4
Middleware SCS-Composite

Este capítulo apresenta o *middleware* SCS-*Composite*. Este *middleware* é a implementação do modelo SCS-*Composite*, definido no capítulo anterior. O modelo SCS possui implementações para componentes locais em C++ (Stroustrup, 2000), Java (Gosling et al., 2005) e C# (International, 2006), e para componentes distribuídos em Lua (Ierusalimschy et al., 2006), C++, Java e C#. Na Seção 4.1, é feita uma síntese do *middleware* SCS destacando diferenças entre as versões para componentes locais e distribuídos. Nas Seções 4.2 e 4.3, serão apresentadas as implementações do SCS-*Composite*, respectivamente, em Java para componentes locais e em Lua para componentes distribuídos. Ao fim deste capítulo, o leitor terá uma visão prática sobre o *middleware* SCS-*Composite* e estará ciente das modificações necessárias para migrar aplicações SCS para SCS-*Composite*.

4.1
Middleware SCS

O *middleware* SCS é a implementação do modelo descrito no Capítulo 3. Para este trabalho iremos utilizar a versão 1.2.1 para componentes distribuídos e 2.0 para componentes locais. É importante destacar que existem diferenças significativas na API e no modelo adotado pelas versões 1.2 e 2.0 do SCS, e que o suporte para componentes locais só foi introduzido na versão 2.0. Augusto et al. (2009) apresenta a implementação do SCS distribuído que já é bastante utilizada em projetos vinculados ao Tecgraf/PUC-Rio. Algumas modificações estão sendo avaliadas para o SCS distribuído e testadas em um protótipo do SCS versão 2.0 para componentes locais. As interfaces e estruturas do *middleware* SCS local (versão 2.0) são apresentadas no Apêndice A.

Dentre as mudanças entre a versão 1.2.1 e 2.0 podemos destacar uma nova abstração de faceta, as operações de ciclo de vida desacopladas da faceta *IComponent*, possibilidade de adicionar mecanismos de *listeners* na interação entre uma faceta e receptáculo e, por fim, introspecção sobre metadados de um componente a partir da abstração de faceta e receptáculo ao invés da faceta *IMetaInterface*.

Na versão 1.X, a interface *IComponent* possui as operações **startup**, **shutdown**, **getFacet**, **getFacetByName** e **getComponentId**. Já na versão 2.0, a interface *IComponent* possui as operações **getFacet** e **getFacets**. As operações responsáveis pelo ciclo de vida de um componente foram desacopladas da faceta *ICom-*

ponent e são oferecidas através da faceta *ILifeCycle*. Estuda-se adicionar algumas outras operações para esta interface como **getMissingDependencies**, **resolveDependencies** e **suspend**.

Uma outra diferença é que o SCS 1.X não possui uma definição de faceta presente na implementação, tratando uma faceta como um objeto do tipo *Object* de CORBA. Assim, o objeto retornado pelas operações **getFacet** e **getFacetByName** é o próprio objeto da aplicação que implementa o serviço representado pela faceta. O objeto *Object* retornado pelas operações de *IComponent* deve ser convertido diretamente para um tipo que oferece as operações do serviço oferecido pela faceta. Ademais, a partir de um *Object* é possível acessar o componente que oferece uma determinada faceta.

Já na versão 2.0, existe uma abstração específica para representar facetas, e a partir dessa abstração é que se obtém o objeto que oferece o serviço representado pela faceta. A interface *IFacet* (Vide Código 4.1) representa a abstração de uma faceta no SCS 2.0. Essa interface apresenta a operação **getDescription** que retorna uma descrição da faceta (identificador único e tipo), a operação **getObject** que retorna a implementação do serviço (o equivalente da faceta da versão 1.X) e **getComponent** que retorna o componente ao qual a faceta pertence.

```
1 package scs2;
2
3 public interface IFacet {
4   FacetDescription getDescription();
5   Object getObject();
6   IComponent getComponent();
7 }
```

Código 4.1: Interface que representa a abstração de uma faceta presente no SCS local

Também, o SCS v. 2.0 oferece um mecanismo adicional de *listeners* para o momento de realização do *binding* horizontal. Desta forma, o usuário da API pode adicionar comportamento após a conexão ou desconexão entre componentes. Este mecanismo é oferecido através da faceta *IReceptacles* através das operações **addConnectionListener** e **removeConnectionListener**.

