3
Modelo SCS com Suporte a Componentes Compostos

Este capítulo apresenta uma especificação formal do modelo de componentes original do SCS (Augusto et al., 2009) (Seção 3.1) e do nosso modelo proposto com suporte a componentes compostos, o SCS-*Composite*(Seção 3.2). Como instrumento da especificação do modelo foram utilizados conceitos de teoria de conjuntos e Prolog, como fatos, relacionamentos e regras. O poder de expressividade do Prolog pode ser útil para validação na fase de modelagem de uma aplicação SCS e fornecer consultas sobre determinada configuração de uma aplicação. Na Seção 3.3, são descritas as características principais do SCS-*Composite*. E, por fim, temos as considerações finais entrando em detalhes sobre as dificuldades encontradas para concepção do SCS-*Composite*.

3.1
Modelo de Componentes SCS

O modelo de componentes SCS é baseado nos modelos COM (Box, 1998) e CORBA *Component Model* - CCM (OMG, 2006) com objetivo de reduzir a complexidade imposta por tais modelos. O SCS tem como objetivo oferecer mecanismos de interação, configuração e introspecção de forma flexível e simples através de um conjunto pequeno de APIs.

Os conceitos principais do SCS são facetas, receptáculos e *bindings*. Na Seção 3.1.1, será apresentada a especificação de um componente SCS, que é formado por um conjunto de facetas e receptáculos. Em seguida, na Seção 3.1.2, serão apresentados os mecanismos de *binding* responsáveis pela interação entre os componentes.

3.1.1
Componente

Um componente SCS é uma unidade lógica pronta para composição e reuso que oferece mecanismos de interação, configuração e introspecção. Um componente é constituído por facetas (serviços oferecidos) e receptáculos (dependências) que interagem entre si através de interfaces explícitas.

De maneira formal, um componente é formado por um conjunto de serviços e dependências. Um componente C disponibiliza seus serviços através de um conjunto de facetas F e suas dependências através de um conjunto de receptáculos R. Desta forma, podemos caracterizar um componente C por F e R da seguinte forma:

$$C = \langle F, R \rangle, com\ F = \{f_1, f_2, \ldots, f_n\}\ e\ R = \{r_1, r_2, \ldots\ r_k\}$$

F define um conjunto de serviços f_k onde cada serviço é definido por uma tupla $\langle identifier, type, component \rangle$ especificado pelo relacionamento **facet (identifier, type, component)** onde **identifier(x)** expressa que x é um identificador, **type(x)** um tipo (análogo a uma interface) e, por fim, **component(x)** o componente que implementa a faceta. R define um conjunto de dependências r_k onde cada dependência é definida pela tupla $\langle identifier, type, cardinality, component \rangle$ especificada pelo relacionamento **receptacle (identifier, type, cardinality, component)**. Nota-se que o relacionamento **receptacle** possui um relacionamento adicional (**cardinality**) que informa a cardinalidade de um receptáculo, isto é, se o receptáculo aceita mais de uma conexão.

O identificador de uma faceta é único no escopo de um componente, ou seja, um componente não possui duas facetas com o mesmo identificador. Da mesma forma, cada receptáculo possui um identificador único. A unicidade do conjunto de facetas e de receptáculos é garantida a partir da regra **isAValidComponent(C,F,R)**, onde F é um conjunto de facetas e R o conjunto de receptáculos de um componente C. A regra **isAValidComponent** retorna que um componente C é válido se não F e R não tiverem nenhum elemento repetido.

Um componente SCS oferece, obrigatoriamente, três facetas. A faceta *IComponent* que define o tipo **componente** em SCS e oferece operações para ativação e desativação de um componente e, para requisição de outras facetas. A faceta *IReceptacles* que define operações para gerenciar conexões de receptáculos; e por fim, a faceta *IMetaInterface* que define operações básicas para introspecção de facetas e receptáculos dos componentes.

A Figura 3.1 ilustra um componente SCS com suas facetas obrigatórias e receptáculos. Como ilustrado, além das facetas obrigatórias podem ser definidas um número variável de facetas e receptáculos.

3.1.2 Mecanismo de Binding

Os *bindings* representam as conexões entre componentes (através de suas facetas e receptáculos), também conhecidos como *bindings* horizontais (vide Capítulo 1). A conexão entre dois componentes é representada através do relacionamento **connection(C1, F, C2, R)** que expressa "o componente C1 possui uma faceta F que se conecta com o componente C2 através do receptáculo R".

Para uma conexão ser válida devem ser satisfeitas duas restrições: a primeira define que uma faceta F de um componente C1 só pode se conectar a um receptáculo R de um componente C2 caso tenham tipos compatíveis. E, a segunda restrição define que a conexão deve respeitar a cardinalidade do receptáculo. Um receptáculo pode ser simples ou múltiplo, isto é, aceitar apenas uma ou um número variável de conexões.

