SCS-Lua - 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 v1.0.0, utilizando a versão Lua 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-Lua baseia-se na versão 5.1 da máquina virtual Lua e em CORBA v2.3, representada pelo ORB OiL v0.4. 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 armazenado em uma variável de nome "*orb*"dentro do módulo *oil*. O processo deve ser feito de acordo com o código do Código 1.

Componentes posteriormente podem realizar um *require* no OiL e acessar o ORB em *oil.orb*.

Código 1: Passagem do ORB para a Biblioteca e Componentes

```
1  local oil = require "oil"
2  local orb = oil.init()
3  oil.orb = orb
```

Essa forma de armazenamento e obtenção do ORB será modificada em versões futuras.

3 Contexto de Componente

Todo componente SCS-Lua é representado por seu "contexto", que é a tabela Lua retornada após a criação de um novo componente. Um Contexto de Componente atua como um envólucro para as facetas e receptáculos de um componente SCS. Para cada faceta e receptáculo, é criada uma tabela de mesmo nome dentro do contexto. Por isso, não é possível utilizar o mesmo nome para uma faceta e um receptáculo. Dessa forma, assumindo-se um componente de nome "component"que tenha uma faceta de nome "foo"e um receptáculo de nome "bar", pode-se acessá-los com o código do Código 2.

Código 2: Acesso Local a Facetas ou Receptáculos de Um Componente

```
1  -- deve-se previamente obter um contexto ou criar um novo componente
2  local component = ...
3  -- acessar facetas ou receptáculos
4  local fooFacet = component.foo
5  local barReceptacle = component.bar
```

Além disso, contextos concentram também o acesso ao identificador do componente e às descrições de facetas e receptáculos. Para tal, são disponibilizadas algumas variáveis:

- `_facetDescs`: Tabela para as descrições de facetas, definidas em IDL. O campo facet_ref provê o objeto CORBA da faceta específica. Indexada pelo nome da faceta. Pode conter o campo extra "key"para designar a chave do endereço corbaloc, caso tenha sido fornecido pelo usuário no ato da criação do componente.
- `_receptacleDescs`: Tabela para as descrições de receptáculos, definidas em IDL. Indexada pelo nome do receptáculo.
- `_componentId`: Tabela com o identificador do componente, definido em IDL.

4 Passos Necessários à Criação de um Componente

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

4.1 Definição do Identificador do Componente

O identificador do componente é uma estrutura definida em IDL (scs.idl) chamada ComponentId, e representada em Lua por uma tabela com os respectivos campos preenchidos. 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".

4.2 Criação de Facetas

Facetas são interfaces CORBA, e devem ser implementadas pelo usuário, como exigido pelas definições Lua desse padrão. Na implementação SCS-Lua, facetas devem ser tabelas do tipo *callable*, ou seja, que possam ser executadas como uma função. Essa execução deve retornar uma nova instância da faceta. Para facilitar esse processo, pode ser utilizada a biblioteca LOOP, que facilita o uso do paradigma de orientação a objetos em Lua.

No ato da criação de um componente, será automaticamente inserido um campo "context"em todas as facetas, com uma referência para o contexto do seu componente. O Contexto pode ser utilizado para acessar outras facetas e o identificador do componente, entre outros dados, como descrito na Seção 3. Um exemplo pode ser conferido no Código 3.

Código 3: Implementação de uma Faceta MyFacet

```
local oo = require "loop.base"

local MyFacet = oo.class{}
function MyFacet:__init()
  return oo.rawnew(self, {})
end

function MyFacet:myMethod()
  -- como acessar o contexto da instância de componente ao qual essa
  -- faceta pertence
  local context = self.context
  -- como acessar e usar outras facetas da mesma instância de componente
  local anotherFacet = context.AnotherFacet
  anotherFacet:anotherMethod()
end
```

O SCS-Lua exige ainda que facetas implementem o método *_component()* de CORBA, definido pelo OiL, mas esse método já é inserido automaticamente em todas as facetas no ato da instanciação do componente. Esse método é o mesmo que o *_get_component()* do ORB JacORB para Java. Em Lua deve-se sempre chamar *_component()*, independente da linguagem do objeto remoto.

