Um Serviço CORBA para Descoberta de Componentes

Nélio Cacho, Thais Batista, Gledson Elias
Departamento de Informática – Universidade Federal do Rio Grande do Norte (UFRN)
Campus Universitário – 59.056-300 – Natal – RN
e-mail: cacho@consiste.dimap.ufrn.br, thais@ufrnet.br, gledson@dimap.ufrn.br

Resumo

O padrão CORBA oferece suporte para descoberta de componentes através dos serviços de Nomes e de Trading e do Repositório de Interfaces. Apesar desses serviços oferecerem um importante suporte para encontrar componentes, eles apresentam algumas restrições. Primeiro, eles são independentes entre si. Portanto, para realização de buscas complexas considerando diferentes critérios como nomes, métodos e propriedades, o programador deve usar esses serviços e combinar os resultados obtidos. Segundo, os serviços apenas trabalham de forma síncrona. Terceiro, os serviços não implementam políticas para balanceamento de carga. Nesse artigo, descrevemos um serviço CORBA para descoberta de componentes que oferece uma maneira simples e uniforme de descobrir componentes considerando diferentes critérios. Este serviço permite tanto buscas síncronas quanto assíncronas, agrupa componentes com propriedades similares em uma coleção e aplica um mecanismo de balanceamento de carga sobre a coleção.

Abstract

Currently, CORBA relies on Naming Service, Trading Service and Interface Repository for discovering components. Although they offer important means to find out components, they present some constraints. First, they are independent from each other. Thus, to perform complex searches according to different criteria, the programmer should use them and combine their results. Second, these services only allow synchronous searching. Third, although the Trading Service groups objects with similar properties, it does not implement policies to favor load balancing. In this paper we describe a CORBA discovering service that provides a simple and uniform way to searching components according different criteria. This service allows the traditional synchronous searching invocation and also offers an asynchronous searching approach, in which a listener is invoked when new components that satisfy a given criterion are registered in the underlying repository. It also groups components with similar properties in a collection and applies a round-robin load balancing mechanism upon the collection.

1. Introdução

O desenvolvimento de software baseado em componentes está centrado na idéia de compor aplicações através da combinação de componentes de software pré-fabricados [9] visando à redução do custo e do tempo de desenvolvimento de software.
Plataformas de Middleware [2] oferecem um poderoso suporte para a construção de sistemas de software baseados em componentes. Nesse contexto, CORBA [3] tem tido destaque por ser independente de linguagem, de plataforma, de fabricante e também por prover transparência de distribuição.
De forma a promover o reuso, é necessário que as plataformas de middleware ofereçam um serviço para descobrir objetos de acordo com diferentes critérios. O suporte atual de CORBA para localização de componentes é fornecido por serviços [4] independentes e

complementares: Nomes e Trading. O Serviço de Nomes permite localização de componentes por nomes e o Serviço de Trading permite localização de componentes de acordo com suas propriedades. O Repositório de Interfaces (RI) oferece a descrição da interface de um objeto. No entanto, como tais serviços e repositórios são independentes uns dos outros, não há uma maneira automática de combinar suas funcionalidades. Para aproveitar o suporte de tais serviços, o programador deve combinar seus resultados. Como citado em [1], a busca por um objeto que implementa um dado conjunto de métodos, exige o uso do Serviço de Trading para encontrar a interface implementada por cada objeto registrado e, em seguida o acesso ao Repositório de Interfaces para determinar as assinaturas dos métodos de cada interface. Dessa forma, o processo torna-se complicado. A falta de uma maneira uniforme e fácil de usar para localização de componentes é um obstáculo para o reuso.

Outro problema desses serviços é não testar se os objetos retornados em uma consulta estão ainda ativos. Além disso, para invocar cada serviço, são necessários vários comandos. Portanto, há necessidade de um serviço de localização mais simples e mais eficiente que o atual suporte que CORBA oferece.

Neste artigo, apresentamos um serviço CORBA para descoberta de componentes que fornece uma maneira simples e uniforme de localizar componentes de acordo com diferentes critérios, e, portanto, reduz significantemente o trabalho do programador no processo de busca. A busca pode ser feita por nome, assinatura de métodos e propriedades. O serviço oferece a opção de retorno de objetos ativos. Enquanto, tradicionalmente, os serviços de localização utilizam invocação síncrona, o serviço apresentado nesse trabalho também oferece busca assíncrona, na qual um objeto *callback (listener)* é invocado quando acontece o registro, no repositório do serviço, de novos componentes que satisfazem um determinado critério.

O serviço CORBA proposto nesse trabalho aproveita a infraestrutura disponibilizada por CORBA utilizando os serviços de Nomes, de Trading e o Repositório de Interfaces. O serviço também pode ser usado de forma independente uma vez que ele mantém um repositório e oferece métodos para registrar e buscar serviços em tal repositório.

Este artigo está estruturado da seguinte forma. A seção 2 apresenta alguns conceitos básicos de CORBA e os seus serviços de localização. A seção 3 apresenta a arquitetura e a funcionalidade do serviço proposto nesse trabalho para descoberta de componentes. A seção 4 apresenta a avaliação de desempenho do serviço. A seção 5 comenta sobre alguns trabalhos relacionados. A seção 6 contém as conclusões.

2. CORBA e os Serviços de Localização

CORBA [3] é um padrão proposto pelo OMG para permitir interoperabilidade entre aplicações em ambientes heterogêneos e distribuídos. Este padrão propõe a separação entre a interface de um objeto e sua implementação. CORBA oferece a Linguagem de Descrição de Interfaces (IDL) para descrição das interfaces dos objetos. A implementação pode ser realizada em qualquer linguagem de programação que tenha mapeamento para CORBA. A arquitetura de CORBA é composta por um conjunto de blocos funcionais que usam o suporte de comunicação oferecido pelo ORB (*Object Request Broker*) – o elemento que gerencia as interações entre objetos, interceptando as invocações dos clientes e direcionando-as para o servidor apropriado. O repositório de interfaces (RI), definido na especificação CORBA, armazena todas as definições IDL dos objetos. CORBA oferece um conjunto de serviços a serem usados no desenvolvimento de aplicações CORBA. Cada serviço é descrito por interfaces IDL.

2.1. Serviço de Nomes

O Serviço de Nomes de CORBA associa nomes a referências de objetos. O serviço oferece métodos para associação de um nome com uma referência e para recuperar referências de objetos. Para exemplificar o uso do Serviço de Nomes, a Figura 1 descreve como obter a referência do objeto *LaserPrinter*.

```
1 CORBA::Object_var nsobj=orb->resolve_initial_references("NameService");
2 CosNaming::NamingContext_var nc=CosNaming::NamingContext::_narrow (nsobj);
3 CosNaming::Name name;
4 name.length (1);
5 name[0].id = CORBA::string_dup ("LaserPrinter");
6 name[0].kind =CORBA::string_dup ("");
7 CORBA::Object_var obj;
8 obj = nc->resolve (name);
```

Figura 1. Usando o Serviço de Nomes.