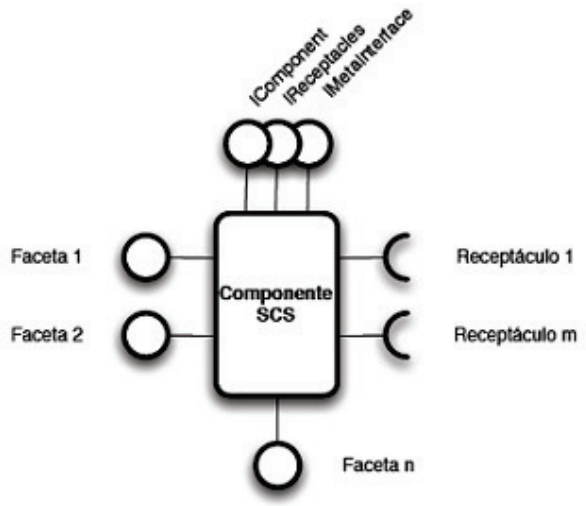

Figura 3.1: Representação de um componente SCS.

O relacionamento que garante a compatibilidade de tipos é definido por **compatible(x,y)** que expressa: "o tipo x é compatível com o tipo y". Este relacionamento tem como base o princípio de substituibilidade de Liskov (Liskov, 1987) que afirma que se S é um subtipo de T, então objetos do tipo T podem ser substistituídos por objetos de tipo S sem alterar nenhuma propriedade desejável do modelo de componentes. Para validar a primeira restrição de conexão no modelo SCS foi adicionada a regra **isAValidConnection(C1,F,C2,R)** definida por:

> isAValidConnection(C1,F,C2,R) :- facet(F,T1,C1), receptacle(R,T2,_,C2), compatible(T1 , T2).

A Figura 3.2 ilustra que um receptáculo pode ser simples ou múltiplo. O **Receptáculo 1** possui cardinalidade múltipla e está conectado com as facetas **Faceta 1** dos componentes B, C e D. O **Receptáculo 2** possui cardinalidade simples e está conectado apenas à **Faceta 2** do componente D.

Para validação da segunda restrição de uma conexão que leva em consideração a cardinalidade de um receptáculo a regra **isAValidConnection** foi redefinida como:

> isAValidConnection(C1,F,C2,R) :- facet(F,T1,C1), receptacle(R,T2,multiple,C2), compatible(T1,T2).

> isAValidConnection(C1,F,C2,R) :- facet(F,T1,C1), receptacle(R,T2,simple,C2), compatible(T1,T2), not(connection(C3,_,C2,R)), not((C1 = C3)).

A conexão entre componentes tem como resultado um *assembly* ou composição. A versão atual do SCS não oferece suporte nativo para definir um componente a partir de uma composição. Tal suporte consiste em um encapsulamento que garante as definições básicas de um componente primitivo para uma composição.

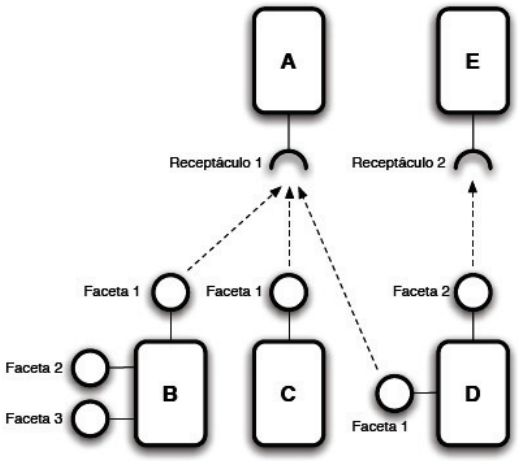

Figura 3.2: Exemplo de conexão entre facetas e receptáculos de componentes SCS.

Esta camada de encapsulamento deve possibilitar duas formas de visão da composição: a primeira, como uma caixa-preta omitindo os subcomponentes e, a segunda, oferecendo mecanismos de configuração e introspecção estrutural sobre os subcomponentes.

A Figura 3.3 ilustra uma composição formada pelos componentes A, B e C. Os componentes A e B possuem facetas (serviços) com o mesmo identificador **idA**. Caso o usuário do SCS original necessite representar esta composição como um componente e, ao mesmo tempo, disponibilizar as facetas de A e B para uma entidade externa, fica a cargo do usuário implementar mecanismos de encapsulamento para esta composição e respeitar a unicidade dos identificadores de sua composição evitando o cenário da Figura 3.3. Esta tarefa adicional aumenta o trabalho de implementação de aplicações SCS que poderiam se beneficiar do conceito de componentes compostos.

