo LDAP e as técnicas e modelo de recuperação da informação utilizados neste trabalho.

2.1 *LDAP*: Lightweight Directory Access Protocol

A nossa proposta, denominada Flex-LDAP, foi baseada na versão 3 do LDAP, que obedece as especificações descritas nas RFCs 2251 (*Protocol*), 2252 (*Mandatory Schema*), 2253 (*Distiguished Names*), 2254 (*LDAP URLs*), 2255 (*Search Filters*), 2256 (*User Schema*), 2829 (*Authentication Methods*), 2830 (*Transport Layer Security and Digest Authentication*), abrangendo o protocolo de comunicação, criação de esquemas, sintaxe, estrutura de diretórios. Sendo o ponto de partida para leitura destas, a RFC2251 [11]. Este conjunto pode ser visto na Figura 2.1.

O LDAP pode ser melhor entendido considerando-se os quatro modelos nos quais se baseia:**Informação**: Descreve a estrutura da informação armazenada em um diretório LDAP;**Nomeação**: Descreve como as informações estão organizadas e identificadas em um diretório LDAP;**Funcional:** Descreve as operações que podem ser realizadas nas informações armazenadas em um diretório LDAP;**Segurança:** Descreve como a informação em um diretório LDAP pode ser protegida de acessos não autorizados.

Figura 2.1: Conjunto de RFCs do LDAP v3.

2.1.1 O Modelo de Informação

A unidade básica de informação armazenada no diretório é a **entrada**, que representa um objeto de interesse do mundo real. Cada entrada contém um ou mais **atributos** que descrevem a entrada. Cada atributo tem um tipo e um ou mais **valores**. O tipo de um atributo, assim como sua descrição é definida por um **esquema**, que dita sua sintaxe. Além de definir que dados podem ser armazenados como o valor de um atributo, a sintaxe do atributo define também como esses valores se comportam durante pesquisas e outras operações no diretório.

Este modelo define as classes de objetos que um servidor de diretório pode armazenar e os atributos. Os atributos são descritos por um esquema. Um esquema define onde as classes de objetos são permitidas no diretório, que atributos elas podem conter, que atributos são opcionais e a sintaxe de cada atributo. A sua verificação garante que todos os atributos exigidos para uma entrada estarão presentes antes dela ser armazenada. O esquema define também a hierarquia e as subclasses dos objetos em que os objetos da estrutura (hierarquia) DIT (*Directory Information*

Tree) podem aparecer.

2.1.2 O Modelo de Nomeação

O modelo de nomeação LDAP define como as entradas são identificadas e organizadas. Estas se apresentam em uma estrutura chamada de árvore de informações de diretório (DIT). As entradas dentro da DIT são organizadas com base em seu nome distinto (DN), um nome único que identifica exclusivamente uma entrada. Os DNs são formados por uma seqüência de nomes distintos relativos (RDN - Relative Distinct Names). Um DN é composto por um seqüência de RDNs separados por vírgulas. Cada RDN em um DN corresponde a um ramo da DIT, da raiz até a entrada do diretório.

Cada seqüência RDN é derivada de atributos da entrada de diretório. No caso mais simples e comum, uma RDN tem a forma de $< nome - atributo >=< valor >$. Um exemplo de uma DIT é mostrado na Figura 2.2. Cada ícone de pasta representa uma entrada no diretório. A entrada dc=ufam,dc=edu,dc=br é denominada raiz da DIT. Os atributos e seus valores são listados dentro de cada entrada.

O Modelo Funcional

O LDAP define operações para acessar e modificar entradas de diretório. As operações do LDAP podem ser divididas em três categorias:

- **Consulta**: inclui as operações de busca usadas para obter informações de um diretório;

- **Atualização**: inclui as operações de adicionar, remover e modificar RDN usadas para atualizar informações em um diretório;

- **Autenticação**: Inclui as operações de ligar, desligar e abandonar conexão, usadas para conectar e desconectar um servidor LDAP. Pode-se estabelecer direitos de acesso e proteger dados.

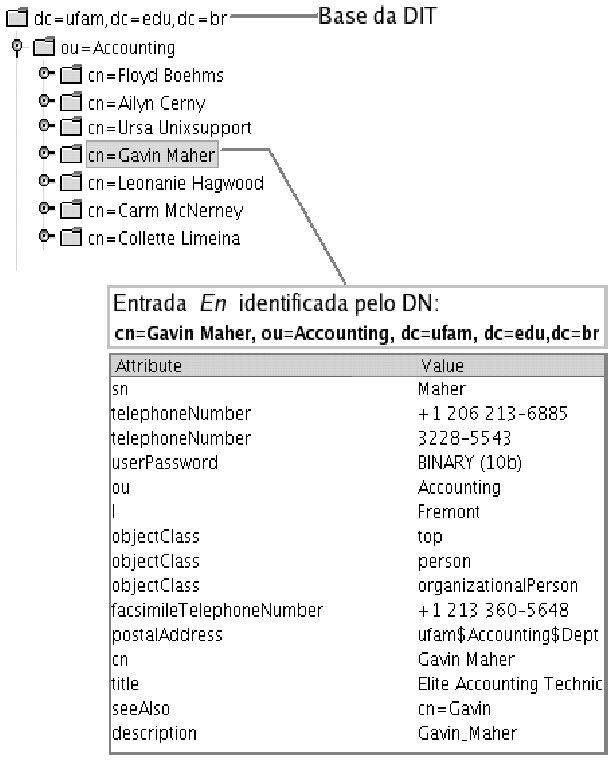

Figura 2.2: Exemplo de uma DIT, com detalhes de uma entrada.

O Modelo Segurança

Este modelo especifica os itens de autenticação, integridade, confidencialidade e autorização referente as transações de informações entre cliente e servidor LDAP. Estes aspectos de segurança são: **Autenticação**: Garantia de que um cliente realmente é quem está se autenticando no servidor LDAP. No LDAP isto é feito através de mensagens *LDAP Bind*, que envia parâmetros de usuário e senha;

Integridade: Garantia que a informação que chega é realmente a mesma que foi enviada.

Confidencialidade: No LDAP existe métodos de criptografia que podem ser utilizados na comunicação entre cliente e servidor.

Autorização: Na versão 3 do LDAP, a autorização não faz parte da especificação padrão do protocolo e, portanto, é uma implementação específica do fornecedor.

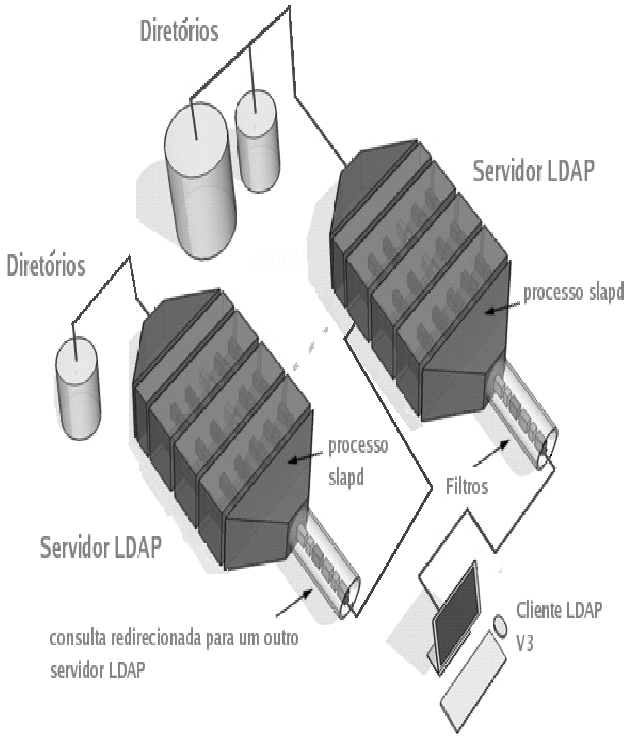

Figura 2.3: Estrutura cliente-servidor LDAP v3.

2.1.3 Detalhamento do Protocolo LDAP

A arquitetura utilizada pelo LDAP é cliente-servidor. Onde um cliente transmite uma requisição, para ser realizada por um ou mais servidores. O servidor (principal) é então responsável por realizar as operações necessárias no diretório. Após o término de todas as operações, o servidor retornar com uma resposta contendo os resultados ou mensagem de erro para o cliente que requisitou.

Na Figura 2.3 existe uma estrutura macro utilizada na comunicação entre cliente e servidor LDAP, assim como seus redirecionamentos para outros servidores LDAP. Um cliente LDAP envia uma consulta para um servidor LDAP, utilizando filtros. O processo slapd realiza o processamento desta consulta, e busca em sua DIT as informações que a atendam. Podendo redirecionar a pesquisa para outro servidor LDAP, que buscará em sua DIT as informações referentes ao filtro passado.

Elementos do Protocolo

O protocolo LDAP é descrito usando *Abstract Syntax Notation 1* [21], e é tipicamente transferido usando um subgrupo de ASN.1, denominado *Basic Enconding Rules* [31]. A seguir serão descritos os tipos de mensagens que o protocolo LDAP utiliza em sua comunicação cliente/servidor.

Elementos comuns

Esta seção descreve o formato de uma PDU LDAPMessage (*Protocol Data Unit*), assim como os tipos de dados que são utilizados nas operações do protocolo.

PDU LDAPMessage

Nas operações entre cliente e servidor LDAP, as mensagens trocadas são encaminhadas em um envelope padrão, o **LDAPMessage**, que é definido conforme gramática do Envelope de Mensagem, que pode ser consultada no Apêndice IV.

A função do **LDAPMessage** é prover um envelope contendo campos comuns requeridos em toda a comunicação do protocolo. Se o servidor recebe uma PDU do cliente em que a TAG de seqüência do LDAPMessage não pode ser recuperada, o *messageID* não poderá ser validado, a *tag protocolOp* não é recuperada como uma requisição, ou a estrutura codificada ou tamanho dos campos de dados são encontradas incorretamente, o servidor deverá retornar uma mensagem de desconexão, com *resultCode protocolError*, e encerra imediatamente a conexão. Em outros casos que o servidor não pode validar a requisição do cliente, o servidor deverá retornar uma resposta apropriada à requisição, com o *resultCode* configurado para *protocolError*. Se o cliente receber um PDU do servidor como não validado, o cliente deverá descartar o PDU, ou poderá encerrar abruptamente a conexão.

Message ID

Todos os envelopes de resposta *LDAPMessage* contém o valor *messageID* da requisição correspondente *LDAPMessage*.

O *messageID* de uma requisição deverá ter um valor diferente dos valores de qualquer outra requisição proeminente em uma sessão LDAP da qual esta mensagem faz parte.

Um cliente não deverá enviar uma segunda requisição com o mesmo *messageID*.

Tipicamente, o cliente incrementa um contador para cada requisição. Um cliente não pode reusar um *messageID* de uma requisição abortada ou de uma operação abandonada até ter recebido uma resposta do servidor.

