Tutorial Básico do SDK Java do SCS

Tecgraf
Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio)
scs-users@tecgraf.puc-rio.br

1 Introdução

Este documento é um tutorial básico sobre a criação de componentes no modelo SCS v1.3.0, utilizando a versão Java da implementação padrão. Não serão encontradas aqui explicações sobre o modelo, as quais encontram-se em documentos específicos. Também não será abordado o uso de serviços específicos desenvolvidos para o auxílio ao uso do modelo, como a infra-estrutura de execução. Essas informações também podem ser obtidas em outros documentos. A implementação Java baseia-se na versão 1.5 da máquina virtual Java e em CORBA v2.3, representada pelo ORB Jacorb v2.3.0 que está incluso na implementação padrão. Este documento assume que o leitor é familiarizado a conceitos de desenvolvimento de *software* baseado em componentes e à terminologia CORBA.

2 Inicialização do ORB

Para a criação e execução do código de um componente, é necessária a inicialização prévia de um ORB. A instância de ORB criada será passada posteriormente para o construtor de um componente SCS. Esse procedimento é ilustrado no Código 1.

Código 1: Criação do ORB

```
1
2 public static void main(String[] args) {
3   Properties props = new Properties();
4   ORB orb = ORB.init(args, props);
5 }
```

3 Passos Necessários à Criação de um Componente

Aqui serão descritos os passos mínimos necessários para a criação de um componente SCS.

3.1 Definição do Identificador do Componente

O identificador do componente é uma estrutura definida em IDL (scs.idl) chamada *ComponentId*, e representada em Java pela classe *ComponentId*. Um identificador de componente conta com os seguintes campos:

- `name`: Nome desejado para o componente.
- `major_version`: Número que define a versão principal do componente.
- `minor_version`: Número que define a versão secundária do componente, possivelmente relacionado a uma sub-versão da versão principal.
- `patch_version`: Número que define a versão de revisão do componente.
- `platform_spec`: *String* contendo quaisquer especificações de plataforma necessárias ao funcionamento do componente.

Os números de versão do componente, quando unificados, devem ser separados por pontos. Ou seja, um componente com versão principal 1, versão secundária 0 e versão de revisão 0 deve ser representado como a *String* "1.0.0".

3.2 Criação do Componente Básico

Todo componente SCS é representado por seu "contexto". Um Contexto de Componente atua como um invólucro local para as facetas e receptáculos de um componente SCS.

O contexto é implementado pela classe scs.core.ComponentContext e seu processo de instanciação engloba a criação de implementações padronizadas para as três facetas básicas: IComponent, IReceptacles e IMetaInterface. Caso o usuário precise utilizar uma implementação diferente de alguma dessas facetas, existe no contexto um método para a atualização de facetas chamado *updateFacet*, descrito na Seção 2.

Como o contexto é quem cria os objetos CORBA, é necessário que tenha acesso ao ORB logo em sua construção, para que possa inserir as facetas básicas e também facetas adicionais, posteriormente. Outro parâmetro obrigatório é o Identificador do Componente (3.1).

Um exemplo de código para a criação de um componente básico pode ser visto no Código 2.

3.3 Criação de Facetas

Facetas são interfaces CORBA, e devem ser implementadas por classes definidas pelo usuário, como exigido pelas definições Java desse padrão. No entanto, a implementação SCS-Java exige também que facetas implementem um construtor que receba um *ComponentContext* (Contexto de Componentes). O Contexto pode ser utilizado para acessar outras facetas e o identificador do componente, entre outros dados, como descrito na Seção **??**. Um exemplo pode ser conferido na Listagem **??**.

Código 2: Implementação de uma Faceta MyFacet

```
1 public class MyFacetServant extends MyFacetPOA {
2
3   private ComponentContext myComponent;
4
5   public MyFacetServant(ComponentContext myComponent) {
6     super();
7     this.myComponent = myComponent;
8   }
9 }
```

O SCS-Java exige ainda que facetas implementem o método _get_component() de CORBA, definido pelo Jacorb, mas o Contexto já fornece um método para suprir essa funcionalidade. Assim, a implementação torna-se trivial e deve ser sempre a mesma, que está exposta na Listagem 3. Existem planos para que esse método seja inserido automaticamente em todas as facetas no futuro, no ato da instanciação do componente.