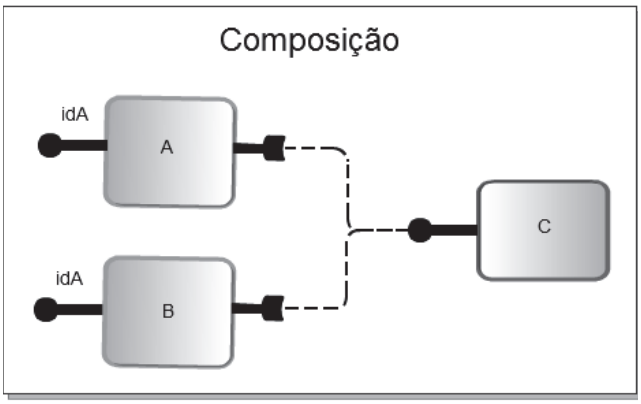

Figura 3.3: Exemplo de uma composição formada pelos componentes A, B e C.

3.2 Modelo de Componentes SCS-Composite

O modelo de componentes original do SCS oferece suporte apenas a componentes primitivos. Este trabalho adiciona o suporte a componentes compostos através do SCS-*Composite* de forma flexível. Segundo Lau et al. (2007), a idéia de componentes compostos é reconhecida como uma boa prática para sistemas baseados em componentes em função de abstrair estruturas complexas e aumentar o reuso.

3.2.1 Componente Composto

Um componente composto é formado por três elementos: uma composição A, um conjunto de facetas F_A e um conjunto de receptáculos R_A, por exemplo:

$$A = \langle \{C_1, C_2\}, F_1 \oplus {}^1R_2, F_2 \oplus R_1 \rangle$$

$$F_A = \{ \{f_1, f_2, ..., f_n\}, F_1 \odot {}^2F_2 \}$$

$$R_A = \{ \{r_1, r_2, ..., r_n\}, R_1 \odot R_2 \}$$

O conjunto F_A pode ser formado por facetas nativas (implementada pelo próprio componente composto), facetas externalizadas de seus subcomponentes (facetas de F_1 ou F_2), ou facetas resultantes da combinação de facetas de subcomponentes. O conjunto de receptáculos R_A pode ser formado por receptáculos nativos, isto é, receptáculos do próprio componente composto, receptáculos de subcomponentes (receptáculos de R_1 ou R_2), e receptáculos resultantes da combinação de receptáculos internos. Desta forma, um componente composto CC é representado pela tupla:

$$CC = \langle A, F_A, R_A \rangle$$

Seguindo a terminologia do Fractal, podemos dizer que um componente CC é formado por uma composição A e uma *membrana* definida pelo conjunto de facetas F_A e de receptáculos R_A. Essa membrana encapsula a composição A e é responsável por disponibilizar duas visões do componente composto que representa: como um componente primitivo, oferecendo as facetas básicas de um componente, ou como uma composição de componentes, explicitando seus subcomponentes e oferecendo mecanismos de configuração e introspecção estrutural.

Um componente composto, além das facetas básicas de um componente primitivo (*IComponent*, *IReceptacles* e *IMetaInterface*), oferece a faceta *IContentController*. Esta nova faceta oferece mecanismos para (i) adicionar e remover subcomponentes (**addSubComponent** e **removeSubComponent**), (ii) realizar o *binding* vertical (**bindFacet** e **bindReceptacle**), (iii) desfazer o *binding* vertical (**unbindFacet** e **unbindReceptacle**) e, (iv) inspecionar subcomponentes (**findComponent** e **getSubComponents**).

Por fim, tanto um componente primitivo quanto um componente composto possuem uma nova faceta obrigatória: *ISuperComponent*. A adição da faceta *ISuperComponent* no modelo representou uma mudança na abordagem de não realizar modificações intrusivas, entretanto foi necessária para garantir regras de conexão (Seção 3.2.4) e compartilhamento de subcomponentes (Seção 3.2.2) do SCS-*Composite*.

3.2.2 Hierarquia de Componentes

Com a introdução do conceito de componentes compostos foi adicionado um novo relacionamento à especificação do modelo de componentes: **subcomponent(x,y)**. Este relacionamento expressa que "x é subcomponente de y". Assim, é possível a navegação de um componente composto para seus subcomponentes. Já para a navegação de um subcomponente para seus componentes compostos foi criada a regra **composite(x,y)** definida por $composite(X, Y) : -subcomponent(Y, X)$. A navegação de um subcomponente para seus componentes compostos é oferecida através da nova faceta *ISuperComponent*.

O modelo SCS-*Composite* considera que alguns componentes podem ser compartilhados por mais de um componente composto. Entretanto, este recurso é oferecido apenas para componentes que não possuem receptáculos em função do problema de ambiguidade descrito na Seção 3.3.5. Para validação do compartilhamento foi criada a regra **isAValidComponentSharing(Sub,CC1,CC2)** que expressa "Sub pode ser compartilhado entre os componentes CC1 e CC2".

```
isAValidComponentSharing(Sub,CC1,CC2) :- subcomponent(Sub,CC1), subcomponent(Sub,CC2), not (receptacle(_,_,_,Sub)), not (CC1 = CC2).
```

Por fim, o SCS-*Composite* oferece o aninhamento de componentes, isto é, um componente composto pode ser encapsulado por outro componente composto. A introspecção estrutural sobre o aninhamento é realizado de forma transitiva, isto é, para um componente composto CC1 que encapsula um componente composto CC2 acessar os subcomponentes de CC2, deve primeiro acessar o componente CC2 e, em seguida, os subcomponentes de CC2.

