2
Trabalhos Relacionados

Neste capítulo será apresentado um levantamento realizado sobre dois modelos de componentes que incorporam o conceito de componentes compostos: Fractal (Bruneton et al., 2006) e OpenCOM (Coulson et al., 2008). Para estudo prático do Fractal e OpenCOM foram utilizadas, respectivamente, as implementações Julia (Bruneton et al., 2006) e OpenCOMJ (Michael Clarke, 2007).

O *framework* de classificação desenvolvido por Crnkovic et al. (2010) apresenta diversos modelos que incorporam o conceito de componentes compostos como Fractal, OpenCOM, Kobra (Atkinson et al., 2001), Koala (van Ommering et al., 2000) e SaveCCM (Hansson et al., 2004). Especificamente, neste trabalho optamos por realizar um levantamento sobre o Fractal e OpenCOM por dois motivos principais.

Em primeiro lugar, nosso estudo experimental com componentes compostos irá utilizar o modelo de componentes SCS que abrange aplicações de propósito geral. Ambos os modelos, Fractal e OpenCOM, apresentam o mesmo domínio de aplicações de propósito geral. Este grupo de modelos oferece mecanismos básicos de especificação e composição de componentes através de suposições sobre formas de interação, suporte a sistemas distribuídos, compilação e implantação em tempo de execução. O segundo motivo foi a disponibilidade de documentação e acesso ao código fonte das implementações dos modelos que nos permite realizar tanto um estudo teórico quanto prático.

Crnkovic et al. (2010) não entra em maiores detalhes sobre o conceito de componentes compostos. Através deste estudo sobre o Fractal e OpenCOM temos como objetivo enumerar características presentes no conceito de componentes compostos e como cada modelo trata essas características. Dentre as características podemos citar:

- Suporte ao desenvolvimento incremental;
- Mapeamento de serviços dos subcomponentes como serviços do componente composto (externalização de serviços);
- Mapeamento de dependências de subcomponentes como dependências do componente composto;

PUC-Rio - Certificação Digital Nº 1012620/CA

– Conectores para *binding* vertical;

– Relacionamento Serviço/Componente;

– Compartilhamento de subcomponentes;

– Navegação entre subcomponente e componente composto;

– Mecanismos de Reconfiguração;

Nas Seções 2.1 e 2.2 serão apresentados os modelos OpenCOM e Fractal. Em seguida, na Seção 2.3, serão analisadas as características descritas acima.

2.1 Fractal

Fractal é um modelo de componentes desenvolvido pela *France Telecom R&D* e INRIA que tem como objetivo oferecer suporte para todo o ciclo de vida de desenvolvimento de software complexo (*design*, implementação, implantação e manutenção/gerenciamento). Dentre seus recursos podemos destacar: (i) o suporte a componentes compostos oferecendo visões da aplicação em diferentes níveis de abstração, (ii) componentes compartilhados, (iii) capacidade de introspecção (monitorar ou controlar um sistema em execução), e, por fim, (iv) capacidade de reconfiguração (implantar e configurar um sistema de forma dinâmica).

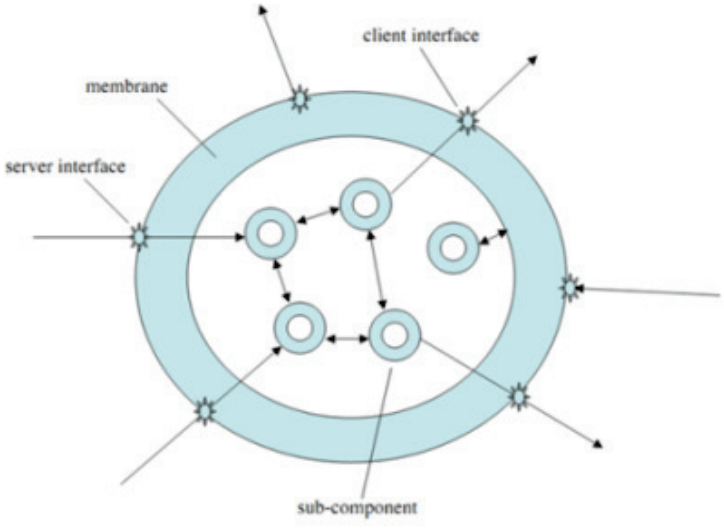


Figura 2.1: Estrutura de um componente no modelo Fractal.