Funcionamento do Protocolo LDAP

Uma operação do protocolo LDAP pode ser resumida, em:

1. O cliente envia uma mensagem do tipo *BindRequest* para o servidor LDAP. Esta função é responsável por realizar a autenticação entre cliente e servidor LDAP. O *BindRequest* é definido conforme gramática no Apêndice C;

2. Em resposta ao *BindRequest* do cliente, o servidor envia uma messagem *BindResponse*, que indica o *status* da requisição de autenticação. O *BindResponse* é definido conforme gramática descrita no Apêndice D. Em caso de sucesso da autenticação, o servidor retorna uma mensagem *LDAPResult*, preenchendo os campos *resultCode*, *Matched DN* e *Error Message*, com os respectivos valores: *Success (0x00), null e null*. O campo *resultCode* pode indicar diversos *status* da requisição de autenticação do cliente, conforme previsto na gramática descrita no Apêndice E. Em casos que o servidor não suporta o protocolo requisitado pelo cliente, o campo *resultCode* conterá o código que representa *protocolError*. Neste caso, em que o cliente recebe um *BindResponse* onde o valor do campo *resultCode* é *protocolError*, ele deverá fechar a conexão com o servidor;

3. Após a confirmação de autenticação do cliente com o servidor, fica estabelecida uma sessão para que os mesmos troquem mensagens entre si. Usaremos o exemplo constante na Figura 2.4, para detalhamento das mensagens trocadas pelo protocolo;

4. A operação de consulta (*Search Operation*) permite que um cliente realize requisições de consulta para um ou mais servidores LDAP, permitindo a leitura de atributos de uma simples entrada no diretório LDAP, várias entradas a

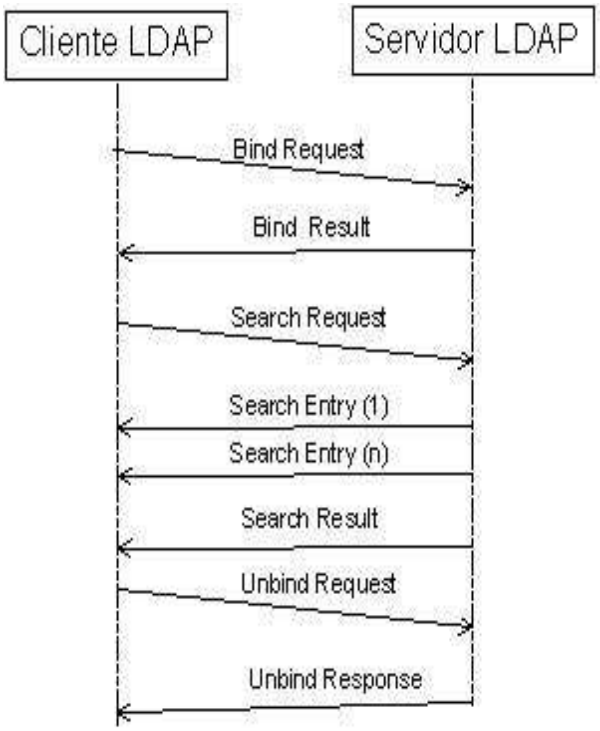

Figura 2.4: Transação de mensagens LDAP.

partir de uma base do diretório ou uma sub-árvore inteira de entradas. Para que um cliente realize uma consulta, é necessário o envio de uma mensagem *SearchRequest* para o servidor. A gramática de uma *SearchRequest* é definida conforme descrita no Apêndice F;

5. Os resultados de uma consulta são enviadas pelo servidor para o cliente em mensagens LDAP, podendo ser *SearchResultEntry, SearchResultReference, ExtendedResponse ou SearchResultDone*. Estas mensagens obedecem às gramáticas descritas nos Apêndices G, H e I;

6. Como a sessão LDAP opera sobre uma sessão orientada a conexão, o servidor retornará para o cliente uma seqüência de respostas separadas em mensagens LDAP. Podendo ser zero ou mais respostas contendo *SearchResultEntry*, uma para cada entrada encontrada na consulta. Após a seqüência de mensagens *SearchResultEntry* seguirá uma resposta contendo um *SearchResultDone*, que contém uma indicação de sucesso, ou detalhando erros que vierem acontecer. Cada entrada retornada em um *SearchResultEntry* conterá todos os atributos

com seus respectivos valores associados, se for especificado para serem retornados, conforme campo indicador (*typesOnly*);

7. Finalmente, após todas as transações efetuadas entre cliente e servidor, o cliente pode sinalizar que deseja terminar a sessão LDAP. Para isto, o cliente envia uma mensagem *UnbindRequest*, que obedece a gramática descrita no Apêndice J. Com esta mensagem enviada para o servidor, o mesmo termina a sessão LDAP.

   No geral, são estas mensagens trocadas pelo protocolo LDAP em suas operações, o que difere são as quantidades de vezes das mensagens trocadas, em casos que exigem o envelopamento de várias entradas encontradas na DIT.

2.2 Recuperação de Informação

A área de Recuperação de Informação ou *Information Retrieval* (RI ou IR) lida com representação, armazenamento, organização e acesso a itens de informação (documentos).

A representação e a organização da informação devem dar ao cliente de um Sistema de Recuperação de Informação (SRI) um acesso fácil à informação de seu interesse. O usuário deve traduzir o que deseja numa forma de consulta (*query*), que possa ser processada por um SRI.

Na sua forma mais comum, esta consulta é escrita utilizando palavras-chave (ou termos de indexação) que resumem a descrição da necessidade de informação do usuário. Dada uma consulta, o principal objetivo do SRI é retornar informações úteis (relevantes) ao usuário. A ênfase é na recuperação de informação e não na recuperação de dados.

Entende-se por Recuperação de dados determinar que documentos de uma coleção contêm as palavras-chave que aparecem na consulta de um usuário. Isto não é o suficiente para suas necessidades de informação, na maioria das vezes. O usuário de um SRI prefere que informações sejam recuperadas sobre determinado assunto, que contenha os dados que aparecem na consulta. O SRI deve, de alguma forma, *interpretar* o conteúdo das informações encontradas nos documentos de uma coleção

e ordená-los de acordo com um grau de relevância para o usuário.

Relevância é a palavra central de um SRI. É objetivo de um SRI recuperar todos os documentos que são relevantes a uma consulta de um usuário e o menor número possível de documentos não relevantes.

Dentre as técnicas de consulta que utilizam indexação, podemos destacar a de arquivos invertidos ou lista invertida. Esta técnica possui um custo de construção e atualização, e é construída antes de qualquer processamento de consulta, pois os modelos dependem diretamente desta lista[1].

Uma lista invertida é um mecanismo orientado a palavra usado para indexar uma coleção de documentos. A estrutura da lista invertida é composta por dois elementos: O vocabulário e as ocorrências. O vocabulário é o conjunto de todas as palavras diferentes encontradas nos documentos. Para cada palavra é criada uma lista contendo as ocorrências dessas palavras em posições dos documentos, palavras ou simplesmente a identificação do documento onde a palavra ocorreu.

2.2.1 Modelos clássicos de RI

Os modelos clássicos de recuperação de informação apresentam estratégias de busca de documentos relevantes para uma consulta. Tanto a consulta feita pelo usuário, quanto os documentos que compõem a coleção a ser pesquisada, são representados pelos seus termos. Os modelos são: booleano, vetorial e probabilístico.

Modelo Booleano

O modelo booleano considera uma consulta como uma expressão booleana convencional, que liga seus termos através de conectivos lógicos **AND, OR e NOT**. No modelo booleano um documento é considerado relevante ou não relevante a uma consulta, não existe resultado parcial e não há informação que permita a ordenação do resultado da consulta.

Este modelo é muito mais utilizado para recuperação de dados do que para recuperação de informação.

[1] Como diretórios LDAPs têm como características as poucas atualizações, o tempo de indexação acaba sendo absorvido, na avaliação geral do desempenho

O protocolo LDAP é baseado neste modelo, conforme visto na RFC 2254**??**. O processamento de consulta no LDAP utiliza diversos operadores lógicos para formação de filtros de consulta, e retorna para o usuário com informações conforme encontradas na ordem de busca na DIT. Para usuários experientes que conhecem a utilização de operadores lógicos e sabem exatamente o que desejam consultar, este modelo é facilmente programável e exato.
As principais dificuldades deste modelo residem no fato que pessoas lidam com conhecimento parcial, o que dificulta a formação de consultas. E o resultado da consulta pode ser nula, ou haver *overload* (problema que dá um falso senso de segurança).

Modelo probabilístico

O modelo probabilístico possui esta denominação por trabalhar com conceitos provenientes da área de probabilidade e estatística. Nele os termos indexados dos documentos e das consultas não possuem pesos pré-definidos. A ordenação dos documentos é calculada pesando dinamicamente os termos da consulta relativamente aos documentos. Esta baseado no princípio da ordenação probabilística (*Probability Ranking Principle*). Neste modelo, busca-se saber a probabilidade de um documento D ser ou não relevante para uma consulta Q_a. Tal informação pode ser obtida assumindo-se que a distribuição de termos na coleção seja capaz de informar a relevância provável para um documento qualquer da cole cão.
Um dos aspectos importantes do raciocínio probabilístico é a complexidade dos métodos de inferência. A manipulação de modelos probabilísticos geralmente leva grande consumo de memória. Mesmo modelos que visam reduzir o esforço computacional de inferência, como redes Bayesianas, ainda possuem elevada complexidade.

Modelo vetorial

O modelo espaço-vetorial [6](ou simplesmente vetorial) foi desenvolvido por Gerard Salton para ser utilizado no sistema de recuperação da informação denominado SMART [6]. No modelo vetorial cada documento é representado por um vetor de termos, e cada termo possuem um valor associado que indica o grau de

importância (peso - *weight*) deste no documento. Cada documento possui um vetor associado que é constituído por pares de elementos na forma ($palavra_1$, $peso_1$), ($palavra_2, peso_2$), ..., ($palavra_n, peso_n$). Estes documentos podem ser organizados, por exemplo, em uma lista invertida.

O peso de um termo em um documento pode ser calculado de diversas formas. Os pesos são usados para computar a similaridade entre cada documento armazenado e uma consulta feita por um cliente. Estes métodos de cálculo de peso geralmente se baseiam no número de ocorrências do termo no documento (freqüência).

Uma das formas de se calcular o peso, dada por Salton e Buckey, tenta balancear características em comum nos documentos (*intra-document*) e características para fazer a distinção entre os documentos (*inter-document*).

Definições:

A freqüência de um termo k, em um documento S que possui n termos, é denominada **TF**, nos cálculos de similaridade a consulta de um usuário, também é considerada um documento, sendo que a freqüência do termo é a quantidade de vezes que ele aparece na consulta. Este termo **TF** pode ser definido como:

$$freq_{(k,S)} \longrightarrow \sum_{Ocorr(k,S)^n} \longrightarrow TF. \tag{2.1}$$

A importância de um termo k para uma coleção $D = \{S_1, \ldots, S_k\}$, onde $k \geq 1$, é denominada **IDF**, podendo ser definida como:

$$log(\frac{N}{n_k}) \longrightarrow IDF \tag{2.2}$$

Onde **N** é o número de termos na coleção D e n_k número de vezes que um termo k ocorreu na coleção D.

TF reflete características intra-documentos e **IDF** dá uma medida de distinções inter-documento. Pesos baseados no produto $TF \times IDF$, são chamados de abordagem **tf-idf**. Cada elemento do vetor de termos é considerado uma coordenada dimensional. Assim, os documentos podem ser colocados em um espaço euclidiano de **n** dimensões (onde **n** é o número de termos) e a posição do documento em cada dimensão são dadas pelo seu peso.

As distâncias entre um documento e outro indicam seu grau de similaridade, ou seja, documentos que possuem os mesmos termos acabam sendo colocados em uma mesma região do espaço e, em teoria, tratam de assuntos similares.

A consulta do usuário também é representada por um vetor. Desta forma, os vetores dos documentos podem ser comparados com o vetor da consulta e o grau de similaridade entre cada um deles pode ser identificado.