3.2.3 Externalização de Facetas e Receptáculos

A faceta nativa de um componente composto é representada pelo mesmo relacionamento **facet (identifier, type, component)** de um componente primitivo. A externalização de um serviço de um subcomponente é realizada pela operação **bindFacet** da faceta *IContentController* e representada pelo relacionamento **exposedFacet(Sub,Fsub,CC,Fcc)** que expressa "**Sub** é um subcomponente de **CC** e possui uma faceta **Fsub** externalizada através da faceta **Fcc** do componente **CC**".

A operação **bindFacet** permite apenas os mapeamentos com aridade 1-1 e n-1. Para a externalização de diversas facetas dos subcomponentes através de uma faceta do componente composto oferecemos a possibilidade do uso de um tipo especial de componente conhecido como conector descrito em maiores detalhes na Seção 3.3.2. Este conector realiza o papel de mediador entre a faceta do componente composto e as facetas dos subcomponentes.

Para o mapeamento de um serviço de um subcomponente através do componente composto deve ser válida a regra **isAValidExposedFacet (Sub, Fsub, CC, Fcc)** que expressa " o **Sub** é um subcomponente de **CC** e pode realizar a externalização da faceta **Fsub** através da faceta **Fcc** do componente **CC**".

> isAValidExposedFacet (Sub,Fsub,CC,Fcc) :- exposedFacet(Sub,Fsub,CC,Fcc), facet(Fsub,T1,Sub), facet(Fcc,T2,CC), subcomponent(Sub,CC),compatible(T1,T2).

De maneira análoga, o receptáculo nativo de um componente composto é representado pelo mesmo relacionamento **receptacle (identifier, type, cardinality, component)** de um componente primitivo. A externalização de uma dependência de um subcomponente é realizada pela operação **bindReceptacle** da faceta *IContentController* e representada pelo relacionamento **exposedReceptacle(Sub,Rsub,CC,Rcc)** que expressa "**Sub** é um subcomponente de **CC** e possui um receptáculo **Rsub** externalizado através do receptáculo **Rcc** do componente **CC**".

A externalização **bindReceptacle** oferece a possibilidade de três tipos de mapeamento: 1-1, 1-n, n-1. Este mapeamento é feito de forma automática pela membrana sem a necessidade do uso de conectores. Para o mapeamento de uma dependência de um subcomponente através do componente composto deve ser válida a regra **isAValidExposedReceptacle (Sub, Rsub, CC, Rcc)** que expressa "**Sub** é um subcomponente de **CC** e pode realizar a externalização do receptáculo **Rsub** através da faceta **Rcc** do componente **CC**".

> isAValidExposedReceptacle (Sub,CC,Rsub,FC) :- receptacle (Rsub, T1, Cardinality, Sub), receptacle(FC, T2, Cardinality, CC), exposedReceptacle(Sub, RSub, CC, Rcc), subcomponent(Sub, CC),compatible(T2, T1).

Da mesma forma que um componente simples, um componente composto após o processo de mapeamento de facetas e receptáculos dos seus subcomponentes deve obedecer a regra **isAValidComponent** (Seção 3.1.1) para garantir a unicidade de identificadores em seu conjunto de facetas e receptáculos.

3.2.4 Regras de Conexão

As conexões no modelo SCS são oferecidas através da faceta *IReceptacles*. A realização de uma conexão é representada pelo relacionamento **connection** definido na seção 3.1.2. No SCS-*Composite*, dois componentes só podem ser conectados se pertencerem a uma mesma composição ou se ambos não pertencerem a nenhum componente composto. Assim, para a conexão de um receptáculo de um componente A com uma faceta de um componente B, uma das condições a seguir precisa ser verdadeira:

1. $SC_A \subset SC_B$
2. $SC_A = \emptyset \wedge SC_B = \emptyset$

SC_A é o conjunto de componentes compostos que encapsulam o componente A e SC_B é o conjunto de componentes compostos que encapsulam o componente B. Como discutido na Seção 2.3.6, optamos por não permitir o compartilhamento de componentes que possuem receptáculos, garantido pela regra **isAValidComponentSharing**. Desta forma, SC_A deve ser um conjunto vazio ou possuir apenas um elemento, e a validação **isAValidConnection** pode ser redefinida como:

> isAValidConnection(C1,F,C2,R) : − not(subcomponent(C1,_)), not(subcomponent(C2,_)), facet(F,T1,C1), receptacle(R,T2,_,C2), compatible(T1,T2).
>
> isAValidConnection(C1,F,C2,R) : − subcomponent(C1,U), subcomponent(C2,U), facet(F,T1,C1), receptacle(R,T2,_,C2), compatible(T1,T2).