As linhas 1 e 2 contêm o código que recupera a referência do Serviço de Nomes. As quatro linhas seguintes contêm o código para determinar o nome do objeto. Finalmente, na linha 8, o método `resolve` é chamado e retorna a referência do objeto.

2.2. Serviço de Trading

O Serviço de Trading de CORBA é semelhante às paginas amarelas de um catálogo telefônico. Ele descreve *ofertas de serviço* que associam referências de objetos com tipos de serviço e também descreve os tipos de serviço através de identificadores de interface IDL e de um conjunto de propriedades. Uma oferta de serviço deve conter valores para propriedades definidas para o tipo de serviço correspondente.
O Serviço de Trading oferece métodos para anunciar um novo serviço (exportar uma oferta de serviço) e para buscar objetos (importar uma oferta de serviço) baseado em propriedades do serviço.
O Serviço de Trading também permite definir uma federação de *traders*. Federações tornam possível objetos anunciarem seus serviços em um grupo.

```
1  CosTrading::Lookup_var lookup = CosTrading::Lookup::_narrow( obj );
2  assert( !CORBA::is_nil(lookup ) );
3  CosTrading::ServiceTypeName_var type = CORBA::string_dup("IDL:Printer:1.0" );
4  CosTrading::Constraint_var constr = CORBA::string_dup( "ppm > 5" );
5  CosTrading::Lookup::Preference_var prefs = CORBA::string_dup( "max ppm" );
6  CosTrading::Lookup::SpecifiedProps desi;
7  desi._d( CosTrading::Lookup::all );
8  CosTrading::PolicySeq policyseq;
9  policyseq.length( 0 );
10 CosTrading::OfferSeq* offers = OL;
11 CosTrading::OfferIterator_ptr offer_itr = OL;
12 CosTrading::PolicyNameSeq* limits = OL;
13 lookup->query(type,constr,prefs,policyseq,desi,100,offers,offer_itr,limits );
```

Figura 2. Usando o serviço de Trading

Para ilustrar o uso do Serviço de Trading de CORBA, a Figura 2 mostra uma consulta que retorna todos os serviços com propriedade *ppm* maior que 5 e *type* igual a IDL:Printer:1.0. As linhas 1 e 2 contêm o código para obter a referência da interface responsável pela busca no Serviço de Trading. A linha seguinte contém o tipo de serviço a ser descoberto. As linhas 4 e 5 especificam a restrição e a ordem em que os resultados devem ser retornados. As próximas duas linhas especificam que todas as propriedades devem ser retornadas. As

linhas 8 e 9 determinam que nenhuma política especial de busca deve ser aplicada. As linhas 10 a 12 contêm o código com as declarações de variáveis usadas para receber o resultado da busca. Na linha 13 o método `query` é invocado.

3. Serviço de Descoberta

O Serviço de Descoberta oferece uma maneira simples e uniforme de buscar componentes, considerando diferentes critérios: nomes de métodos, assinaturas de métodos, exceções e propriedades não-funcionais.
Esse serviço oferece os mecanismos básicos do Serviço de Trading tais como interfaces para exportar e importar serviços, para administrar e interagir com *traders* e/ou outros serviços CORBA remotos agrupados em uma federação. Além disso, o serviço oferece um mecanismo de *callback* para notificar a aplicação quando acontece o registro de um novo componente que satisfaz um critério previamente estabelecido.
O Serviço de descoberta pode ser usado em tempo de desenvolvimento ou em tempo de execução. Desta forma, o mecanismo de descoberta não retorna uma lista de referências de objetos, mas uma lista de identificadores de ofertas. Em tempo de desenvolvimento, o programador pode invocar o método `describe` da interface *Register* de forma a obter a descrição da interface denotada por um identificador particular. Em tempo de execução, o programador pode invocar o método `getServiceReference` para recuperar a referência de cada identificador.

3.1. Arquitetura

A Figura 3 ilustra a arquitetura do Serviço de Descoberta. O serviço dispõe de quatro interfaces e uma delas (*Link*) usa serviços oferecidos pelos repositórios CORBA padrão e por outros serviços de localização. As interfaces oferecidas pelo serviço de Descoberta são similares as que compõem o Serviço de Trading de CORBA: *Register*, *Lookup*, *Link* e *Admin*.

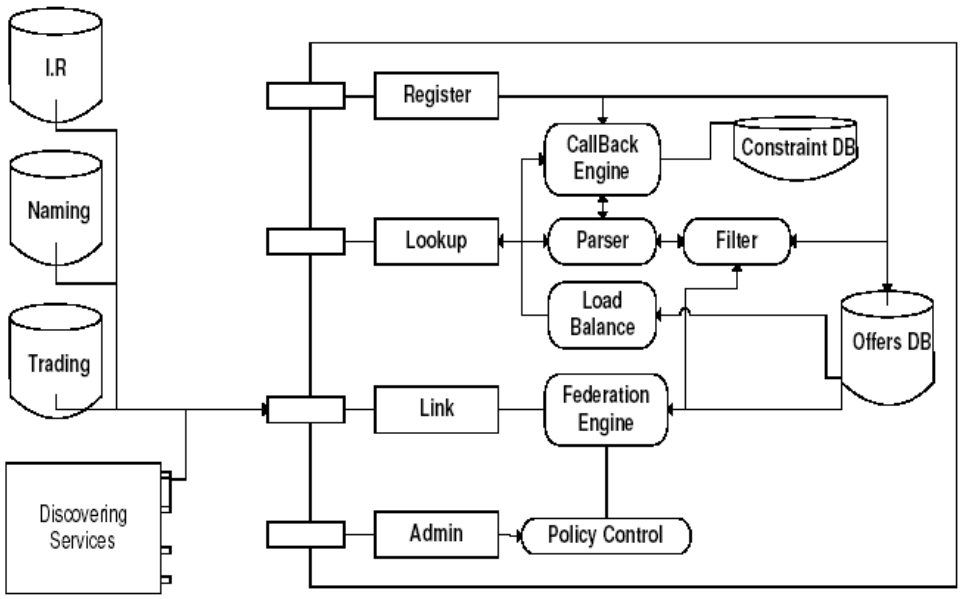

Figura 3. Arquitetura do Serviço de Descoberta.

A principal finalidade da interface *Register* é armazenar novas ofertas de serviço no repositório *Offers DB*. Essa interface agrupa novos serviços de acordo com suas propriedades funcionais e não-funcionais. Esse agrupamento será usado pelo módulo de balanceamento de carga (*Load Balance*) para determinar o componente que satisfaz um requisito específico. Outra função da interface *Register* é informar ao módulo `Callback Engine` que novas ofertas de serviço foram registradas.