Os documentos mais similares (mais próximos no espaço) à consulta são considerados relevantes para o usuário e retornados como resposta para ela.

Uma das formas de calcular a proximidade entre os vetores é testar o ângulo entre estes vetores. No modelo original, é utilizada uma função denominada de *cosine vector similarity* que calcula o produto dos vetores de documentos através da fórmula:

$$similaridade(\vec{Q}, \vec{D}) = \frac{\sum_{k=1}^{n} w_{qk} \times w_{dk}}{\sum_{k=1}^{n} (w_{qk})^2 \times \sum_{k=1}^{n} (w_{dk})^2} \tag{2.3}$$

Onde,

- $\vec{Q}$ representa o vetor de termos da consulta;
- $\vec{D}$ é o vetor de termos do documento;
- w_{qk} são os pesos dos termos da consulta; e
- w_{dk} são os pesos dos termos do documento.

Depois dos graus de similaridade terem sido calculados, é possível montar uma lista ordenada (*ranking*) de todos os documentos e seus respectivos graus de relevância à consulta, da maior para a menor relevância.

Vantagens do modelo vetorial: Como estratégia de encontro parcial (função de similaridade), onde o usuário não tem idéia como formular sua consulta, e procura baseá-la em termos de interesse. Estes termo s são palavras-chave que o usuário utiliza para expressar sua consulta. Por exemplo: Informática Manaus.

O modelo vetorial tem características que podem ser aplicadas no processamento de consulta, pois existe a atribuição de pesos aos termos, que melhora o desempenho e devolve um conjunto de respostas ordenados de acordo com o grau de similaridade entre a consulta e os documentos relevantes encontrados na coleção de pesquisa.
As dificuldades encontradas neste modelo estão na ausência de ortogonalidade entre os termos, isto poderia encontrar relações entre termos que aparentemente não têm nada em comum, também por ser um modelo generalizado, e ainda pode acontecer de um documento relevante não conter termos da consulta.

Capítulo 3

Proposta *Flex-LDAP*

A necessidade de consultas mais amigáveis, onde um cliente possa realizar buscas de informações utilizando apenas valores, sem a necessidade de especificar atributos e operadores lógicos, é a principal motivação deste trabalho. Para isto, dois caminhos podem ser adotados. O primeiro envolve a realização de adaptações diretamente na versão disponível do suíte LDAP, principalmente no servidor de processamento de consultas. O segundo caminho é a criação de um *middleware* que incorpora as funcionalidades necessárias para processar consultas não estruturadas e transformá-las em consultas que possam ser processadas por um servidor LDAP padrão.

Nós optamos em desenvolver e incorporar ao processo de consultas em diretórios um *middleware*, que chamaremos de Flex-LDAP. Esta proposta nos traz as vantagens de independência do servidor LDAP, e permite, ainda, que sistemas de diretórios implantados possam usufruir do *middleware* sem grandes impactos.

O Flex-LDAP deve ser capaz de entender e responder tanto consultas não estruturadas quanto consultas oriundas de clientes LDAP padrão. Assim, as funcionalidades de um servidor LDAP devem estar inseridas no Flex-LDAP, respeitando as especificações da versão 3o protocolo LDAP.

O ganho de desempenho no processamento de consultas que observamos e

apresentamos neste trabalho vem, em parte, da foram como implementamos tais funcionalidades. Pois usamos técnicas de Recuperação de Informação baseadas no modelo vetorial.

A escolha do modelo vetorial foi baseada na experiência adquirida em trabalhos de RI expostas em [1]. Comumente, este modelo é utilizado em máquinas de buscas, com suas respectivas extensões e/ou apoio de outros modelos.

A arquitetura do Flex-LDAP é cliente-servidor. No entanto, as mensagens oriundas de clientes LDAP passam a não ser mais enviadas diretamente para um servidor LDAP, e sim para o Flex-LDAP.

Após o devido processamento da consulta, o Flex-LDAP interage com o servidor LDAP que tem a base de informações (DIT), e com as respostas adquiridas do servidor LDAP, o Flex-LDAP aplica o modelo vet orial em cima do resultado oriundo do servidor LDAP, buscando classificar os termos da consulta realizada pelo cliente. Retornando ao cliente uma relação de respostas ordenadas pelo grau de similaridade entre consulta e entradas existentes na DIT.
Sob o ponto de vista do cliente LDAP, nada se altera, pois o padrão do protocolo permanece intacto. Assim, a mesma sintaxe de consulta utilizada para servidores LDAPs, pode ser utilizada para consultar no Flex-LDAP, com a vantagem do cliente não mais se preocupar em conhecer os atributos da base (DIT) onde irá realizar a consulta. Direcionando seu foco de esforço apenas para os valores que deseja consultar.

Uma estrutura macro de nossa proposta pode ser vista na Figura 3.1. Nesta figura, observamos que o cliente passa a interagir com o Flex-LDAP, e não mais com o servidor LDAP. Para que esta interação ocorra de modo transparente para o cliente, há apenas a necesidade de ajustar a configuração do Flex-LDAP e do servidor LDAP no quesito portas de comunicação do socket.

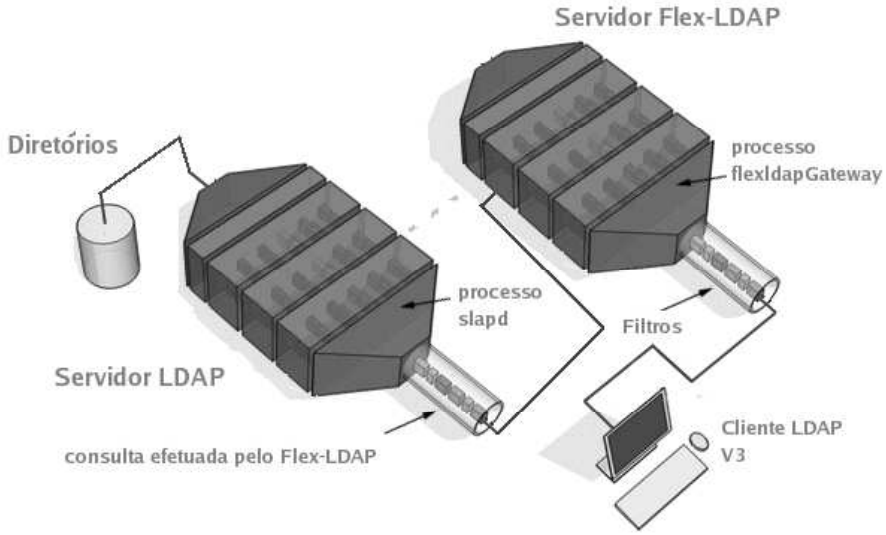

Figura 3.1: Estrutura Macro do Flex-LDAP.

3.1 Definições

Considerando um diretório D implementado conforme as especificações LDAP. Definimos este diretório como uma coleção de entradas, conforme descrito abaixo:

$$D = \{e_1, e_2, \ldots, e_n\}, n \geq 1$$

Cada entrada e_n é um conjunto de pares atributo-valores, que definimos como:

$$e_n = \{\langle A_1, v_{11}\rangle, \langle A_1, v_{k1}\rangle, \ldots, \langle A_l, v_{1l}\rangle, \langle A_l, v_{kl}\rangle\}, k \geqslant 1 \text{ e } l \geqslant 1.$$

Onde k representa a seqüência de valores v_{kl} de um atributo A_l.
Nós consideramos um esquema LDAP S_d como o conjunto de todos atributos que compõe qualquer entrada armazenada no diretório D. Definido como:

$$S_d = \{A_l | A_l \text{ é um atributo de alguma entrada } e_n \in D\}$$

Nós consideramos uma consulta Q como um conjunto de palavras-chave (ou termos), conforme:

$$Q = \{t_1, t_2, \ldots, t_k\}, k \geq 1$$

Esta definição de um diretório nos permite ignorar os detalhes de como sua estrutura é representada, e será utilizada na especificação de nossa proposta.

3.2 Especificação do Flex-LDAP

Para atender os objetivos descritos no início do trabalho, nossa propost a é composta de dois módulos. Estes módulos formam o *middleware* denominado Flex-LDAP, cuja implementação deverá participar de um sistema LDAP existente. Estes componentes de um sistema integrado com Flex-LDAP pode ser visto na Figura 3.2. Nesta figura o cliente passa a comunicar-se com o Flex-LDAP. Este, por sua vez, deve realizar o processamento da consulta do cliente (módulo de processamento de consulta), ajustá-la para os padrões que o servidor compreende. Enviar a consulta ajustada ao servidor. Receber do servidor as respostas da consulta. Processar esta resposta para gerar uma saída ordenada por um grau de similaridade.

Antes de qualquer processamento de consulta faz-se necessário a utilização do módulo de indexação, que deve ser executado previamente, pois cria novas entradas na DIT que são necessárias para o processamento de consultas. Esta s novas entradas geram um aumento de carga para a DIT, entretanto são essênciais para o modelo vetorial e também responsável pelo ganho de eficiência. O segundo intermedeia a comunicação entre um cliente LDAP e um Servidor LDAP, que é responsável por receber as consultas de clientes LDAP, e em seguida realiza o processo de extração de palavra-chave para obtenção das melhores respostas utilizando o modelo vetorial, que serão enviadas ao cliente solicitante.

Figura 3.2: Componentes do Flex-LDAP.

3.2.1 Módulo de Indexação do Flex-LDAP

Assumimos a existência de um diretório D (DIT) contendo um conjunto de entradas[1] E_n. Sendo que cada entrada E_n contém:

$$E_n = \{\langle A_1, v_{11}\rangle, \ldots, \langle A_1, v_{1k}\rangle, \ldots, \langle A_l, v_{l1}\rangle, \ldots, \langle A_l, v_{lk}\rangle\}, k \geq 1 \text{ e } l \geq 1$$

Este conjunto E_n representa os atributos com seus respectivos valores, sendo que um mesmo atributo pode aparecer mais de uma vez dentro de uma entrada, mas com valores diferenciados. Logo, dentro de uma entrada pode aparecer diversos atributos e seus valores, como por exemplo na Figura 3.3, onde o atributo *telephoneNumber* aparece duas vezes com os valores "$+1\ 206\ 213 - 6885$"e "$3228 - 5543$".