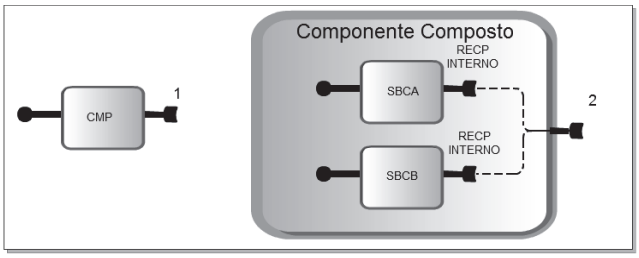

Figura 3.4: Os dois tipos de receptáculos presentes no SCS-*Composite*

Ainda sobre os mecanismos de *binding* é importante destacar que o SCS-*Composite* possui um novo tipo de receptáculo: o receptáculo externalizado (Vide 2 na Figura 3.4). O funcionamento de um receptáculo de um subcomponente externalizado é diferente do funcionamento de um receptáculo implementado por um componente (Vide 1 na Figura 3.4). Quando um receptáculo externalizado RC do componente C recebe uma faceta F para ser conectada realiza a conexão. Após realizar a conexão, cria um objeto *proxy* F' da faceta F. Este objeto *proxy* é criado

para evitar a conexão direta entre receptáculos de subcomponentes e facetas de componentes localizados fora do componente C que venham a ser conectados a eles. Esta ligação direta iria infrigir a regra 1 do modelo SCS, pois tornaria possível a conexão de um subcomponente de uma composição com um componente de fora da composição.

A criação de um *proxy* para permitir a conexão de receptáculos de subcomponentes com componentes externos ao componente composto gera uma nova regra **connectByComposite (Fext, Cext, Rsub, Sub, Rcc, CC)**. Esta regra define que "a faceta **Fext** do componente **Cext** externo ao componente composto **CC** foi enviada ao receptáculo **Rsub** de um subcomponente **Sub** através do receptáculo **Rcc** do componente composto **CC**".

```
connectedByComposite( Fext, Cext, Rsub, Sub, Rcc, CC ):- connection(
Cext, Fext, CC, Rcc), exposedReceptacle( Sub, Rsub, CC, Rcc).
```

3.3 Diretrizes do Modelo

O modelo SCS-*Composite* é baseado nos modelos Fractal e OpenCOM (vide Capítulo 2). Uma característica importante do modelo é prezar por poucas modificações na base do modelo SCS. Desta forma, espera-se facilitar o processo de migração das aplicações SCS. As próximas subseções têm como objetivo analisar o modelo SCS-*Composite* de acordo com as características definidas na Seção 2.3.

3.3.1 Suporte ao Desenvolvimento Incremental

O suporte ao desenvolvimento incremental possibilita que determinada aplicação, inicialmente, seja composta por apenas componentes primitivos e à medida que for evoluindo torne possível a coexistência de componentes primitivos e compostos.

O suporte a esta funcionalidade é importante por oferecer flexibilidade para desenvolvedores iniciantes, desenvolvimento de aplicações simples de forma rápida e a possibilidade de migração incremental de aplicações desenvolvidas utilizando apenas componentes primitivos. A Figura 3.5 ilustra os diferentes tipos de conexões entre facetas e receptáculos de componentes compostos e primitivos.

A conexão entre componentes no SCS-*Composite* segue a regra **connect** definida na Seção 3.2. Esta regra estabelece que uma conexão entre dois componentes só pode ser realizada se os dois componentes em questão pertencerem a uma mesma composição ou se ambos não pertencerem a nenhuma composição. A coexistência dessas duas condições permite o desenvolvimento incremental de aplicações baseadas em componentes compostos.

Como dito anteriormente um dos nossos objetivos é realizar o menor número de alterações no modelo SCS. Porém, para garantia das duas restrições de conexão no

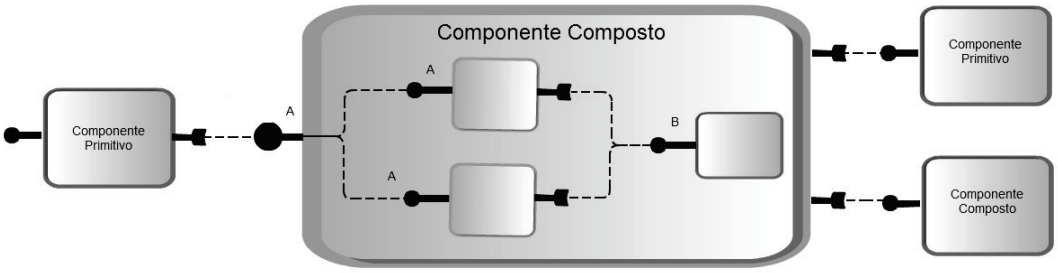

Figura 3.5: Exemplos de conexões entre componentes primitivos e compostos.

