SCS-Java - Tutorial - Básico

C. Augusto, R.Cerqueira
Tecgraf
Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio)
rcerq@inf.puc-rio.br

1 Introdução

Este documento é um tutorial básico sobre a criação de componentes no modelo SCS, 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 SCS-Java baseia-se na versão 1.6 da máquina virtual Java e em CORBA v2.3, representada pelo ORB Jacorb 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, que deverá ser repassado a um *ComponentBuilder*. Esse procedimento será demonstrado em mais detalhes adiante.

3 Contexto de Componente

Todo componente SCS-Java é representado por seu "contexto". Um Contexto de Componente (classe *ComponentContext*) atua como um envólucro para as facetas e receptáculos de um componente SCS, e fornece acesso interno a essas mesmas facetas e receptáculos. Além disso, concentra também o acesso ao identificador do componente, ao objeto CORBA da faceta *IComponent* e ao Construtor de Componentes utilizado para sua construção (mais detalhes na Seção 4.3). Para tal, são disponibilizados alguns mapas e objetos:

- `Map<String, FacetDescription> FacetDescriptions`: Mapa para as descrições de facetas, definidas em IDL. O campo facet_ref provê o objeto CORBA da faceta específica. Essas descrições são criadas automaticamente pelo Construtor de Componentes (ver Seção 4.3). Indexado pelo nome da faceta.
- `Map<String, ExtendedFacetDescription> ExtendedFacetDescriptions`: Mapa para as descrições extendidas de facetas. Essas descrições são fornecidas pelo criador do componente (exceto para as facetas básicas), e incluem a classe implementadora da faceta. Objetos de descrições extendidas são apenas Java, pois não estão definidos em IDL. Indexado pelo nome da faceta.
- `Map<String, ReceptacleDescription> ReceptacleDescriptions`: Mapa para as descrições de receptáculos, definidas em IDL. Indexado pelo nome do receptáculo.
- `Map<String, Object> Facets`: Mapa para os *servants* das facetas, que são objetos Java. Indexado pelo nome da faceta.
- `Map<String, Receptacle> Receptacles`: Mapa para os objetos Java que representam os receptáculos. Indexado pelo nome do receptáculo.
- `ComponentId`: Identificador do componente, definido em IDL.
- `ComponentBuilder`: Referência para o Construtor de Componentes utilizado para criar o contexto, facetas e receptáculos. O Construtor de Componentes, por sua vez, mantém referências para o ORB e o POA utilizados.

4 Passos Necessários à Criação de um Componente

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

4.1 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`: Octeto que define o número principal da versão do componente.
- `minor_version`: Octeto que define a versão secundária do componente, possivelmente relacionado a uma sub-versão da versão principal.
- `patch_version`: Octeto 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".

4.2 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

Listing 1: 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 }
```

acessar outras facetas e o identificador do componente, entre outros dados, como descrito na Seção 3. Um exemplo pode ser conferido na Listagem 1.

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 2. Existem planos para que esse método seja inserido automaticamente em todas as facetas no futuro, no ato da instanciação do componente.

Listing 2: 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.

4.3 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 *ExtendedFacetDescriptions* 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 *ExtendedFacetDescriptions*, um *array* de *ReceptacleDescriptions* 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 *ExtendedFacetDescriptions*. 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 3.

Listing 3: 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[2];
6
7  ExtendedFacetDescription componentExtDesc =
8    new ExtendedFacetDescription("IComponent", "IDL:scs/core/IComponent:1.0",
9    "scs.core.servant.IComponentServant");
10 ExtendedFacetDescription myFacetExtDesc =
11   new ExtendedFacetDescription("MyFacet", "IDL:mymodule/MyFacet:1.0",
12   "mypackage.MyFacetServant");
13
14 facetDescs[0] = componentExtDesc;
15 facetDescs[1] = myFacetExtDesc;
16
17 // receptacle descriptions
18 ReceptacleDescription[] receptacleDescs = new ReceptacleDescription[1];
19
20 ReceptacleDescription myReceptDesc =
21   new ReceptacleDescription("MyReceptacle",
22     "IDL:expectedmodule/ExpectedFacet:1.0", true, null);
23
24 receptacleDescs[0] = myReceptDesc;
25
26 // component id
27 ComponentId cpId = new ComponentId("MyComponent", (byte)1, (byte)0, (byte)0,
28   "none");
29
30 // utilização da API
31 ComponentBuilder builder = new ComponentBuilder(poa, orb);
32 ComponentContext newInstance = builder.newComponent(facetDescs, receptacleDescs,
33   cpId);
```