Cada entrada E_n é identificada de forma única dentro do diretório D por uma chave denominada DN (*Distinguished Name*), que é formada pelo caminho na DIT desde a folha até a raiz, um exemplo disto pode ser visto na Figura 3.3, onde um DN contém o valor *cn=Gavin Maher, ou=Accounting, dc=ufam,dc=edu,dc=br*. A extração de entradas na DIT formarão um conjunto $DN_s = \{\langle pk_1, dn_1\rangle, \ldots, \langle pk_s, dn_k\rangle\}, k \geq 1$ e $s \geq 1$, sendo que pk_s representa uma chave primária gerada pelo módulo indexador. Em cada entrada E_n, o módulo indexador realizará a extração dos valores v_{lk} de cada atributo A_l, que serão gerados *tokens* a partir deles, formando um conjunto para cada termo K_n extraído, sendo:

[1] objetos contidos na DIT

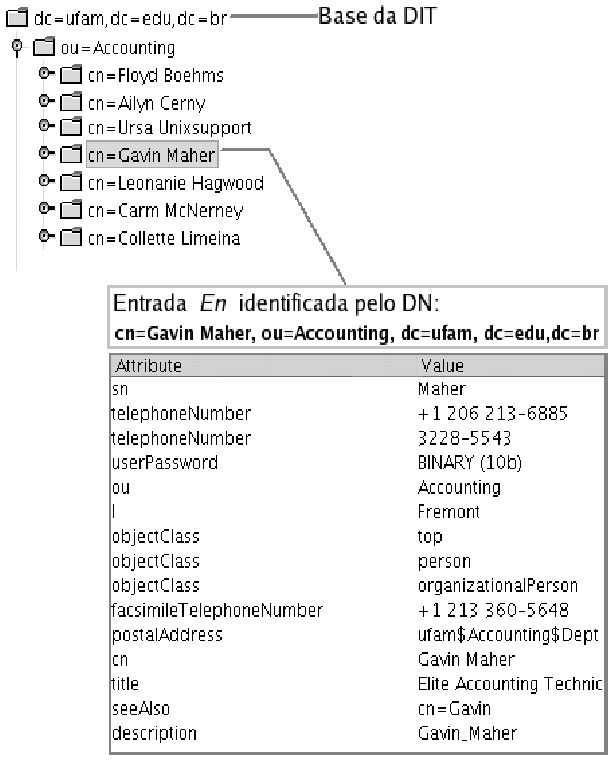

Figura 3.3: Exemplo de uma DIT, com detalhes de uma entrada.

$$K_n = \{\ldots, \langle pk_s, f_{K_n \in pk_s} \rangle, \ldots\}$$

Este conjunto K_n é formado pelo código que identifica um *DN*, onde o termo K_n apareceu, e sua respectiva freqüência $f_{K_n \in DN}$. Como exemplo, podemos compor um conjunto baseado na Figura 3.3 para o termo *Accounting*, que conterá a seguinte formação:

Sendo:

$$DN_s = \{\ldots, \langle pk_1(\mathbf{001}), "cn = GavinMaher, ou = Accounting, dc = ufam, dc = edu, dc = br" \rangle, \ldots\}$$

Temos:

$$K_n(Accounting) = \{\ldots, \langle \boldsymbol{pk(001)}, f_{K_n \in pk(001)}(\mathbf{3}) \rangle, \ldots\}$$

Inferindo que o termo *Accounting* apareceu três vezes na entrada *cn=Gavin Maher, ou=Accounting, dc=ufam, dc=edu, dc=br*.

Ao final da indexação teremos um conjunto V, denominado vocabulário, contendo $V = \{K_1, \ldots, K_n\}$, que corresponde ao conjunto de todos os termos distintos

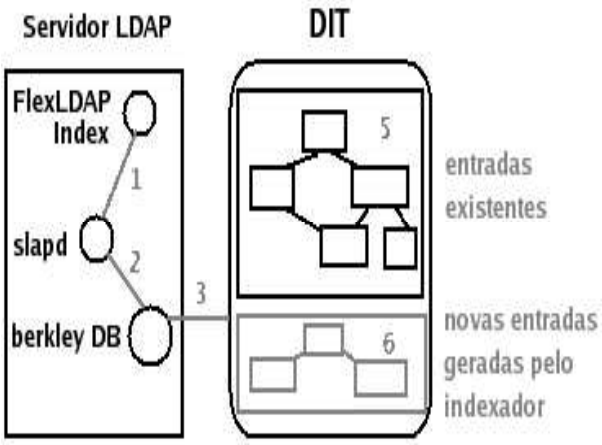

Figura 3.4: Estrutura macro do indexador.

encontrados na DIT.

Detalhe de Implementação do Módulo de Indexação

O módulo de indexação é o principal responsável pelo funcionamento adequado do módulo de processamento de consultas do Flex-LDAP. Sem a indexação da DIT, o processamento de consulta não funciona; pois, existem informações (entradas) que são geradas na DIT que servem para o processamento do modelo vetorial e recuperação de informações.

O módulo de indexação obedece o seguinte procedimento:

Faz-se necessário que o módulo indexador conecte-se ao serviço LDAP existente, para isto foi criado um arquivo de configuração denominado *flexldap.prop* que contém informações importantes para extração de entradas e operação de todo o Flex-LDAP. Tais informações são referentes à: endereço IP ou *hostname* do servidor LDAP a ser indexado, porta TCP, conta de usuário administrador e sua respectiva senha e a *baseDN* a partir da qual o módulo iniciará sua indexação;

O módulo ao autenticar-se ao servidor LDAP inicia a procura de todas as entradas existentes na DIT. De cada entrada serão extraídos os valores de seus atributos.

Estes valores irão compor o vocabulário da lista invertida; lembrando que o vocabulário é formado por termos distintos. A cada termo distinto encontrado será computado em qual entrada ocorreu este termo e a quantidade de vezes que ocorreu em uma determinada entrada;

A lista invertida que será gerada no módulo indexador é armazenada na própria DIT do servidor indexado. Optamos pela integração e padronização em LDAP, para que não fosse necessária a utilização de bases de dados externas. Assim, nõ deve haver maiores interferências externas no desempenho de nossa proposta. Para tal, foi criado um esquema LDAP que representa uma lista invertida na DIT, descrito no Apêndice K. Este esquema obedece as especificações de sintaxe do LDAP versão 3. Caso contrário, não seria possível armazenar informações em uma DIT LDAP;

Após a indexação da DIT, uma sub-árvore estará contida internamente nesta DIT, onde estarão valores necessários para o módulo de processamento de consulta. A estrutura desta sub-árvore pode ser vista na Figura 3.5. Nesta pode se observar uma base DIT com dc=ufam,dc=edu,dc=br, que podemo s considerar como raiz e a partir desta temos as entradas já existentes e as novas entradas geradas pelo módulo de indexação do Flex-LDAP, composta por ou=FlexLDAP com suas respectivas herdeiras ou=documents e ou=listinv. Nestas sub-árvores se pode encontrar todas as entradas indexadas necessárias para o modelo vetorial. Detalhes do algoritmo pode ser visto no Apêndice A.

3.2.2 Módulo de Processamento de Consultas do Flex-LDAP

A consulta efetuada por um cliente LDAP obedece a uma linguagem de consulta, que no mínimo é necessária a especificação do par atributo-valor, sendo que entre este par é necessária a atribuição de um operador lógico (or, not, and e outras extensões).

Definimos esta consulta Q_o como $\{\ldots, \langle A, O, V\rangle, \ldots\}$. Onde A representa

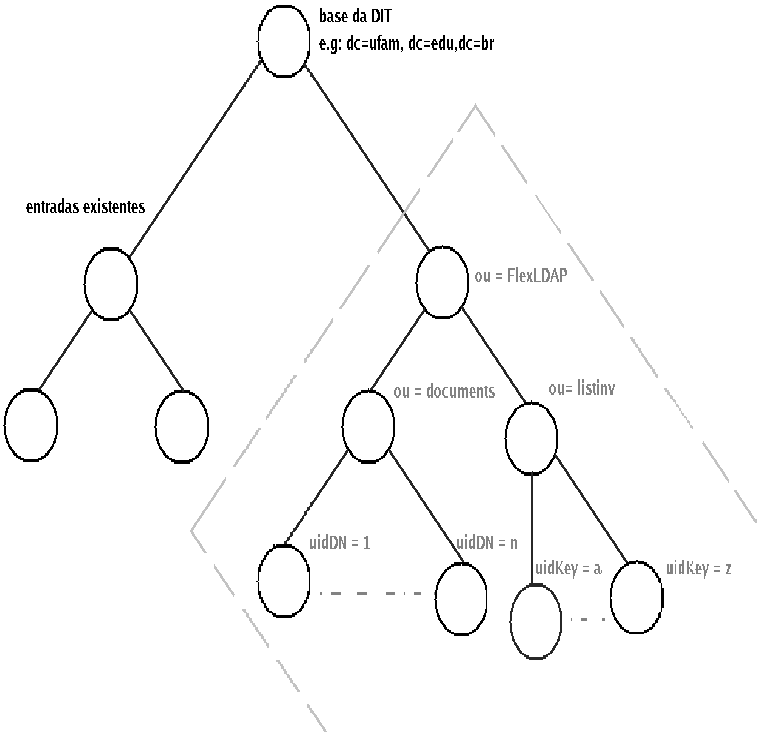

Figura 3.5: Exemplo de uma Sub-árvore do Flex-LDAP.

um atributo, O o operador lógico (extensões) e V o valor a ser consultado.

Um sistema LDAP que recebe este tipo de consulta pode utilizar dois formatos por: casamento, onde cada valor e atributo da consulta são casados com os encontrados na DIT; e o por busca por DN, que utiliza o atributo DN que possui índice. Estes tipos de consulta influênciam a performance do LDAP, pois em casos de consultas por DN utiliza-se do sistema de indexação existente no *backend* utilizado pelo servidor LDAP, o que torna a consulta mais rápida.

O método de consulta por casamento será mais lento, por não ter índices criados para todos os atributos possíveis. Na configuração do servidor LDAP pode-se adicionar mais atributos ao sistema de indexação, mas tornar-se inviável a indexação para todos os atributos existentes.

Nossa proposta de processamento de consulta é totalmente baseada em DN, independente da consulta efetuada pelos clientes LDAP. Isto é possível pois existe

a indexação prévia da DIT, realizada pelo módulo indexador. E o módulo de processamento de consulta do Flex-LDAP conhece o esquema que foi implementado na DIT, e com isto realiza consultas diretas utilizando DN.

O processamento de consulta do Flex-LDAP é inteiramente baseado no LDAP versão 3, a parte de processamento interno do modelo vetorial é transparente para todo o sistema. Pois, este processo faz parte do processo de servidor que possui internamente funções de processamento do modelo vetorial.

Na Figura 3.6 é representada a mensagem *BindRequest* efetuada por um cliente LDAP para o Flex-LDAP, isto é uma mensagem de autenticação ao servidor, passando parâmetros de usuário, senha e base da DIT. O Flex-LDAP autentica e responde para o cliente LDAP, usando as especificações do LDAP v3, o que garante a coexistência das implementações dos clientes e servidores LDAPs.

Após a autenticação no Flex-LDAP, o cliente submete uma mensagem *SearchRequest* com o filtro de consulta para o servidor Flex-LDAP.
Neste momento o Flex-LDAP extrai os valores da consulta Q_o oriunda do cliente LDAP e descarta os atributos A e operadores lógicos e extensões O, aproveitando somente os valores, formando um conjunto $Q = \{k_1, \ldots, k_n\}$ onde $n \geq 1$.

Este conjunto Q será utilizado para formar as n consultas q que serão efetuadas para um servidor LDAP. O método de consulta utilizado pelo Flex-LDAP será por DN. Definimos a formação de cada consulta q sendo:

$$q \longleftarrow "uidKey = " + k_n + ", ou = listinv, ou = flexldap"$$

Observa-se na Figura 3.6 que o Flex-LDAP também realiza sua autenticação no servidor LDAP, por isto a existência das mensagens de *BindRequest e BindResult* entre o Flex-LDAP e o servidor.