Um componente Fractal é uma unidade lógica que oferece uma ou mais interfaces. As interfaces são o ponto de acesso ao componente e possuem um tipo (semelhante a interfaces em Java). O Fractal distingue entre interfaces

PUC-Rio - Certificação Digital Nº 1012620/CA

servidoras que oferecem serviços e interfaces clientes que invocam serviços em outros componentes.

Como ilustrado na Figura 2.1, um componente é definido como uma membrana que oferece interfaces para inspecionar e reconfigurar seus elementos internos. Esta membrana encapsula um conjunto finito de outros componentes (chamados subcomponentes). A membrana possui interfaces internas e externas, sendo que as interfaces internas só podem ser acessadas pelos subcomponentes que fazem parte do conteúdo da membrana.

A API do Fractal possui interfaces de diversos tipos. Dentre elas encontram-se as interfaces que oferecem serviços para introspecção, instanciação, configuração e interação entre os componentes. As interfaces de introspecção e configuração oferecem mecanismos para inspecionar e configurar os recursos internos (atributos, *bindings*, subcomponentes e ciclo de vida) de um componente. As interfaces para instanciação de componentes funcionam como *factories* (um tipo de componente fábrica). Estas fábricas tornam possível a criação de componentes a partir da definição de um conjunto de interfaces que o compõem. Para cada interface devem ser especificados o nome (identificador único), papel (cliente ou servidora), tipo, obrigatoriedade e cardinalidade[1].

Especificamente, entraremos em maiores detalhes sobre as interfaces *ContentController* (Código 2.1), *SuperController* (Código 2.2) e *BindingController* (Código 2.3). Um componente composto deve oferecer, obrigatoriamente, estas três interfaces. A partir destas interfaces é possível realizar mecanismos de introspecção e configuração sobre os subcomponentes bem como mapeamento de serviços e dependências de subcomponentes através do componente composto.

A interface *ContentController* define cinco operações: três operações de inspeção **getFcSubComponents**, **getFcInternalInterfaces** e **getFcInternalInterface** que retornam, respectivamente, os subcomponentes e interfaces; e, as operações de configuração **addFcSubComponent** e **removeFcSubComponent**.

```
1 package org.objectweb.fractal.api.control;
2
3 interface ContentController {
4
5   any[] getFcInternalInterfaces ();
6   any getFcInternalInterface (string itfName) throws
        NoSuchInterfaceException;
7   Component[] getFcSubComponents ();
8   void addFcSubComponent (Component c)
9     throws IllegalContentException, IllegalLifeCycleException;
```

[1]A cardinalidade de uma interface de tipo T indica quantas interfaces de tipo T um componente pode ter.

PUC-Rio - Certificação Digital Nº 1012620/CA

```
10   void removeFcSubComponent (Component c)
11     throws IllegalContentException, IllegalLifeCycleException;
12 }
```

Código 2.1: Interface *ContentController* que faz parte do conjunto de interfaces para controle de conteúdo do Fractal

A interface *SuperController* define a operação **getFcSuperComponents** onde dado um subcomponente são retornados componentes compostos que o encapsulam. Através desta interface é oferecida a navegação no sentido *filho* $=\triangleright$ *pai*.

```
1 package org.objectweb.fractal.api.control;
2
3 interface SuperController {
4    Component[] getFcSuperComponents ();
5 }
```