Por fim, a inspeção sobre facetas e receptáculos é oferecida através da faceta *IMetaInterface* no SCS v. 1.X. O SCS v. 2.0 não apresenta esta interface para inspeção. A inspeção é realizada através das facetas e receptáculos, que possuem o método **getDescription** para recuperar metadados. Os metadados de uma faceta e receptáculo são representados, respectivamente, pelas classes **FacetDescription** e **ReceptacleDescription**. A classe **FacetDescription** possui os atributos nome e tipo de interface. Já a classe **ReceptacleDescription** possui os atributos nome, tipo de interface, cardinalidade e limite de conexões.

4.2 Middleware SCS-Composite

O *middleware* SCS-*Composite* é a extensão do SCS que oferece suporte a componentes compostos. Para atender os requisitos do modelo proposto no Capítulo 3, foram necessárias, basicamente, três modificações principais: alterações na estrutura do componente, na fábrica de componentes e no mecanismo de *binding* horizontal. A Seção 4.2.1 descreve a mudança na estrutura de um componente e, a Seção 4.2.2 apresenta detalhes sobre as novas regras no mecanismo de *binding* horizontal.

4.2.1 Estrutura do Componente

Como visto no Capítulo 3, foi introduzida a faceta obrigatória *ISuperComponent* (Código 4.2) tanto para componentes primitivos quanto compostos. Como nossa abordagem inicial era não realizar modificações intrusivas na essência do SCS, em um primeiro momento, optamos por adiar a adição desta nova faceta na especificação do modelo. Porém, avaliando melhor vimos que ela é importante para garantir algumas diretrizes do modelo descritas na Seção 3.3. Dentre as funcionalidades desta faceta podemos citar: permite a navegação no sentido *filho* $\Rightarrow$ *pai*; auxilia no teste de conexão realizado no *binding* de uma faceta e receptáculo, verificando se pertencem ao mesmo componente composto; e, é importante no mecanismo de garantir o compartilhamento de subcomponentes apenas entre componentes sem receptáculos.

```
1 module scs {
2   module core{
3     typedef sequence<IComponent> Components;
4     interface ISuperComponent {
5       void addSuperComponent(in IComponent cmp);
6       void removeSuperComponent(in string cmp);
7       IComponentSeq getSuperComponents();
8     };
9   };
10 };
```

Código 4.2: Interface *ISuperComponent* descrita em IDL responsável por oferecer instrospecção de um componente composto apartir de um subcomponente

O componente composto possui as mesmas facetas do componente primitivo e a faceta *IContentController* (Código 4.3). Esta faceta oferece mecanismos de configuração e introspecção sobre os subcomponentes de um componente. Diferente do componente primitivo, o componente composto possui um identificador único. Optamos por não adicionar este identificador ao componente primitivo para facilitar a

migração incremental de aplicações. Na versão local, este identificador é a referência do objeto que representa o componente composto. Já na versão distribuída, o identificador é a concatenação do horário do sistema em milisegundos no momento da criação do componente composto mais a IOR (*Interoperable Object Reference*) do componente. Este identificador do componente composto é importante para um subcomponente verificar se faz parte da mesma composição de um outro subcomponente remoto garantindo assim as restrições de conexão entre dois componentes do modelo SCS-*Composite*.

```
1 module scs {
2   module core{
3     typedef sequence<IComponent> Components;
4     typedef unsigned long MembershipId;
5     typedef unsigned long BindingId;
6     interface IContentController {
7       string getId();
8       MembershipId addSubComponent(in IComponent obj);
9       void removeSubComponent(in MembershipId id);
10      Components getSubComponents();
11      IComponent findComponent(in MembershipId id);
12      BindingId bindFacet(in MembershipId subcomponent,in string
            internalFacetName,in string externalFacetName);
13      void unbindFacet(in BindingId id);
14      BindingId bindReceptacle(in MembershipId subcomponent,in
            string internalReceptacleName,in string
            externalReceptacleName);
15      void unbindReceptacle(in BindingId id);
16    };
17  };
18 };
```

Código 4.3: Interface *IContentController* responsável pela abstração de membrana no modelo SCS

Ademais, foram apresentados, na Seção 3.2, os conceitos de faceta e receptáculo externalizados presentes em um componente composto. Uma faceta é externalizada através da operação **bindFacet**. Esta operação recebe como parâmetros: o *membership* do subcomponente que possui o serviço, o identificador da faceta e um novo identificador para acessar a faceta a partir do componente composto. Esta faceta funciona como um *proxy* para a faceta de um subcomponente adicionando uma indireção. Este *proxy* repassa as requisições para o subcomponente que realmente implementa o serviço e retorna o componente composto caso uma entidade externa consulte qual componente implementa o serviço. Desta forma, o *proxy* permite a navegação da faceta externalizada para as outras facetas do componente composto.