A função da interface *Lookup* é consultar os repositórios usados pelo Serviço de Descoberta para localizar objetos que satisfaçam determinados critérios. Dois tipos de consultas podem ser feitas: síncrona e assíncrona. Para consultas síncronas, a interface *Lookup* invoca o módulo `Parser` que decompõe os critérios de busca e os encaminha para o módulo `Filter` processar os resultados. Para consultas assíncronas, a interface *Lookup* invoca o módulo `CallBack Engine` de forma a registrar o critério de busca no *constraint DB*. Esse módulo usa o módulo `Parser` para encontrar componentes que satisfazem os critérios estabelecidos.

Após processar os resultados, a interface *Lookup* retorna a lista de identificadores de ofertas. O balanceamento de carga é opcional e é usado apenas quando o cliente especifica na restrição da busca que o resultado deve conter os serviços menos sobrecarregados. Nesse caso, quando o módulo `Parser` termina o processo de filtragem, ele encaminha os resultados para o módulo `Load Balance`. Esse módulo ordena os resultados usando um mecanismo *round-robin*. Depois da ordenação, o módulo `Load Balance` retorna os resultados para a interface *Lookup*. Para manter o controle do uso dos serviços, o módulo `Load Balance` mantém um contador de uso que é incrementado sempre que o método `getServiceReference` é invocado para um determinado identificador de oferta (*OfferId*).

A interface *Admin* mantém os mecanismos de proteção interna do Serviço de Descoberta. O módulo `Policy Control` pode limitar o uso dos recursos de forma a evitar uma sobrecarga na busca que trata uma grande quantidade de dados. Os parâmetros *max_return_offers* e *max_match_card* são usados para setar os limites recursos utilizados.

A interface *Link* permite a interação com outros Serviços de Descoberta e com os três repositórios CORBA: Nomes, Trading e o Repositório de Interfaces. Ele também permite a propagação da busca para outros mecanismos.

A Figura 4 exibe um resumo do modelo de classes em notação UML do Serviço de Descoberta. A classe *Lookup* é responsável por interpretar a consulta, executá-la e retornar os serviços selecionados. Para isso, essa classe é composta por outras três classes: (1) *Parser*, responsável por interpretar e executar as restrições da busca; (2) *Filter*, responsável por aplicar ao resultado fornecido pela classe *Parser* as funções de ordenação e de balanceamento de carga; (3) *ActivityControl*, responsável por verificar, quando necessário, se todos os serviços fornecidos estão disponíveis. As classes *Admin* e *Link* são responsáveis, respectivamente, pela proteção do Serviço de Descoberta e em permitir que o Serviço de Descoberta interaja com outros serviços. Todas as inserções, modificações e remoções das interfaces e serviços são realizadas pela classe *Register*. Esta classe é composta pelas classes *OffersDB*, *ConstraintDB* e *CallBackEngine*. A classe *OfferDB* armazena as ofertas que são compostas por uma lista de instâncias da classe *Offer* juntamente com uma instância da classe *Index*. Para armazenar as propriedades de cada oferta a classe *Index* possui uma lista de instâncias da classe *IndexKey*, que representam as propriedades dos *Offers*. Para cada *IndexKey* podem existir uma ou mais propriedades armazenadas em *KeyValues*. A classe *KeyValues* é um par formado pelo valor da propriedade juntamente com um identificador de um *Offers*. Cada *Offers* pode ser representado por uma interface ou por um serviço. Além de armazenar as propriedades dos

Offers, *IndexKey* também relaciona propriedades com gatilhos, uma vez que uma propriedade pode estar em mais de um gatilho. O controle dos gatilhos é realizado pelas classes *ConstraintDB* e *Callback Engine* que interagem com a classe *Lookup* para realizar a inserção de novos gatilhos.

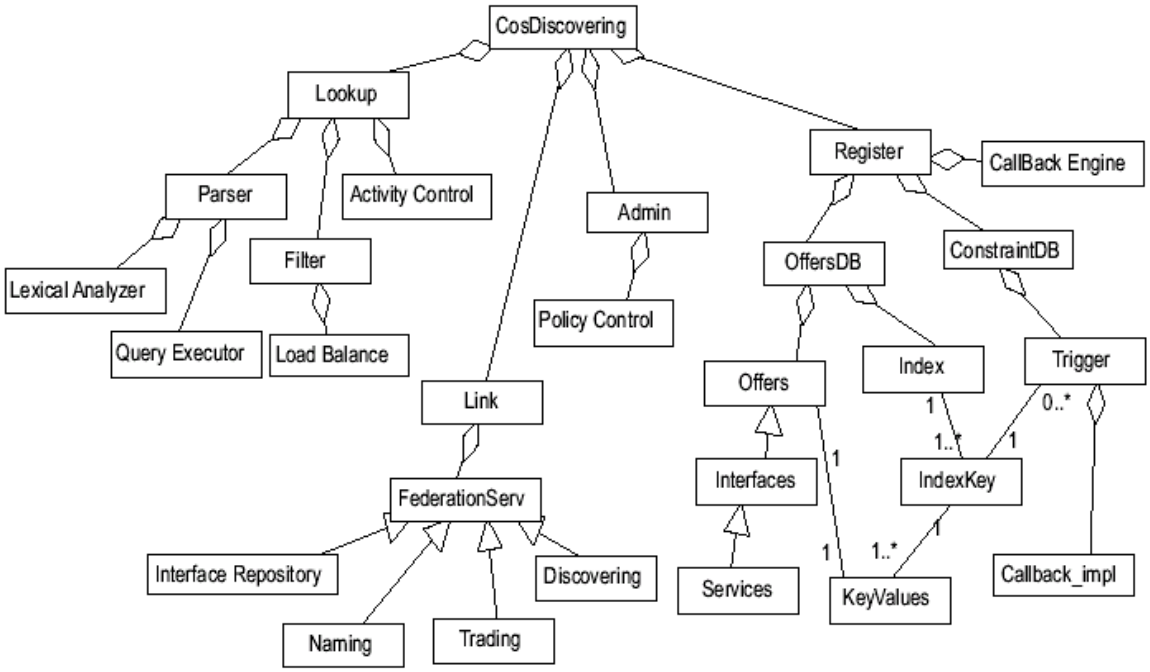

Figura 4. Diagrama de Classes do Serviço de Descoberta.

3.2. Funcionalidade

Como ilustrado na Figura 3, o Serviço de Descoberta oferece quatro interfaces: *Register*, *Lookup*, *Admin* e *Link*. Nesta seção, descreveremos cada uma dessas interfaces mostrando seus métodos e como eles podem ser utilizados.