Código 2.2: Interface *SuperController* que faz parte do conjunto de interfaces para controle de conteúdo do Fractal

Por fim, a API do Fractal possui a interface *BindingController* (Vide Código 2.3). Um componente oferece esta interface para realizar a interação entre as interfaces clientes e servidoras (*binding* horizontal) e mapeamento de serviços e dependências dos subcomponentes (*binding* vertical).

```
1 package org.objectweb.fractal.api.control;
2
3 interface BindingController {
4    string[] listFc ();
5    any lookupFc (string clientItfName)
6       throws NoSuchInterfaceException;
7    void bindFc (string clientItfName, any serverItf)
8       throws NoSuchInterfaceException, IllegalBindingException,
           IllegalLifeCycleException;
9    void unbindFc (string clientItfName)
10      throws NoSuchInterfaceException, IllegalBindingException,
           IllegalLifeCycleException;
11 }
```

Código 2.3: Interface para realizar *bind* e *unbind* entre interfaces clientes e outros componentes através de *bindings* primitivos.

A interface *BindingController* define operações de inspeção e configuração. A operação **listFc** retorna os nomes das interfaces clientes de um componente e a operação **lookupFc** recebe como parâmetro o nome de uma interface cliente de um componente e retorna a interface servidora que está

PUC-Rio - Certificação Digital Nº 1012620/CA

conectada a esta interface. Para configuração dos *bindings* são oferecidas as operações **bindFc** e **unbindFc**.

A operação **bindFc** realiza a conexão entre uma interface cliente de um componente e a interface servidora de outro componente bem como o mapeamento de uma interface de um subcomponente com a interface da membrana (*binding* vertical), e recebe como parâmetro o nome de um interface cliente de um componente e a interface servidora de outro componente. Nota-se que o Fractal oferece as mesmas operações para os *bindings* horizontais e verticais. A operação **unbindFc** recebe como parâmetro o nome da interface cliente de um componente e desconecta todas interfaces servidoras associadas.

2.2 OpenCOM

OpenCOM é um modelo de componentes genérico desenvolvido pela Universidade de Lancaster que oferece suporte a uma vasta gama de aplicações de diferentes domínios. Nesse modelo um componente é uma unidade de funcionalidades encapsuladas que interagem com outros componentes através de pontos de interação servidores e clientes.

Os conceitos principais do modelo OpenCOM são interfaces, receptáculos e conexões (*binding* entre interfaces e receptáculos). Uma interface expressa os serviços oferecidos por um componente, um receptáculo as suas dependências e as conexões suas interações. È importante ressaltar que um componente não possui duas interfaces de mesmo tipo, esta mesma regra se aplica aos receptáculos.

O OpenCOM oferece como recurso um *kernel* padrão em tempo de execução (Vide Figura 2.2). Este *kernel* gerencia a criação e remoção de componentes e, auxilia na gerência das conexões. Ademais, existe uma interface gráfica que apresenta a aplicação como um grafo onde os componentes são nós e os *bindings* arestas. A partir desta representação é possível realizar introspecção sobre a estrutura da aplicação em tempo de execução.

Um componente OpenCOM possui quatro interfaces obrigatórias: *IConnections*, *ILifeCycle*, *IMetaInterface* e *IUnknown*. Os componentes interagem através das operações da interface *IConnections* (**connect** e **disconnect**). A partir da interface *ILifeCycle* o componente oferece mecanismos para adicionar comportamento no momento de inicialização e finalização (**startup** e **shutdown**). Para inspeção, o componente possui a interface *IMetaInterface* que oferece informações sobre os metadados de um componente (descrições das interfaces e receptáculos). Por fim, através da interface *IUnknown* é possível a navegação entre facetas de um componente.