SCS-*Composite* tivemos que adicionar uma faceta obrigatória para todo componente: *ISuperComponent*. A partir desta interface o componente consegue informar quais componentes o encapsulam. Como dito no Capítulo 2, optamos pela mesma decisão do OpenCOM em oferecer este recurso, ao contrário do Fractal.

3.3.2 Mapeamento de Serviços dos Subcomponentes como serviços do componente composto e Conectores Exógenos

O mapeamento de serviços de subcomponentes através de interfaces do componente composto trata-se do *binding* vertical onde um serviço de um subcomponente é disponibilizado como um serviço do componente composto (vide Capítulo 1). O SCS-*Composite*, assim como no Fractal, oferece suporte nativo ao *binding* vertical com aridade 1-1 e n-1 através da operação **bindFacet** da interface *IContentController* e o uso de conectores para o *binding* vertical com aridade 1-n. A Figura 3.6 ilustra o cenário onde um subcomponente externaliza sua faceta com um identificador **hello** através de duas facetas do componente composto com identificadores, respectivamente, **externalHelloA** e **externalHelloB**. A aridade deste relacionamento é n-1.

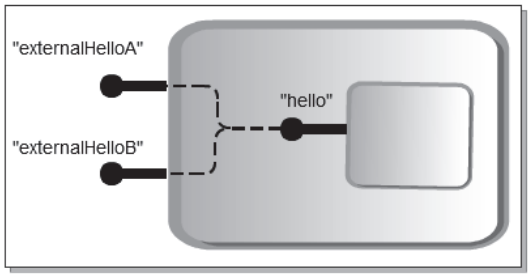

Figura 3.6: Mapeamento de uma faceta de um subcomponente para facetas externas diferentes.

A figura 3.7 ilustra como deve ser realizado o *binding* com aridade 1-n com o uso de conectores exógenos. O cenário possui dois componentes C1 e C2 que oferecem uma faceta de tipo A (Código 3.1). Para estas facetas serem externalizadas por apenas uma interface do componente composto o usuário da API deve implementar

um componente conector. Este conector deve implementar a mesma interface A de C1 e C2 e servirá como um mediador entre os serviços dos subcomponentes e a faceta do componente composto.

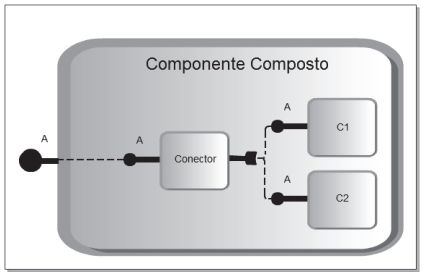

Figura 3.7: Funcionamento de um conector no processo de externalização de facetas.

C1 e C2 devem implementar a interface A e o conector fica responsável por implementar como funcionará o fluxo de comunicação entre o componente composto e subcomponente. O Código 3.1 faz uma demonstração a partir de um pseudocódigo em Java de como funcionaria este esquema. A chamada a operação **f** da faceta externalizada **A** do componente composto que agrega C1 e C2 deve ter como resultado uma saída de texto em C1 e C2.

```
1 interface A{
2   void f(String str);
3 }
4
5 Conector implements A
6 {
7   void f(String str){
8     // Envia str para C1 e C2.
9     //Como o método é void não espera retorno de C1 e C2
10  }
11 }
12
13 C1 implements A
14 {
15  void f(String str){
16    //imprime str
17  }
18 }
19
20 C2 implements A
21 {
22  void f(String str){
23    //imprime str
24  }
25 }
```

Código 3.1: Exemplo de uso de um conector para realizar o *binding* vertical com aridade 1-n.

Como trabalho futuro, o usuário da API terá disponível uma biblioteca auxiliar para criação de conectores. A idéia é que a partir desta API, o usuário tenha acesso a conectores padrões (Vide Seção 2.3.4)e, também, seja possível customizar conectores para sua própria aplicação.

Por fim, um componente primitivo segue uma regra para definição de suas facetas, onde dado um componente C constituído por um conjunto de facetas $F = \{f_1, f_2, ...f_n\}$, cada faceta possui um identificador único. Da mesma forma, um componente composto respeita esta regra e a operação **bindFacet** não permite que sejam externalizadas duas facetas com o mesmo nome.

