Integração do Serviço de Diretório LDAP com o Serviço de Nomes CORBA*

Gustavo Scalco Isquierdo
scalco@ime.usp.br

Francisco C. R. Reverbel
reverbel@ime.usp.br

Departamento de Ciência da Computação
Instituto de Matemática e Estatística da Universidade de São Paulo

Abstract

This paper presents an approach for the integration of the LDAP directory service and the CORBA naming service. We describe a CORBA naming service implementation that stores the bindings between names and object references in an LDAP directory. The name–reference bindings become accessible to both CORBA clients (through the naming service interfaces) and LDAP clients (through the LDAP API). Descriptive attributes may be added to the directory entries that represent name–reference bindings. LDAP clients may use directory search facilities to obtain references whose entries meet certain conditions. Such conditions may involve the name bound to the reference or other attributes of the directory entries.

Keywords: LDAP, CORBA, naming service, interoperable systems, middleware

Resumo

Este artigo apresenta uma abordagem para a integração do serviço de diretório LDAP com o serviço de nomes CORBA. Descrevemos a implementação de um servidor de nomes CORBA que armazena num diretório LDAP as associações entre nomes e referências para objetos. As associações nome–referência ficam acessíveis tanto para clientes CORBA (através das interfaces do serviço de nomes) como para clientes LDAP (através da API do LDAP). Atributos descritivos podem ser adicionados às entradas do diretório que representam associações nome–referência. Clientes LDAP podem utilizar as facilidades de busca no diretório para obter referências cujas entradas satisfaçam determinadas condições. Essas condições podem envolver o nome associado à referência ou outros atributos presentes nas entradas do diretório.

Palavras chave: LDAP, CORBA, serviço de nomes, sistemas interoperáveis, *middleware*

1 Introdução

Um serviço de diretório mantém uma base de dados com informações sobre objetos e fornece interfaces para busca, inserção, remoção e atualização dessas informações. O LDAP (*Lightweight Directory Access Protocol*) [5, 6, 11] é um serviço de diretório derivado do X.500 [9] e padronizado pela IETF (*Internet Engineering Task Force*). Ele tem um conjunto de facilidades

* Trabalho desenvolvido com apoio financeiro parcial da FAPESP (Fundação de Amparo à Pesquisa do Estado de São Paulo), proc. no. 98/06138-2 (projeto SIDAM).

de busca bastante rico, que permite a realização de buscas complexas em todo o diretório ou em partes dele.

O serviço de nomes CORBA [2, 16] mapeia nomes em referências de objetos CORBA. Ele é freqüentemente comparado a uma lista telefônica, que associa nomes de assinantes a seus correspondentes números de telefone. Trata-se de um serviço bastante simples, que não oferece tantas facilidades de busca quanto um serviço de diretório. Para obtermos uma referência para um objeto CORBA é necessário que saibamos, de antemão, um nome completo do objeto. Temos de saber completa e exatamente um nome sob o qual o objeto foi registrado no serviço de nomes.

Com o objetivo de integrar esses dois serviços, construímos um servidor CORBA que implementa o serviço de nomes e emprega um diretório LDAP para armazenar, de forma persistente, as associações entre nomes e referências para objetos CORBA. Armazenadas num diretório LDAP, tais informações ficam acessíveis para uma gama mais ampla de clientes. Elas estão disponíveis não apenas para clientes no domínio do servidor de nomes CORBA, mas para todo cliente com acesso ao diretório LDAP. Clientes que só precisem de "busca por nome" poderão obter referências para objetos através da interface do serviço de nomes CORBA, simples e independente de linguagem de programação. Clientes interessados em facilidades de busca mais avançadas poderão utilizar diretamente a API mais complexa do LDAP.

Este artigo está organizado da seguinte maneira: a seção 2 sumariza conceitos sobre diretórios e LDAP, a seção 3 apresenta o serviço de nomes CORBA, a seção 4 descreve como integramos o LDAP e o serviço de nomes CORBA, a seção 5 discute trabalhos relacionados e, finalmente, a seção 6 traz nossas considerações de encerramento.

2 Diretórios e o Serviço LDAP

Um diretório pode ser definido como uma base de dados global, que pode estar fisicamente distribuída, mas mantém uma centralização lógica, cujo objetivo é armazenar e fornecer informações sobre objetos. Não há restrições quanto à natureza dessas informações. Embora sejam muito usados por aplicações interessadas em armazenar e recuperar informações sobre recursos de redes, os serviços de diretório não são de modo algum limitados ou especificamente destinados a essa área de aplicação.

Dois aspectos diferenciam os diretórios dos bancos de dados convencionais: o valor mais alto da razão consultas/alterações e a inexistência de transações atômicas. Os serviços de diretório se distinguem dos serviços de nomes por oferecerem mais recursos de busca (pesquisa) e recuperação de informações.

2.1 O Serviço de Diretório X.500

A arquitetura do X.500, representada na figura 1, prevê dois agentes: o *Directory User Agent* (DUA) e o *Directory Service Agent* (DSA). O primeiro é uma aplicação cliente, através da qual usuários (pessoas ou aplicações) manipulam as diversas entradas do diretório. O segundo é uma aplicação servidora que gerencia a base de dados do diretório e provê serviços de diretório aos clientes. Esses agentes são implementados como processos da camada de aplicação do modelo OSI, conforme a recomendação ITU X.511 [9]. A interação entre um DUA e um DSA é feita por meio de um protocolo da camada de aplicação do modelo OSI, protocolo esse denominado

DAP (*Directory Access Protocol*). De forma análoga, a interação entre dois DSAs é feita através do protocolo DSP (*Directory Service Protocol*).

