ProLDAP
API em Prolog para directórios LDAP

Pedro Patinho <Pedro.Patinho@alunos.uevora.pt>
Salvador Abreu <spa@di.uevora.pt>

Departamento de Informática
Universidade de Évora
Setembro de 2001

Resumo

Num Sistema de Informação são guardados diversos tipos de dados, com características diferentes. Há informação em constante mutação, e há informação que se mantém inalterada por largos períodos de tempo. É neste sentido que se torna necessária a utilização de vários tipos de sistemas de armazenamento para esses dados. O LDAP é um sistema distribuído e altamente optimizado para guardar informação que tende a permanecer inalterada ao longo do tempo e, por isso, tem vindo a ser utilizado cada vez mais pelas organizações para guardar informação deste tipo. A implementação de uma API para LDAP utilizando programação em lógica tem em vista a integração no ISCO [1, 3, 2], a linguagem para construção de Sistemas de Informação utilizada na Universidade de Évora. Com esta API passamos a poder utilizar o LDAP, de forma transparente, como repositório de informação.

1 Introdução

Os Sistemas de Informação tomam cada vez mais um papel decisivo no funcionamento das organizações. A Universidade de Évora tem levado a cabo a construção de um Sistema de Informação que abranja a maior diversidade de dados possível, de modo a permitir acesso informatizado a toda a informação disponível na Universidade. É óbvio que, para cada tipo de informação disponível, haverá sistemas mais ou menos adequados para a armazenar. O ISCO é uma linguagem que permite a construção de Sistemas de Informação Heterogénea, através da utilização de diferentes repositórios de informação, de acordo com o tipo de informação a guardar. Os tipos de repositório implementados ou previstos incluem bases de dados relacionais acedidas por ODBC ou nativamente, no caso do PostgreSQL, LDAP, DNS e SNMP, entre outros.

O objectivo principal deste trabalho foi desenvolver uma API flexível que permitisse o uso do LDAP no ISCO, como repositório de informação. O ISCO é implementado em Prolog, pelo que a linguagem utilizada para o desenvolvimento deste projecto foi o Prolog. A utilização do Prolog, para além de permitir uma relativa simplicidade de utilização, adiciona certos mecanismos não existentes noutras linguagens, como a unificação, o backtracking ou mesmo o facto de a linguagem ser interpretada (apesar de a implementação de Prolog utilizada permitir a compilação nativa), o que permite uma utilização do tipo "linha de comando" ou a aplicação de técnicas de meta-programação.

Para ser efectiva, a API deve incluir todas as funções relevantes da API original do LDAP, acrescida dos benefícios já referidos, inerentes a uma linguagem de programação

declarativa. Como qualquer API, destina-se a ser usada por programadores, mas é essencial que seja fácil de compreender e utilizar.

As funções essenciais a disponibilizar serão a inserção, modificação, remoção, actualização e procura de registos. Sendo o LDAP um sistema acessível através da rede, é necessário disponibilizar também as funções de estabelecimento de ligação ao servidor, autenticação e desconexão do servidor LDAP. Foram implementadas também outras funções que se revelaram necessárias para um aperfeiçoamento da API.

Numa fase futura estão ainda planeadas funções para a consulta da estrutura das classes LDAP (*schema*), de modo a permitir a geração das classes equivalentes no ISCO (o LDAP não permite a alteração *on-line* da estrutura da base de dados, pelo que não serão contempladas funções para esta finalidade). Esta restrição não é muito grave pois a perspectiva pretendida para a utilização por via de aplicações desenvolvidas em ISCO é a de ignorar o facto que certas relações estão implementadas usando directórios LDAP, sendo estas encaradas como simples predicados Prolog (com algumas restrições.)

2 Arquitectura

Antes de detalhar a API do *ProLDAP*, importa explicar brevemente qual a estrutura duma aplicação desenvolvida em ISCO: um programa ISCO consiste num conjunto de *definições de classes*, que poderão ser de várias categorias:

- **internas.** As instâncias destas classes correspondem a tuplos que irão ser representados externamente usando uma base de dados relacional particular para cada aplicação.
- **externas.** As classes externas têm os seus tuplos representados externamente, tal como as *internas*, com a diferença que o programador tem a possibilidade de especificar qual a fonte de informação a utilizar para a classe em particular.
- **calculadas.** As instâncias destas classes serão *geradas* de cada vez que se fizer uma interrogação à relação descrita pela classe. A semântica associada a estas classes é semelhante à dum predicado Prolog "normal". Classes calculadas podem ser encaradas, em termos de funcionalidade, como um sobreconjunto do conceito de *view* em SQL.