3.2.1. Register. Os métodos da interface *Register* estão ilustrados na tabela 1.

Tabela 1. Métodos da Interface Register

Nome de Métodos e Parâmetros	Descrição
OfferId insertService(in Object reference, in PropertySeq properties);	Insere services
OfferId insertInterface(in CORBA::InterfaceDef:: FullInterfaceDescription interfaceDef, in PropertySeq properties)	Insere interfaces
remove(in OfferId id)	Remove uma oferta do repositório.
modifyReference(in OfferId id, in Object reference)	Modifica a referência associada com uma oferta.
modifyProperty(in OfferId id, in PropertyNameSeq del_list, in PropertySeq modify_list)	Modifica as propriedades de uma oferta.
modifyInterface(in OfferId id, in CORBA::InterfaceDef::FullInterfaceDescription interfaceDef)	Modifica a interface de uma oferta.
ObjectDescription describe(in OfferId id);	Retorna a descrição de um serviço.

A maioria dos serviços de localização de objetos como os Serviços de Nomes e de Trading apenas armazenam referências de objetos ativos. No entanto, esses serviços não verificam se os objetos estão realmente ativos. Desta forma, o cliente pode receber uma referência de um objeto que não está mais ativo.

Para solucionar esse problema, o Serviço de Descoberta não considera apenas ofertas associadas com objetos ativos, mas também todos os objetos ou interfaces que podem ser descritos por propriedades funcionais (métodos, parâmetros, etc) e não-funcionais. Ele também pode retornar a referência de um objeto ativo. Para isso o cliente usa a função *activity* na restrição de busca e o serviço disponibiliza o método `getServiceReference` para fornecer a referência do serviço.

Para registrar novos serviços, a interface *Register* oferece os métodos `insertService` e `insertInterface`. O método `insertService` recebe uma referência de um objeto ativo e uma seqüência de propriedades não-funcionais. Ele retorna uma *OfferId* que identifica a oferta inserida. As propriedades funcionais não são passadas como parâmetro, pois o próprio Serviço de Descoberta as infere e realiza a inserção. Isto só é possível porque todos os objetos CORBA ativos têm uma `FullInterfaceDescription` associada. As propriedades funcionais inseridas são descritas na Tabela 3.

O método `insertInterface` é usado para inserir interfaces. Esse método recebe uma *FullInterfaceDescription* e um conjunto de propriedades não-funcionais. Ele retorna um identificador da oferta *OfferId*. Apesar de parecer estranho associar propriedades não-funcionais a uma interface, uma vez que pode ser que não exista implementação associada, nada impede que o usuário, através do método `modifyReference`, associe uma implementação a uma interface. Tal método recebe um identificador de oferta e uma referência para um objeto ativo. O método `modifyInterface` é usado para associar uma interface a um objeto quando tal objeto tem uma interface diferente da inserida anteriormente.

A Figura 5 ilustra um exemplo do registro de novos serviços usando o Serviço de Descoberta.

```
1  CosDiscovering::Register_var register = compo->getRegister();
2  assert( !CORBA::is_nil( register ) );

4  ::CosDiscovering::PropertySeq seq;
5  seq.length(2);
6  seq[0].name = CORBA::string_dup( "Domain" );
7  seq[0].value << CORBA::Any::from_string("dimap.ufrn.br",0);
8  seq[1].name = CORBA::string_dup( "ppm" );
9  seq[1].value<< (CORBA::Long)5;

11 CORBA::Object_var ref = poa->id_to_reference(oid.in());
12 register->insertService(ref, seq);
```

Figura 5. Usando a interface Register.

As duas primeiras linhas do código mostrado na Figura 5 são usadas para recuperar a referência da interface *Register*. As linhas 4 e 5 mostram a criação de um array de propriedades que pode armazenar dois itens. Nas próximas quatro linhas, o array recebe os valores das propriedades *Domain* e *ppm*. Na linha 11, a referência para um objeto CORBA é recuperada. Na linha 12, o método `insertService` é usado para registrar esse objeto com as propriedades *Domain* e *ppm*.

Os métodos `remove` e `modify` oferecem maneiras de, respectivamente, remover e modificar serviços registrados previamente. O método `remove` recebe a identificação da oferta (*OfferId*) do serviço. O método `modify` recebe a identificação da oferta (*OfferID*) do serviço, a lista (*PropertyNameSeq*) das propriedades a serem removidas e a lista (*PropertySeqlist*) com as novas propriedades.

O método `describe` é usado para obter uma descrição completa da oferta de serviço. Este método recebe uma identificação de oferta (*OfferId*) e retorna uma estrutura

(*ObjectDescription*) composta por dois campos. O primeiro campo, *description*, armazena todas as propriedades funcionais de um objeto como operações, parâmetros, etc. Este campo é definido pela especificação CORBA: *CORBA::InterfaceDef: :FullInterfaceDescription*. O segundo campo, *properties*, consiste em uma seqüência de pares nome/valor com as propriedades não-funcionais de uma oferta de serviço.

3.2.2. Lookup. A interface *Lookup* implementa o mecanismo de busca do Serviço de Descoberta. A tabela 2 contém a descrição IDL da interface *Lookup*.

Tabela 2. Descrição IDL da interface Lookup

Nome de Métodos e Parâmetros	Descrição
void search(in string constr, out OfferIdSeq result);	Busca Síncrona.
constraintId search_back(in string constr, in Callback obj)	Busca Assíncrona.
void remove_callback(in constraintId idreg)	Remove uma restrição registrada no mecanismo callback.
Object getServiceReference(in OfferId idoffer)	Recupera a referência de um componente ativo.

O método `search` implementa o mecanismo de busca síncrona. Esse método recebe uma *string* com as restrições (critérios de busca) que componentes devem satisfazer, e retorna uma lista de identificadores de ofertas. O parâmetro *constr* descreve uma combinação de critérios de busca. O Serviço de Descoberta permite todos os tipos de restrições definidas pela *CORBA constraint language*. Uma restrição pode assumir uma combinação dos seguintes valores:

- nomes e valores de propriedades
- literais de vários tipos de dados (String, Number, Boolean)
- teste da existência de uma propriedade (exists)
- comparação de propriedades, literais ou ambos (igual, String mais próxima(~), todos os operadores relacionais)
- expressões booleanas (and, or, not)
- operadores aritméticos básicos (adição, subtração, multiplicação e divisão)
- teste se um componente é membro de um conjunto (in)

Para o último item listado acima, a especificação determina que o operador *in* pode ser aplicado apenas se o operador à esquerda é de um tipo básico (string, number ou boolean) e o operador da direita é uma seqüência do mesmo tipo. De forma a flexibilizar o critério de busca, o Serviço de Descoberta permite que o operador à esquerda assuma uma seqüência de qualquer tipo. Desta forma, restrições como ((Age > 10)in(Sex == 'M')) são válidas para o Serviço de Descoberta que retorna todos os componentes com Age > 10 em um conjunto de componentes com Sex == 'M'.
Entre os nomes e valores de propriedades que uma restrição pode assumir, há a lista de propriedades (descritas na Tabela 3) que são automaticamente inseridas pela interface *Register* ou pelo módulo *Link* quando da existência de um *link* com opção `ModeHop` igual a *clone*.