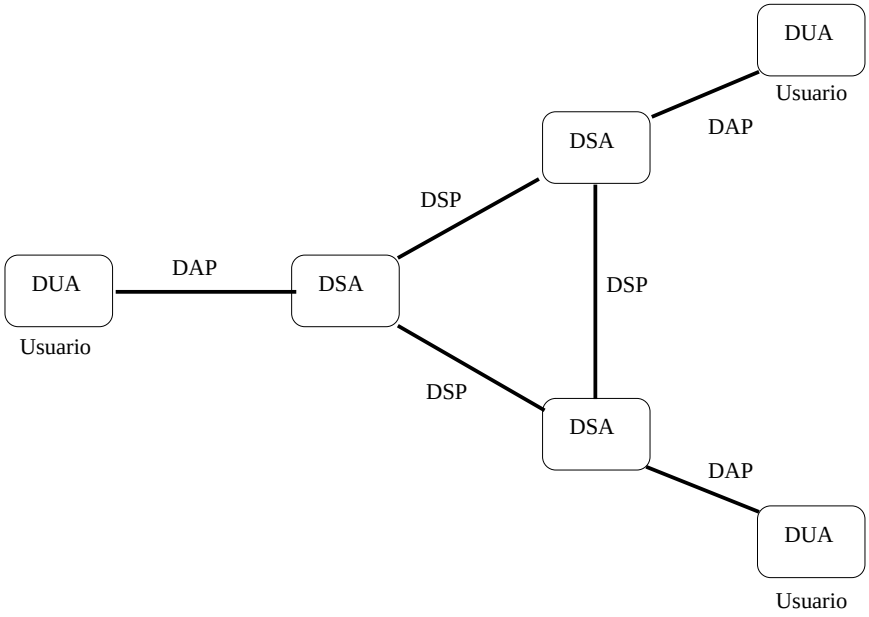

Figura 1: Os agentes do serviço de diretório X.500.

2.2 O Serviço de Diretório LDAP

Em 1988, quando foi publicada a especificação do o serviço de diretório X.500, o custo computacional desse serviço era incompatível com equipamentos menores. Era muito difícil rodar clientes X.500 nos microcomputadores disponíveis na época. O *Lightweight Directory Access Protocol* (LDAP), como o nome sugere, foi originalmente desenvolvido como um *front end* com menor custo computacional que o DAP do X.500 (figura 1).

Desenvolvida na Universidade de Michigan, a primeira versão do LDAP [23] já rodava sobre TCP/IP e não sobre a pilha de protocolos OSI. Através dela, clientes se comunicavam com um servidor *gateway*, que traduzia as solicitações recebidas via TCP/IP para as correspondentes operações X.500 e as enviava para um DSA, via protocolo DAP sobre a pilha OSI.

A escassez de implementações e a baixa aceitação do X.500 e da pilha de protocolos OSI motivou a construção de servidores LDAP *standalone*. Com a popularização de tais servidores, o LDAP deixou de ser uma mera alternativa ao DAP do X.500 e ganhou o status de serviço de diretório completo, passando a competir com o X.500. A versão 2 do protocolo LDAP foi definida nas RFCs 1777 [24], 1778 [3] e 1779 [10]. A API C para o LDAP foi definida na RFC 1823 [4]. A versão 3 do LDAP [21], aprovada pela IETF como proposta de padrão Internet para serviços de diretório, teve grande aceitação entre fornecedores e usuários de sistemas distribuídos. Uma definição de API Java [22] foi recentemente publicada como *Internet draft*.

2.2.1 O Modelo de Dados do LDAP

Todas os dados no diretório são armazenados em “entradas”, sendo que cada entrada pertence a pelo menos uma “*object class*”. Cada entrada tem uma coleção de atributos, que de fato

mantém as informações da entrada. A figura 2 representa uma entrada de um diretório. As *object classes* a que essa entrada pertence definem os atributos que ela pode ou deve conter. Esse modelo, herdado quase sem alteração do X.500, é extensível: definindo-se novas *object classes*, pode-se adicionar ao diretório qualquer tipo de informação.

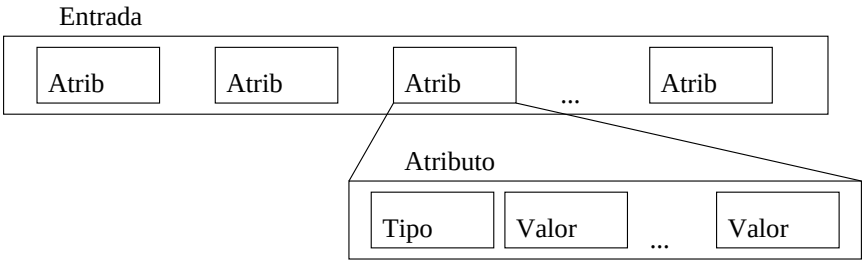

Figura 2: Uma entrada, seus atributos e valores.

Cada atributo tem um tipo e um ou mais valores. O tipo de um atributo determina os valores admissíveis para o atributo e define como esses valores são comparados, além de especificar se pode ou não haver mais de um valor para o atributo numa mesma entrada do diretório.

Toda entrada deve ter um atributo `objectClass`, que especifica as *object classes* a que a entrada pertence. A cada *object class* está associado um conjunto de atributos obrigatórios, que devem estar presentes em toda entrada que pertença a essa *object class*, e um conjunto de atributos opcionais, que podem ou não estar presentes numa entrada que pertença à *object class*.

Uma *object class* A pode ser subclasse de outra *object class* B. Neste caso A herda os atributos obrigatórios e os atributos opcionais de B, além de ter seus próprios atributos obrigatórios e opcionais.

2.2.2 O Modelo de Nomes do LDAP

O modelo de nomes especifica como as informações num diretório são organizadas e referenciadas. No caso do LDAP, esse modelo pressupõe um diretório organizado em árvore, com nomes hierárquicos. As entradas do diretório são os nós da árvore. Cada entrada tem um *relative distinguished name* (RDN), que é relativo ao seu nó pai, e um *distinguished name* (DN), que especifica um caminho da raiz até a entrada.

O RDN de uma entrada é formado por um ou mais atributos da entrada. Duas "entradas irmãs" não podem ter o mesmo RDN. O DN de uma entrada é a concatenação dos RDNs da seqüência de nós que começa na própria entrada e vai até uma entrada diretamente subordinada à raiz da árvore.

A figura 3 mostra uma parte de um diretório. Nessa figura os RDNs são formados por um só atributo, o atributo "`dc`" (*domain component*) da entrada. Ela mostra entradas com RDNs "`dc=edu`", "`dc=br`", "`dc=unicamp`" e "`dc=usp`". O DN "`dc=usp, dc=br`" denota a entrada no canto inferior direito. A figura 3 mostra também uma entrada *alias*, que referencia outra entrada. A existência de *aliases*, permitida pelo LDAP, quebra a organização hierárquica do espaço de nomes.