Código 3: O Método CORBA _get_component()

```
1  @Override
2  public org.omg.CORBA.Object _get_component() {
3    return this.myComponent.getIComponent();
4  }
```

Obviamente, facetas devem ainda implementar seus métodos definidos em IDL.

3.4 Utilização do Construtor de Componentes

O Construtor de Componentes fornece uma API para a criação ou "instanciação"de um novo componente. Por instância de componente, nos referimos a um Contexto de Componente com todas as suas facetas, receptáculos e objetos internos criados, e seus mapas preenchidos. O processo de instanciação engloba a criação de descrições de facetas, a instanciação das classes que implementam as facetas, a criação de objetos CORBA referentes às facetas e a criação de receptáculos.

Para realizar esse trabalho, é necessário informar ao Construtor de Componentes quais facetas e receptáculos fazem parte do componente. Esses dados são representados por descrições "estendidas"de facetas e descrições de receptáculos. As descrições estendidas de facetas recebem este nome pois já existe o termo *FacetDescription* na IDL do modelo SCS. No entanto, o tipo *ExtendedFacetDescription* não está definido em IDL, pois esta é uma classe existente somente na implementação SCS-Java. Descrições estendidas são compostas pelo nome da faceta, o nome da interface da faceta e o nome completo (incluindo pacote) da classe que a implementa, para que o objeto que representa a faceta possa ser instanciado.

Descrições de receptáculos são iguais às definidas em IDL, com o único detalhe de que, em sua criação, não é necessário fornecer o *array connections*, podendo-se passar *NULL*. Todo receptáculo é representado por uma instância da mesma classe, *Receptacle*, a qual já está implementada.

O método do Construtor de Componentes para a criação / instanciação de um novo

componente chama-se *newComponent*. Esse método recebe um *array* de *ExtendedFacetDescription*, um *array* de *ReceptacleDescription* e o identificador do componente. Em certos casos, pode ser importante para o componente que uma faceta seja carregada antes de outras. Por isso, o Construtor de Componentes instanciará as facetas na ordem do *array*.

Não é necessário inserir as facetas básicas (IComponent, IReceptacles e IMetaInterface) no *array* de *ExtendedFacetDescription*. O Construtor automaticamente insere essas facetas e suas descrições no componente. Caso o usuário deseje utilizar uma implementação diferente de alguma dessas facetas, basta inserir a descrição apropriada que essa será utilizada no lugar da padrão.

Um outro detalhe é que, como o Construtor cria os objetos CORBA, é necessário que tenha acesso ao ORB. Para que os componentes tenham acesso ao ORB e POA utilizados, o Construtor fornece métodos para o acesso a eles. Todo Contexto de Componente guarda uma referência para o Construtor que o criou, e assim facetas podem acessá-lo para obter uma referência ao ORB ou POA, conforme suas necessidades.

Um exemplo de código para a criação dos *arrays* de descrições, identificador do componente e utilização da API pode ser visto na Listagem 4.

Código 4: Instanciação de um Novo Componente

```
1  ORB orb = ...; // referência para o ORB, já inicializado
2  POA poa = ...; // referência para o POA
3
4  // facet descriptions
5  ExtendedFacetDescription[] facetDescs = new ExtendedFacetDescription[1];
6  facetDescs[0] =
7    new ExtendedFacetDescription("MyFacet", "IDL:mymodule/MyFacet:1.0",
8    "mypackage.MyFacetServant");
9
10 // receptacle descriptions
11 ReceptacleDescription[] receptacleDescs = new ReceptacleDescription[1];
12 receptacleDescs[0] =
13   new ReceptacleDescription("MyReceptacle",
14     "IDL:expectedmodule/ExpectedFacet:1.0", true, null);
15
16 // component id
17 ComponentId cpId = new ComponentId("MyComponent", (byte)1, (byte)0, (byte)0,
18   "none");
19
20 // utilização da API
21 ComponentBuilder builder = new ComponentBuilder(poa, orb);
22 ComponentContext newInstance = builder.newComponent(facetDescs, receptacleDescs,
23   cpId);
```

4 Exemplo Completo

Demonstraremos aqui o uso mais simples para um componente: apenas uma faceta além das três facetas básicas. Não será criado nenhum receptáculo, apesar da existência da faceta IReceptacles. Exemplos mais complexos poderão ser encontrados nas *demos* do projeto.