Da forma análoga, um receptáculo é externalizado através da operação **bindReceptacle**. Esta operação recebe como parâmetros: o *membership* do subcompo-

nente que possui a dependência, o identificador da dependência e um novo identificador para acessar a dependência a partir do componente composto. Esta operação, também, adiciona uma indireção, pois funciona como um repassador de dependências para os receptáculos dos subcomponentes associados. Para aplicações com alto grau de aninhamento as indireções criadas pela externalização de facetas e receptáculos podem representar uma sobrecarga de desempenho. Como trabalho futuro, pode-se investigar mecanismos para otimizar facetas e receptáculos externalizados.

Ao fim das operações **bindFacet** e **bindReceptacle** é retornado um identificador relacionado ao novo mapeamento. Este identificador é utilizado para a remoção dos mapeamentos através das operações **unbindFacet** e **unbindReceptacle**.

Com a modificação da estrutura de um componente foi necessária a alteração da geração de componentes. Assim, a fábrica deve gerar dois tipos de componentes: primitivos e compostos. Primeiro, serão apresentadas as modificações sobre a fábrica do SCS local e, em seguida, sobre a fábrica do SCS distribuído.

Um dos principais objetivos da fábrica de componentes do SCS local é oferecer uma migração fácil do paradigma de orientação a objetos para orientação a componentes. Desta forma, para definir um componente basta ser adicionado o método **getInstance** a uma classe. O Código 4.4 demonstra a definição de um componente através da classe **Calculator** que implementa a interface *ICalculator*. Para definir um componente **Calculator**, basta adicionarmos o método **getInstance**. Na linha 7 temos acesso ao contexto da aplicação que oferece uma fábrica de componentes (linhas 8 e 9). A partir desta fábrica, o usuário define as facetas e receptáculos (linhas 14 e 15) e, por fim, retorna um componente SCS com as facetas básicas, facetas definidas pelo usuário e receptáculos.

```
1 public Calculator implements ICalculator
2 {       ...
3   public static IComponent getInstance()
4   {
5     IComponent self = null;
6     final Calculator tst = new Calculator();
7     IComponentContext cnx = SCS.getInstance();
8     IComponent cmp = cnx.lookup("component_manager");
9     IFacet fct = cmp.getFacet(IComponentAssemblerFactory.
          interface_id);
10    IComponentAssembler assembler = ((IComponentAssemblerFactory)
          fct.getObject()).create();
11
12    try {
13      assembler.startComponent();
14      assembler.addFacet(new FacetDescription("calculator",
            ICalculator.class), tst);
15      assembler.addReceptacle(new ReceptacleDescription("operation
            ", IOperation.class, 0, false));
16      self = assembler.finishComponent();
```

```
17      } catch (Exception ex){...}
18      return self;
19    }
20          ...
21 }
```

Código 4.4: Método **getInstance** implementado pelo usuário para criar um componente.

Caso o usuário queira criar um componente composto o método **create** (Código 4.4 - linha 10) foi sobrecarregado. Existe uma versão sem argumentos que retorna um componente primitivo e outra versão que recebe uma **String** com o valor *primitive* ou *composite* onde o usuário define explicitamente qual tipo de componente deseja.

Para o SCS distribuído, a versão SCS em Lua oferece um módulo **scs.core.base** que possui o método **newComponent**. Basicamente, **newComponent** recebe como parâmetro um conjunto de facetas, receptáculos e retorna a instância de um componente primitivo. Para a criação de componentes compostos foi criado o módulo **scs.core.composite** que possui o método **newCompositeComponent**. Esta operação retorna um componente composto com as facetas obrigatórias de um componente primitivo e, adicionalmente, a faceta *IContentController*.