Assim como o modelo de dados, o modelo de nomes do LDAP deriva diretamente do X.500. Entretanto, o X.500 requer uma hierarquia de entradas definida segundo critérios de distribuição geográfica e organizacional, sendo padronizados os atributos que podem aparecer nos RDNs de entradas em vários níveis da hierarquia. Exemplificando: entradas diretamente subordinadas à

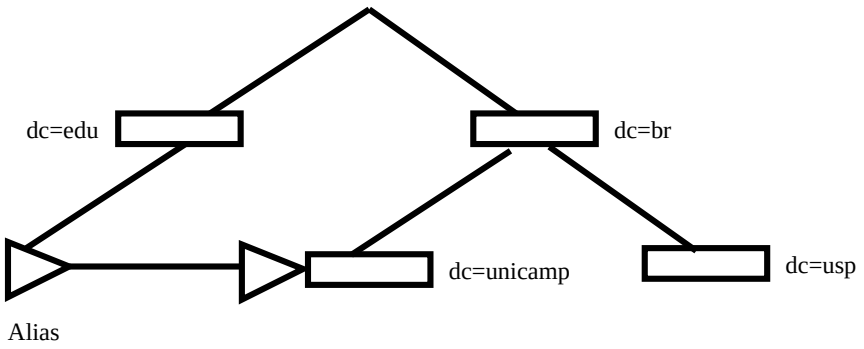

Figura 3: Organização das entradas de diretório usando uma estrutura em árvore.

raiz de um diretório X.500 têm RDN formado pelo atributo "`c`" (*country*). Já o LDAP permite maior flexibilidade, pois não impõe restrições sobre a hierarquia de entradas. Ela pode ser como a do X.500, pode ser um sistema de nomes completamente "achatado" (com todas as entradas diretamente subordinadas à raiz), ou pode ser algum outro sistema hierárquico.

2.2.3 O Esquema do LDAP

O LDAP é permite a definição de *object classes* e de tipos de atributos de acordo com as necessidades das aplicações ou usuários do serviço. Cada implementação do LDAP deve oferecer ao administrador do serviço de diretório mecanismos para definir *object classes* adequadas às necessidades específicas de sua organização. O conjunto de definições de *object classes* é a especificação do esquema do diretório. O processo de extensão do esquema envolve a atribuição de identificadores de objeto aos novos elementos do esquema e a escolha dos nomes desses elementos, bem como a criação de um arquivo de esquema local, com definições de novos tipos de atributos (se necessário) e definições de novas *object classes*. Essas definições devem ter o formato especificado em [20].

Cada *object class* ou tipo de atributo tem um identificador de objeto (*object identifier*, ou OID) globalmente único. Um OID é escrito como uma seqüência de números decimais separados por pontos e representa um caminho da raiz até um nó de uma árvore de identificadores. OIDs com o mesmo formato são empregados em muitos contextos, geralmente nos protocolos descritos por meio da ASN.1 [8]. Para adicionar seus próprios elementos ao esquema LDAP, uma organização deve solicitar que a *Internet Assigned Numbers Authority* (IANA) lhe atribua um OID. A organização poderá então alocar OIDs para seus próprios objetos, desde que tais OIDs tenham como prefixo o OID da organização.

Neste trabalho foi necessário adicionarmos certos elementos ao esquema LDAP. O OID atribuído pela IANA à nossa organização é `1.3.6.1.4.1.9242`. Tivemos portanto autonomia para alocar OIDs da forma `1.3.6.1.4.1.9242.1.1.1.`*x* para tipos de atributos e OIDs da forma `1.3.6.1.4.1.9242.1.1.2.`*x* para *object classes*. A tabela 1 mostra a estrutura de nossa sub-árvore de OIDs.

Além de ter um identificador globalmente único, cada elemento adicionado ao esquema deve ter um nome, um identificador textual mnemônico. O administrador de um serviço de diretório deve escolher nomes que não colidam com os nomes dos elementos padronizados do esquema e que tenham baixa probabilidade de colisão com nomes escolhidos em outras instalações. A convenção usual é empregar prefixos dependentes do nome da organização no DNS, isto é, na

OID	Utilização
1.3.6.1.4.1.9242	Instituto de Matemática e Estatística da USP
1.3.6.1.4.1.9242.1	atividades de pesquisa
1.3.6.1.4.1.9242.1.1	elementos do LDAP
1.3.6.1.4.1.9242.1.1.1	tipos de atributos
1.3.6.1.4.1.9242.1.1.1.*x*	um tipo de atributo
1.3.6.1.4.1.9242.1.1.2	object classes
1.3.6.1.4.1.9242.1.1.2.*x*	uma object class

Tabela 1: Estrutura de nossa sub-árvore de OIDs.

organização `ime.usp.br`, recomenda-se usar "`brUspImeUmNome`" em vez de um "`umNome`".

3 O Serviço de Nomes CORBA

O serviço de nomes CORBA [2, 16] é um dos *Object Services* da *Object Management Architecture* [2, 12, 18] definida pelo *Object Management Group* (OMG). Ele serviço mantém associações entre nomes e referências para objetos CORBA, permitindo que objetos tenham nomes significativos do ponto de vista de uma aplicação.

Uma associação nome–referência é denominada *name binding*. A mesma referência pode estar associada a vários nomes, mas cada nome identifica exatamente uma referência. Um contexto de nomes é um objeto CORBA que contém *name bindings*. Em outras palavras, cada contexto de nomes implementa uma tabela que mapeia nomes em referências para objetos. Um nome nessa tabela pode estar associado tanto a uma referência para um objeto definido por uma aplicação como a uma referência para outro contexto implementado pelo serviço de nomes. Isto significa que contextos podem formar uma hierarquia análoga à hierarquia de diretórios num sistema de arquivos: contextos correspondem a diretórios que podem conter arquivos (objetos de aplicação) ou outros diretórios (outros contextos).