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

4.3 Utilização da API

A biblioteca representada pelo módulo Lua "scs.core.base"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 tabelas internas criadas e preenchidas. O processo de instanciação engloba a criação de descrições de facetas, a instanciação das facetas, a criação de objetos CORBA referentes às facetas e a criação de receptáculos.

Para realizar esse trabalho, é necessário informar à biblioteca quais facetas e receptáculos fazem parte do componente. Esses dados são representados por descrições de

facetas e descrições de receptáculos, definidos na IDL do modelo SCS. As descrições de facetas devem ser quase iguais às definidas em IDL, necessitando apenas de um campo adicional chamado *class* e podendo opcionalmente especificar um campo *key*. Portanto, devem ser tabelas Lua compostas pelos itens a seguir:

- `name`: Nome desejado para a faceta.
- `interface_name`: Nome completo da interface CORBA, incluindo módulos. Exemplo: "IDL:scs/core/IComponent:1.0".
- `class`: Tabela Lua do tipo *callable*, que implementa os métodos da faceta e retorna uma nova instância da mesma ao ser chamada.
- `key`: String opcional que define uma chave para o endereço *corbaloc* dessa faceta.

Descrições de receptáculos também devem ser quase iguais às definidas em IDL, necessitando de apenas um campo adicional chamado *type*. Esse campo deve receber uma *string* e informa qual tipo de receptáculo deve ser utilizado, dentre as seguintes opções:

- `Receptacle`: Receptáculo que suporta apenas uma conexão.
- `ListReceptacle`: Receptáculo LOOP que suporta múltiplas conexões.
- `HashReceptacle`: Receptáculo LOOP que suporta múltiplas conexões.
- `SetReceptacle`: Receptáculo LOOP que suporta múltiplas conexões.

Dentre os tipos de receptáculos múltiplos, não há diferenças para o usuário, já que todos são tabelas Lua e a biblioteca se encarrega do tratamento específico para cada tipo. Esses tipos são fornecidos pela biblioteca LOOP e se diferenciam principalmente em relação à geração de identificadores. Por isso, planejamos remover essa opção de tipo futuramente. Para mais detalhes sobre os tipos de receptáculos múltiplos suportados pelo LOOP, o seu manual deve ser consultado. Um outro detalhe das descrições de receptáculos é que não é necessário fornecer o *array connections*, podendo-se passar *nil*.

O método da biblioteca para a criação / instanciação de um novo componente chama-se *newComponent*. Esse método recebe uma tabela de descrições de facetas, uma tabela de descrições de receptáculos (ambas as tabelas indexadas pelo nome da faceta ou receptáculo) e o identificador do componente.

Não é necessário inserir as facetas básicas (IComponent, IReceptacles e IMetaInterface) na tabela de descrições de facetas. A biblioteca automaticamente insere essas facetas e suas descrições no componente. Caso o usuário deseje utilizar uma implementação diferente de alguma dessas facetas ou incluir uma chave para definir um endereço *corbaloc* para elas, basta inserir a descrição apropriada na tabela que essa será utilizada no lugar da padrão.

Um outro detalhe é que, como a biblioteca cria os objetos CORBA, é necessário que tenha acesso ao ORB. Para que os componentes tenham acesso ao ORB utilizado, o mesmo deve ser definido em uma variável específica, como mencionado na Seção 2.

Um exemplo de código para a criação das descrições, identificador do componente e utilização da API pode ser visto no Código 4.

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 no Código 5.