  Adicionalmente, as classes calculadas podem, por se tratar efectivamente de predicados Prolog, ter efeitos secundários tais como efectuar algum cálculo (daí o seu nome) ou ter efeitos secundários, como por exemplo produzir como output a representação XML ou HTML dum termo Prolog arbitrário.[1]

Esta uniformização de diversas formas de obter e armazenar informação usando uma sintaxe sistemática tem várias vantagens, pois permite programar aplicações sem ter de se preocupar com a forma como a informação é representada de forma persistente. Pode-se assim encarar o ISCO como um sistema de desenvolvimento de aplicações capaz de usar e semânticamente unificar bases de dados heterogéneas.

A inclusão dum "back-end" LDAP para as relações *externas* do ISCO é portanto um passo importante na direcção de viabilizar o uso generalizado dos dados contidos no Sistema de Informação Integrado da Universidade de Évora.

2.1 Ferramentas utilizadas

Nesta secção descrevem-se brevemente algumas das ferramentas ou bibliotecas sobre as quais assenta a implementação descrita neste artigo.

[1] Esta linha de ideias conduz a trabalho muito interessante no âmbito das linguagens de interrogação baseadas em XML, como o Semantic Web [**?**].

2.1.1 OpenLDAP

Um serviço de directório é uma base de dados especializada em leitura, pesquisa e navegação pelos registos. Os directórios tendem a conter informação descritiva baseada em atributos, e incluem capacidades avançadas de filtragem de registos.

O LDAP [7] é um protocolo que foi desenvolvido para colmatar as dificuldades inerentes ao funcionamento do X.500 sobre TCP/IP, tendo surgido inicialmente como uma gateway para X.500, mas que posteriormente ganhou uma implementação e servidores próprios. Dadas as suas baixas exigências em termos de capacidade de processamento, tem sido cada vez mais utilizado nas organizações, especialmente para guardar informação relacionada com os utilizadores (dados pessoais, passwords, preferências, etc.) ou recursos (edifícios, salas, computadores, impressoras, etc.). Os dados são guardados de forma hierárquica, de modo a existir a noção de "está contido em", isto é, por exemplo, uma determinada pessoa X trabalha no departamento Y que se situa no edifício W, que por sua vez está na cidade Z do país K. Podemos observar melhor a estrutura hierárquica do LDAP na figura 1. A informação poderá ser distribuída, de forma a que cada unidade dentro da organização possa conter a informação que lhe diz respeito (semelhante à distribuição utilizada nos servidores DNS).