Os *name bindings* correspondem aos arcos de um grafo dirigido denominado grafo de nomes, cujos nós representam contextos de nomes ou objetos de aplicação. Os arcos que saem de um contexto de nomes representam os *name bindings* nesse contexto. Para cada associação nome–referência temos um arco que é rotulado pelo nome e aponta para o objeto denotado pela referência. Contextos de nomes podem aparecer como nós internos ou como nós folha do grafo de nomes, mas objetos de aplicação só podem aparecer como nós folha.

Fixado um contexto inicial, pode-se percorrer o grafo de nomes até um dado nó (contexto ou objeto de aplicação) alcançável a partir do contexto inicial. A seqüência de nomes nos arcos percorridos forma um *pathname*, um nome composto que identifica univocamente o nó alcançado. Um nó pode ter múltiplos *pathnames*.

3.1 Interfaces do Serviço de Nomes

As interfaces do serviço de nomes, definidas em IDL [2, 15], estão no módulo `CosNaming`, parcialmente[1] apresentado na figura 4. Essa figura reproduz integralmente a principal interface

[1] As interfaces `BindingIterator` e `NamingContextExt` foram omitidas por falta de espaço.

```
module CosNaming {

    typedef string Istring;
    struct NameComponent {
        Istring id;
        Istring kind;
    };
    typedef sequence<NameComponent> Name;
    enum BindingType {
        nobject,
        ncontext
    };
    struct Binding {
        Name binding_name;
        BindingType binding_type;
    };
    typedef sequence<Binding> BindingList;
    interface BindingIterator; // forward declaration

    interface NamingContext {

        enum NotFoundReason { missing_node, not_context,  not_object };

        exception NotFound { NotFoundReason why; Name rest_of_name; };
        exception CannotProceed { NamingContext ctx; Name rest_of_name; };
        exception InvalidName {};
        exception AlreadyBound {};
        exception NotEmpty {};

        void bind(in Name n, in Object obj)
            raises(NotFound, CannotProceed, InvalidName, AlreadyBound);
        void rebind(in Name n, in Object obj)
            raises(NotFound, CannotProceed, InvalidName);
        void bind_context(in Name n, in NamingContext nc)
            raises(NotFound, CannotProceed, InvalidName, AlreadyBound);
        void rebind_context(in Name n, in NamingContext nc)
            raises(NotFound, CannotProceed, InvalidName);
        Object resolve(in Name n)
            raises(NotFound, CannotProceed, InvalidName);
        void unbind(in Name n)
            raises(NotFound, CannotProceed, InvalidName);
        NamingContext new_context();
        NamingContext bind_new_context(in Name n)
            raises(NotFound, AlreadyBound, CannotProceed, InvalidName);
        void destroy() raises(NotEmpty);
        void list(in unsigned long how_many,
                  out BindingList bl, out BindingIterator bi);

    };

    ...

};
```

Figura 4: Interfaces do serviço de nomes.

do serviço de nomes, a interface `NamingContext`. Com relação aos elementos do módulo `CosNaming`, os seguintes pontos devem ser observados:

- Um `Name` é um nome composto, um caminho que identifica um objeto a partir de um contexto inicial. Um `Name` é uma seqüência de `NameComponents`.
- Cada `NameComponent` consiste num par de cadeias de caracteres: um `id` e um `kind`. O campo `id` é considerado a "parte principal" do `NameComponent`. O propósito original do campo `kind` era descrever de algum modo o objeto associado ao `NameComponent`. Hoje, entretanto, muitos consideram que a distinção entre `id` e `kind` foi um erro no projeto do serviço de nomes e recomendam que sempre se coloque a cadeia vazia no campo `kind`.
- Cada contexto de nomes guarda somente associações entre `NameComponents` e referências para objetos CORBA. Já um nome composto (`Name`) não é armazenado num único contexto de nomes, pois ele identifica um caminho de um contexto inicial até um objeto. Embora exista uma associação entre o nome composto e esse objeto, as informações sobre essa associação ficam distribuídas ao longo dos contextos do caminho.

3.2 Resolução de Nomes

A maioria das operações da interface `NamingContext` recebe um `Name` como parâmetro. Nessas operações o `Name` é interpretado no contexto alvo da chamada, isto é, tomando-se como ponto de partida o `NamingContext` sobre o qual a operação foi invocada.

A interpretação de um `Name` num dado contexto é denominada resolução do nome. O serviço de nomes busca nesse contexto uma associação que envolva o primeiro `NameComponent` do nome. Se existir tal associação, o `NameComponent` estará ligado a uma referência para outro contexto ou para um objeto de aplicação. Caso o nome tenha outros `NameComponents`, a referência associada ao primeiro componente deve apontar para outro contexto, no qual o serviço de nomes buscará uma associação envolvendo o segundo componente do nome, e assim sucessivamente. O processo de resolução terminará quando o último componente do `Name` for resolvido e produzirá como resultado a referência associada ao último componente.

Para uma operação op que é invocada sobre um contexto de nomes ctx e usa um `Name` com componentes c_1, c_2, ..., c_n, a resolução do nome é definida recursivamente por

$$\mathsf{ctx} \rightarrow \mathsf{op}(c_1/c_2/\ldots/c_n,\ldots) \;\equiv\; \mathsf{ctx} \rightarrow \mathsf{resolve}(c_1) \rightarrow \mathsf{op}(c_2/\ldots/c_n,\ldots)$$

A operação op pode ser qualquer uma das operações da interface `NamingContext` que recebe um `Name` como parâmetro de entrada: `bind`, `rebind`, `resolve`, `unbind`, etc.

Um contexto de nomes pode conter (e tipicamente contém) associações envolvendo objetos CORBA implementados por outros servidores. Em particular, um `NamingContext` residente em um dado servidor de nomes pode conter referências para `NamingContexts` residentes em outros servidores de nomes. No caso geral, a resolução de nomes é um processo distribuído, pois os contextos percorridos para se resolver os componentes de um nome podem estar distribuídos por vários servidores.