PUC-Rio - Certificação Digital Nº 1012620/CA

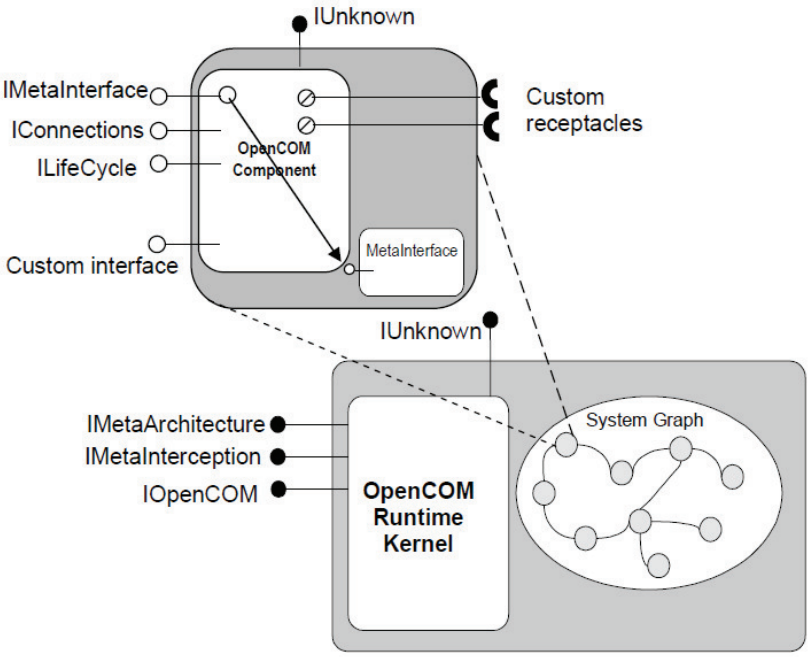


Figura 2.2: Arquitetura do modelo de componentes OpenCOM (Michael Clarke, 2007).

Ademais, o *kernel* do OpenCOM provê um metamodelo de interceptadores através da interface *IMetaInterception* e um metamodelo da arquitetura através da interface *IMetaArchitecture*. Interceptadores tornam possível adicionar pré e pós-comportamento a uma interface de um componente. O metamodelo de arquitetura oferece suporte a inspeção sobre conexões entre componentes.

O conceito de componentes compostos está presente através do mecanismo de *Component Framework* (CF) (Vide Figura 2.3). Ao contrário do Fractal, as aplicações modeladas utilizando OpenCOM não possuem como requisito obrigatório possuir ao menos um componente composto. O *design* de um CF é baseado nas seguintes propriedades:

1. Um CF em OpenCOM é um componente composto;

2. Um CF oferece um interface adicional para inspeção e adaptação dinâmica da arquitetura local do componente composto;

3. A integridade de cada componente do CF é garantida após uma reconfiguração dinâmica. Para garantia desta integridade são implementadaos mecanismos de bloqueio de escrita/leitura através de semáforos.

Um CF é implementado como uma extensão de um componente primitivo possuindo as mesmas interfaces obrigatórias. Adicionalmente, possui a interface *ICFMetaInterface* (Código 2.4) e contém em sua estrutura interna uma

PUC-Rio - Certificação Digital Nº 1012620/CA

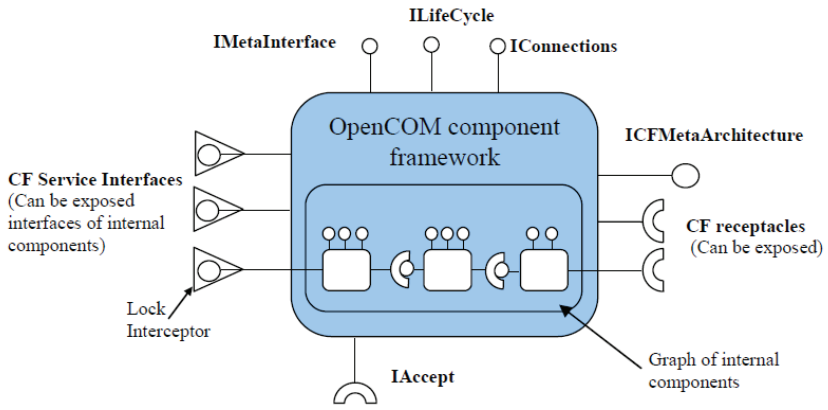


Figura 2.3: Um *Component Framework* em OpenCOM.