O Código 6 implementa a faceta IHello, que conta com apenas um método, *sayHello*. Além disso, realiza a criação do componente. O código é bastante similar ao apresen-

Código 4: Instanciação de um Novo Componente

```
local scs    = require "scs.core.base"

-- criação da Faceta
local MyFacet = ...

-- criação das descrições
local facetDescs = {}
facetDescs.MyFacet  = {
  name = "MyFacet",
  interface_name = "IDL:mymodule/MyFacet:1.0",
  class = MyFacet
}
local receptDescs = {}
receptDescs.MyReceptacle = {
  name = "MyReceptacle",
  interface_name = "IDL:expectedmodule/ExpectedInterface:1.0",
  is_multiplex = false,
  type = "Receptacle"
}

local componentId = {
  name = "MyComponent",
  major_version = 1,
  minor_version = 0,
  patch_version = 0,
  platform_spec = ""
}

oil.main(function()
  -- cria uma thread para que o ORB passe a aguardar chamadas remotas
  oil.newthread(orb.run, orb)

  -- cria o componente
  local instance = scs.newComponent(facetDescs, receptDescs, componentId)
end)
```

tado nos Códigos 3 e 4.

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 no Código 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.

Código 5: IDL do Componente Hello

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

Código 6: Criação do Componente Hello

```
local oo  = require "loop.base"
local oil = require "oil"

-- inicialização do ORB
-- porta e host apenas para fins do exemplo
local orb = oil.init({host = "localhost", port = 1050})
oil.orb = orb

-- carga das IDLs no ORB
orb:loadidlfile("scs.idl")
orb:loadidlfile("hello.idl")

-- implementação da faceta IHello
local Hello = oo.class{name = "World"}
function Hello:sayHello()
  print("Hello " .. self.name .. "!")
end

-- criação das descrições de facetas e receptáculos
local facetDescs = {}
facetDescs.IHello  = {
  name = "IHello",
  interface_name = "IDL:scs/demos/helloworld/IHello:1.0",
  class = Hello
}
local receptDescs = {}

-- criação do ComponentId
local cpId = {
  name = "Hello",
  major_version = 1,
  minor_version = 0,
  patch_version = 0,
  platform_spec = ""
}

-- função main
oil.main(function()
  -- instrução ao ORB para que aguarde por chamadas remotas (em uma nova "thread")
  oil.newthread(orb.run, orb)

  -- cria o componente
  instance = scs.newComponent(facetDescs, receptDescs, cpId)

  -- modificação do nome a ser exibido na mensagem da faceta Hello
  instance.IHello.name = "User"

  -- publicação do IOR para que a faceta IHello do componente possa ser
  -- encontrada. Observação: podemos exportar qualquer faceta, pois temos
  -- o método _component para obter a faceta IComponent e, com ela,
  -- pode-se obter outras facetas(esse passo pode ser substituído por outras
  -- formas de publicação, como a publicação em um serviço de nomes, por
  -- exemplo).
  oil.writeto("hello.ior", orb:tostring(instance.IHello))
end)
```

Código 7: Utilização do Componente Hello

```
local oil = require "oil"

-- inicialização do ORB
local orb = oil.init()

-- carga das IDLs no ORB
orb:loadidlfile("scs.idl")
orb:loadidlfile("hello.idl")

-- função main
oil.main(function()
  -- assume-se que o arquivo que contém o IOR (publicado pelo código
  -- anterior) esteja disponível. O arquivo pode ter sido criado em
  -- outra máquina e, nesse caso, tem de ser copiado manualmente
  -- (pode-se também utilizar um método diferente de publicação,
  -- como um serviço de nomes).
  local iHelloIOR = oil.readfrom("hello.ior")

  -- obtenção das facetas IHello e IComponent
  local iHelloFacet = orb:newproxy(iHelloIOR,
    "IDL:scs/demos/helloworld/IHello:1.0")
  -- precisamos utilizar o método narrow pois estamos recebendo um
  -- org.omg.CORBA.Object
  local icFacet = orb:narrow(iHelloFacet:_component())

  -- inicialização do componente.
  icFacet:startup()

  -- com o componente inicializado, podemos utilizá-lo à vontade.
  -- note que não é possível modificar o campo "name" da classe Hello
  -- remotamente, pois o campo não está definido em IDL (nem há um
  -- método "setter").
  iHelloFacet:sayHello()
end)
```