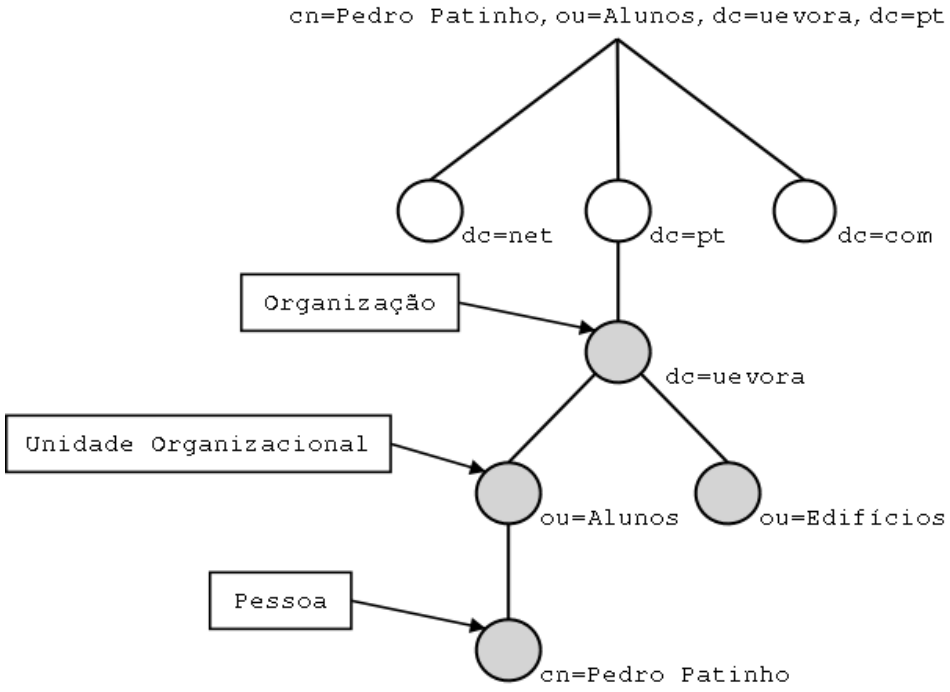

Figura 1: Representação hierárquica dos registos num servidor LDAP

Relativamente a outros tipos de base de dados, o LDAP apresenta algumas características vantajosas:

- A informação é guardada de forma hierárquica, permitindo a noção de *está contido em* ou *pertence a*.
- Possibilidade de atribuição de mais que um registo (valor) a cada campo (atributo);
- Possibilidade de inclusão de atributos não contemplados no *schema* (configurável pelo administrador);
- Listas de Controle de Acesso (ACLs) baseadas nos atributos e não nas tabelas (não existe a noção de tabela no LDAP);

- Cada registo (entrada) pode pertencer a várias classes (semelhante à herança múltipla).

O OpenLDAP [5] é uma implementação do protocolo LDAP que inclui um servidor próprio, uma API em C, e um servidor para replicação da base de dados remotamente. É um projecto levado a cabo por um grupo de voluntários, que recentemente tiveram o contributo da Universidade do Michigan, que os deixou a cargo do código fonte da sua implementação do LDAP, que já estava bastante estável e com suporte à versão 3 deste protocolo. Assim, o OpenLDAP, na sua versão 2.0, passou a suportar comunicação segura via SASL ou TLS (o Kerberos já era suportado anteriormente), bem como a exportação da estrutura da base de dados (*schema*), que não eram suportados anteriormente.

2.1.2 GNU Prolog

O GNU Prolog [4] é uma implementação robusta e eficiente da linguagem Prolog, com vários extras, como as restrições sobre domínios finitos, ou a compilação dos programas. Trabalhos em curso visam ainda a integração de Contextos no GNU Prolog, ou a execução distribuída (paralela) de *goals*.

A funcionalidade de maior relevo do GNU Prolog é a excelente interface para construção de predicados em linguagem C, que nos permitiu construir a API *ProLDAP* tendo como base a API em C fornecida pelo OpenLDAP.

2.2 Funcionamento e Formas de Utilização da API

Embora possa funcionar isoladamente, a API *ProLDAP* destina-se a funcionar como parte integrante do ISCO, de modo a permitir a utilização transparente de directórios LDAP enquanto repositórios de informação ISCO. De um modo geral, podemos considerar o *ProLDAP* como uma interface entre o ISCO e os servidores LDAP. Podemos observar o método de funcionamento do sistema na figura 2.

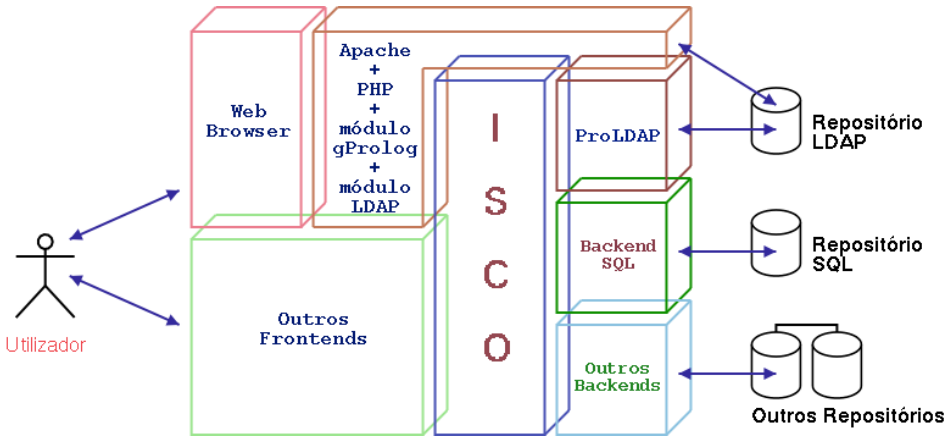

Figura 2: Utilização do *ProLDAP* pelo ISCO.

