3
O sistema SCS

Todos os mecanismos de adaptação propostos neste trabalho dependem de uma modularização adequada das aplicações para que possam ser aplicados. Além disso, a natureza tipicamente distribuída das aplicações que pretendemos atender, justifica que eles sejam oferecidos em conjunto com uma infra-estrutura básica de comunicação. Ao invés de implementar esses serviços partindo do zero, decidimos que seria mais adequado utilizar um sistema de componentes existente que, além de facilitar a organização dos módulos das aplicações em componentes reutilizáveis, oferecesse serviços essenciais de comunicação distribuída de forma transparente.

O sistema de componentes escolhido foi o SCS [25] (Sistema de Componentes de Software), um sistema também desenvolvido por nosso grupo, cujo objetivo é prover uma infra-estrutura de distribuição, instalação, configuração e execução de componentes que seja ao mesmo tempo flexível e simples de usar. O SCS usa CORBA como base para seu mecanismo de comunicação distribuída e conta com implementações tanto em Lua quanto em Java. A escolha deste sistema se mostrou adequada pois implementações *CORBA* vem sendo largamente utilizadas no mercado e já se mostraram soluções maduras e confiáveis. *CORBA* conta com implementações em diversas linguagens de programação, o que proporciona um certo grau de independência de linguagem ao processo de desenvolvimento de componentes. Finalmente, alguns trabalhos já foram desenvolvidos com sucesso no sentido de prover interoperabilidade entre componentes *CORBA* e componentes de outros sistemas, como COM e EJB (podemos citar como exemplos os sistemas *LuaORB* [4], *Vienna Component Framework* [40] e *Flexible Packaging* [41]). Esses trabalhos garantem a possibilidade de implementar também interoperabilidade entre componentes SCS e componentes desses sistemas.

O sistema SCS é claramente influenciado pelos modelos CCM (*CORBA Component Model*) e *OpenCOM*. As abstrações utilizadas para representar

os serviços oferecidos e dependências dos componentes são um subconjunto daquelas utilizadas no CCM e descritas na seção 2.2, sendo que o único tipo de conexão suportado é o tipo baseado em interface e as portas declaradas podem ser facetas ou receptáculos. Além disso, assim como no CCM, no SCS as portas dos componentes são descritas utilizando a linguagem de definição de interfaces OMG IDL.

O SCS disponibiliza um modelo dinâmico de configuração de aplicações sob a forma de operações que permitem estabelecer e remover conexões entre os componentes em tempo de execução. Esse modelo permite que sistemas em execução sejam inteiramente reconfigurados através da substituição de seus componentes. O modelo de configuração do SCS é dito de baixa granularidade pois não permite alterações na estrutura interna dos componentes, apenas reconexões entre eles. A maneira pela qual o SCS disponibiliza seus serviços básicos de gerência de componentes é bastante similar à utilizada pelo *OpenCOM*: Enquanto componentes *OpenCOM* disponibilizam operações de ciclo de vida e de conexão através das interfaces *ILifeCycle* e *IReceptacles*, componentes SCS disponibilizam as mesmas operações através da faceta *IComponent*. Esta faceta, como ilustra a listagem 3.1, oferece as seguintes operações:

- *startup*() e *shutdown*(): Invocadas, respectivamente, logo após a instanciação e imediatamente antes da remoção da instância do componente.
- *connect*(*in string receptacle*, *in Object obj*): Conecta um componente externo a um receptáculo específico do componente, retornando um identificador para a conexão estabelecida.
- *disconnect*(*in ConnectionId id*): Desconecta um componente externo de um receptáculo específico do componente, a partir do identificador da conexão.
- *getConnections*(*in string receptacle*): Obtém descritores de todas as conexões estabelecidas em um receptáculo do componentes. Os descritores de conexão são formados de um identificador e uma referência para o componente conectado.
- *getFacet*(*in string facetinterface*): Obtém uma referência a uma faceta oferecida pelo componente a partir do nome da interface implementada por ela.
- *getFacetByName*(*in string facet*): Obtém uma referência a uma faceta oferecida pelo componente a partir do seu nome.