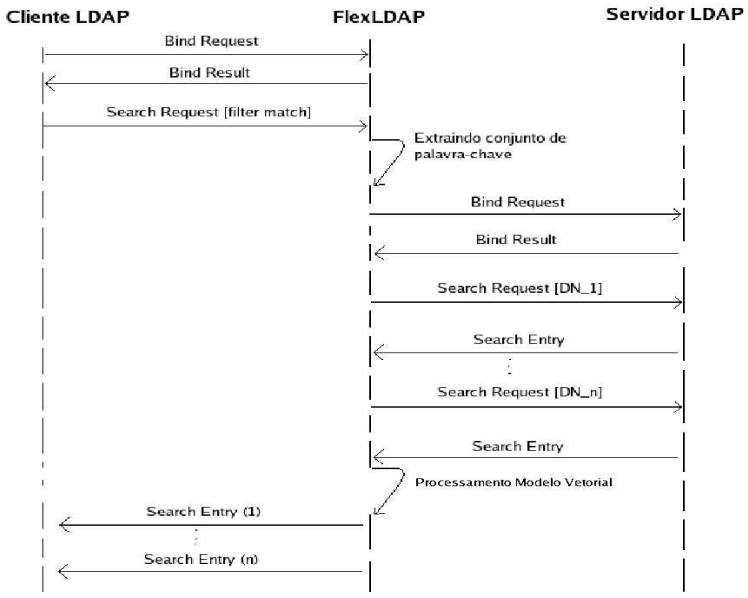

Figura 3.6: Transação de Mensagens FlexLDAP em uma Consulta.

Após isto, o Flex-LDAP submete as novas consultas formuladas por ele, utilizando mensagens *SearchRequest* e recebe a resposta a cada consulta q com mensagens *SearchEntry* do servidor LDAP. A cada resposta, é obtida uma entrada e_n, que terá os valores de freqüência $tf_{k_n ind n_n}$ (quantidade de vezes que apareceu) do termo k_n em uma determinada entrada dn_n. Além disto, será obtido o valor de idf_{k_n} do termo k_n para a coleção D. Todos estes valores foram criados pelo módulo de indexação. Os valores TF e IDF serão utilizados para calcular o peso w que é definido como:

$$w = tf_{k_n ind n_n} \times idf_{k_n}$$

Este peso representa quão relevante é a entrada dn_n para o termo k_n da consulta Q.

Nesta mesma entrada pode ocorrer que o termo k_n tenha aparecido em mais de uma entrada dn_n, logo é necessário um vetor que acumule as informações de entrada-peso. O FlexLDAP gera um acumulador $AC = \{\langle ac_1, peso\rangle, \ldots, \langle ac_n, peso\rangle\}$.

Este processo será realizado para todos os termos existentes na consulta Q, obtendo ao final um conjunto AC que representa a lista de entradas relevantes para a consulta Q.

Este conjunto AC é codificando para ASN.1/BER e transmitido como mensagem *SearchEntry* para o cliente LDAP que originou a consulta.

Algorithm 1: Processamento de Consulta

```
   Q := {k_1, ..., k_n};
 2: while (Q.hasMoreKeywords())
        {
 4:     k_n ⟵ Q.nextKeyword();
        q ⟵ "uidKey = " + k_n + ", ou = listinv, ou = flexldap";
 6:     e_n ⟵ ldapsearch(q);
        while (e_n.hasMoreAttributes())
 8:         {
            dn_n ⟵ e_n.group(1);
10:         tf_{k_n ind n_n} ⟵ e_n.group(2);
            idf ⟵ e_n.idf;
12:         peso ⟵ tf × idf;
            if (AC.containKey(dn_n))
14:             {
                At ⟵ AC.containKey(dn_n);
16:             AC ⟵ {AC − At} ⋃ {dn_n, peso};
                }
18:         else
                AC ⟵ AC ⋃ {dn_n, peso};
20:         }
        }
```

Detalhe da Implementação do Processamento de Consultas do FlexLDAP

O módulo de processamento de consultas faz parte do serviço Flex-LDAP, que é uma implementação de um serviço LDAP, conforme RFC 2253 [13]. Este serviço, denominado *FlexLDAPGateway*, implementa um *socket*, configurado na porta 389 utilizando o protocolo TCP, que é a porta padrão para o serviço LDAP. Logo, o servidor LDAP existente deverá ser reconfigurado para responder em outra porta diferente da 389, pois por padrão clientes LDAP efetuam consultas para a porta 389. Para nossos experimentos reconfigurar a porta do servidor para um funcionamento adequado de todo o processo de consultas é a única alteração a ser realizada nas configurações no sistema LDAP padrão, para que ele possa interagir com o Flex-LDAP.

Existe um arquivo de configuração onde os parâmetros de portas do Flex-LDAP e do LDAP devem ser configurados, o arquivo tem o nome de flexldap.prop. Após sua execução, o serviço FlexLDAP espera que algum cliente se conecte com sua porta.

Para que o servidor Flex-LDAP realize o processamento de consultas é necessário que um cliente LDAP faça uma interação de mensagens LDAP, normalmente, conforme especificações da RFC 2153 da versão 3 do protocolo LDAP. Como por exemplo:

$$ldapsearch - hserverflex - p3189 - x - D"cn = Manager, dc = ufam, dc = edu, dc = br - wsecret'(|(cn = carrie*)(cn = *pai))'$$

Nesta consulta um cliente LDAP se conecta a um servidor com nome de *serverflex* e porta TCP 3189, que no caso é o serviço *FlexLDAPGateway*, e passa como parâmetro: usuário, senha, base e o filtro de consulta. O filtro de consulta, neste exemplo, está estruturado com o operador lógico **OR**, e duas especificações de atributos, sendo respectivamente para os atributos CNs, onde os valores *carrie* e *pai* foram atribuídos, representando um interesse do cliente em entradas que possuam o

atributo *cn* com valores iniciando com a palavra *carrie*, e entradas com o atributo *cn* terminando com a palavra *pai*. Para este exemplo, o módulo de processamento de consultas deverá extrair os valores *carrie* e *pai*, que serão considerados as nossas palavras-chave para formulação da resposta do modelo vetorial.

A extração destas palavras é através da interceptação das mensagens LDAP, conforme Figura 3.6. Após as mensagens LDAP *BindRequest*, *BindResult* e a *SearchRequest*, existe a decomposição do filtro de consulta para obtenção das palavras-chave. O processamento do modelo vetorial, após a extração das palavras-chave procede da seguinte forma:

1. Realizar conexão com o serviço LDAP especificado no arquivo flexldap.prop, conforme variáveis *flexldap.service.host, port, rootuser, rootpw,basedn*;

2. Para cada termo (palavra-chave) é realizada uma consulta LDAP direta especificando o **DN**, que no caso será montado com a seguinte estrutura: *uid-Key=[termo],ou=listinv, ou=flexldap*;

3. O resultado da consulta DN é para obtenção dos valores de **IDF** do termo, freqüência e **id** do documento onde ocorreu a freqüência, sendo que para obter a freqüência e **id** do documento é necessária a interação na quantidade de entradas que o termo apareceu, para tal o atributo *freqInDN* foi especificado como multivalorado;

4. Para cada interação no atributo *freqInDN* é realizado o cálculo de peso, que é obtido pelo produto da freqüência e **IDF** do termo;

5. Em uma tabela *Hash* é acumulado o peso para cada entrada encontrada no resultado da consulta, para um termo especifico;
6. Para cada termo (palavra-chave) são realizados os mesmos passos a partir do item 2;

7. Ao final das consultas, a tabela *Hash* é percorrida para gerar uma lista em ordem decrescente pelo peso de cada documento. Com isto, obtém-se a lista ordenada com as melhores entradas para a consulta realizada por um cliente.

De posse da lista ordenada dos documentos o módulo encapsula os documentos em forma de mensagens LDAP e as envia para o cliente LDAP solicitante, conforme se pode observar na Figura 3.6.

Capítulo 4

Experimento e Resultado

4.1 Configuração

Descrevemos nosso experimento, incluíndo o hardware utilizado, o servidor LDAP software, o FlexLDAP software, o cliente LDAP e a configuração de *benchmarking*.

O servidor LDAP e o FlexLDAP software executaram sobre sistema operacional Linux na distribuição Fedora Core 3, uma CPU Intel Pentium IV 2.0 MHz, 512MB de memória RAM e disco rígido de 120 GB. O cliente LDAP foi executado sobre uma CPU Intel Pentium IV 3.0 GHz, 256MB de memória RAM, com disco rígido de 60GB executando sobre um linux Fedora core 4. Os servidores e clientes foram conectados através de uma rede 100 Mb/s fastethernet (cabo cruzado); onde somente as duas máquinas utilizaram a rede, descartando com qualquer *broadcast ou throughput* de outras máquinas.

4.1.1 Servidor LDAP

Existem diversas distribuições comerciais de servidores LDAP, incluíndo *Netscape Directory Server, Novell LDAP services*. Nós escolhemos o OpenLDAP 2.3. Este é um suite de cliente e servidor LDAP de código aberto. O que nos permite verificar detalhes de implementação para entendimento do protocolo de comunicação, conforme recomendações das RFCs.

O servidor é baseado em um *stand-alone LDAP UNIX daemon (slapd)* para o serviço de diretório. O serviço de replicação é também suportado por um *UNIX daemon*, o *slurpd*.

O *slapd* consiste de duas distintas partes: um *frontend* que é o servidor de protocolo de comunicação com os clientes LDAP; e um banco de dados *backend* que pode ser escolhido na configuração do servidor LDAP. O OpenLDAP possui suporte as bases Berkeley DB e GDBM. O LDBM é um banco de dados com alta performance de disco que utiliza em baixo nível hash ou b-tree em seu banco de dados. Neste trabalho usamos o LDBM, denominado Berkley DB com versão 4.

4.1.2 Cliente LDAP

A arquitetura cliente-servidor usada em nosso experimento é mostrada na Figura 4.1. Um processo mestre coordena os processos de cliente e gera um relatório de performance. Os parâmetros de configuração são definidos em um arquivo.
Com isto, o processo mestre constroi os comandos de linha com argumentos para que o processo cliente LDAP execute as consultas especificas para um serviço LDAP. Cada cliente lê os comandos e inicia uma comunicação com o processo mestre. Após todos os clientes terem sido iniciados, o processo mestre instrui os clientes a iniciarem as consultas. Ele aguarda o término do experimento, enquanto recebe todas as informações de todos os clientes LDAP. Cada cliente realiza consulta ao servidor de diretório LDAP, periodicamente.

4.1.3 Metodologia

Nosso experimento tem o objetivo de analisar a performance e corretude da proposta FlexLDAP. Para tal, iremos comparar dois sistemas LDAP, sendo o primeiro um sistema composto por cliente e servidor LDAP, conforme as recomendações das RFCs na versão 3 do LDAP. O outro será composto também por cliente-servidor LDAP, só que entre um cliente e servidor LDAP estará o servidor FlexLDAP.

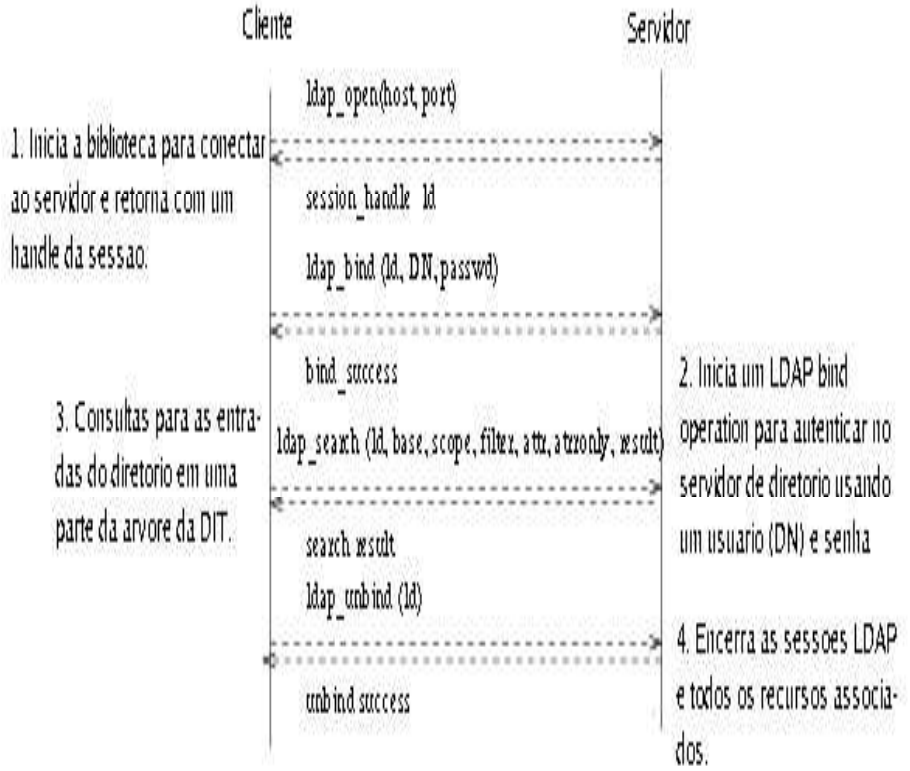

Figura 4.1: Arquitetura Cliente-Servidor utilizada no Experimento.