A nível da programação, esta ocorre em duas camadas: uma (alto nível) em Prolog e outra (mais baixo nível) em C. De uma forma mais precisa, cada predicado da API *ProLDAP* é implementado na linguagem C, utilizando funções da API nativa do OpenLDAP assim como da API de interface C do GNU Prolog.

Um programa em ISCO é compilado para Prolog, sendo as declarações de classes transformadas em vários predicados, um para cada operação pretendida (acesso em leitura, inserção, remoção, alteração). Da mesma forma que, no caso dum "back-end" ODBC ou PostgreSQL, é gerado código que constroi instruções SQL para interrogar uma base de

dados, no caso dum "back-end" LDAP serão gerados predicados que implementem as mesmas funcionalidades, usando desta feita a API aqui descrita. Posteriormente será dado um exemplo mais elucidativo.

3 Implementação da API

Como já foi referido anteriormente, a API conta com funções (predicados) de ligação, autenticação, e desconexão do servidor, inserção, remoção, modificação e pesquisa de registos. A seguir são descritas as várias funções que permitem cada um destes propósitos.

Cada função devolve o valor lógico verdade no caso de a operação ser efectuada com sucesso, e devolve o valor lógico falso e lança uma excepção, caso não seja possível executar a operação. Em caso de haverem várias soluções, foram usados mecanismos de não-determinismo, de modo a que as soluções sejam devolvidas em backtracking.

A notação utilizada será semelhante à utilizada no manual do GNU Prolog, ou seja, os argumentos de entrada serão prefixados com um `+`, os de saída com um `-`, e os de entrada/saída com um `?`.

3.1 Ligação

`l_init(+HOST, +PORT, -LD)`

Liga ao servidor `HOST` na porta `PORT`, e devolve um inteiro `LD`, que representa um índice de um vector onde são guardados apontadores para estruturas internas do LDAP. Com a utilização deste vector, é possível manter ligações a vários servidores LDAP em simultâneo.

São também disponibilizadas duas versões reduzidas desta função:

- `l_init(+HOST, -LD)`, que liga ao servidor `HOST` na porta 389 (porta predefinida para o LDAP);
- `l_init(-LD)`, que liga ao computador local (*localhost*) na porta 389.

Por exemplo:

```
% Liga ao servidor ‘‘ldap.uevora.pt’’ na porta 389 e
% devolve o identificador de ligação LD.

l_init('ldap.uevora.pt', LD).
```

3.2 Autenticação

`l_bind(+LD, +DN, +PASSWORD)`

Esta função permite a autenticação de um utilizador ao servidor LDAP. `LD` é um inteiro devolvido pelo `l_init`, `DN` é o *Distiguished Name*, a chave que identifica inequivocamente um registo LDAP, e `PASSWORD` é a password associada a esse `DN` (normalmente representa uma pessoa, mas poderá ser qualquer outra coisa contida no directório LDAP.

É ainda disponibilizada uma versão reduzida, `l_bind(+LD)`, que identifica o utilizador como anónimo.

Exemplo:

```
% Autentica-se na ligação identificada pelo inteiro LD
% (devolvido pelo l_init) com o DN e a PASSWORD especificados

l_bind(LD, 'cn=Pedro Patinho,ou=alunos,dc=uevora,dc=pt', 'abcd123').
```

3.3 Desconexão

`l_close(+LD)`

Utiliza o identificador de ligação `LD` devolvido pelo `l_init` para desligar do servidor LDAP.