5 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 IHello. Sua IDL está disponível na

listagem 4.

Listing 4: IDL do Componente Hello

```
module scs{
  module demos{
    module helloworld {
      interface IHello {
        void sayHello();
      };
    };
  };
};
```

Para implementar a faceta IHello, que conta com apenas um método, *sayHello*, criamos a classe HelloServant.java, que pode ser visualizada na listagem 5. O código é bastante similar ao apresentado nas listagens 1 e 2.

Listing 5: A Faceta IHello

```
package scs.demos.helloworld.servant;

import scs.core.servant.ComponentContext;
import scs.demos.helloworld.IHelloPOA;

public class HelloServant extends IHelloPOA {

  private String name = "World";
  private ComponentContext myComponent;

  public HelloServant(ComponentContext myComponent) {
    super();
    this.myComponent = myComponent;
  }

  public void setName(String name) {
    this.name = name;
  }

  @Override
  public void sayHello() {
    System.out.println("Hello " + name + "!");
  }

  @Override
  public org.omg.CORBA.Object _get_component() {
    return myComponent.getIComponent();
  }
}
```

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 3 e pode ser conferido na Listagem 6.

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 7.

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

Listing 6: 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("IHello",
19         "IDL:scs/demos/helloworld/IHello:1.0",
20         "scs.demos.hello.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     instance.getFacets().get("IHello").setName("User");
32
33     // publicação do IOR para que a faceta IHello do componente possa ser
34     // encontrada. Observação: podemos exportar qualquer faceta, pois temos
35     // o método _get_component para obter a faceta IComponent e, com ela,
36     // pode-se obter outras facetas(esse passo pode ser substituído por outras
37     // formas de publicação, como a publicação em um serviço de nomes, por
38     // exemplo).
39     // Assume-se que writeToFS seja uma função que escreva uma String em
40     // um arquivo.
41     Hello helloObj = instance.getFacetDescriptions().get("IHello").facet_ref;
42     String helloIOR = orb.object_to_string(helloObj);
43     writeToFS(helloIOR, "hello.ior");
44
45     // instrução ao ORB para que aguarde por chamadas remotas
46     orb.run();
47   }
48   catch (Exception e) {
49     e.printStackTrace();
50     System.exit(1);
51   }
52 }
```

Listing 7: 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     // Assume-se que readFromFile seja uma função que leia o conteúdo
20     // de um arquivo para uma String.
21     String helloIOR = readFromFile("hello.ior");
22
23     // obtenção das facetas IHello e IComponent
24     // precisamos utilizar o método narrow pois estamos recebendo um
25     // org.omg.CORBA.Object
26     IHello helloFacet = IHelloHelper.narrow(orb.string_to_object(helloIOR));
27     IComponent icFacet = IComponentHelper.narrow(helloFacet._get_component());
28
29     // inicialização do componente.
30     icFacet.startup();
31
32     // com o componente inicializado, podemos utilizá-lo à vontade.
33     // note que o método setName da classe HelloServant não pode ser utilizado
34     // remotamente, pois não está definido em IDL.
35     helloFacet.sayHello();
36   }
37   catch (Exception e) {
38     e.printStackTrace();
39     System.exit(1);
40   }
41 }
```