Em sistemas LDAP são comuns as operações modificar, adicionar, apagar, comparar e consultar. Em nosso experimento consideramos aplicações pseudo-estáticas, como *white-pages*, o que é comumente baseada com o LDAP e também pela motivação deste trabalho, que é construir uma forma de consultas fléxiveis com o LDAP.

Uma simples operação de consulta envolve uma seqüência de quatro operações: ldap-open, ldap-bind, um ou mais ldap-search e ldap-unbind (Figura 4.1).

A operação *ldap-open* abre uma conexão com o servidor LDAP, e retorna uma sessão para uso futuro. O *ldap-bind* é responsável pela autenticação. A operação ldap-bind permite a um cliente identificar-se ao servidor LDAP utilizando um *Distinguished Name* e uma senha. O LDAP suporta uma variedade de métodos de autenticação. Em nosso experimento foi usado o método por senha.

Quando uma operação *bind* é completada com sucesso, o servidor de diretório guarda a sua identificação até outra operação *ldap-bind* ou a sessão LDAP for terminada por uma chamada *ldap-unbind*.

Quanto a performance dos sistema consideramos as métricas de tempo de conexão, tempo de processamento, tempo de resposta. O tempo de conexão definimos como o tempo de envio da requisição ldap-open até a operação ldap-bind com sucesso. O tempo de processamento foi definido como o tempo requerido para uma operação ldap-search também como o tempo de transferência do resultado para o cliente.

O tempo total para uma operação *ldap-search*, da operação *ldap-open* até *ldap-unbind* definimos como tempo de resposta.

As medidas que reflete a performance dos sistemas LDAP são conexão, processamento e latência de resposta e o *throughput* do servidor, representado pela

média de requisições por segundo.

Em nosso experimento cada operação de consulta envolve todos os quatros passos: *ldap-open, ldap-bind, ldap-search e ldap-unbind*.

Quanto a precisão comparamos as respostas dos sistemas, sendo que as respostas obtidas pelo sistema composto apenas por cliente-servidor LDAP consideramos como referência.

A massa de consultas efetuadas para os dois sistemas foram geradas aleatoriamente pelo software *DirectoryMark*. O software gera as consultas baseado em dicionários de nomes, onde realiza combinações para construir os filtros de consultas.

4.2 Resultado dos Experimentos

Para realizarmos os testes construimos uma DIT composta por 100.000 entradas geradas a partir de dicionários do programa *DirectoryMark*.

As consultas efetuadas para os sistemas Flex-LDAP e Open-LDAP foram baseadas nas consultas geradas também pelo *DirectoryMark*.

4.2.1 Performance

Observamos que o tempo de conexão e autenticação entre os sistemas são próximos, sendo que o Flex-LDAP apresentam um tempo mais alto para este processo, conforme se pode observar na Figura 4.2. Podemos considerá-los equivalentes em tempo, pois seguimos a mesma especificação do protocolo LDAP versão 3. Acreditamos que a perda de performance do Flex-LDAP tenha haver com a linguagem utilizada em nossa implementação, que no caso foi Java. Enquanto que a linguagem utilizada no servidor OpenLDAP é **C**. Extraimos os pacotes referentes a consulta LDAP, observando através do programa *tcpdump*. Onde

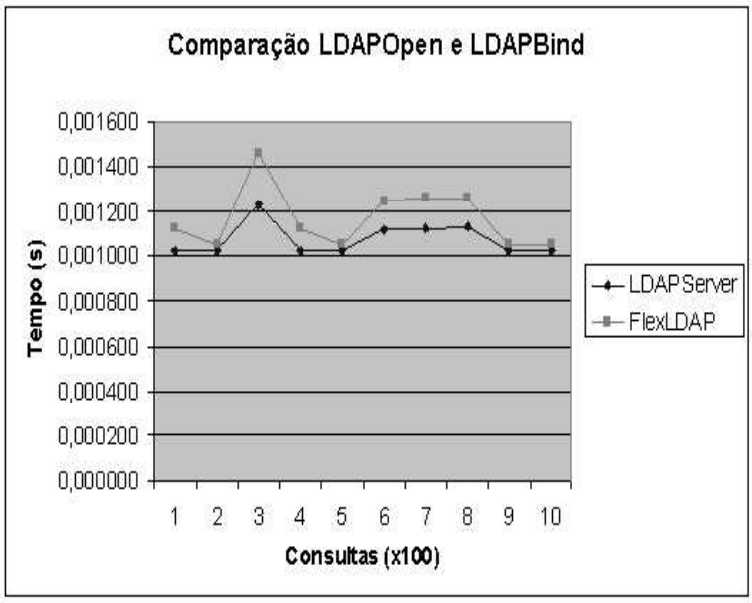

Figura 4.2: Comparação LDAPBind e LDAPOpen entre LDAP e Flex-LDAP.

os tempos referentes as mensagens LDAP utilizada nesta operação foram computados em nossa análise. Observamos que existe um significativa diferença entre os sistemas, pois em todas as consultas o Flex-LDAP possui um tempo de resposta menor que um sistema composto somente por um servidor LDAP em sua versão 3. Na Figura 4.3 podemos observar esta diferença, onde um sistema composto apenas por LDAP tem um tempo máximo de 3,9s para uma operação LDAPSearch. Enquanto para um sistema composto com o Flex-LDAP o tempo para a mesma operação cai para um tempo máximo de 0,11s. Este tempo é atingido pela caracterização que conseguimos na implementação do Flex-LDAP, onde todas as operações de LDAPSearch são baseadas em DN, diferente de uma consulta do LDAP, que pode utilizar consultas baseada em casamento de atributos.

Os tempos de desconexão entre sistemas, o Flex-LDAP possui um tempo maior de desconexão pelo motivo de existir mais operações LDAP que um sistema LDAP, mas o tempo total da operação de uma consulta, incluíndo LDAPOpen, LDAPBind, LDAPSearch e LDAPUnbind, o Flex-LDAP ganha pois a maior latência entre os sitemas está na operação de LDAPSearch. Na Figura 4.4 existe uma comparação dos maiores tempos alçandos entre os dois sistemas, o que mostra

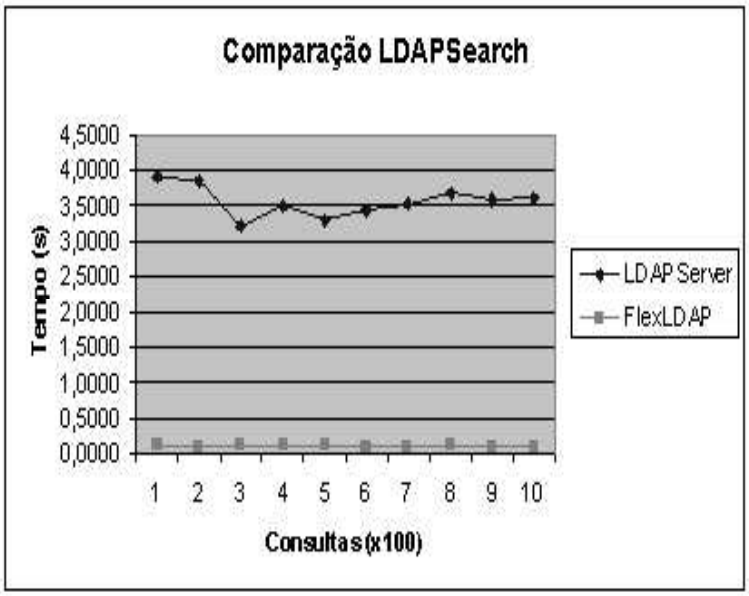

Figura 4.3: Comparação LDAPSearch entre LDAP e Flex-LDAP.

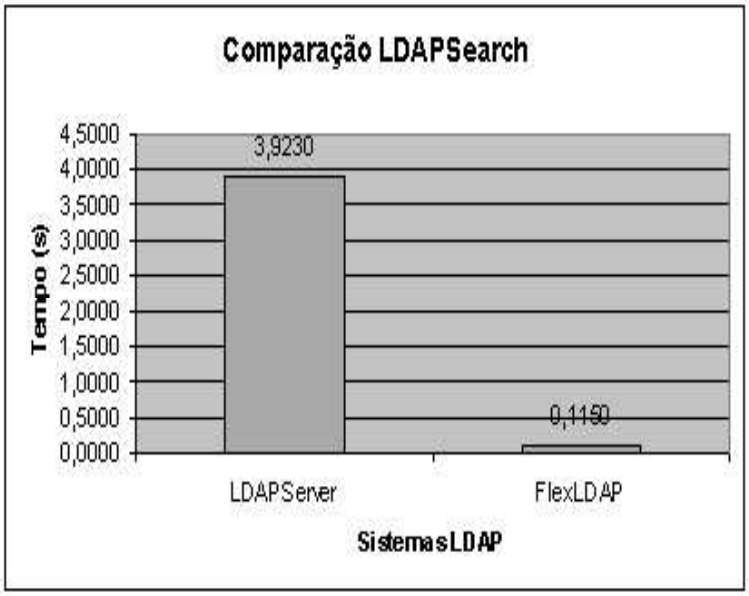

Figura 4.4: Comparação LDAPSearch entre LDAP e Flex-LDAP.

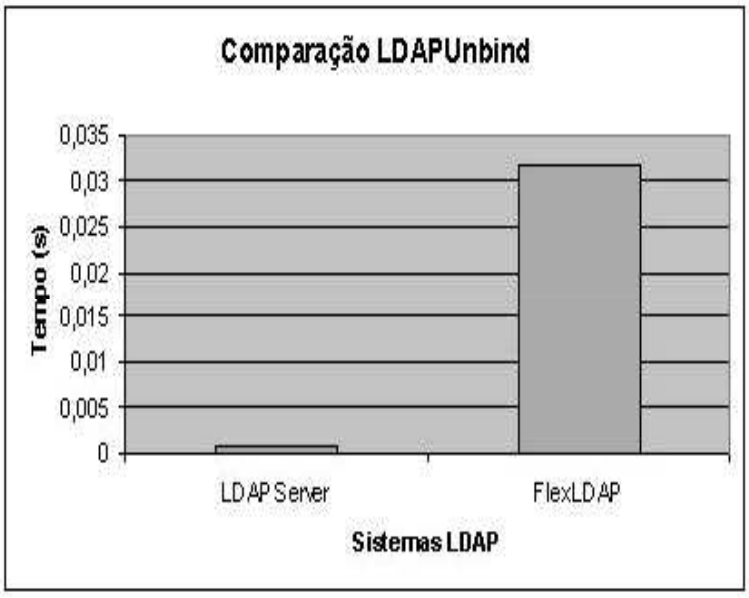

Figura 4.5: Comparação LDAPUnbind entre LDAP e Flex-LDAP.

um ganho de performance relevante para um sistema composto com o Flex-LDAP.