4 Integração do LDAP com o Serviço de Nomes CORBA

Implementações do serviço de nomes CORBA precisam armazenar, de modo persistente, as associações nome–referência existentes nos contextos de nomes. Podem, por exemplo, guardar essas associações em arquivos do sistema operacional subjacente. Ou usar um sistema gerenciador de bancos de dados para esse propósito.

Construímos uma implementação do serviço de nomes CORBA que armazena as associações nome–referência num diretório LDAP. Nosso servidor de nomes é portanto um cliente do serviço de diretório LDAP. Esta abordagem foi motivada pelas seguintes considerações:

Eficiência. Um diretório LDAP é um tipo de banco de dados especialmente otimizado para consultas. As associações nome–referência num `NamingContext` CORBA são utilizadas principalmente por uma operação de consulta, a resolução de nomes.

Flexibilidade. As associações nome–referência ficarão acessíveis tanto para clientes CORBA como para clientes LDAP. Clientes CORBA usarão as interfaces do serviço de nomes. Clientes LDAP poderão inserir mais atributos nas entradas do diretório que representam associações nome–referência. Poderão também utilizar facilidades de busca em diretórios para obter referências cujas entradas satisfaçam determinadas condições. Essas condições podem ser expressas em termos do `NameComponent` ao qual a referência está associada, ou em termos de atributos adicionais presentes nas entradas do diretório.

A arquitetura utilizada é de três camadas (*three-tier*): uma aplicação cliente CORBA utilizará o serviço de nomes através das interfaces definidas pelo OMG. A comunicação entre o cliente CORBA e o servidor de nomes será feita através do protocolo empregado no ambiente CORBA, tipicamente o IIOP. O servidor de nomes mantém as associações nome–referência num diretório LDAP, e interage com o diretório através da API do LDAP. A comunicação entre o servidor de nomes CORBA e o servidor de diretório é feita através do protocolo LDAP. A figura 5 ilustra essa arquitetura.

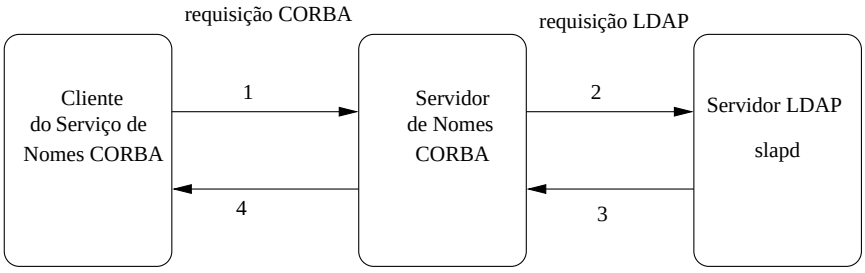

Figura 5: As três camadas da arquitetura de integração LDAP-CORBA.

As subseções seguintes discutem os seguintes aspectos da implementação de nosso servidor de nomes:

1. A definição do formato das entradas do diretório LDAP que conterão informações sobre contextos de nomes e associações nome–referência.
2. A modularização do código do servidor, de modo que boa parte dele seja independente do serviço de armazenamento persistente no qual são mantidos os contextos de nomes e as associações nome–referência. Com isso uma parte significativa do código do servidor

não dependerá do LDAP. Essa parte do código poderá ser reutilizada para se implementar um servidor de nomes que mantém num banco de dados os contextos e as associações nome–referência.

3. A utilização de um esquema de *caching* de modo a evitar que toda chamada ao serviço de nomes ocasione uma requisição ao servidor usado para armazenamento persistente de informações, no caso o LDAP.

4. A efetiva utilização, por parte do servidor de nomes, do *Portable Object Adapter* (POA), o elemento de CORBA que é responsável pela criação e pela interpretação de referências para objetos [2, 15]. O servidor deve empregar um POA com as políticas adequadas, de modo que as referências CORBA para `NamingContexts` sejam persistentes.

4.1 *Object Classes* para Contextos e Bindings

Definimos *object classes* específicas para as entradas do diretório que contêm contextos de nomes e associações nome–referência. A figura 6 mostra a definição da primeira dessas *object classes*, que denominamos `corbaCosContext`[2]. Esta *object class* especifica o formato das entradas do diretório correspondentes aos `NamingContexts` do serviço de nomes CORBA.

```
objectclass ( 1.3.6.1.4.1.9242.1.1.2.1
        NAME 'corbaCosContext'
        DESC 'LDAP entry for a CORBA naming context'
        SUP top
        STRUCTURAL
        MUST ( corbaId $ corbaBase ) )
```

Figura 6: A *object class* `corbaCosContext`.

Uma entrada com esse formato é mantida para cada contexto de nomes armazenado no diretório. Ela tem a função de registrar a existência de um dado contexto de nomes e de associar um identificador a esse contexto. Por ser derivada da *object class* `top`[3], `corbaCosContext` herda de `top` o atributo obrigatório `objectClass`. Além deste, um `corbaCosContext` tem somente dois atributos, também obrigatórios. O primeiro é um `corbaId`, uma cadeia de caracteres numéricos que distingue este contexto dos demais contextos implementados pelo mesmo servidor de nomes. O segundo é o atributo `corbaBase`, uma cadeia de caracteres que identifica o servidor de nomes que implementa este contexto. A função deste atributo é possibilitar que mais de um servidor de nomes CORBA utilize o mesmo servidor LDAP para armazenamento de associações nome–referência. A figura 7 ilustra o caso em que vários servidores de nomes acessam o mesmo diretório LDAP. O atributo `corbaBase` distingue as entradas dos diferentes servidores nos diretório.

Uma entrada com *object class* `corbaCosContext` é adicionada ao diretório quando se cria um novo `NamingContext` e é apagada quando este é destruído. A figura 8 apresenta as definições dos atributos desta *object class*.

[2] O verdadeiro nome dessa *object class* é `brUspImeCorbaCosContext`, escolhido conforme a convenção apresentada ao final da seção 2.2.3. Em benefício da legibilidade omitimos o prefixo `brUspIme` na descrição deste e dos demais elementos que adicionamos ao esquema do LDAP.

[3] A *object class* `top` é definida pela RFC 2256 [19] e tem somente o atributo obrigatório `objectClass`.