Por exemplo,

```
% Encerra a ligação identificada pelo inteiro LD
% (devolvido pelo l_init)

l_close(LD).
```

3.4 Inserção de registos

`l_add(+LD, +DN, +DADOS)`

Adiciona registos ao directório LDAP. `LD` é o inteiro devolvido pela função `l_init`, `DN` é o *Distinguished Name* do registo a inserir, e `DADOS` é um termo em Prolog que contem os campos a inserir. Cada campo a inserir será um sub-termo da forma `nome_do_campo(valor1, valor2, ..., valorN)`.

```
% Introduz um registo no servidor

l_add(LD, 'cn=Pedro Patinho,ou=alunos,dc=uevora,dc=pt',
   dados( cn('Pedro Patinho'),
          objectclass(top, account, posixAccount,
                      shadowAccount, uevoraEntity,
                      uevoraPerson),
          uid('ppatinho'),
          password('xpto123'),
          morada('Rua do Álamo, 24'),
          curso('Engenharia Informática)
        )
).
```

3.5 Remoção de registos

`l_del(+LD, +DN)`

Esta função remove o registo identificado pelo *Distiguished Name* `DN`. `LD` é o identificador de ligação devolvido pela função `l_init`.

Exemplo:

```
% Remove o registo introduzido no exemplo anterior

l_del(LD, 'cn=Pedro Patinho,ou=alunos,dc=uevora,dc=pt').
```

3.6 Actualização ou Modificação de Registos

`l_modify_add(+LD, +DN, +DADOS)`

Este predicado adiciona um ou mais campos ao registo identificado pelo *Distiguished Name* `DN`. A sintaxe é igual à da função `l_add`.

`l_modify_del(+LD, +DN, +DADOS)`
: Remove um ou mais campos (ou valores dos campos) ao registo identificado pelo *Distiguished Name* `DN`. A sintaxe é igual à da função `l_add`. Para remover campos ou valores dos campos é necessário que se saiba os valores exactos a remover.

`l_modify_replace(+LD, +DN, +DADOS)`
: Modifica os valores de um ou mais campos ao registo identificado pelo *Distiguished Name* `DN`. A sintaxe é igual à da função `l_add`. Depois da execução desta função, os campos alterados terão unicamente os valores indicados em `DADOS`, por isso não é necessário conhecer os valores antigos dos campos.

Para além destes predicados, é ainda disponibilizada uma versão genérica que combina a funcionalidade dos três anteriores:

`l_modify(+LD, +DN, +DADOS, +TIPO)`
: Em que os três primeiros argumentos têm o mesmo significado que nas outras três funções, e `TIPO` é um átomo que especifica o tipo de modificação desejada, podendo tomar os seguintes valores:

  - *`a`* ou *`add`*, para executar uma adição de campos ou valores (como o `l_modify_add`);
  - *`d`* ou *`del`*, para executar uma remoção de campos ou valores (como o `l_modify_del`);
  - *`r`* ou *`replace`*, para executar a modificação de valores dos campos (como o `l_modify_replace`).

Adicionalmente, pode ser necessário alterar o *DN* dum determinado registo. Tal objectivo pode ser atingido pelo uso do predicado:

`l_modify_dn(+LD, +DN_VELHO, +DN_NOVO)`
: Modifica o *Distinguished Name* do registo identificado por `DN_VELHO`, para `DN_NOVO`.

O uso destes predicados pode ser ilustrado pelo seguinte exemplo:

```
% -- Adiciona o campo ‘‘idade’’, com o valor ‘‘23’’ ao registo
l_modify(LD, 'cn=Pedro Patinho,ou=alunos,dc=uevora,dc=pt',
         dados(idade(23)), add).

% -- Modifica o valor do campo ‘‘uid’’, para ‘‘pp’’
l_modify(LD, 'cn=Pedro Patinho,ou=alunos,dc=uevora,dc=pt',
         dados(uid('pp')), replace).
```

3.7 Procura de registos

A procura de registos é uma operação assíncrona (são efectuados passos separados para o envio do pedido e para a recepção dos resultados), por isso tem de ser feita através de diversas funções, que são geralmente chamadas em sequência, podendo no entanto haver execução de "goals" entre o que inicia a pesquisa no servidor LDAP e o que "recolhe" os resultados.