Esta demonstração será baseada na demo *Hello*, que implementa um componente carregável em contêiner (parte da infra-estrutura de execução). O código apresentado a seguir é uma versão modificada dessa demo, para que possa ser carregado manualmente, sem o uso de um contêiner.

O componente Hello tem quatro interfaces: IComponent, IReceptacles, IMetaInterface e apenas uma interface própria, de nome Hello. Sua IDL está disponível na Listagem 5.

Código 5: IDL do Componente Hello

```
1 module scs{
2   module demos{
3     module helloworld {
4       interface Hello {
5         void sayHello();
6       };
7     };
8   };
9 };
```

Para implementar a faceta Hello, que conta com apenas um método, *sayHello*, criamos a classe HelloServant.java, que pode ser visualizada na Listagem 6. O código é bastante similar ao apresentado nas Listagens **??** e 3.

Além da implementação da faceta, é necessário um código de criação de componente. Esse código, que tipicamente será incluído na função *main* do programa, é muito similar ao da Listagem 4 e pode ser conferido na Listagem 7.

Por fim, temos o código "cliente", que acessa o componente. Note que esse código pode ser CORBA puro, não é necessária a criação de um componente para acessar outro componente. Um exemplo desse tipo de código pode ser visto na Listagem 8.

Neste exemplo, a mensagem “Hello User!” será exibida somente na máquina servidor. O código cliente apenas terá a chamada *sayHello()* completada corretamente e

Código 6: A Faceta Hello

```
1  package scs.demos.helloworld.servant;
2
3  import scs.core.servant.ComponentContext;
4  import scs.demos.helloworld.HelloPOA;
5
6  public class HelloServant extends HelloPOA {
7
8    private String name = "World";
9    private ComponentContext myComponent;
10
11   public HelloServant(ComponentContext myComponent) {
12     super();
13     this.myComponent = myComponent;
14   }
15
16   public void setName(String name) {
17     this.name = name;
18   }
19
20   public void sayHello() {
21     System.out.println("Hello " + name + "!");
22   }
23
24   @Override
25   public org.omg.CORBA.Object _get_component() {
26     return myComponent.getIComponent();
27   }
28 }
```

será finalizado sem erros.

5 Notas sobre os procedimentos de compilação e execução

É importante ressaltar que o JacORB exige:

- compilar (tanto o código servidor quanto o código cliente) utilizando o parâmetro **-Xbootclasspath/p:<localização do jacorb.jar>** no compilador Java (por exemplo, no Eclipse basta colocar o JAR do JacORB com prioridade maior que o JRE Runtime).
- executar (tanto o código servidor quanto o código cliente) passando alguns parâmetros na JVM, conforme mostra a Listagem 9. Esses parâmetros garantem o

uso da implementação CORBA do JacORB ao invés do uso da implementação básica provida pela Sun.