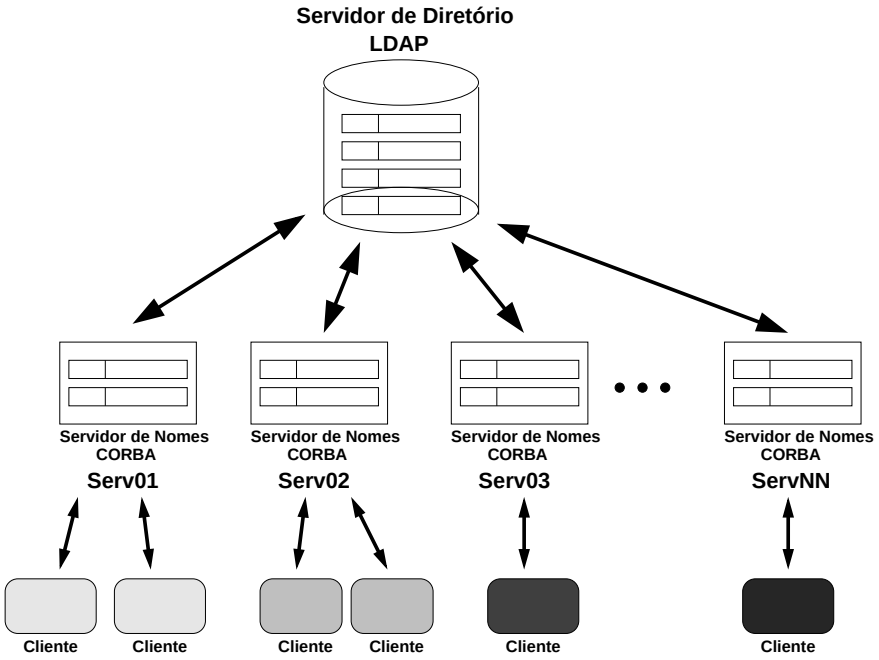

Figura 7: Vários servidores de nomes CORBA compartilhando um diretório LDAP.

```
attributetype ( 1.3.6.1.4.1.9242.1.1.1.1
        NAME 'corbaId'
        DESC 'Identifies a CORBA naming context'
        EQUALITY numericStringMatch
        ORDERING caseIgnoreOrderingMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.36{4} )

attributetype ( 1.3.6.1.4.1.9242.1.1.1.2
        NAME 'corbaBase'
        DESC 'Identifies a CORBA naming server'
        EQUALITY caseExactMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 )
```

Figura 8: Definições de atributos para a *object class* `corbaCosContext`.

```
objectclass ( 1.3.6.1.4.1.9242.1.1.2.2
        NAME 'corbaCosBinding'
        DESC 'LDAP entry for a CORBA name-objref binding'
        SUP top
        STRUCTURAL
        MUST ( corbaName $ corbaKind $ corbaIor $
               corbaType $ corbaCtxId $ corbaBase )
        MAY  ( cn $ description ) )
```

Figura 9: A *object class* `corbaCosBinding`.

A figura 9 mostra a definição da segunda *object class* que adicionamos ao esquema do LDAP. Essa *object class*, denominada `corbaCosBinding`, especifica o formato das entradas com associações nome–referência. Ela usa as definições de atributos da figura 10.

```
attributetype ( 1.3.6.1.4.1.9242.1.1.1.3
        NAME 'corbaName'
        DESC 'The id field of a CORBA name component'
        EQUALITY caseExactMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 )

attributetype ( 1.3.6.1.4.1.9242.1.1.1.4
        NAME 'corbaKind'
        DESC 'The kind field of a CORBA name component'
        EQUALITY caseExactMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 )

attributetype ( 1.3.6.1.4.1.9242.1.1.1.5
        NAME 'corbaIor'
        DESC 'The IOR bound to the name'
        EQUALITY caseIgnoreIA5Match
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.26 )

attributetype ( 1.3.6.1.4.1.9242.1.1.1.6
        NAME 'corbaType'
        DESC 'Tells if the IOR refers to a naming context or to an app object'
        EQUALITY booleanMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.7 )

attributetype ( 1.3.6.1.4.1.9242.1.1.1.7
        NAME 'corbaCtxId'
        DESC 'Identifies the naming context a binding belongs to'
        EQUALITY numericStringMatch
        ORDERING caseIgnoreOrderingMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.36{4} )
```

Figura 10: Definições de atributos para a *object class* `corbaCosBinding`.

Os atributos obrigatórios `corbaName` e `corbaKind` são cadeias de caracteres que armazenam um `NameComponent` do serviço de nomes. Eles correspondem aos campos `id` e `kind` do `NameComponent`. O atributo obrigatório `corbaIor` é uma referência CORBA para o objeto (contexto ou objeto de aplicação) associado ao `NameComponent`, convertida para cadeia de caracteres. O atributo `corbaType`, também obrigatório, é um valor booleano que indica se `corbaIor` se refere a um contexto ou objeto de aplicação. O atributo obrigatório `corbaCtxId` tem a função de identificar o contexto de nomes ao qual esta associação nome–referência se vincula. Ele contém o valor do atributo `corbaId` de uma entrada `corbaCosContext`. O último atributo obrigatório, `corbaBase`, tem a mesma função de seu homônimo na *object class* `corbaCosContext`: ele identifica o servidor de nomes responsável por esta associação nome–referência.

Como atributos opcionais da *object class* `corbaCosBinding`, temos `cn` (*common name*) e `description`, ambos com o formato de cadeia de caracteres. No *common name* pode-se armazenar um nome adicional para o objeto CORBA referenciado pela entrada `corbaCosBinding`. O `description` pode conter uma breve descrição desse objeto, um comentário sobre a sua funcionalidade, versão, etc. Outros atributos opcionais poderão ser adicionados se necessário.

A terceira e última *object class* que definimos descreve o formato de uma entrada especial,

cujo único propósito é armazenar, de modo persistente, o valor de um contador. O servidor de nomes usa esse contador para gerar o próximo identificador de contexto, ou seja, o campo `corbaId` da próxima entrada `corbaCosContext` que ele adicionará ao diretório. A figura 11 apresenta a definição dessa *object class*, que denominamos `corbaNextCtxId`.

```
objectclass ( 1.3.6.1.4.1.9242.1.1.2.3
        NAME 'corbaNextCtxId'
        DESC 'corbaId of next corbaCosContext entry for CORBA server
              corbaBase'
        SUP top
        STRUCTURAL
        MUST ( corbaNextId $ corbaBase ) )
```

Figura 11: A *object class* `corbaNextCtxId`.