De forma similar ao *OpenCOM* e ao CCM, os componentes SCS oferecem funcionalidades reflexivas básicas, que permitem inspecionar em tempo de execução os tipos das facetas e receptáculos que eles oferecem. A faceta através da qual essa funcionalidade é oferecida no SCS é homônima à sua interface equivalente no *OpenCOM*: *IMetaInterface*. As operações oferecidas por esta faceta, também listadas na listagem 3.1, são as seguintes:

- *getFacets*(): Obtém uma lista de descritores das facetas oferecidas pelo componente. O descritor de uma faceta é composto pelo seu nome, pelo nome da interface que ela implementa e por uma referência a ela.
- *getFacetsByName*(*in NameList names*): Obtém uma lista de descritores das facetas que são oferecidas pelo componente e que têm seu nome contido na lista de nomes fornecida como parâmetro.
- *getReceptacles*(): Obtém uma lista de descritores dos receptáculos declarados pelo componente. O descritor de um receptáculo é composto pelo seu nome, pelo nome da interface que ele implementa, por um indicador que sinaliza se ele aceita conexões múltiplas e por uma lista de descritores das conexões estabelecidas.
- *getReceptaclesByName*(*in NameList names*): Obtém uma lista de descritores dos receptáculos que são declarados pelo componente e que tem seu nome contido na lista de nomes fornecida como parâmetro.

```
1  module SCS {
2          exception StartupFailed {};
3          exception ShutdownFailed {};
4          exception InvalidName {
5                  string name;
6          };
7          exception InvalidConnection {};
8          exception AlreadyConnected {};
9          exception ExceededConnectionLimit {};
10         exception NoConnection {};
11
12         typedef unsigned long ConnectionId;
13         typedef sequence<string> NameList;
14         typedef sequence<octet> OctetSeq;
15
16         struct FacetDescription {
17                 string name;
18                 string interface_name;
19                 Object facet_ref;
20         };
```

```
21          typedef sequence<FacetDescription> FacetDescriptions;
22
23          struct ConnectionDescription {
24                  ConnectionId id;
25                  Object objref;
26          };
27          typedef sequence<ConnectionDescription>
28                  ConnectionDescriptions;
29
30          struct ReceptacleDescription {
31                  string name;
32                  string interface_name;
33                  boolean is_multiplex;
34                  ConnectionDescriptions connections;
35          };
36          typedef sequence<ReceptacleDescription>
37                  ReceptacleDescriptions;
38
39          interface IComponent {
40                  void startup() raises (StartupFailed);
41                  void shutdown() raises (ShutdownFailed);
42                  Object getFacet(in string facet_interface);
43                  Object getFacetByName(in string facet);
44
45                  ConnectionId connect(in string receptacle,
46                          in Object obj)
47                          raises (InvalidName,
48                                  InvalidConnection,
49                                  AlreadyConnected,
50                                  ExceededConnectionLimit);
51                  void disconnect(in ConnectionId id)
52                          raises (InvalidConnection,
53                                  NoConnection);
54                  ConnectionDescriptions getConnections(
55                          in string receptacle)
56                          raises (InvalidName);
57          };
58          interface IMetaInterface {
59                  FacetDescriptions getFacets();
60                  FacetDescriptions getFacetsByName(
61                          in NameList names)
62                          raises (InvalidName);
63                  ReceptacleDescriptions getReceptacles();
64                  ReceptacleDescriptions getReceptaclesByName(
65                          in NameList names)
66                          raises (InvalidName);
67          };
```

```
68 };
```

Listing 3.1: Interfaces *SCS::IComponent* e *SCS::IMetaInterface*

Além do modelo de componentes descrito, o SCS define um conjunto de serviços que tem como objetivo permitir o gerenciamento e configuração de suas aplicações. Esses serviços são disponibilizados no ambiente de execução sob a forma de componentes comuns. Maiores detalhes os serviços do SCS podem ser encontrados no relatório técnico [25].