4.2.2 Mecanismo de binding

No SCS-*Composite* podemos diferenciar dois tipos de receptáculos: um tipo especificado pelo usuário no momento de *design* de um componente (Vide 1 em Figura 4.1) e outro tipo que representa a externalização de um ou mais receptáculos dos subcomponentes (Vide 2 em Figura 4.1). O primeiro diz respeito ao receptáculo já existente no SCS atual e é encontrado tanto em um componente primitivo quanto composto. O segundo tipo de receptáculo é encontrado apenas em componentes compostos.

As duas regras de conexões estabelecidas na seção 3.2 representam uma grande alteração nos mecanismos básicos do SCS. Basicamente, elas definem que dois componentes só podem interagir caso façam parte do mesmo supercomponente ou não pertençam a nenhum componente componente composto. Foi necessário adicionar este teste na operação **connect** da faceta *IReceptacles*.

O novo tipo de receptáculo presente em componentes compostos diz respeito ao receptáculo de um subcomponente que foi externalizado através de determinado componente composto. O comportamento deste receptáculo é parecido com o anterior, entretanto, para a operação **connect** e **disconnect** são introduzidos alguns passos. Na operação **connect**, o receptáculo além de realizar a conexão com uma faceta, também requisita a conexão desta faceta aos receptáculos dos subcomponentes. Já na operação **disconnect**, o receptáculo além de realizar a

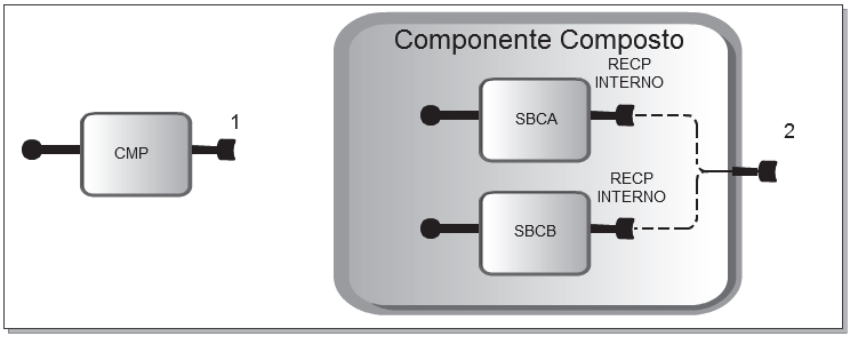

Figura 4.1: Os dois tipos de receptáculos presentes no SCS-*Composite*

desconexão de uma faceta conectada, também, solicita a desconexão desta faceta nos receptáculos dos subcomponentes.

Enfrentamos algumas dificuldades quando o receptáculo do componente composto repassava as conexões para os subcomponentes. O subcomponente não aceitava a conexão repassada em função da faceta oferecida ser de um componente que se encontrava fora do componente composto. Para a solução deste problema quando um receptáculo externalizado de um componente composto A recebe uma faceta F de um componente B, primeiramente, é realizado o *binding* entre os componentes A e B. Em seguida, cria-se um *proxy* da faceta F. Este *proxy* repassa todas as requisições para o componente B. Porém, quando consultado sobre qual componente o implementa, este *proxy* retorna o componente composto A. Segundo a regra de conexão definida na Seção 3.2.4, um subcomponente pode aceitar conexões de *proxies* do componente composto.

Um aspecto negativo do mecanismo de *proxy* criado pelo componente composto é em relação a navegação de facetas a partir de um receptáculo de um subcomponente. Dado um receptáculo é possível acessar suas facetas conectadas e, consequentemente, os componentes que as implementam. Entretanto, como em nossa implementação o *proxy* altera a identidade do componente que implementa a faceta, quando um subcomponente realizar a navegação sobre um *proxy* irá chegar ao próprio componente composto. A navegação de um subcomponente através de um *proxy* nos leva a uma informação incoerente sobre a configuração da aplicação.

Para contornar este problema de incoerência recomendamos ser feita uma navegação de forma indireta. Este tipo de navegação consiste de três passos. Primeiro, após um receptáculo interno recuperar uma faceta para navegação deve verificar se a mesma é implementada pelo componente composto. Caso seja, deve navegar para o componente composto e, em seguida, para o receptáculo do componente composto que representa o receptáculo interno. Depois destes passos, o receptáculo interno tem acesso à faceta desejada. Seria interessante como trabalho futuro investigar uma forma melhor de realizar este tipo de navegação, como por exemplo, através de uma API auxiliar ou um mecanismo alternativo ao *proxy*.