Tabela 3. Nomes de Propriedades - nomes reservados

Nome	Descrição	Possíveis Valores
serviceId	ID do serviço	Ex.:IDL:Account:1.0
servicename	Nome do serviço	Ex.:Account
versionumber	Número da versão do serviço	Ex.:1.0

operationame	Nome da operação	Ex.:deposit
operatiomode	Modo da operação	NORMAL ou ONEWAY
paramname	Nome do parâmetro	Ex.:valuetotal
parammode	Modo do parâmetro	PARAM_IN, PARAM_OUT or PARAM_INOUT
paramtype	Tipo do Parâmetro	tk_shot, tk_long, tk_string, etc.
operationexceptionname	Nome da exceção emitida por uma operação	Ex.: InvalidObjectRef
attributename	Nome de um atributo	Ex.: AccountState
attributetype	Tipo do atributo	tk_shot, tk_long, tk_string, etc.
lastavailability	Data da ultima disponibilidade	21-03-2000-15:30:50.

Essas propriedades podem guiar o processo de busca em direção a um resultado adequado uma vez que elas permitem o detalhamento do critério de busca. Por exemplo, a restrição *((paramname == 'AccountNumber')in(operation == 'deposit'))* especifica que apenas devem ser retornadas ofertas que tenham o método `deposit` com um parâmetro igual a *AccountNumber*. O parâmetro *constraint* também deve permitir o uso de funções para determinar a ordem de retorno dos resultados para o usuário. As seguintes funções podem ser usadas:

- funções de ordenação 'min' e 'max' recebem como parâmetro uma expressão que refere-se a propriedades da oferta. As ofertas são retornadas em uma ordem descendente (max) ou ascendente (min).
- 'random' as ofertas são retornadas em uma ordem aleatória.
- 'first' as primeiras ofertas descobertas são retornadas.
- 'with' recebe uma expressão booleana. As ofertas que tenham o valor verdadeiro precedem as que têm o valor falso.

A Figura 6 contém um código que implementa, usando o Serviço de Descoberta, as mesmas consultas feitas usando o Serviço de Nomes e de Trading previamente apresentados na seção 2. As linhas 1 a 3 recuperam a referência do Serviço de Descoberta e da interface *Lookup*. Nas linhas seguintes a restrição é definida setando-se os valores de algumas propriedades reservadas como *servicename* e *serviceId* e propriedades definidas pelo usuário como *ppm*. Na linha 6, o método `search` é invocado. Esse método retorna uma lista de identificadores de ofertas *OfferId*.
Além das funções CORBA padrão, o Serviço de Descoberta também oferece a função *activity*. Essa função seleciona componentes ativos. Por exemplo, a restrição *activity(max (ppm) (ppm > 10)and(servicename == 'Printer'))* retorna todas as ofertas de serviços que tenham *ppm* maior que 10 e o nome do serviço igual a *Printer*. As funções *max(ppm)* e *activity* determinam que a lista resultante deve começar com o serviço ativo com maior valor de *ppm*. A verificação de atividade de um serviço é realizada quando da aplicação da função *activity* ou quando da utilização da função `getServiceReference`. Em ambos os casos, o serviço verifica se o objeto está ativo via a invocação do método `non_existent`, presente em todos os serviços CORBA, que retorna verdadeiro para serviços indisponíveis e falso caso contrário. Para as situações em que tal método retorna verdadeiro, o Serviço de Descoberta registra na propriedade *lastavailability* a data/hora de sua indisponibilidade. Esta propriedade pode ser usada, por exemplo, na restrição de busca *(lastavailability < 3600)and(ppm > 10)* para recuperar os serviços com *ppm* maior que 10 e que estejam ativos *(lastavailability = 0)* ou que estavam disponíveis na última hora.

```
1 CosDiscovering::DiscoveringComponents_var compo= CosDiscovering::
DiscoveringComponents::_narrow (obj);
2 CosDiscovering::Lookup_var search = compo->getLookup();
3 assert( !CORBA::is_nil( search ) );
4 const char* constraint = "max(ppm)((servicename == 'LaserPrinter') or
((serviceId == 'IDL:Printer:1.0')and(ppm > 5)))";
5 ::CosDiscovering::OfferIdSeq_var result;
6 search->search(constraint, result);
```

Figura 6. Usando o Serviço de Descoberta.

Outra função oferecida pelo Serviço de Descoberta é a de balanceamento de carga. Essa função aplica um algoritmo *round-robin* para ordenar o resultado da busca. O primeiro elemento da lista ordenada é o serviço com menos carga. Um exemplo do uso de tal função: *loadbalance((ppm > 10)and(servicename == 'Printer'))*.

O mecanismo de busca síncrona é útil para as situações em que a aplicação encontra um componente que satisfaça suas necessidades de utilização. No entanto, para as situações em que não existe ainda um componente disponível, a aplicação tem que escolher entre seguir, sem o componente, ou ficar bloqueado realizando consultas sucessivas. Se optar por seguir, a aplicação terá que, freqüentemente, verificar a existência do componente. Essa verificação além de adicionar um *overhead* extra, uma vez que se trata de invocações remotas, também adiciona mais complexidade ao código que terá agora que alterar, seguidamente, seu fluxo natural para verificar a disponibilidade de um componente.

Para minimizar essa problemática, o Serviço de Descoberta oferece, através do método `search_back`, o mecanismo de busca assíncrona. Este método recebe uma restrição e uma referência para um objeto *callback*. A restrição é armazenada no *constraint DB* que retorna um identificador de tal restrição (*Constraint Id*).

```
interface Callback { oneway void result_back(in OfferIdSeq result);  };
```

Figura 7. Interface Callback.

O objeto *Callback* deve implementar a interface descrita na Figura 7. A invocação do método `result_back` ocorre sempre que há o registro de um novo componente que satisfaz as restrições passadas como parâmetro do método `search_back`. O parâmetro *result* irá receber a lista de identificadores de ofertas.

Dessa forma se uma aplicação precisar imprimir um documento, mas não existir uma impressora disponível, a aplicação pode apenas implementar a interface *Callback* e no método `result_back` inserir o código necessário para impressão, pois este código só será invocado quando os critérios fornecidos ao método `search_back` forem satisfeitos. Isso evita tanto o bloqueio quanto o acréscimo na complexidade da aplicação que não terá mais que modificar seu fluxo para verificar a disponibilidade de componentes.

Para remover uma restrição no mecanismo de busca assíncrona, utiliza-se o método `remove_callback`. Ele pode ser invocado informando apenas o identificador da restrição.