3.1 SCS Java

Na implementação Java convencional do SCS, as facetas dos componentes são implementadas por objetos CORBA. Para implementar esses objetos, o desenvolvedor descreve a interface das facetas em IDL e utiliza uma ferramenta para gerar o código dos esqueletos dos objetos. Dentre as classes geradas para um esqueleto, está uma classe que declara métodos abstratos como os da interface da faceta e deve ser estendida pelo desenvolvedor para fornecer a implementação desses métodos. Além da implementação das facetas, o desenvolvedor deve fornecer também classes com a implementação das funcionalidades básicas do componente, como a inicialização e finalização do componente e a criação de suas facetas e receptáculos.

Para ilustrar o uso dessa versão do SCS, descrevemos a seguir a implementação de um exemplo simples, composto apenas pelo componente *FooBar*. O primeiro passo é a definição das interfaces das facetas do componente em IDL. No exemplo utilizaremos duas facetas *Foo* e *Bar*, cuja interface é exibida na listagem 3.2.

```
1 module foobar {
2         interface Foo {
3                 string foo();
4         };
5         interface Bar {
6                 string bar();
7         };
8 };
```

Listing 3.2: Interface das facetas *Foo* e *Bar*

A seguir, utilizamos a ferramenta de geração de código de um ORB Java para gerar esqueletos para essas facetas e fornecemos classes com a implementação delas estendendo as classes geradas *FooPOA* e *BarPOA* (listagens 3.3 e 3.4).

```
1 package foobar.component;
2 import foobar.FooPOA;
3
4 public class FooServant extends FooPOA {
5         public String foo() {
6                 return "foo";
7         }
8 }
```

Listing 3.3: Implementação da faceta *Foo*

```
1 package foobar.component;
2 import foobar.BarPOA;
3
4 public class BarServant extends BarPOA {
5         public String bar() {
6                 return "bar";
7         }
8 }
```

Listing 3.4: Implementação da faceta *Bar*

Para finalizar a implementação do componente, é necessário implementar as funcionalidades básicas do componente. Como o componente do exemplo conta apenas com facetas, precisamos apenas implementar os métodos de inicialização e finalização do componente e o método de criação de facetas, que são declarados na classe abstrata *IComponentServant*, fornecida pela implementação do SCS Java. A listagem 3.5 exibe a implementação do componente *FooBar*. O método *createFacets* (linha 17) cria uma lista com os descritores das facetas *Foo* e *Bar*, sendo que cada descritor é composto pelo nome da faceta, o nome de sua interface e uma referência para o objeto com a sua implementação. Os métodos de inicialização e finalização (*doStartup* e *doShutdown*, linhas 41 e 46) no exemplo não realizam nenhuma operação. Finalmente, os métodos *getFoo* e *getBar* (linhas 50 e 65, respectivamente) instanciam os objetos CORBA que implementam as facetas *Foo* e *Bar*. Esses métodos não são declarados na classe *IComponentServant*.

```
1 package foobar.component;
2
3 import java.util.ArrayList;
```

```
4  import org.omg.CORBA.Object;
5  import org.omg.PortableServer.POAPackage.ServantNotActive;
6  import org.omg.PortableServer.POAPackage.WrongPolicy;
7  import scs.core.FacetDescription;
8  import scs.core.servant.IComponentServant;
9  import foobar.Bar;
10 import foobar.BarHelper;
11 import foobar.Foo;
12 import foobar.FooHelper;
13
14 public class FooBarComponent extends IComponentServant{
15
16         @Override
17         protected ArrayList<FacetDescription> createFacets() {
18                 ArrayList<FacetDescription> facets = new
19                 ArrayList<FacetDescription>();
20
21                 FacetDescription fooFacetDescription =
22                         new FacetDescription();
23                 fooFacetDescription.interface_name =
24                         "foobar::Foo";
25                 fooFacetDescription.name = "Foo";
26                 fooFacetDescription.facet_ref = getFoo();
27                 facets.add(fooFacetDescription);
28
29                 FacetDescription barFacetDescription =
30                         new FacetDescription();
31                 barFacetDescription.interface_name =
32                         "foobar::Bar";
33                 barFacetDescription.name = "Bar";
34                 barFacetDescription.facet_ref = getBar();
35                 facets.add(barFacetDescription);
36
37                 return facets;
38         }
39
40         @Override
41         protected boolean doStartup() {
42                 return true;
43         }
44
45         @Override
46         protected boolean doShutdown() {
47                 return true;
48         }
49
50         private Object getFoo() {
```

```
51                  try {
52                          FooServant fooServant = new FooServant();
53                          Foo foo = FooHelper.narrow(
54                                  this._poa().
55                                  servant_to_reference(fooServant));
56                          return foo;
57                  } catch (ServantNotActive e) {
58                          e.printStackTrace();
59                  } catch (WrongPolicy e) {
60                          e.printStackTrace();
61                  }
62                  return null;
63          }
64
65          private Object getBar() {
66                  try {
67                          BarServant barServant = new BarServant();
68                          Bar bar = BarHelper.narrow(
69                                  this._poa().
70                                  servant_to_reference(barServant));
71                          return bar;
72                  } catch (ServantNotActive e) {
73                          e.printStackTrace();
74                  } catch (WrongPolicy e) {
75                          e.printStackTrace();
76                  }
77                  return null;
78          }
79  }
```