4.3 Exemplo de Uso

Nesta seção são apresentados dois exemplos de uso do *middleware* SCS-*Composite*. Na Seção 4.3.1 é apresentado o mecanismo de externalização de uma faceta de um subcomponente. E, na Seção 4.3.2, o mecanismo de externalização de um receptáculo de um subcomponente.

4.3.1 Mapeamento de Facetas

A Figura 4.2 ilustra um exemplo de externalização de um faceta de um subcomponente através de um componente composto. Para implementação deste exemplo foi utilizado o *middleware* SCS-*Composite* para componentes locais em Java. O exemplo é formado por três componentes: A, B e C. O componente A encapsula os subcomponentes B e C. O componente C oferece a faceta **helloFctC** do tipo *IHello* que se conecta com B através do receptáculo **helloRecp**. Por fim, B externaliza sua faceta **printFctB** do tipo *IPrint* através da faceta **externalPrint** de A. A operação **printHello** acessada a partir da faceta **externalPrint** deve requisitar a operação **sayHello** da faceta **helloFctC**. O código-fonte completo encontra-se no Apêndice B.1.

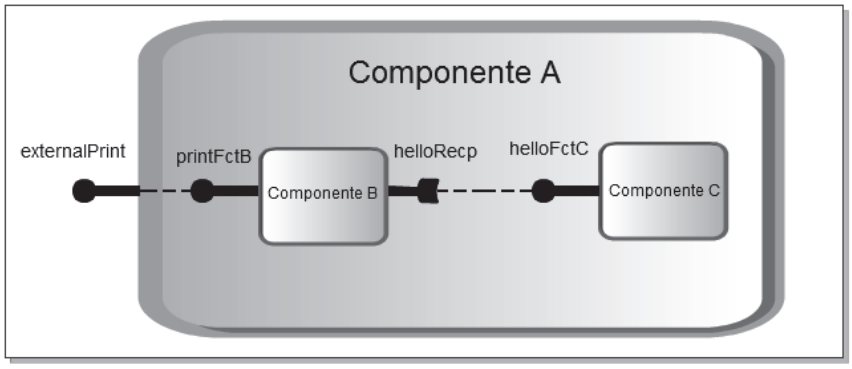

Figura 4.2: Arquitetura do exemplo de uso do SCS-*Composite* Local

O Código 4.5 apresenta a instanciação dos componentes A, B e C (linhas 2 - 4). Na linha 8 temos acesso aos mecanismos de introspecção e configuração do componente composto A através da faceta *IContentController*. Desta forma, são adicionados os componentes B e C (linhas 7 e 8) ao componente composto A. Em seguida, é feita a conexão entre faceta **helloFctC** do componente C e receptáculo **helloRecp** de B (linhas 10 - 13); e, a externalização da faceta **helloFctB** do componente B através do componente composto A (linha 15). Ao término do procedimento de configuração temos acesso ao serviço **helloFctC** do componente C como se fosse um serviço oferecido pelo componente A (linhas 17 - 19).

```
1              ...
2            IComponent cmpA = (IComponent) A.getInstance();
3            IComponent cmpB = (IComponent) B.getInstance();
4            IComponent cmpC = (IComponent) C.getInstance();
5            IContentController compositeService = (
                 IContentController) cmpA.getFacet(
                 IContentController.interface_id).getObject();
6
7            compositeService.addSubComponent(cmpB);
8            compositeService.addSubComponent(cmpC);
9
10           IFacet fct = cmpC.getFacet("helloFctC");
11           IReceptacles recps = (IReceptacles)cmpB.getFacet(
                 IReceptacles.interface_id).getObject();
12           IReceptacle recp = recps.getReceptacle("helloRecp");
13           recp.addConnection(fct);
14
15           compositeService.bindFacet(0,"helloFctB","
                 externalHello");
16
17           IFacet helloFacet = cmpA.getFacet("externalHello");
18           IPrint helloImpl = (IPrint) helloFacet.getObject();
19           helloImpl.printHello();
20             ...
```

Código 4.5: Classe *main* responsável por executar a aplicação.