Os métodos previamente descritos retornam uma lista de identificadores de ofertas. Esses identificadores têm diferentes usos de acordo com o momento que eles serão usados. Em tempo de desenvolvimento, os identificadores são usados pelo programador para conhecer os métodos, parâmetros e exceções de um objeto. Em tempo de execução, esses identificadores devem ser substituídos por referências de objetos que implementam as funcionalidades requisitadas. O método `getServiceReference` deve ser usado para recuperar tais referências. Ele recebe um identificador de oferta (*OfferId*) e retorna uma referência para um objeto CORBA.

3.2.3. Admin. A interface *Admin* oferece formas de administrar as políticas de proteção ao Serviço de Descoberta. Tais políticas permitem proteger o serviço contra sobrecarga causada por buscas que tratam com uma grande quantidade de dados. Os métodos oferecidos por essa interface são descritos na Tabela 4.

Tabela 4. Métodos da Interface Admin

Nome de Métodos e Parâmetros	Descrição
set_max_return_offers(in unsigned long value)	Especifica o número máximo de ofertas retornadas como resultado do processo de busca.
set_max_match_card(in unsigned long value)	Especifica o número máximo de ofertas sobre as quais a busca deve ser aplicada.
set_max_hop_count(in unsigned long value)	Especifica o número máximo de serviços de descoberta a serem usados no processo de busca.
set_if_no_local(in boolean value)	Especifica que outro serviço de busca deve ser usado apenas se o resultado da busca local é vazio

`set_max_return_offers`, `set_max_match_card` e `set_max_hop_count` são métodos usados para configurar o controle das buscas. O primeiro é usado para indicar o número máximo de ofertas que podem ser retornadas em uma busca. O segundo é usado para especificar o número máximo de comparações que podem ser realizadas. O terceiro é usado para determinar o número máximo de serviços visitados num caminho de busca. O método `set_if_no_local` especifica que um outro Serviço de Descoberta será usado apenas se o resultado da busca local é vazio.

3.2.4. Link. O Serviço de Descoberta reusa os repositórios CORBA e também permite o agrupamento de outros Serviços de Descoberta em uma federação. O objetivo da federação é agrupar alguns Serviços de Descoberta e suas ofertas de forma a oferecer um mecanismo de busca mais amplo. Tal mecanismo é estruturado em um grafo onde os nós representam os Serviços de Descoberta e as ligações representam o caminho entre os nós. Os caminhos são usados pelo mecanismo de busca.
Portanto, a interface *Link* permite se definir as ligações entre os Serviços de Descoberta e/ou outros repositórios CORBA. A Tabela 5 exibe os métodos dessa interface.

Tabela 5. Métodos da Interface Link

Nome de Métodos e Parâmetros	Descrição
void addLink(in string name, in Object target, in TypeServer type, in ModeHop mode, in long timeofreload);	Insere uma ligação
void removeLink(in string name)	Remove uma ligação.

O método `addLink` insere uma ligação. Ele recebe como parâmetro o nome da ligação e a referência do objeto que implementa um dos tipos de serviços suportados, a declaração do tipo do servidor, o tipo de *hop*(salto) que será usado e o tempo de recarga das ofertas. O Serviço de Descoberta implementa dois tipos de ligações. A primeira apenas propaga uma busca enquanto que a segunda clona o repositório. Desta forma, o parâmetro *ModeHop* pode assumir um dos seguintes valores: *Search* ou *Clone*. A interface *Link* implementa ligação com quatro tipos de servidores: Repositório de Interfaces, Nomes, Trading e Serviço de Descoberta. O parâmetro *TypeServer* especifica qual serviço será usado. Seus possíveis valores: *SIRServer*, *SName*, *STrader*, *SDiscovering*. O parâmetro *timeofreload* deve ser usado apenas quando o parâmetro *ModeHop* é igual a *Clone*. Nesse caso, a

ligação será visitada novamente no tempo determinado por *timeofreload*. O método removeLink é usado para remover uma ligação.

```
1 CosDiscovering::Link_var link = Discovering1->getLink();
2 assert( !CORBA::is_nil( link ) );
3 CosDiscovering::Lookup_var lookup=Discovering2->getLookup();
4 assert( !CORBA::is_nil( lookup ) );
5  link->addlink(CORBA::string_dup  ("link1"),  lookup,    CosDiscovering::Link::
Sdiscovering , CosDiscovering::Link::Search, 0);
```

Figura 8. Interface Link.

A Figura 8 mostra um exemplo do uso do método addLink. A referência da interface *Link* é obtida nas linhas 1 e 2. As duas linhas seguintes recuperam outra referência da interface *Lookup* para onde a busca será propagada. Na linha 5, o método addLink é invocado.
Os métodos set_if_no_local e set_max_hop_count são usados para determinar quando uma busca será propagada e quantas ligações serão usadas.

4. Avaliação de Desempenho

No sentido de avaliar o desempenho na busca de componentes usando o Serviço de Descoberta, definimos três cenários experimentais. No primeiro cenário, analisamos o comportamento do serviço na busca de componentes. No segundo, analisamos o tempo para inserção de novas ofertas de componentes e no último, avaliamos o comportamento do serviço diante da utilização do mecanismo de busca assíncrona. Neste experimento comparamos o desempenho do Serviço de Descoberta com o serviço de *Trading* do MICO sob as mesmas condições. O experimento foi realizado num PC-Pentium IV com 512Mb RAM, sob Linux-Mandrake 9.0 com o ORB Mico 2.3.11.
Para analisar o tempo de resposta de cada consulta, inserimos no Serviço de Descoberta e no serviço de *Trading* do MICO o mesmo conjunto de ofertas de componentes. Este conjunto é composto por apenas um componente com cinco propriedades não funcionais. A mesma *constraint*, composta de propriedades não funcionais, foi usada como critério de busca para os dois serviços. Os valores representados nos gráficos foram obtidos pela mensuração do tempo de execução de cada método.
A Figura 9 mostra o comportamento do tempo de resposta do método search, no Serviço de Descoberta, e do método query, no serviço de *Trading* do MICO, com o aumento do número de componentes ofertados. O método *search* é utilizado de duas formas. Na primeira forma ele retorna uma lista de *OffersId* que satisfazem uma dada *constraint*. Na segunda, retorna uma lista de *OffersId* produzido pelo mecanismo de balanceamento de carga usando a metodologia *Round-robin*. A Figura 9 mostra que existe uma diferença substancial entre o tempo de resposta do MICO e do Serviço de Descoberta, sob o mesmo número de componentes ofertados. A figura também mostra que quando aplicado o mecanismo de balanceamento de carga, existe um pequeno *overhead* provocado pelo tempo extra, necessário para ordenar o resultado segundo a metodologia *round-robin*.