composição de componentes. É importante destacar que um CF pode ser visualizado externamente como um componente primitivo. Este recurso simplifica a composição de arquiteturas hierárquicas e promove o reuso.

```
1  public interface ICFMetaInterface extends IUnknown{
2
3    IUnknown find_component(String pIntf);
4    int get_internal_components(Vector<IUnknown> ppComps);
5    IUnknown getComponentPIUnknown(String CompName);
6    String getComponentName(IUnknown CompIUnk);
7    int get_bound_components(IUnknown comp, Vector<CFMetaInterface.
       ConnectedComponent> ppConnections);
8    int get_internal_bindings(Vector<Long> pConnIDS);
9    int get_detailed_exposed_interfaces(Vector<CFMetaInterface.
       ExposedInterface> ppIntfs);
10   int get_exposed_interfaces(Vector<String> ppIntfs);
11   int get_exposed_receptacles(Vector<CFMetaInterface.
       ExposedReceptacle> ppComps);
12   long local_bind( IUnknown pIUnkSource,IUnknown pIUnkSink,String
       InterfaceType);
13   boolean break_local_bind(long connID);
14   IUnknown create_component(String componentType,String
       componentName);
15   boolean insert_component(IUnknown pCompReference);
16   boolean delete_component(IUnknown pIUnknown);
17   boolean expose_interface(String rintf,IUnknown pComp);
18   boolean unexpose_interface(String rintf,IUnknown pComp);
19   boolean unexpose_all_interfaces();
20   boolean expose_receptacle(String rintf,IUnknown pComp,String
       recpType);
21   boolean unexpose_receptacle(String rintf,IUnknown pComp);
22   boolean unexpose_all_receptacles();
23 }
```

PUC-Rio - Certificação Digital Nº 1012620/CA

Código 2.4: Interface *ICFMetaInterface* que oferece operações de configuração e inspeção de um CF

A interface *ICFMetaInterface* oferece operações para configurar e inspecionar a composição de subcomponentes. Especificamente, vamos entrar em maiores detalhes sobre as operações desta interface que incorporam o conceito de componentes compostos no modelo. A adição e remoção de subcomponentes é feita a partir das operações **create_component** que cria um componente com tipo e identificador único dentro do CF, **insert_component** que insere um componente previamente instanciado e **delete_component** que remove um subcomponente.

As interações entre subcomponentes dentro de um CF devem ser realizadas através das operações **local_bind** e **break_local_bind**. Tais operações garantem que somente subcomponentes pertencentes ao mesmo CF possam realizar conexões. Por fim, para a externalização de serviços e dependências dos subcomponentes o usuário da API deve utilizar as operações **expose_interface** e **expose_receptacle**. E, para remoção da externalização as operações **unexpose_interface** e **unexpose_receptacle**.

As operações para inspeção de um CF retornam informações sobre subcomponentes, conexões internas e detalhes sobre os mapeamento de serviços e dependências dos subcomponentes. As operações **find_component**, **getComponentPIUnknown** e **getComponentName** retornam, respectivamente, um subcomponente dada sua interface, a interface *IUnknown* de um componente dado seu identificador único e o nome de um componente dada uma interface *IUnknown*. A operação **get_internal_components** disponibiliza todos os subcomponentes de um CF.

Para verificar os componentes conectados a determinado subcomponente existe a operação **get_bound_components**. A operação **get_internal_bindings** retorna todas as conexões internas entre os subcomponentes. Por fim, as operações **get_detailed_exposed_interfaces**, **get_exposed_interfaces** e **get_exposed_receptacles** informam detalhes sobre o mapeamento. Por exemplo, a operação **get_exposed_interfaces** torna possível descobrir qual subcomponente externalizou determinada interface de um CF.