É importante destacar que a interação entre os componentes B e C só é possível por serem encapsulados pelo componente A obedecendo as regras de conexão entre componentes estabelecidas na Seção 3.1.2. Também, o mapeamento de um serviço interno (linha 15) cria uma indireção em função do mapeamento do serviço de um subcomponente através do componente composto. O escopo deste trabalho não abrange a implementação de mecanismos de otimização para este.

4.3.2 Mapeamento de Receptáculos

A Figura 4.3 ilustra um exemplo de externalização de um receptáculo de um subcomponente através de um componente composto. Para implementação deste exemplo foi utilizado o *middleware* SCS-*Composite* para componentes distribuídos em Lua. O exemplo é formado por três componentes: A, B e C. O componente A encapsula o subcomponentes B que oferece a faceta **fctB** e receptáculo **recpB**. O componente C oferece a faceta **fctC** e receptáculo **recpC**. O componente B externaliza sua faceta através de **fctA** e receptáculo através de **recpA**.

Basicamente, o exemplo consiste em um "ping-pong" entre os componentes A e C. O componente A inicia a execução através do método **ping** que realiza uma

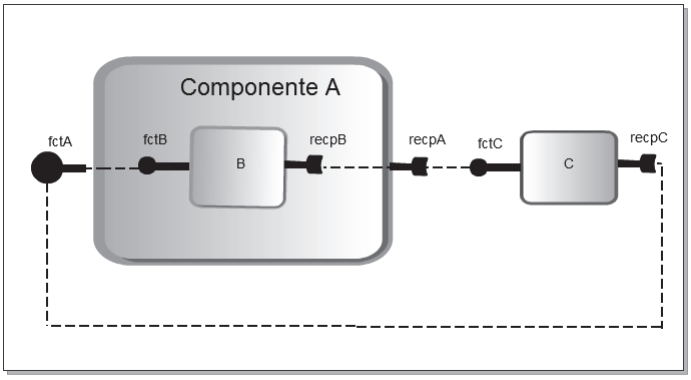

Figura 4.3: Arquitetura do exemplo de uso do SCS-*Composite* Distribuído.

chamada ao método **pong** do componente C. Ao receber a chamada o método **pong** de C realiza a chamada ao método **ping** do componente A. O código-fonte completo do exemplo encontra-se no Apêndice B.2.

O Código 4.6 demonstra a execução da aplicação. Primeiramente, são recuperadas as referências CORBA dos componentes A, B e C (linhas 2-4). Entre as linhas 6 e 10 é acessada a faceta *IContentController* do componente A e feito o mapeamento de serviços e dependências. Os receptáculos *PingPongReceptacle* de A e C são acessados entre as linhas 12 e 15. Em seguida, nas linhas 17 e 18 são recuperadas as facetas *PingPongServer* dos componentes A e C e realizada a conexão.

```
1                    ...
2 local cmpA = orb:newproxy(oil.readfrom("componentA.ior"))
3 local cmpB = orb:newproxy(oil.readfrom("componentB.ior"))
4 local cmpC = orb:newproxy(oil.readfrom("componentC.ior"))
5
6 local compositeFct  = cmpA:getFacetByName("IContentController")
7 compositeFct = orb:narrow(compositeFct,"IDL:scs/core/
      IContentController:1.0")
8 compositeFct:addSubComponent(cmpB)
9 compositeFct:bindFacet(0,"PingPongServer","PingPongServer")
10 compositeFct:bindReceptacle(0,"PingPongReceptacle","
      PingPongReceptacle")
11
12 local recpsCmpA = cmpA:getFacetByName("IReceptacles")
13 recpsCmpA = orb:narrow(recpsCmpA,"IDL:scs/core/IReceptacles:1.0")
14 local recpsCmpC = cmpC:getFacetByName("IReceptacles")
15 recpsCmpC = orb:narrow(recpsCmpC,"IDL:scs/core/IReceptacles:1.0")
16
17 ppCmpA = cmpA:getFacetByName("PingPongServer")
18 ppCmpC = cmpC:getFacetByName("PingPongServer")
19
20 recpsCmpA:connect("PingPongReceptacle", ppCmpC)
21 recpsCmpC:connect("PingPongReceptacle", ppCmpA)
22
```

```
23 ppCmpA:start()
24                    ...
```

Código 4.6: Script Lua que demonstra o mapeamento de um receptáculo de um subcomponente através de um receptáculo de um componente composto.