`l_search(+LD, +DN_BASE, +ESPACO_PESQISA, +FILTRO, +ATRIBUTOS, +ATTRS_ONLY, -MSG_ID)`

Activa a procura dos registos que estejam de acordo com o `FILTRO` especificado. `DN_BASE` é o *Distinguished Name* que identifica o nó da árvore LDAP onde a pesquisa deve começar, e `ESPACO_PESQUISA` é um inteiro que define o espaço da árvore a pesquisar. `ESPACO_PESQUISA` pode tomar um dos seguintes valores:

- `0`, para procurar apenas no nó `DN_BASE`;
- `1`, para pesquisar nos nós que são filhos directos do nó `DN_BASE`;
- `2`, para procurar em toda a sub-árvore abaixo do nó `DN_BASE`.

`FILTRO` é uma string normal que define as condições necessárias para que os registos sejam aceites (por exemplo, "mail=*" ou mesmo "uid=pp").

`ATRIBUTOS` é uma lista com os atributos (campos) a mostrar dos registos encontrados. Se a lista for vazia, mostra todos os atributos encontrados. `ATTRS_ONLY` é um inteiro que especifica se queremos receber só os nomes dos atributos ou também os valores (`1` mostra só os nomes, `0` mostra os valores também).

`MSG_ID` é um identificador interno utilizado pelas funções seguintes, que irão buscar os resultados da pesquisa.

`l_result(+LD, +MSG_ID, +ALL, +TIMEOUT, -RES_TYPE, -RESULT)`

Esta é a função que recebe efectivamente os registos pesquisados. `MSG_ID` é o identificador devolvido pela função `l_search`. O argumento `ALL` especifica se queremos receber todos os registos de uma só vez (`ALL = 1`) ou se queremos recebê-los um a um (`ALL = 0`). Neste último caso, teremos que chamar repetidas vezes a função `l_result` para receber todos os registos.

`TIMEOUT` especifica o tempo máximo a esperar pelo resultado da pesquisa. Caso o tempo seja excedido, será lançada uma excepção. `TIMEOUT` pode tomar uma das seguintes formas:

- Um termo da forma "`timeval(SEGs, MSEGs)`", em que `SEGs` é o nº de segundos a esperar e `MSEGs` é o nº de micro-segundos a esperar (somam-se os dois valores para obter o tempo total);
- O nº inteiro `0`, para especificar que não há limite de tempo, ou seja, a operação só termina quando receber todos os dados que está à espera.

`RES_TYPE` é um inteiro que especifica qual a função chamada antes do `l_result` (nesta fase de implementação será sempre a função `l_search`).

`RESULT` é um inteiro utilizado pelas funções seguintes, para obter os dados propriamente ditos.

`l_entry(+LDX, +RESULT, -ENTRY)`

Processa os dados recebidos pela função `l_result` associados ao identificador `RESULT` devolvido por esta. Devolve o identificador `ENTRY` da entrada (registo) que chegou primeiro. As entradas seguintes podem ser obtidas em backtracking.

`l_attribute(+LD, +ENTRY, -ATRIB)`

Para cada entrada devolvida pela função `l_entry`, lê os nomes dos atributos (pedidos na função `l_search`) um a um, em backtracking. O nome do atributo será unificado com o argumento `ATRIB`.

`l_values(+LD, +ENTRY, +ATRIB, -VALUES)`

Lê os valores do atributo `ATRIB` relativo à entrada `ENTRY`. Os valores são devolvidos na lista `VALUES`.

`l_value(+LD, +ENTRY, +ATRIB, -VALUE)`

Igual à função `l_values`, mas devolve apenas um valor do atributo `ATRIB`. Os outros valores são devolvidos em backtracking.

`l_dn(+LD, +ENTRY, -DN)`

Unifica o argumento `DN` com o *Distinguished Name* (uma string) da entrada `ENTRY`.

`l_count_entries(+LDX, +RESULT, -COUNT)`

Unifica o argumento `COUNT` com o número de entradas recebidas pela função `l_result`. `RESULT` é o identificador devolvido por essa função.

Na figura 3 está um exemplo de funcionamento de todas estas funções sem uso explícito de "backtracking". O predicado `exemplo/3` é não-determinístico (no sentido em que pode produzir várias soluções para o mesmo input), podendo gerar todas as soluções por "backtracking".