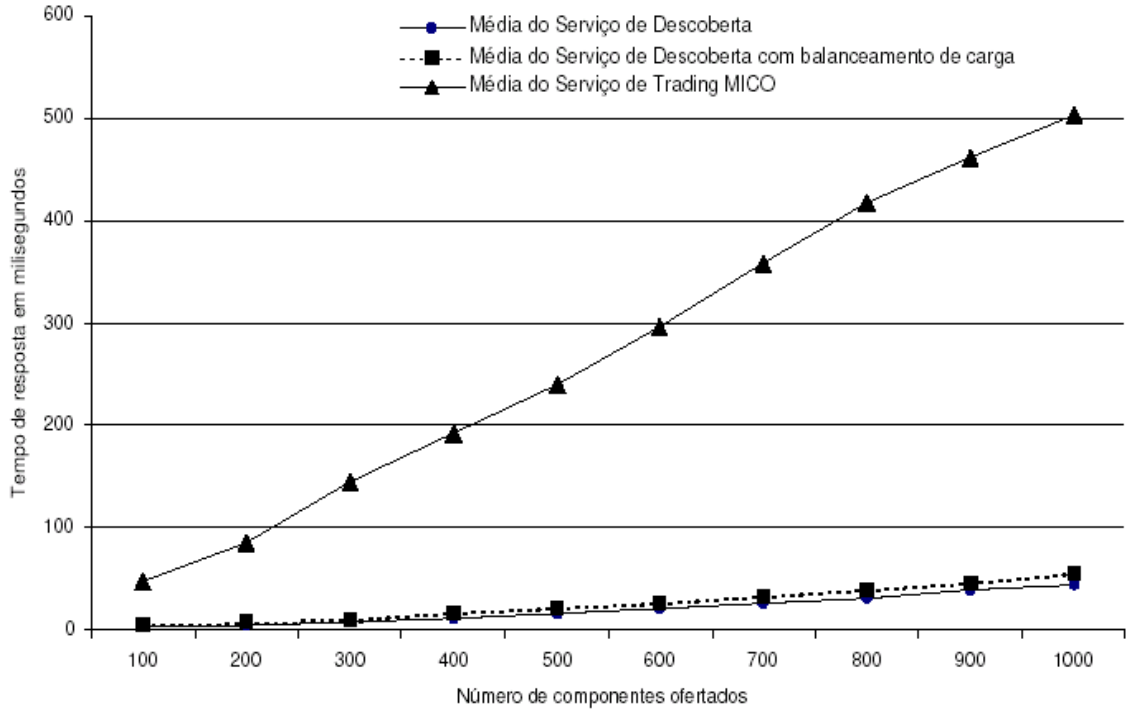

Figura 9. Avaliação na busca de componentes

A Figura 10 representa o tempo para registrar uma nova oferta de componente. Nesta avaliação não foram inseridas *constraints* de *callback*. O gráfico mostra que sob as mesmas condições, o tempo de resposta do Serviço de Descoberta é constante mesmo com o aumento no número de componentes ofertados.

5. Trabalhos Relacionados

Existem vários trabalhos com enfoque em aumentar o Serviço de Trading de CORBA para ampliar sua funcionalidade [5,6,10]. No entanto, nenhum deles propõe um novo serviço CORBA para descoberta de objetos que ofereça uma maneira simples e uniforme de especificar um conjunto de critérios de busca e, adicionalmente, ofereça mecanismos para balanceamento de carga e busca assíncrona.

Em [6] é proposto um modelo de serviço para busca de componentes que faz o papel do Serviço de Nomes e de Trading e inclui informação semântica descrevendo o comportamento do componente e suas dependências. Este serviço funciona em um ambiente baseado no modelo de Componentes de CORBA (CCM). Apesar dele oferecer capacidades semelhantes aos Serviços de Nomes e de Trading, ele difere do nosso Serviço de Descoberta porque oferece uma solução proprietária que não aproveita os repositórios existentes disponibilizados por tais serviços. Além disso, ele não implementa busca assíncrona. O seu principal alvo é a semântica. Esse aspecto ainda não é contemplado no Serviço de Descoberta apresentado nesse trabalho.

A integração de balanceamento de carga com o Serviço de Trading de CORBA tem sido tópico de pesquisa de vários trabalhos [5,8]. Em [5], é proposta uma integração do *trader* e de um módulo de balanceamento de carga de forma a adaptar a alocação das solicitações dos clientes a servidores adequados considerando a atual carga de uso do sistema. O foco desse trabalho é o aumento da qualidade de serviço em termos de desempenho. O trabalho implementa várias estratégias de balanceamento de carga em uma plataforma CORBA.

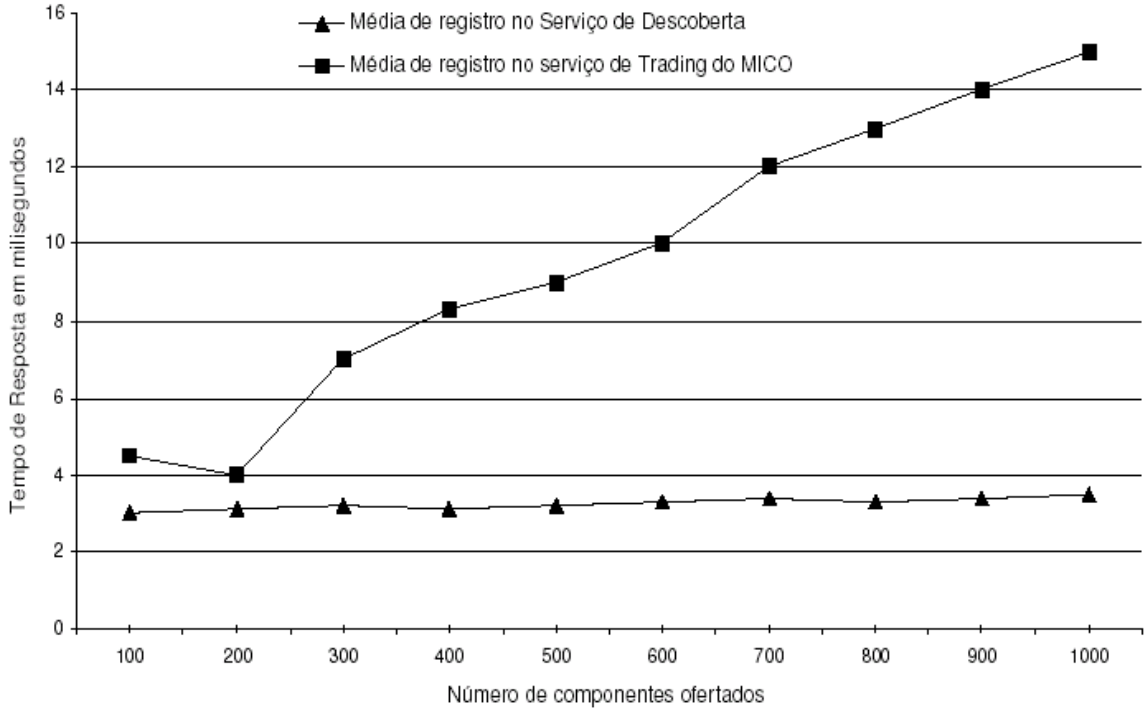

Figura 10. Avaliação no registro de oferta de compoentes

Um módulo é usado para combinar os resultados retornados pelo Serviço de Trading e os resultados retornados pelo módulo de balanceamento de carga. Esse trabalho difere do nosso em vários pontos. Primeiro, ele apenas implementa busca por propriedades. Segundo, os módulos *trader* e balanceamento de carga são independentes, então, é necessário um módulo combinador para compor seus resultados. O Serviço de Descoberta apresentado nesse trabalho encapsula o módulo de balanceamento de carga.