3.3.3 Relacionamentos Serviço/Componente e Subcomponente/Componente Composto

O relacionamento **facet(identifier,type,component)** definido na Seção 3.1.1 representa o conceito de faceta (serviço) no modelo SCS. De acordo com este relacionamento é possível acessar o componente que implementa determinada faceta. No modelo SCS-*Composite* é apresentado o conceito de faceta externalizada. A externalização de uma faceta F = **facet (fA, typeA, cmpA)** de um subcomponente **cmpA** através do componente composto **cmpB** com identificador fA' adiciona um novo relacionamento F' = **facet (fA', typeA, cmpB)**. Nota-se que uma faceta externalizada possui o mesmo comportamento de um faceta básica garantido a integridade do relacionamento Serviço/Componente. Este mesmo comportamento de uma faceta nativa e externalizada é presente também nos modelos Fractal e OpenCOM.

O relacionamento Subcomponente/Componente composto garante mecanismos para verificar a estrutura hierárquica de uma aplicação. Assim, optamos por oferecer este recurso através das interface *ISuperComponent* e *IContentController*. A primeira permite a navegação no sentido *filho* $\Rightarrow\triangleright$ *pai* e a segunda a navegação *pai* $\Rightarrow\triangleright$ *filho*.

3.3.4 Mapeamento de Dependências de Subcomponentes como Dependências do Componente Composto

O mapeamento de dependências internas através de interfaces externas é realizado através da operação **bindReceptacle** (vide Seção 3.2). Tanto no mapeamento de serviços quanto de dependências, criamos novas operações para realizar o *binding* vertical (**bindFacet** e **bindReceptacle**), diferentemente do Fractal que oferece todas operações de *binding* sem distinguir entre *bindings* horizontais e verti-

cais. Esta nossa decisão preza por tornar a API explícita para o usuário do modelo SCS-*Composite*.

Nossas operações de *binding* entre receptáculos internos e externos contemplam os mapeamentos ilustrados na Figura 2.6 no Capítulo 2. Caso o usuário queira realizar um tipo de fluxo de comunicação mais complexo pode utilizar a mesma abordagem de conectores para o fluxo de comunicação, assim como descrito na Seção 3.3.2 sobre o *binding* vertical de facetas.

A Figura 3.8 ilustra o *binding* entre um receptáculo de um subcomponente e de um componente composto. Um componente composto pode requisitar dependências para um subcomponente. Em 1 o receptáculo **RECP INTERNO** é externalizado através do receptáculo **RECP EXTERNO** do componente composto CC. Em 2 é realizado o *binding* horizontal entre CC e CB. O receptáculo **RECP EXTERNO** ao receber a faceta **FCT B** do componente CB realiza a conexão e, em seguida, repassa um *proxy* da faceta **FCT B** para os receptáculos dos subcomponentes. Como dito na Seção 3.2.4, este *proxy* foi criado para evitar a conexão direta de receptáculos de subcomponentes de um componente composto C com facetas implementadas por um componente de fora do componente C.

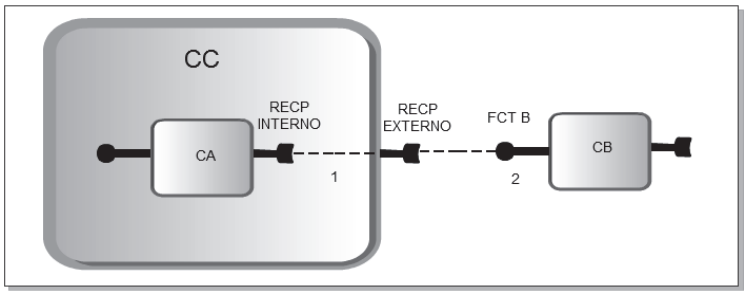

Figura 3.8: Representação de externalização de um receptáculo de um subcomponente.

Um recurso do modelo SCS é a navegação entre facetas a partir das conexões de um receptáculo. A indireção realizada no momento que o receptáculo externo repassa a faceta para os subcomponentes faz com que este recurso de navegação seja perdido. Desta forma, na Figura 3.8, a verificação sobre qual componente oferece o serviço ao subcomponente CA retorna CC ao invés de CB. O usuário da API sabendo que o serviço foi repassado, deve inspecionar de maneira diferente o subcomponente CA. Primeiro, o usuário deve navegar do subcomponente CA para o componente composto, em seguida, para o receptáculo externo, e, por fim, para o componente que oferece o serviço.

3.3.5 Compartilhamento de subcomponentes

O SCS-*Composite* oferece compartilhamento de componentes com a restrição de que para um subcomponente ser compartilhado não deve possuir receptáculos.

O Fractal oferece este recurso sem nenhuma restrição. Analisando bem esta característica chegamos a conclusão de que é necessária esta restrição para solucionar alguns problemas de ambiguidade (Vide Seção 2.2.6).

Desta forma, optamos por deixar o SCS-*Composite* com compartilhamento restritivo. A interface *ISuperComponent* (vide Código 4.2) é importante para garantir a integridade desta regra a medida que no processo de adição de um novo subcomponente em um componente é necessária a validação desta restrição, isto é, verificar se este subcomponente é encapsulado por um outro componente e, caso seja, se possui algum receptáculo.

