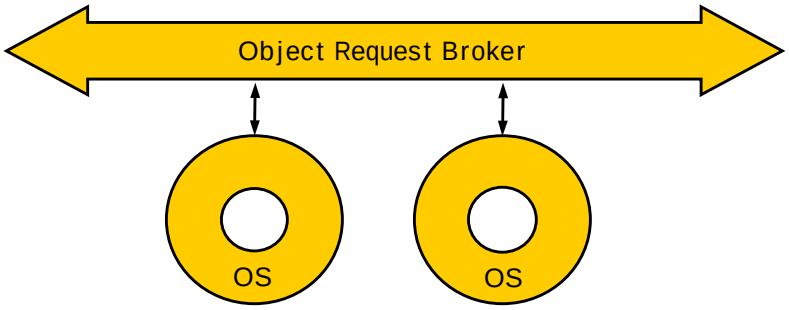
CORBAServices
Serviço de Nomes

Sumário

1 - Introdução ao Serviço de Nomes
2 - “CosNaming” e Formato de Nomes
3 - Resolução de Nomes
4 - Outros Métodos
5 - Interface “BindIterator“
6 - Considerações Finais

1 - Introdução ao Serviço de Nomes

- Os serviços CORBA estendem o *core* do CORBA com um conjunto de ferramentas úteis para muitas aplicações distribuídas.
  - ... seu papel é permitir associar nome a objeto e posteriormente permitir que o objeto seja encontrado através do seu nome;
  - ... a associação entre um nome e um objeto é denominada *binding* pelo Serviço de Nomes.



... 1 - Introdução ao Serviço de Nomes

- ... o servidor que retém uma referência de um objeto, pode registrá-la dentro do serviço de nomes, fornecendo o nome pelo qual será registrado no serviço de nomes.
- ... aplicações utilizam nomes para procurar por objetos no serviço de nomes.
- ... o serviço de nomes tem duas interfaces IDL que permitem qualquer componente do sistema utilizar usas facilidades para resolver nomes e/ou para cria novos “bindings”

2 - “CosNaming” e Formato de Nomes

- As interface fornecidas pelo Serviço de Nomes são definidas dentro do modulo IDL “CosNaming”

```
// IDL
module CosNaming {
    interface BindingIterator {...};
    interface NamingContext{...};
};
```

- Um nome sempre está relacionado a um *contexto de nomes*.
- Um contexto de nomes é um objeto ao qual pode-se atribuir um nome.
- Contextos de nomes são organizados através de um *grafo de nomes*, formando uma estrutura hierárquica.

... 2 - “CosNaming” e Formato de Nomes

- Esta construção fornece a noção de nome completo (“compound name”) - um conceito muito comum em diferentes sistemas:
  - por exemplo, em Unix um nome completo pode representar uma hierarquia de diretórios, do tipo “aaa/bbb/ccc” ;
  - sendo “aaa” e “bbb” diretórios e “ccc” podendo ser um diretório ou arquivo.
- No serviço de nomes, um nome completo tem uma forma mais abstrata: uma sequência IDL de componentes de nomes.

... 2 - “CosNaming” e Formato de Nomes

- Uma componente de nomes é definida como uma struct, “NameComponent”, da seguinte forma:

```
// IDL
typedef string Istring;
struct NameComponent {
   Istring id;
   Istring kind;
};
```

- onde, “id” representa o nome do componente;
- “kind” é usado para distinguir nomes que tem o mesmo “id” (geralmente usado na aplicação).

... 2 - “CosNaming” e Formato de Nomes

- Um nome é definido como uma sequência de componentes:

```
typedef sequence<NameComponent> Name;
```

- Além dos tipos e estrturas já definidos, o serviço de nomes especifica a estrutura “binding”:

```
enum BindingType {nobject, ncontext};
struct Binding {
  // NameComponent binding_name;
  Name binding_name;
  BindingType binding_type;
};
```

... 2 - “CosNaming” e Formato de Nomes

- “BindingType” distingue dois tipos de “bindings”, já que tanto objetos quanto contextos podem ser inseridos no grafo de contextos:
  - a interface “NamingContext” especifica a maioria dos serviços providos pelo serviço de nomes.
  - um objeto “NamingContext”, não exige um nó raiz, mas cada nome precisa ser único.