Em relação ao suporte para buscar componentes CORBA, o trabalho apresentado em [1] tem algumas similaridades com o presente trabalho. Ele também oferece uma função *search* que busca objetos que satisfazem diferentes critérios de busca incluindo nomes de objetos, assinaturas de métodos e propriedades. Ele também explora os repositórios dos serviços de Nomes, de Trading e de Interfaces. No entanto, ele é dependente de linguagem e não endereça busca assíncrona nem balanceamento de carga. Além disso, ele não propõe um novo serviço CORBA e não mantém um repositório próprio. Por não possuir um repositório, o mecanismo de busca sempre utiliza repositórios CORBA para resolver uma solicitação de busca.

O Serviço de Descoberta proposto nesse trabalho aproveita os repositórios CORBA, mas também mantém seu próprio repositório. Esse aspecto introduz flexibilidade porque permite que o serviço trabalhe independentemente dos serviços existentes.

O trabalho descrito em [10] propõe um framework CORBA que suporta a criação e a localização dinâmica de serviços, bem como balanceamento de carga e tolerância a falhas. Ele mantém um serviço de diretório que permite localizar serviços por nomes e por propriedades. O balanceamento de carga é implementado usando um monitor central que recebe dados de módulos de monitoramento individual que monitoram a carga de um servidor. O framework é baseado no modelo de *chain stores*. Em contraste, o Serviço de Descoberta procurou utilizar os padrões existentes, seguindo a especificação OMG dos serviços de Trading, de Nomes e do Repositório de Interface.

6. Conclusões

O atual suporte de CORBA para descoberta de componentes é fornecido pelo Repositório de Interfaces, pelo Serviço de Nomes e pelo Serviço de Trading. Eles mantêm repositórios que oferecem diferentes informações sobre componentes disponíveis e também oferecem métodos para inspeção nos repositórios. A independência de tais repositórios e a ausência de uma maneira simples e uniforme de buscar componentes de acordo com critérios diferentes e complexos é um obstáculo para programadores reusarem componentes disponíveis.

Nesse artigo, apresentamos um serviço CORBA para descoberta de componentes que tem como objetivo oferecer uma forma simples de determinar buscas complexas considerando nomes de objetos, assinaturas de métodos e propriedades. De forma a aproveitar as informações disponíveis nos repositórios CORBA, o Serviço de Descoberta é construído sobre serviços CORBA para localização de objetos. Apesar do Serviço de Descoberta ser construído sobre serviços CORBA, ele não é fortemente dependente desses serviços, pois pode ser usado independentemente uma vez que mantém seu próprio repositório.

O Serviço de Descoberta oferece funcionalidades também providas por outros serviços CORBA atualmente disponíveis tais como registro, busca, administração e ligação. Adicionalmente, o Serviço de Descoberta implementa uma estratégia de balanceamento de carga e um mecanismo de busca assíncrona. O mecanismo de busca assíncrona evita que o programador tenha de repetir, periodicamente, as mesmas mensagens de busca, o que pode ser um significante obstáculo para o desempenho e escalabilidade do serviço. O mecanismo de balanceamento de carga implementa um algoritmo *round-robin* que distribui solicitações uniformemente para componentes disponíveis. As vantagens de usar tal algoritmo é que ele é simples e determinístico.

Em termos de desempenho, apresentamos resultados de testes de desempenho que demonstram que apesar do serviço implementar buscas complexas, ele não apresenta qualquer perda de desempenho.

O objetivo de oferecer um novo serviço CORBA para descoberta de componentes é liberar o programador do trabalho envolvido em invocar diferentes serviços de forma a buscar um componente usando diferentes critérios de busca. Além disso, a funcionalidade do Serviço de Descoberta ultrapassa a funcionalidade dos atuais serviços de busca disponibilizados por CORBA uma vez que eles não permitem busca assíncrona nem balanceamento de carga.

Como trabalho futuro, pretendemos incluir outros algoritmos de balanceamento de carga e considerar aspectos adicionais que influenciam na carga de um componente. Pretendemos também avaliar o Serviço de Descoberta usando federações de *trader* em larga escala. Estamos trabalhando atualmente com a investigação do uso de ontologias para aumentar a semântica do Serviço de Descoberta.

Referências

1. Batista, T., Cerqueira, R., Rodriguez, N.: Enabling Reflection and Reconfiguration In CORBA. In Proceedings of the 2nd Workshop on Reflective and Adaptative Middleware - ACM/IFIP/USENIX International Middleware Conference, ISBN 85-87926-03-9, pp 125 - 129, Rio de Janeiro, RJ, June 2003.
2. Bernstein, P.: Middleware. Communications of the ACM, 39(2), February (1996).
3. OMG: The Common Object Broker Architecture and Specification Technical Report Revision 2.2, (1998)

4. OMG: CORBA Services: Common Object Services Specification Technical Report formal/97-07-04, (1997)
5. Thiben, D. and Neukirchen, H.: Managing Services in Distributed Systems by Integrating Trading and Load Balancing. Fifth IEEE Symposium on Computers and Communications (ISCC 2000). Antibes, France, July (2000).
6. Kebbal, D. and Bernard, G.: Component Search Service and Deployment of Distributed Applications. In Proc. 3rd International Symposium on Distributed Objects and Applications (DOA'01), Roma, Italy, September 17-20, 2001.
7. Lukkien, J,Tranmanh, T., Verhoeven, P and Peters, P.: Service Discovery Mechanisms: two case studies. Proceedings of the 2002 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA), Las Vegas, pp. 1187-1192; 2002.
8. Schiemann, B.: A New Approach for Load Balancing in Heterogeneous Distributed Systems. Proc.Workshop on Trends in Distributed Systems, Aachen, Germany, 1996.
9. Szyperski, C.: Component Software: Beyond Object-Oriented Programming",Addison-Wesley, 1998.
10. Shankaran, N. and Klefstad, R.: ZEUS: A CORBA Framework for Service Location and Creation. Proceedings of The 2004 International Symposium on Applications and the Internet (SAINT), Tokyo Japan, January, 2004.