Um CF tem como papel importante garantir que a composição de seus subcomponentes sempre esteja em um estado válido. Atraves do receptáculo *IAccept*, um CF realiza um teste de consistência sobre uma determinada configuração de seus subcomponentes. Para executar o teste, o CF requisita o estado atual de seus subcomponentes (todas as conexões e *bindings* verticais)

PUC-Rio - Certificação Digital Nº 1012620/CA

para o *kernel*. Em posse da configuração é possível verificar a mesma atende as regras do modelo OpenCOM.

2.3 Características de Componentes Compostos

Está seção está dividida em subseções que descrevem características identificadas nos modelos Fractal e OpenCOM e, questionamentos surgidos na fase de *design* do nosso estudo experimental. A investigação sobre como Fractal e OpenCOM tratam cada característica serviram de base para o *design* de nosso modelo experimental. O formato de cada subseção é:

- Breve descrição sobre a característica;
- Análise sobre como o Fractal e OpenCOM tratam a característica;
- Decisão realizada para o *design* do SCS com suporte a componentes compostos.

2.3.1 Suporte ao Desenvolvimento Incremental

Esta dimensão define se o modelo permite que uma aplicação inicialmente seja composta por apenas componentes primitivos e, à medida que for evoluindo torne possível a coexistência de componentes primitivos e compostos. O projeto Julia (modelo Fractal) trata como requisito obrigatório a existência de pelo menos um componente composto encapsulando toda a aplicação. Por outro lado, o OpenCOM oferece a flexibilidade no desenvolvimento de aplicações oferecendo o suporte ao desenvolvimento incremental.

O recurso de desenvolvimento incremental é interessante para desenvolvedores iniciantes que não são familiarizados com conceitos de modelos de componentes (facetas, receptáculos, *bindings*). Em um primeiro momento, pode ser complicado entender a abstração de componentes compostos. Esta flexibilidade permite ao desenvolvedor iniciante, primeiramente, se familiarizar com os conceitos básicos do paradigma de orientação a componentes e após ultrapassar esta curva de aprendizagem entender o conceito de componentes compostos.

Também, a flexibilidade desta dimensão é uma ótima alternativa para o desenvolvimento de aplicações simples. Por exemplo, a restrição imposta pelo Julia faz com que em um simples "Hello World", o desenvolvedor da aplicação se preocupe com tarefas de definição e instanciação de um componente composto, encapsulamento de subcomponentes e, mapeamento de serviços e dependências internas.

PUC-Rio - Certificação Digital Nº 1012620/CA

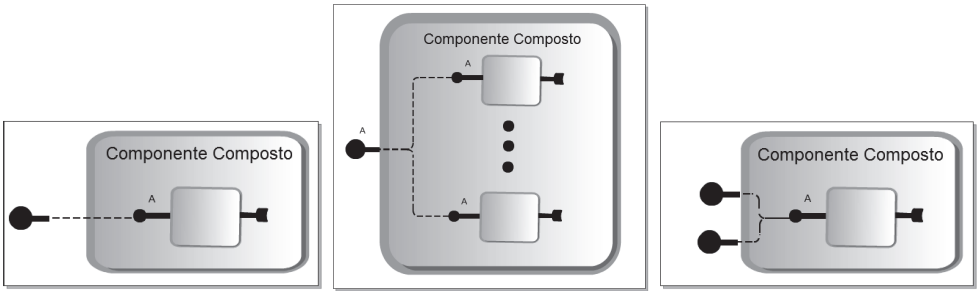
Por fim, outro benefício desta dimensão é a possibilidade de migração incremental de aplicações desenvolvidas utilizando apenas componentes primitivos para aplicações com suporte a componentes compostos.