4.2.2 Precisão

Em todas as consultas efetuadas para o Flex-LDAP o resultado trouxe todas as respostas obtidas para um sistema LDAP em sua versão 3. Isto deve ao fato que a DIT é indexada totalmente, não utilizamos indexação parcial do diretório. Logo, todos os termos foram indexados e apareceram no resultado das consultas para termos existentes na DIT. Com isto, o conjunto resposta do Flex-LDAP contém o conjunto resposta do LDAP versão 3, e mais outras respostas encontradas, pois ele traz outros resultados que o LDAP versão 3 não consegue encontrar, por utilizar operadores lógicos.

Capítulo 5

Conclusões e Trabalhos Futuros

5.1 Conclusões

Apresentamos uma solução para consultas flexíveis para o protocolo LDAP, onde é possível a utilização de diretórios, com os mais diversificados esquemas, utilizando clientes LDAP sem a necessidade de conhecimento prévio do esquema utilizado.

Por adotarmos a construção de um *middleware* para interagir entre um cliente e servidor LDAP, a sua inserção num sistema de diretórios não acarreta em grandes traumas, sendo o principal impacto o aumento da DIT, e uma reconfiguração da porta de comunicação do servidor LDAP. Ficando transparente para o cliente, o Flex-LDAP. Passando este a poder realizar consultas não estruturadas. Tornar-se possível a construção de interfaces de consultas simples, e com expressividade nos resultados. Não sendo necessária a construção de sítios com diversos campos.

Apesar da camada a mais, entre cliente LDAP e Servidor LDAP, que é o Flex-LDAP, o tempo de resposta a consulta cai significativamente. Em nossos experimentos, o Flex-LDAP apresenta tempo de resposta, com ganhos de aproximadamente de 97% do tempo apresentado pelo servidor LDAP. Logo, além da flexibilidade da consulta obtém-se o ganho na performance das consultas de clientes LDAP.

Os sistemas atuais que utilizem o LDAP na versão 3 podêm utilizar de imediato o *middleware* Flex-LDAP.

5.2 Trabalhos Futuros

A principal questão que avaliamos não ser tão positiva é o fato de necessitarmos criar novas entradas na DIT, referenciando os índices usados pelo Flex-LDAP. Com isso, pode ocorrer um crescimento significativo do volume da DIT, e talvez não desejado. Para contornar esta questão, sugerimos que sejam realizados estudos sobre compressão de dados, métodos de podas para melhoria no processo de indexação.

Em nossos experimentos, onde não utilizamos qualquer método de compreensão ou outros métodos que diminua o tamanho do armazenamento das informações, tivemos um aumento de aproximadamente 40% do tamanho da DIT inicial, isto pode ser siginificativo para plataformas que possuem limitações de unidades de disco. Apesar, que os módulos de processamento de consulta rodam em servidores, que geralmente possuem grandes unidades de disco.

Com esta solução é possível a utilização em infra-estrutura de telefônia móvel, onde existe a necessidade de simplicidade em interface do cliente móvel. Por isto, sugerimos que esta solução seja utilizada em servidores de GPRS, que possibilita que clientes móveis usufruam de aplicações IP.

Aplicações de localizações, como mapas e outras informações armazenadas em diretórios podem ser uma boa utilização para testes desta nossa proposta.

Referências Bibliográficas

[1] R. Baeza-Yates e B. Ribeiro-Neto. *Modern Information Retrieval.* Addison Wesley, 1999.

[2] Clayton Donley. *LDAP Programming, Management and Integration.* Manning PublicationsCo, 2003.

[3] IETF (2005). The Internet Engineering Task Force, disponível na url: http://www.ietf.org/. Acessado em 19 de Dezembro de 2005.

[4] Calado, P., da Silva, A. S., Vieira, R. C., Laender, A . H. F., & Ribeiro-Neto, B. A . (2002). Searching Web databases by structuring keyword-based queries. In Proceedings of the 11th international conference on information and knowledge management (pp. 26-33). McLean, VA, USA: ACM Press.

[5] Agrawal, S., Chaudhrui, S., & Das, G. (2002). DBXplorer: A system for keyword-based search over relational databases. In Proceedings of the 18th international conference on data engineering (pp. 5-16). San Jose, CA, USA: IEEE Computer Society.

[6] Salton, G., & McGill, M.J. (1983). Introduction to modern information retrieval. New York, USA: McGraw-Hill.

[7] Yahoo! People Search (2005). Site do Yahoo People Search, disponÃvel na url: http://people.yahoo.com/. Acessado em 01 de Novembro de 2005.

[8] Chaudhuri, S., & Gravano, L. (1999). Evaluating top-k selection queries. In Proceedings of 25th international conference on very large data bases VLDB99 (pp. 397-410). Edinburgh, UK: Mogran Kaufman.

[9] Dar,S.,Entin,G.,Geva, S.,& Palmin, E. (1998). DTLs DataSport: Database exploration using plain language. In Proceedings of 24th international conference on very large data bases VLDB98 (pp. 645-649). New York, USA: Morgan Kayfman.

[10] EduPerson Object Class (2005). The EDUCASE/Internet2 eduPerson task force has the mission of defining an LDAP object class that includes widely-used person attributes in higher education. Acessado em 18 de Novembro.

[11] M. Wahl, T. Howes, and S. Kille.Lightweight Directory Access Protocol (v3): Protocol RFC 2251, December 1997. http://www.rfc-editor.org/in-notes/rfc2251.txt.

[12] M. Wahl, A. Coulbeck, T. Howes, S. Kille. Lightwight Directory Access Protocoll (v3): Attribute Syntax Definitions. RFC 2252, December 1997. http://www.rfc-editor.org/in-notes/rfc2252.txt.

[13] M. Wahl, S. Kille, T. Howes.Lightwight Directory Access Protocoll (v3): UTF-8 String Representation of Distinguished Names.RFC 2253, December 1997.http://www.rfc-editor.org/in-notes/rfc2253.txt.

[14] T. Howes. Lightwight Directory Access Protocol (v3): The String Representation of LDAP Search Filters. RFC 2254, December 1997. http://www.rfc-editor.org/in-notes/rfc2254.txt.

[15] T. Howes, M. Smith. The LDAP URL Format. RFC 2255, December 1997. http://www.rfc-editor.org/in-notes/rfc2255.txt.

[16] M. Wahl. A Summary of the X.500(96) User Schema for use with LDAPv3. RFC 2256, December 1997.http://www.rfc-editor.org/in-notes/rfc2256.txt.

[17] M. Smith. Definition of the inetOrgPerson LDAP Object Class. RFC 2798, April 2000. http://www.rfc-editor.org/in-notes/rfc2798.txt.

[18] David H. Crocker. ARPA Internet Text Messages. RFC 822, August 1982. http://www.faqs.org/rfcs/rfc822.html.

[19] Mark Valence. Posting auf openldap-devel@openldap.org betreffs Implementing object-based access control, August 1999.http://www.openldap.org/lists/openldap-devel/199908/msg00110.html.

[20] M. Wahl, T. Howes, and S. Kille. RFC 2251:Lightweight Directory Access Protocol (v3).http://www.ietf.org/rfc/rfc2251.txt, December 1997.

[21] Information Processing — Open Systems Interconnection: Specification of Abstract Syntax Notation One (ASN.1). 2nd Draft International Standard 8824.

[22] Open systems interconnection: Specification of basic encoding rules for abstract syntax notation one (asn.1). CCIT Recommendation X.209, 1992.

[23] JNDI Sun Microsystems, Inc. (Hrsg.): *Java Naming and Directory Interface (JNDI).*

[24] DIT Michigan (2005). Site da Universidade de Michigan, Online Directory, disponível na url: http://directory.umich.edu/. Acessado em 01 de Novembro de 2005.

[25] DirectoryMark. The LDAP Server Benchmarking Tool , disponível na url: http://www.mindcraft.com/directorymark/. Acessado em 19 de Dezembro de 2005.

[26] DIT Harvard (2005). Site da Universidade de Harvard, Online Directory, disponível na url: https://www.directory.harvard.edu/phonebook/. Acessado em 19 de Dezembro de 2005.

[27] DIT Notre Dame (2005). Consulta de diretórios da Universidade de Notre Dame, disponível na url:http://eds.nd.edu/search/adv_search.shtml. Acessado em 01 de Novembro de 2005.

[28] Florescu, D., Kossmann, D., & Manolescu, I. (2000). Integrating Keyword Search into XML Query Processing. WWW9/ Computer Networks, 33(1-6), 119-135.

[29] XML-QL (1998): A Query Language for XML, especificação disponível na url: http://www.w3.org/TR/1998/NOTE-xml-ql-19980819/. Acessado em 21 de Novembro de 2005.

[30] Middleware (2005). Object Web Open Source Middleware , disponível na url: http://middleware.objectweb.org/. Acessado em 23 de Novembro de 2005.

[31] BER. Information technology:ASN.1 encoding rules:Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER), disponível na url: http://www.itu.int/ITU-T/studygroups/com17/languages/X.690-0207.pdf. Acessado em 24 de Novembro de 2005.

Apêndice A

Algoritmo do Módulo de Indexação do Flex-LDAP

Algorithm 2: Indexador

```
1: n := totalEntriesDIT;
2: D := {E_1, ..., E_n};
3: DN_s:= {};
4: V := {};
5: while (D.hasMoreEntries())
6:     {
7:     E_n := D.nextEntry();
8:     dn := E_n.DN;
9:     DN_s:= DN_s ∪ {p, dn};
10:    while (E_n.hasMoreAttributes())
11:        {
12:        A_i := E_n.nextAttribute();
13:        while (A_i.hasMoreValues())
14:            {
15:            v_{A_i} := A_i.nextValue();
16:            T := TOKENIZER(v_{A_i});
17:            while (T.hasMoreToken())
18:                {
19:                tk := T.nextToken();
20:                if (tk ⊂ V)
21:                    {
22:                    K := V ⋂ tk;
23:                    if (dn ⊂ K)
24:                        K_{f_{tk(DN)}} += 1;
25:                    else
26:                        K := K ∪ {DN, 1};
27:                    }
28:                else
29:                    {
30:                    K := {DN, 1};
31:                    V := V ∪ K;
32:                    }
33:                }
34:            }
35:        }
36:    }
```

Apêndice B

Envelope LDAPMessage

```
LDAPMessage ::= SEQUENCE {

                messageID       MessageID,
                protocolOp      CHOICE {
                        bindRequest     BindRequest,
                        bindResponse    BindResponse,
                        unbindRequest   UnbindRequest,
                        searchRequest   SearchRequest,
                        searchResEntry  SearchResultEntry,
                        searchResDone   SearchResultDone,
                        searchResRef    SearchResultReference,
                        modifyRequest   ModifyRequest,
                        modifyResponse  ModifyResponse,
                        addRequest      AddRequest,
                        addResponse     AddResponse,
                        delRequest      DelRequest,
                        delResponse     DelResponse,
                        modDNRequest    ModifyDNRequest,
                        modDNResponse   ModifyDNResponse,
                        compareRequest  CompareRequest,
                        compareResponse CompareResponse,
```

```
                abandonRequest  AbandonRequest,
                extendedReq     ExtendedRequest,
                extendedResp    ExtendedResponse },
        controls        [0] Controls OPTIONAL }

MessageID ::= INTEGER (0 .. maxInt)

maxInt INTEGER ::= 2147483647 -- (2^^31 - 1) --
```