Caso sejam pretendidas as soluções não uma-a-uma, mas todas em conjunto, é trivial usar as técnicas de programação do Prolog para dar um goal como, por exemplo, `setof(V, exemplo(DN,A,V), Vs`. Neste caso teremos um novo goal não-determinístico que irá gerar soluções para as variáveis `A` e `Vs`, dando-lhe a interpretação de "nome do atributo" e "conjunto de valores que esse atributo toma".

```
exemplo(DN, ATTR, VALUES) :-
   l_init(LDX),   % liga ao localhost
   l_bind(LDX),   % identifica-se como anónimo
   l_search(LDX, 'dc=di,dc=uevora,dc=pt', 2, 'objectclass=*', [], 0, RES),
   % procura na sub-árvore começada no DN especificado, com o filtro
   % ''objectlass=*''. Pede para mostrar todos os atributos (lista vazia)
   % e recebe os nomes e valores dos atributos (0)
   l_result(LDX, RES, 1, 0, _, MSG_ID), % recebe todas as entradas de uma vez
   l_count_entries(LDX, MSG_ID, COUNT). % conta o nº de entradas recebidas
   l_entry(LDX, MSG_ID),  % processa a primeira entrada
   l_attribute(LDX, MSG_ID, ATTR),  % lê o primeiro atributo
   l_values(LDX, MSG_ID, ATTR, 0, VALUES),  % lê a lista de valores do atributo
   l_dn(LDX, MSG_ID, DN).  % lê o DN da entrada
```

Figura 3: Exemplo de utilização da API *ProLDAP*para pesquisa

4 Aplicação da API

Como já foi referido, o *ProLDAP*tem como objectivo principal a integração no ISCO. Como tal, existem vários tipos de informação para os quais é mais adequada a utilização do backend *ProLDAP*, devido às suas poucas ou nenhumas necessidades de actualização (relativamente a cada pessoa):

- Informação para autenticação de pessoas (passwords, impressões digitais, etc.);
- Dados relativos à distribuição geográfica dos edifícios da Universidade de Évora;
- Dados para localização de recursos dentro dos edifícios (computadores, impressoras, etc.), de modo automatizado;
- Preferências pessoais dos utilizadores de qualquer serviço on-line (por exemplo, notícias numa página web), de modo a disponibilizar em primeiro plano os materiais que são mais apreciados pelo utilizador.

Para além destas aplicações dentro do SIIUE e do ISCO, podemos ainda considerar a utilização isolada do *ProLDAP*. Com o desenvolvimento desta API, passou a ser possível criar aplicações em Prolog que comuniquem com um servidor LDAP. A programação utilizando o *ProLDAP*revela-se bastante mais simples do que com a API nativa em linguagem C, nomeadamente porque cria uma camada de abstracção que elimina a necessidade de nos preocuparmos com as estruturas internas do protocolo LDAP.

Supondo que pretendemos anular o "password" de todos os alunos que tiveram um resultado "anulado" numa disciplina qualquer leccionada pelo Departamento de Informática no ano lectivo 2000/01, seria possível produzir esse efeito com o predicado "`ripa/1`" visível na figura 4, em ISCO. Este predicado, quando invocado como "`ripa(ALUNO)`", sendo `ALUNO` uma variável livre, irá suceder repetidamente, instanciando essa variável com todos os valores para os quais as condições anteriormente referidas se verificarem.

```
ripa(A) :-
  departamento(sigla='di', id=DI),
  disciplina(id=C, departamento=DI),
  sac_aluno(número=A),
  sac_inscrito(aluno=A, cadeira=C, ano_lectivo=2000),
  sac_resultado(aluno=A, cadeira=C, ano_lectivo=2000, resultado=anulado),
  sac_login(número=A, login=L),
  posixAccount(uid=L) := userPassword='*'.
```

Figura 4: Exemplo de uso da API LDAP no ISCO

Aqui novamente, caso se pretenda fazer o mesmo para todos os alunos nestas condições, bastará executar o "goal" "`bagof(A, ripa(A), As)`", ficando na variável `As` a lista de alunos nessas condições.