Código 7: Criação do Componente Hello

```
1  public static void main(String[] args) {
2    try {
3      // inicialização do ORB
4      Properties props = new Properties();
5      // porta e host apenas para fins do exemplo
6      props.put("org.omg.CORBA.ORBInitialPort", "1050");
7      props.put("org.omg.CORBA.ORBInitialHost", "localhost");
8
9      ORB orb = ORB.init(args, props);
10
11     POA poa = POAHelper.narrow(orb.resolve_initial_references("RootPOA"));
12     poa.the_POAManager().activate();
13
14     // criação de descrições de facetas e receptáculos
15     ExtendedFacetDescription[] extDescs = new ExtendedFacetDescription[1];
16     ReceptacleDescription[] recepDescs = new ReceptacleDescription[0];
17     extDescs[0] =
18       new ExtendedFacetDescription("Hello",
19         "IDL:scs/demos/helloworld/Hello:1.0",
20         "scs.demos.helloworld.servant.HelloServant");
21
22     // criação do ComponentId
23     ComponentId cpId =
24       new ComponentId("Hello", (byte)1, (byte)0, (byte)0, "none");
25
26     // criação do Construtor de Componentes e instanciação do componente
27     ComponentBuilder builder = new ComponentBuilder(poa, orb);
28     ComponentContext instance = builder.newComponent(extDescs, recepDescs, cpId);
29
30     // modificação do nome a ser exibido na mensagem da faceta Hello
31     HelloServant helloImpl =
32       (HelloServant) context.getFacets().get(FACET_HELLO);
33     helloImpl.setName("User");
34
35     // publicação do IOR para que a faceta Hello do componente possa ser
36     // encontrada. Observação: podemos exportar qualquer faceta, pois temos
37     // o método _get_component para obter a faceta IComponent e, com ela,
38     // pode-se obter outras facetas(esse passo pode ser substituído por outras
39     // formas de publicação, como a publicação em um serviço de nomes, por
40     // exemplo).
41     org.omg.CORBA.Object helloObj =
42       instance.getFacetDescs().get("Hello").facet_ref;
43     String helloIOR = orb.object_to_string(helloObj);
44     FileWriter file = new FileWriter("hello.ior");
45     file.write(helloIOR);
46     file.flush();
47     file.close();
48
49     // instrução ao ORB para que aguarde por chamadas remotas
50     orb.run();
51   }
52   catch (Exception e) {
53     e.printStackTrace();
54     System.exit(1);
55   }
56 }
```

Código 8: Utilização do Componente Hello

```
1  public static void main(String[] args) {
2    try {
3      // inicialização do ORB
4      Properties props = new Properties();
5      // porta e host apenas para fins do exemplo
6      props.put("org.omg.CORBA.ORBInitialPort", "1051");
7      props.put("org.omg.CORBA.ORBInitialHost", "localhost");
8
9      ORB orb = ORB.init(args, props);
10
11     POA poa = POAHelper.narrow(orb.resolve_initial_references("RootPOA"));
12     poa.the_POAManager().activate();
13
14     // assume-se que o arquivo que contém o IOR (publicado na listagem
15     // anterior) esteja disponível. O arquivo pode ter sido criado em
16     // outra máquina e, nesse caso, tem de ser copiado manualmente
17     // (pode-se também utilizar um método diferente de publicação,
18     // como um serviço de nomes).
19     BufferedReader in = new BufferedReader(new FileReader("hello.ior"));
20     String iHelloIOR = in.readLine();
21
22     // obtenção das facetas Hello e IComponent
23     // precisamos utilizar o método narrow pois estamos recebendo um
24     // org.omg.CORBA.Object
25     Hello iHelloFacet = HelloHelper.narrow(orb.string_to_object(iHelloIOR));
26     IComponent icFacet = IComponentHelper.narrow(iHelloFacet._get_component());
27
28     // inicialização do componente.
29     icFacet.startup();
30
31     // com o componente inicializado, podemos utilizá-lo à vontade.
32     // note que o método setName da classe HelloServant não pode ser utilizado
33     // remotamente, pois não está definido em IDL.
34     iHelloFacet.sayHello();
35   }
36   catch (Exception e) {
37     e.printStackTrace();
38     System.exit(1);
39   }
40 }
```

Código 9: Parâmetros na Sun JVM para forçar o uso do JacORB

```
1 -Djava.endorsed.dirs=<diretório contendo o JAR do jacorb> \
2  -Djacorb.home=<diretório contendo o JAR do jacorb> \
3  -Dorg.omg.CORBA.ORBClass=org.jacorb.orb.ORB \
4  -Dorg.omg.CORBA.ORBSingletonClass=org.jacorb.orb.ORBSingleton
```