Como anteriormente, `corbaBase` é um atributo obrigatório que especifica um certo servidor de nomes. O atributo obrigatório `corbaNextId` é o próximo identificador de contexto que será utilizado por esse servidor de nomes. A figura 12 mostra a definição desse atributo.

```
attributetype ( 1.3.6.1.4.1.9242.1.1.1.8
        NAME 'corbaNextId'
        DESC 'corbaId for next corbaCosContext entry'
        EQUALITY numericStringMatch
        ORDERING caseIgnoreOrderingMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.36{4} )
```

Figura 12: Atributo para a *object class* `corbaNextCtxId`.

É interessante notar que o esquema de *object classes* que definimos é análogo a um esquema relacional. A figura 13 mostra um esquema relacional que poderia ser utilizado para armazenar contextos de nomes e *bindings* num banco de dados.

```
table corbaCosContext( corbaId, corbaBase )

table corbaCosBinding ( corbaName, corbaKind, corbaIor,
                        corbaType, corbaCtxId, corbaBase, cn, description )

table corbaNextCtxId ( corbaNextId, corbaBase )
```

Figura 13: Esquema relacional para armazenamento de contextos e *bindings*.

Todas as tabelas deste esquema têm chaves primárias compostas. A primeira dessas tabelas, `corbaCosContext`, tem chave primária formada por suas duas colunas. O mesmo ocorre com a terceira tabela, `corbaNextCtxId`. A chave primária da tabela `corbaCosContext` é constituída pelas colunas `corbaName`, `corbaKind`, `corbaCtxId` e `corbaBase`. Estas duas últimas colunas formam uma chave estrangeira, que referencia a tabela `corbaCosContext`.

4.2 O Módulo de Acesso ao Repositório Persistente

Este módulo oferece uma interface de acesso a um "repositório abstrato" que contém contextos e associações nome–referência. Ele oculta do restante do servidor de nomes a API e os detalhes do serviço de armazenamento persistente empregado. Esta abordagem tem duas vantagens:

1. Ela permite que o restante do código do servidor de nomes seja independente do LDAP e possa ser reutilizado para implementar servidores de nomes baseados em outros serviços de armazenamento.
2. Ela evita que a complexidade e as peculiaridades da API do LDAP permeiem todo o código do servidor de nomes.

Todas as funções de acesso ao repositório abstrato têm o prefixo "`repository_`" em seus nomes. A relação dessas funções é apresentada abaixo.

`repository_init()`: Inicializa a biblioteca de acesso ao serviço de armazenamento subjacente, lendo de um arquivo de configuração os parâmetros que forem necessários.

`repository_connect()`: Abre uma conexão com o serviço de armazenamento subjacente.

`repository_disconnect()`: Encerra a conexão com o serviço de armazenamento subjacente.

`repository_cosBinding_insert()`: Insere no repositório uma associação nome–referência vinculada a dado contexto.

`repository_cosBinding_lookup()`: Procura pelo nome um objeto vinculado a um dado contexto, retornando uma referência para esse objeto e o seu tipo (objeto de aplicação ou contexto).

`repository_cosBinding_remove()`: Remove do repositório uma associação nome–referência vinculada a um dado contexto.

`repository_cosContext_insert_new()`: Cria um novo contexto e o insere no repositório.

`repository_cosContext_insert()`: Insere no repositório um dado contexto.

`repository_cosContext_lookup()`: Procura por um contexto no repositório.

`repository_cosContext_remove()`: Remove do repositório um dado contexto.

`repository_new_id_context()`: Cria um novo identificador de contexto.

`repository_list()`: Lista todos os objetos e contextos vinculados a um um contexto especificado.

4.3 O Esquema de *Caching* no Servidor de Nomes

Para evitar que toda chamada ao serviço de nomes ocasione outra para o repositório abstrato e, conseqüentemente, uma requisição ao servidor LDAP, implementamos um esquema simples de *caching*.Empregamos duas tabelas de *hash* em memória, denominadas `the_context_table` e `the_name_table`. Na primeira são mantidas as últimas m entradas `corbaCosContext` acessadas pelo servidor de nomes, onde m é um parâmetro de configuração do servidor. Na segunda tabela de *hash* são mantidas as últimas n entradas `corbaCosBinding` acessadas pelo servidor de nomes, onde n é outro parâmetro de configuração do servidor de nomes.

Todos os acessos ao repositório são mediados pelas funções que mantém o *cache*. O conjunto dessas funções forma uma camada que envolve o repositório e oferece ao restante do servidor de nomes uma API semelhante à do repositório. A camada de gerenciamento do *cache* oferece, por exemplo, uma função que procura um *binding* pelo nome. Essa função faz primeiro uma busca na tabela `the_name_table` e só chama `repository_cosBinding_lookup()` se

o *binding* procurado não estiver em memória. Caso o *binding* não esteja em memória e seja encontrado no repositório, ele é adicionado a `the_name_table`.

4.4 Utilização do POA pelo Servidor de Nomes

Para os objetos `NamingContext`, o servidor de nomes usa um POA com as seguintes políticas: `PERSISTENT`, `USER_ID`, `NON_RETAIN`, `USE_DEFAULT_SERVANT` e `MULTIPLE_ID`.

- A política `PERSISTENT` foi selecionada para que as referências `NamingContext` permaneçam válidas ao longo de sucessivas execuções de um servidor.
- Com a política `USER_ID`, o servidor pode determinar o campo *object id* das referências para `NamingContexts`. O servidor usa esse recurso para embutir nessas referências o atributo `corbaId` de uma entrada `corbaCosContext` no diretório.
- As políticas `NON_RETAIN` e `USE_DEFAULT_SERVANT` fazem com que um servente *default* trate todas as requisições para `NamingContexts`. Essa escolha é motivada por razões de escalabilidade e simplicidade de implementação. Para saber qual `NamingContext` encarnar a cada requisição, o servente *default* inspeciona o *object id* alvo da requisição.
- A política `MULTIPLE_ID` foi selecionada porque o servente default tratará requisições para vários `NamingContexts`, com diferentes *object ids*.