Noutra linha, convém ilustrar algum código gerado pelo compilador de ISCO para uma definição de classe que tenha sido declarada "`external(ldap)`". Consideremos a definição da figura 5. Esta definição corresponde à definição habitual dum registo de utilizador

```
mutable, external(ldap) class posixAccount.
  cn: text.            key.
  uid: text.           not_null.
  uidNumber: int.      not_null.
  gidNumber: int.      not_null.
  homeDirectory: text. not_null.
  userPassword: text.
  loginShell: text.
  gecos: text.
  description: text.
```

Figura 5: Declaração em ISCO da classe `posixAccount`

com os campos Posix. A esta definição, o compilador ISCO irá fazer corresponder vários predicados, um para cada operação (consulta, inserção, remoção, alteração), dos quais ilustramos a estrutura do mais comum, a consulta, na figura 6 (ver página 11).

5 Conclusões e Trabalho Futuro

Apesar de a API estar completamente funcional, estão previstas algumas adições a curto e médio prazo:

1. Integração total com o ISCO, em particular a finalização do compilador na geração de código que faz uso do "back-end" LDAP.
2. Possibilidade de importação do esquema utilizado no servidor LDAP para o ISCO.
3. Possibilidade de alteração do esquema pelo ISCO, através de mecanismos externos, pois o protocolo LDAP não permite a alteração remota do seu esquema.

A implementação da API *ProLDAP* tem-se revelado robusta, tendo sido exercitada com sucesso em múltiplas situações.

```
posixAccount(CONN, A,B,C,D,E,F,G,H,I, WANT) :-
  <construção do filtro LDAP (var. FILTRO)>
  <construção da lista de atributos pretendidos (var. ATRIBs)>
  l_search(CONN, FILTER, 2, FILTRO, ATRIBs, 0, RES),
  l_result(CONN, RES, 1, 0, _, MSG),
  l_entry(CONN, MSG),           % backtrackável!
  l_value(CONN, MSG, cn, A),
  l_value(CONN, MSG, uid, B),
  ...
  l_value(CONN, MSG, description, I).
```

Figura 6: Código Prolog para consulta da classe `posixAccount`

No decorrer do desenvolvimento do projecto foram sempre testadas várias formas de chegar ao mesmo objectivo, tendo-se optado sempre pela forma que ao mesmo tempo parecia ser mais simples de utilizar, e menos complexa em termos computacionais.

A utilização de duas camadas de programação (Prolog sobre C) permite-nos modificar uma das camadas sem alterar a outra (por exemplo, se for necessário alterar alguma função na camada em C, podemos sempre manter o protótipo do predicado em Prolog), o que leva a uma grande facilidade de expansão da API sem modificar as partes que já estão feitas. Qualquer alteração no próprio protocolo LDAP não implica uma mudança na interface da API *ProLDAP*.

Referências

[1] Salvador Abreu. A Logic-based Information System. In Enrico Pontelli and Vitor Santos-Costa, editors, *2nd International Workshop on Practical Aspects of Declarative Languages (PADL'2000)*, volume 1753 of *Lecture Notes in Computer Science*, pages 141–153, Boston, MA, USA, January 2000. Springer-Verlag.

[2] Salvador Abreu. A practical language for heterogeneous information system construction. 14th International Conference on Applications of Prolog. Tokyo, Novembro 2001.

[3] Salvador Abreu and Joaquim Godinho. Logic-based Network Configuration and Management. In *The 7th International Congress of European University Information Systems*, Berlin, March 2001. Humboldt University.

[4] Daniel Diaz and Philippe Codognet. GNU Prolog: Beyond Compiling to C. In *2nd International Workshop on Practical Aspects of Declarative Languages (PADL'2000).*

[5] The OpenLDAP Project: *OpenLDAP 2.0 Administrator's Guide*, The OpenLDAP Foundation, Setembro de 2000

[6] H. Johner, L. Brown, F. Hinner, W. Reis, J. Westman: *Understanding LDAP*, International Technical Support Organization (IBM Redbook), Junho de 1998

[7] Vários autores: *RFC2251: "The Lightweight Directory Access Protocol (v3)"*, Network Working Group, Dezembro de 1997