Listing 3.5: Implementação do componente *FooBar*

A instanciação do componente é efetuada através dos serviços de gerência de ambiente de execução mencionados no fim da seção anterior. Esse processo não será exibido pois como esses serviços não foram implementados neste trabalho, sua análise foge do escopo do capítulo. A listagem 3.6 mostra um cliente que consome o componente *FooBar*. Esse cliente é implementado como um cliente de um objeto CORBA comum:

- Na linha 2, lemos o IOR do objeto CORBA que implementa a faceta *IComponent* do componente *FooBar* de um arquivo.
- Na linha 3, obtemos uma referência para esse objeto a partir do IOR lido.
- Na linha 4, utilizamos a classe utilitária *IComponentHelper* para converter esse objeto para a interface *IComponent*. Tanto a classe utilitária

quanto a interface foram geradas por um ORB Java a partir da IDL do sistema SCS.

– Nas linhas 6 e 7, chamamos o método *getFacetByName*, definido na interface *IComponent*, passando os nomes das facetas *Foo* e *Bar* para obter referências para os objetos CORBA que as implementam.

– Nas linhas 8 e 9, utilizamos as classes utilitárias *FooHelper* e *BarHelper* para converter esses objetos para as interfaces *Foo* e *Bar*. Tanto as classes utilitária quanto as interfaces foram geradas por um ORB Java a partir da IDL do componente *FooBar*.

– Nas linhas 11 e 12, efetuamos chamadas às duas facetas do componente, imprimindo o resultado no console.

```
1  ORB orb = ORB.init(new String[] {}, null);
2  String ior = FileUtil.read("foobar.ior");
3  org.omg.CORBA.Object obj = orb.string_to_object(ior);
4  IComponent component = IComponentHelper.narrow(obj);
5
6  org.omg.CORBA.Object fooObj = component.getFacetByName("Foo");
7  org.omg.CORBA.Object barObj = component.getFacetByName("Bar");
8  Foo foo = FooHelper.narrow(fooObj);
9  Bar bar = BarHelper.narrow(barObj);
10
11 System.out.println(foo.foo());
12 System.out.println(bar.bar());
```

Listing 3.6: Cliente das facetas *Foo* e *Bar*