3.3.6 Mecanismos de Reconfiguração

No Capítulo 2, fizemos um levantamento sobre como o Fractal e OpenCOM tratam esta característica. Ambos modelos oferecem suporte à reconfiguração dinâmica de baixo nível. O nosso trabalho não abrange a modelagem e implementação de tais mecanismos, entretanto, como trabalho futuro seria interessante investigar tal recurso. Também, seria interessante investigar mecanismos de reconfiguração como o FScript do Fractal, que oferece a possibilidade de realizar operações com semântica ACID de mais alto nível.

Para permitir tais mecanismos de reconfiguração, seria interessante investigar a adição da operação **suspend** ao ciclo de vida de um componente. Esta operação deixaria um componente em um estado válido para reconfiguração, isto é, qualquer requisição feita seria bloqueada e o componente poderia ser alterado garantindo a consistência da aplicação. Em uma primeira análise, a operação **suspend** pode causar um efeito colateral em função do nosso modelo oferecer o compartilhamento de subcomponentes. Por exemplo, é natural imaginar que operação **suspend** realizada sobre o componente composto CC que encapsula o componente C irá suspender o subcomponente C. A suspensão do componente C pode modificar o comportamento de outros componentes que o encapsulam. É necessário investigar qual a melhor forma de oferecer mecanismos para reconfiguração dinâmica no SCS-*Composite*.

3.4 Considerações Finais

Neste capítulo apresentamos a especificação do SCS e de sua extensão com suporte a componentes compostos: SCS-*Composite*. O estudo realizado no Capítulo 2 sobre o Fractal e OpenCOM levando em consideração aspectos do conceito de componentes compostos foi fundamental para as decisões de projeto que definiram o modelo SCS-*Composite*.

No SCS original, uma composição é o resultado da conexão de componentes e não pode ser classificada como um componente por falta de um mecanismo que a encapsule. Este mecanismo deve garantir que as restrições de consistência de um

componente sejam cumpridas e possibilitar duas formas de visão da composição. A primeira, como uma caixa-preta, deve omitir os componentes da composição e oferecer as facetas básicas de um componente. Já a segunda, deve explicitar os componentes da composição e oferecer mecanismos de configuração e introspecção estrutural sobre os mesmos.

Para preencher a falta de expressividade do conceito de componentes compostos no SCS, foi realizada a extensão do modelo SCS apresentado formalmente na Seção 3.2. Basicamente, foi adicionada a abstração de uma membrana sobre uma composição e uma nova faceta *IContentController*. Esta faceta oferece mecanismos para a configuração e introspecção estrutural de subcomponentes da composição e, para o mapeamento de serviços e dependências.

O mapeamento de serviços e dependências é conhecido como *binding* vertical. Após o *binding* vertical de uma faceta de um subcomponente é criada uma nova faceta no componente composto que oferece o serviço implementado pelo subcomponente. O componente composto pode funcionar como uma caixa-preta onde o usuário não precisa saber se o serviço está sendo realizado por um subcomponente ou pelo próprio componente composto. Com a introdução do *binding* vertical para receptáculos, foi criado um novo tipo de receptáculo (vide Seção 3.2.4). Este novo tipo de receptáculo aceita conexões para o componente composto e repassa as dependências para os subcomponentes.

Primeiramente, tínhamos como base para o projeto implementarmos uma extensão do modelo pouco intrusiva com objetivo de facilitar a migração das aplicações SCS. Entretanto, em várias situações foram necessárias modificações nos mecanismos básicos do SCS. A primeira foi decorrente da mudança na estrutura do componente primitivo. O componente primitivo no SCS-*Composite* possui uma faceta obrigatória adicional: *ISuperComponent*. Esta faceta, também, é obrigatória para componentes compostos. Através desta faceta um subcomponente pode acessar os componentes compostos que o encapsulam proporcionando uma navegação hierárquica de uma aplicação. Este tipo de navegação hierárquica é fundamental para validação sobre as restrições de conexões e compartilhamento de subcomponentes.

Por fim, a partir das diretrizes do modelo proposto, nota-se que o SCS-*Composite* possui forte influência do Fractal. Por exemplo, ele oferece suporte à externalização de um serviço ou dependência de um subcomponente e a possibilidade do uso de componentes conectores para o *binding* vertical (Vide Seção 3.3.2). Este tipo de componente funciona como um mediador entre as facetas dos subcomponentes e a faceta de um componente composto permitindo o *binding* vertical com aridade 1-n. Também, assim como no Fractal, o SCS-*Composite* oferece navegação do subcomponente para o componente composto importante para os mecanismos de introspecção estrutural e, compartilhamento de subcomponentes entre os componentes compostos. Entretanto, o nosso modelo proposto também possui algumas características oriundas do OpenCOM como suporte ao desenvolvimento incremental e uma API intuitiva oferecendo operações diferentes para o *binding* horizontal e

vertical.