Após o *binding* entre os componentes A e C, dá-se início à execução da aplicação. A operação **start** é solicitada pela faceta **ppCmpA** do componente A (linha 23) que realiza um **ping** no componente C. Maiores detalhes sobre a implementação da interface **PingPongServer** encontram-se no Apêndice B.2.

4.4 Considerações Finais

Neste capítulo foram demonstradas duas implementações do modelo de componentes SCS-*Composite*: uma versão para componentes locais (SCS Local 2.0) e outra para componentes distribuídos (SCS Distribuído 1.2.1). O SCS para componentes locais apresenta uma abstração de faceta bem definida, operações de ciclo de vida desacopladas da faceta *IComponent*, mecanismos de *listeners* e introspecção sobre as facetas e receptáculos a partir das facetas *IFacet* e *IReceptacle*.

Basicamente, o *middleware* SCS-*Composite* se diferencia do *middleware* SCS em três aspectos: estrutura do componente, forma de criação de um componente e mecanismo de *binding* horizontal com os testes de restrição definidos na Seção 3.2. As alterações na estrutura de um componente e mecanismos de *binding* representam mudanças na base do SCS. Embora nosso objetivo inicial fosse não realizar este tipo de modificação intrusiva, o estudo realizado nos Capítulos 2 e 3 nos mostrou que tais modificações eram necessárias para garantir características importantes do modelo de componentes compostos como suporte ao desenvolvimento incremental, relacionamento Subcomponente/Componente Composto e compartilhamento de componentes.

Com as modificações, um componente primitivo tem mais uma faceta obrigatória: *ISuperComponent*. Através dela é possível a navegação no sentido "filho" $\Rightarrow\triangleright$ "pai". Este tipo de navegação auxilia no teste de conexão realizado na interação entre dois componentes e na garantia de que um componente só pode ser compartilhado caso não tenha receptáculos. Também, é um importante mecanismo para inspeção da arquitetura de uma aplicação. As mudanças em relação à estrutura do componente primitivo e adição do componente composto implicaram na implementação de uma nova fábrica de componentes que oferece suporte para a geração de ambos os tipos de componentes.

Um conceito importante introduzido foi o mecanismo de *proxies* de um receptáculo externo. Ao receber uma conexão de uma faceta, um receptáculo externo repassa *proxies* para os subcomponentes. Este mecanismo é necessário para o subcomponente aceitar a conexão de uma faceta implementada por um componente externo à sua composição. Entretanto, estes *proxies* podem gerar

inconsistências quando for realizada a inspeção hierárquica sobre uma arquitetura, como por exemplo, na navegação de um receptáculo interno R ao componente que implementa uma faceta conectada a R (Vide Seção 4.2.2). É necessário um estudo mais aprofundado sobre como lidar com as incoerências introduzidas por um *proxy* oferecendo mecanismos de introspecção mais precisos.

Na Seção 4.3, os exemplos têm como objetivo demonstrar como o usuário da API SCS deve lidar com os novos conceitos introduzidos. A forma de manipulação sobre componentes primitivos é feita da mesma forma que no SCS sem suporte a componentes compostos. Assim, caso o usuário do SCS-*Composite* queira desenvolver aplicações com apenas componentes primitivos pode trabalhar como se estivesse desenvolvendo com o SCS original. E, caso necessário, pode adicionar componentes compostos a aplicação de forma incremental. Na próxima seção, será apresentado um exemplo de uso com um estudo mais aprimorado sobre as funcionalidades do *middleware* SCS-*Composite*.

Por fim, destacamos que os mapeamentos de facetas e receptáculos dos subcomponentes adicionam indireções em uma aplicação. Quando um serviço externalizado é acessado existe uma sobrecarga decorrente do componente composto repassar a solicitação para o subcomponente. Da mesma forma, quando um receptáculo externalizado recebe uma faceta para conexão, tem como tarefa repassar esta faceta para os subcomponentes. Como trabalho futuro, é importante investigar mecanismos para otimizar tais indireções decorrentes do mapeamento de facetas e receptáculos de subcomponentes.