Apêndice C

Gramática *BindRequest*

```
BindRequest ::= [APPLICATION 0] SEQUENCE {
version                 INTEGER (1 .. 127),
name                    LDAPDN,
        authentication          AuthenticationChoice }

AuthenticationChoice ::= CHOICE {
                simple                  [0] OCTET STRING,
                        -- 1 and 2 reserved
sasl                    [3] SaslCredentials }
        SaslCredentials ::= SEQUENCE {
mechanism               LDAPString,
   credentials             OCTET STRING OPTIONAL }
```

Os parâmetros do *BindRequest* são:

version: Um número de versão indicando a versão do protocolo utilizado na sessão. Este documento descreve a versão 3 do protocolo LDAP. Não acontece uma negociação de versão, o cliente apenas informa em qual versão do LDAP deseja trabalhar, caso o cliente solicite ao protocolo à versão 2, um servidor que suporte a versão 2 não retornará qualquer campos específicos do protocolo LDAP em sua versão 3;

name: O DN (*Distiguished Name*) de um objeto que representa um usuário com credenciais para autenticação no servidor LDAP. Este campo pode conter valor nulo, em caso de usuário *anonymous*.

authentication: Informações utilizadas para autenticar o objeto name. Em caso de solicitação de autenticação pelo cliente ao servidor. O servidor retornará uma mensagem *BindResponse* indicando o status da autenticação para o cliente. Para alguns mecanismo de autenticações do tipo SASL, pode ser necessário que o cliente invoque o *BindRequest* várias vezes. Neste documento não detalharemos o modelo de segurança por não ser o foco desse trabalho. Por outro lado, a autenticação simples provê um mínimo de segurança, onde o conteúdo da autenticação (senha) consiste de uma string em forma de texto limpo, ou seja sem nenhuma criptografia.

Apêndice D

Gramática *BindResponse*

```
BindResponse ::= [APPLICATION 1] SEQUENCE {
   COMPONENTS OF LDAPResult,
   serverSaslCreds    [7] OCTET STRING OPTIONAL }
```

Um *BindResponse* é constituído dos componentes de um *LDAPResult*, que por sua vez é definido conforme a gramática descrita no Apêndice E.

Apêndice E

Gramática *LDAPResult*

```
LDAPResult ::= SEQUENCE {  resultCode      ENUMERATED {
               success           (0),
               operationsError          (1),
               protocolError      (2),
               timeLimitExceeded         (3),
               sizeLimitExceeded          (4),
       compareFalse      (5),
       compareTrue               (6),
       authMethodNotSupported     (7),
         strongAuthRequired        (8),
       -- 9 reserved --
       referral         (10),  -- new
       adminLimitExceeded      (11),  -- new
       unavailableCriticalExtension  (12),  -- new
       confidentialityRequired       (13),  -- new
       saslBindInProgress            (14),  -- new
       noSuchAttribute               (16),
       undefinedAttributeType        (17),
       inappropriateMatching         (18),
       constraintViolation           (19),
       attributeOrValueExists        (20),
```

```
invalidAttributeSyntax        (21),
-- 22-31 unused --
noSuchObject                  (32),
aliasProblem                  (33),
invalidDNSyntax               (34),
-- 35 reserved for undefined isLeaf --
aliasDereferencingProblem     (36),
-- 37-47 unused --
inappropriateAuthentication   (48),
invalidCredentials            (49),
insufficientAccessRights      (50),
busy                    (51),
unavailable            (52),
        unwillingToPerform            (53),
        loopDetect             (54),
-- 55-63 unused --
namingViolation               (64),
objectClassViolation          (65),
notAllowedOnNonLeaf           (66),
notAllowedOnRDN               (67),
entryAlreadyExists            (68),
objectClassModsProhibited     (69),
-- 70 reserved for CLDAP --
affectsMultipleDSAs           (71), -- new
-- 72-79 unused --
other                   (80) },
-- 81-90 reserved for APIs --
matchedDN       LDAPDN,
errorMessage    LDAPString,
referral        [3] Referral OPTIONAL }
```

Apêndice F

Gramática *SearchRequest*

```
SearchRequest ::= [APPLICATION 3] SEQUENCE {    baseObject      LDAPDN,
     scope           ENUMERATED {
      baseObject             (0),
      singleLevel            (1),
      wholeSubtree           (2) },
      derefAliases    ENUMERATED {
             neverDerefAliases      (0),
            derefInSearching       (1),
                     derefFindingBaseObj    (2),
             derefAlways            (3) },
       sizeLimit       INTEGER (0 .. maxInt),
     timeLimit       INTEGER (0 .. maxInt),
     typesOnly       BOOLEAN,
     filter          Filter,
        attributes      AttributeDescriptionList }

Filter ::= CHOICE {
               and             [0] SET OF Filter,
or             [1] SET OF Filter,
               not             [2] Filter,
               equalityMatch   [3] AttributeValueAssertion,
```

```
                substrings      [4] SubstringFilter,
                greaterOrEqual  [5] AttributeValueAssertion,
                lessOrEqual     [6] AttributeValueAssertion,
                present         [7] AttributeDescription,
                approxMatch     [8] AttributeValueAssertion,
                extensibleMatch [9] MatchingRuleAssertion }


SubstringFilter ::= SEQUENCE {
                type            AttributeDescription,
                -- at least one must be present
                substrings      SEQUENCE OF CHOICE {
                        initial [0] LDAPString,
                        any     [1] LDAPString,
                        final   [2] LDAPString } }


  MatchingRuleAssertion ::= SEQUENCE {
                matchingRule    [1] MatchingRuleId OPTIONAL,
                type            [2] AttributeDescription OPTIONAL,
                matchValue      [3] AssertionValue,
                dnAttributes    [4] BOOLEAN DEFAULT FALSE }
```

Os parâmetros de uma mensagem *SearchRequest* são:

baseObject: Um DN que define o ponto inicial, chamado de objeto base, da pesquisa. O objeto base é um nó dentro da DIT; *scope*: O escopo da consulta a ser realizada. Especifica a profundidade a pesquisar dentro da DIT a partir do objeto base. Há três opções:

baseObject: somente o objeto base é examinado;

singleLeve: somente o descendente imediato do objeto base é examinado; o objeto base em si não é examinado;

WholeSubtree: O objeto base e todos os seus descendentes serão examinados.

derefAliases: Especifica se os aliases não são referenciados. Ou seja, o objeto real de interesse, apontado por uma entrada de alias, é examinado.

sizeLimit: Restringe um número máximo de entradas a serem retornadas como resultado em uma consulta. Um valor zero atribuído neste campo indica que o cliente não especificou restrições de tamanho. Servidores podem forçar um número máximo de entradas a serem retornadas.

timeLimit: restringe o tempo máximo (em segundos) permitida para a consulta. Um valor zero neste campo indica que não existe restrição de tempo.

typesOnly: Um flag que indica se o resultado conterá atributos mais valores, ou apenas atributos. Quando atribuído o valor *TRUE* somente os atributos serão retornados (nenhum valor), ao contrário quando atribuído o valor *FALSE* ambos serão retornados, atributos e valores.

filter: Especifica o critério que uma entrada precisa corresponder para ser retornada de uma busca. O filtro de busca é uma combinação lógica de declarações de valores de atributos. Uma declara cão de valor de atributo testa igualdade, menor que ou igual e etc. do valor de um atributo.

attributes: Especifica os atributos que devem ser retornados das entradas que atenderem aos critérios de busca. Como uma entrada pode ter muitos atributos, isso permite ao usuário ver somente os atributos nos quais eles estão interessados.

Apêndice G

Gramática SearchResultEntry

```
SearchResultEntry ::= [APPLICATION 4] SEQUENCE { objectName      LDAPDN,
      attributes      PartialAttributeList }

  PartialAttributeList ::= SEQUENCE OF SEQUENCE {
                type    AttributeDescription,
                vals    SET OF AttributeValue }
```

Apêndice H

Gramática SearchResultReference

```
SearchResultReference ::= [APPLICATION 19] SEQUENCE OF LDAPURL
        -- at least one LDAPURL element must be present
```

Apêndice I

Gramática SearchResultDone

SearchResultDone ::= [APPLICATION 5] LDAPResult

Apêndice J

Gramática UnbindRequest

UnbindRequest ::= [APPLICATION 2] NULL

Apêndice K

Esquema LDAP para representa Listas invertidas em uma DIT

Autor: Péricles Oliveira

Data: 29 de Setembro de 2005

Mestrado em Informática Universidade Federal do Amazonas

Conteúdo do arquivo flexldap.schema

```
attributetype ( 1.3.6.1.4.1.4203.666.1.100 NAME 'uidKey'
        DESC 'proposal draft: identifier of  keyword'
        EQUALITY caseIgnoreMatch
        SUBSTR caseIgnoreSubstringsMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.15{32768})
attributetype ( 1.3.6.1.4.1.4203.666.1.101 NAME 'freqInDN'
        DESC 'proposal draft: frequency keyword in LDAP directory, specified in
        EQUALITY caseIgnoreMatch
        SUBSTR caseIgnoreSubstringsMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.15{32768})
attributetype ( 1.3.6.1.4.1.4203.666.1.102 NAME 'idf'
        DESC 'proposal draft: idf keyword to LDAP directory'
        EQUALITY caseIgnoreMatch
        SUBSTR caseIgnoreSubstringsMatch
```

```
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.15{32768})
attributetype ( 1.3.6.1.4.1.4203.666.1.103 NAME 'uidDN'
        DESC 'proposal draft: identifier od DN'
        EQUALITY caseIgnoreMatch
        SUBSTR caseIgnoreSubstringsMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.15{32768})
attributetype ( 1.3.6.1.4.1.4203.666.1.104 NAME 'descriptionDN'
        DESC 'proposal draft: description DN'
        EQUALITY caseIgnoreMatch
        SUBSTR caseIgnoreSubstringsMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.15{32768})

objectclass ( 1.3.6.1.4.1.4203.666.1.100 NAME 'invertedList'
        SUP top STRUCTURAL
        MUST uidKey
        MAY ( freqInDN \$ idf  )
        )
objectclass ( 1.3.6.1.4.1.4203.666.1.100 NAME 'documents'
        SUP top STRUCTURAL
        MUST uidDN
        MAY ( descriptionDN)
        )
```

Este esquema define dois objectClasses, sendo o primeiro chamado de *invertedList*, onde são definidos os atributos: *uidKey* que representa um termo encontrado no diretório LDAP, após o processo de indexação. A entrada composta por uidKey=[termo],ou=listinv,ou=flexldap,..., representa uma DN (*Distiguished Name*) que conterá as informações de freqüência deste termo em cada entrada no diretório LDAP, para isto foi utilizado o atributo *freqInDN* que recebe a quantidade de vezes que o termo aparece em uma determinada entrada. E por último o atributo *idf* que representa a importância do termo *uidKey* para coleção (diretório LDAP),

calculado conforme descrito na seção sobre o modelo vetorial.

O segundo *objectClass* foi chamado de *documents*, e possui os atributos *uidDN* e *descriptionDN*, sendo respectivamente o código seqüência (gerado pelo flexldap) de uma DN existente no diretório e a descrição desta DN (DN obtida da DIT indexada), que no caso representado pela própria DN encontrada no diretório LDAP indexado.