- Especificação da Interface IDL do Serviço de Nomes:

```
interface NamingContext {
  enum NotFoundReason {missing_node,not_context,not_object};
  exception NotFound { ... ... };
  exception CannotProceed { ... ... };
};
```

... 2 - "CosNaming" e Formato de Nomes

- É muito comum que as implementações de Serviços de Nomes contenha um contexto de nomes raiz, equivalente ao diretório "\" do sistema de arquivos do UNIX.
- A interface "NamingContext" fornece as seguintes operações para:
  - 1 - fazer o *bind* de um nome para uma referência de objeto,
  - 2 - resolver um nome para encontrar uma referência de objeto,
  - 3 - desfazer o bind entre um nome e objeto, e
  - 4 - listar os nomes dentro de um contexto.

3 - Resolução de Nomes

- A resolução de nomes é um processo pelo qual obtém-se uma referência de um objeto, buscando-o pelo nome.
  - ... a operação resolve(...) retorna a referência para o objeto registrado no servidor relativa ao nome passado como argumento - o objeto retornado é do tipo CORBA::Object_ptr.
  - ... portanto, este objeto deve sofrer um *downcasting* (ou *narrowing*) antes de ser usado pela aplicação.
- Já o método bind(...) cria um "binding" entre o nome o objeto no servidor de nomes.
- O método bind_context(...) realiza tarefa semelhante ao método bind, criando um *binding* entre um nome e um contexto específico.

... 3 - Resolução de Nomes

- O primeiro argumento passado para o método “bind” pode ser um nome completo do objeto - desta forma todas os componentes, à excessão do último, são usadas para encontrar o contexto onde o objeto será adicionado.
- O último componente especifica a referência do objeto no contexto desejado.
- Uma excessão é retornada se o nome já existe no contexto.

... 3 - Resolução de Nomes

- O método rebind(...) cria um “binding” entre o nome e o objeto que já está no contexto.
- O método rebind_context(...) cria um “binding” entre o nome e um contexto específico que já está no contexto.
- Já o método unbind(...) remove o “binding” entre um nome e um objeto que ele resolve.
- Para remove um objeto contexto e seu estado, usa-se destroy( ).

4 - Outros Métodos

- Dois métodos sao utilizadods para criar contextos de nomes:
  - new_context( ) - cria um novo contexto, sem entrada no grafo (i.e. sem *binding* para nenhum nome,
  - bind_new_context(...) - cria um novo contexto e realiza um *bind* com um nome especificado.
- O método destroy( ) remove o contexto de nomes no qual é invocado. O contexto alvo deve está vazio.
- O método list(...) obtém uma lista dos "bindings" (nomes) no contexto de nomes. A quantidade de "bindings" listados pode ser especificados no argumento do método list.

... 4 - Outros Métodos

- Dois métodos sao utilizadods para criar contextos de nomes:
  - new_context( ) - cria um novo contexto, sem entrada no grafo (i.e. sem *binding* para nenhum nome,
  - bind_new_context(...) - cria um novo contexto e realiza um *bind* com um nome especificado.
- O método destroy( ) remove o contexto de nomes no qual é invocado. O contexto alvo deve está vazio.
- O método list(...) obtém uma lista dos "bindings" (nomes) no contexto de nomes. A quantidade de "bindings" listados pode ser especificados no argumento do método list.

5 - Interface "BindIterator "

- Analisando a interface "NamingContext", vemos que o método list( ) retorna uma lista de "bindings".
- A interface "BindingIterator" pode ser utilizada para o acesso às entradas não-listadas no contexto de nomes.
- A especificação IDL é:

```
interface BindingIterator  {
  boolean next_one (out Binding b);
  boolean next_n
    (in unsigned long how_many,  out BindingList bl );
  void destroy ( );
};
```

... 5 - Interface "BindIterator "

- Os métodos next_one(...) e next_n( ) podem ser usados para o acesso às entradas não-listadas por list(...).
  - cada entrada irá sera retornada pelo menos uma vez.
  - o método next_one(...) retorna "true" se a entrada pode ser retornada e o "binding" associado é retornado. Caso contrário ela retorna "false".
  - O método next_n( ) retorna "true" se as n-entradas podem ser retornadas, caso contrário ela retorna "false". Em caso afirmativo, uma lista de *bindings* é retornada.
- O objeto "BindingIterator" pode ser removido por chamar a operação destroy( ).

6 - Considerações Finais

- O serviço de nomes provê uma forma elegante de resolver o problema de endereçamento em sistemas cliente/servidor, já que a relação nome/objeto estará acessivel em um servidor de nomes.
- Assim, o serviço de nomes evita que os clientes precisem conhecer os endereços de diversos servidores de objetos distribuídos.