4.5 *Software* Utilizado na Implementação

Nosso protótipo foi implementado em C, com o compilador gcc, versão egcs-1.1.2. Empregamos o OpenLDAP 2.0.7, derivado do LDAP 3.3 da Universidade de Michigan, e o ORB ORBit 0.5.1. O desenvolvimento e os testes foram feitos sob o sistema operacional Linux, tendo sido utilizadas as distribuições Red Hat 6.2 e Debian 2.2, bem como várias versões de kernel 2.2.x (de 2.2.14 a 2.2.18).

5 Trabalhos Relacionados

O *Trading Object Service* [14] padronizado pelo OMG, freqüentemente comparado a uma lista telefônica de "páginas amarelas", oferece recursos avançados para clientes CORBA buscarem referências para objetos, especificando propriedades dos objetos desejados. Nosso trabalho permite que as facilidades de busca do LDAP sejam empregadas para se obter referências para objetos CORBA. Embora essas facilidades fiquem acessíveis a clientes LDAP, elas não são disponibilizadas a clientes CORBA, por meio de interfaces IDL.

Em [1], Haase *et. al.* apresentam um serviço de diretório baseado em CORBA. Os autores definiram uma interface IDL para acesso ao diretório, interface esta similar à API Java para o LDAP [22]. Este trabalho se assemelha ao nosso por empregar uma arquitetura *three-tier*, com um servidor CORBA na camada intermediária e um servidor LDAP na camada mais distante dos clientes. Ele se distingue do nosso pelas interfaces oferecidas pelo servidor na camada intermediária. No trabalho de Haase *et. al.*, a camada intermediária procura disponibilizar, para clientes CORBA, a funcionalidade da API do LDAP. Em nosso trabalho, essa camada implementa as interfaces do serviço de nomes padronizado pelo OMG.

6 Considerações Finais

Este artigo apresentou uma abordagem para a integração do serviço de diretório LDAP com o serviço de nomes CORBA. Implementamos um servidor de nomes CORBA que armazena num diretório LDAP as associações entre nomes e referências para objetos. Esta solução foi motivada por considerações de eficiência e flexibilidade. Ela permite, por exemplo, que um diretório global seja usado para facilitar a construção e a administração de federações de servidores de nomes.

Referências

[1] O. Haase, A. Schrader, K. Geihs, and R. Jans. Mobility support with CORBA directories. In T. Znati and R. Simon, editors, *Communication Networks and Distributed Systems Modelling and Simulation*, pages 20–29. The Society for Computer Simulation International, January 2000.

[2] M. Henning and S. Vinoski. *Advanced CORBA Programming with C++*. Addison Wesley Longman Inc., January 1999.

[3] T. Howes, S. Kille, W. Yeong, and C. Robbins. *The String Representation of Standard Attribute Syntaxes*. RFC 1778, IETF, March 1995.

[4] T. Howes and M. Smith. *The LDAP Application Programming Interface*. RFC 1823, IETF, August 1995.

[5] T. Howes and M. Smith. *LDAP Programming Directory — Enabled Applications with Lightweight Directory Access Protocol*. Macmillan Technical Publishing, Indianapolis, IN, 1997.

[6] T. A. Howes, M. C. Smith, and G. S. Good. *Understanding and Deploying LDAP Directory Services*. Macmillan Technical Publishing, 1999.

[7] G. S. Isquierdo. Integração do Serviço de Diretório LDAP com o Serviço de Nomes CORBA. Dissertação de Mestrado, Instituto de Matemática e Estatística da Universidade de São Paulo, Outubro 2001.

[8] ITU-T. *Abstract Syntax Notation One (ASN.1) — Specification of Basic Notastion — Recommendation X.680*. International Telecommunication Union, 1993.

[9] ITU-T. *Information Technology — Open Systems Interconnection — The Directory — Recommendations X.500–X.521*. International Telecommunication Union, 1993.

[10] S. Kille. *A String Representation of Distinguished Names*. RFC 1779, IETF, March 1995.

[11] P. Loshin. *Big Book of Lightweight Directory Access Protocol LDAP RFCs*. Morgan Kaufmann, 2000.

[12] OMG. *Discussion of the Object Management Architecture*. OMG, January 1997. OMG document formal/00-06-41.

[13] OMG. *C Language Mapping Specification*. OMG, June 1999. OMG document formal/99-07-35.

[14] OMG. *Trading Object Service Specification*. OMG, May 2000. OMG document formal/00-06-27.

[15] OMG. *The Common Object Request Broker: Architecture and Specification — Revision 2.6*. OMG, December 2001. OMG document formal/01-12-35.

[16] OMG. *Naming Service Specification*. OMG, February 2001. OMG document formal/01-02-65.

[17] OpenLDAP Foundation. OpenLDAP. `http://www.openldap.org`.

[18] S. Vinoski. CORBA: Integrating Diverse Applications Within Heterogeneous Environments. *IEEE Communications*, 14(2), February 1997.

[19] M. Wahl. *A Summary of the X.500(96) User Schema for use with LDAPv3*. RFC 2256, IETF, December 1997.

[20] M. Wahl, A. Coulbeck, T. Howes, and S. Kille. *Lightweight Directory Access Protocol (v3): Attribute Syntax Definitions*. RFC 2252, IETF, December 1997.

[21] M. Wahl, T. Howes, and S. Kille. *Lightweight Directory Access Protocol (v3)*. RFC 2251, IETF, December 1997.

[22] R. Weltman, C. Tomlinson, J. Sermersheim, M. Smith, and T. Howes. *The Java LDAP Application Program Interface*. Internet draft, IETF, March 2000.

[23] W. Yeong, T. Howes, and S. Kille. *X.500 Lightweight Directory Access Protocol*. RFC 1487, IETF, July 1993.

[24] W. Yeong, T. Howes, and S. Kille. *Lightweight Directory Access Protocol*. RFC 1777, IETF, March 1995.