Em função dos benefícios acima vistos, optamos por adicionar o suporte a componentes compostos para a nossa extensão do SCS da mesma forma que o OpenCOM.

2.3.2 Mapeamento de Serviços dos Subcomponentes como Serviços do Componente Composto

Uma funcionalidade de um componente composto é tornar possível o mapeamento de serviços dos subcomponentes através de interfaces do componente composto. A Figura 2.4 ilustra os três tipos de mapeamentos identificados em nosso trabalho de acordo com sua aridade: (i) o mapeamento do tipo 1-1 onde um serviço de um subcomponente é externalizado através de uma interface do componente composto, (ii) o mapeamento do tipo 1-n onde uma interface do componente composto representa um serviço de um ou mais subcomponentes, permitindo que a interface externa controle de forma coletiva as interfaces dos subcomponentes e (iii) o mapeamento do tipo n-1 onde um serviço de um subcomponente é externalizado através de uma ou mais interfaces do componente composto.

Figura 2.4: Mapeamentos de serviços dos subcomponentes através de interfaces do componente composto.



2.4(a): Mapeamento 1-1 - Um serviço de um subcomponente é externalizado por uma interface do componente composto.

2.4(b): Mapeamento 1-n - Uma interface do componente composto representa um serviço de mesmo tipo de um ou mais subcomponentes.

2.4(c): Mapeamento n-1 - Um serviço de um subcomponente é externalizado através de interfaces diferentes do componente composto.

Para exemplificar os tipos de mapeamento de serviços podemos citar o exemplo de uso, o CAS, que será descrito em maiores detalhes no Capítulo 5. Em um cenário temos a captura de uma apresentação realizada em uma sala através de quatro câmeras: duas do lado esquerdo (C1 e C2) e duas do lado di-

PUC-Rio - Certificação Digital Nº 1012620/CA

reito (C3 e C4) (Vide Figura 2.5). A sala representa um componente composto que encapsula quatro subcomponentes que representam, respectivamente, as câmeras C1, C2, C3 e C4.

Nesta aplicação muitas vezes será necessário realizar operações sobre as quatro câmeras (iniciar, pausar, parar e verificar status) de uma só vez. Este comportamento representa a externalização de um serviço de um ou mais subcomponentes através de uma interface do componente composto (aridade 1-1 e 1-n). Entretanto, em algumas situações será necessário realizar operações específicas sobre câmeras levando em consideração sua localização. Por exemplo, parar a gravação das câmeras localizadas apenas no lado esquerdo da sala. Este comportamento representa o mapeamento n-1, onde o usuário que gerencia a sala acessa uma mesma interface de um subcomponente através de interfaces diferentes do componente composto.

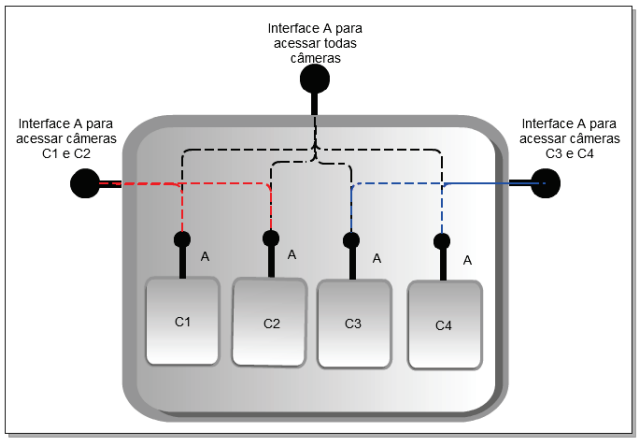


Figura 2.5: Exemplo de um cenário do CAS demonstrando a necessidade de disponibilizar serviços internos por interfaces externas diferentes.

O Código 2.5 mostra um pseudocódigo em Julia do cenário ilustrado na Figura 2.5. Este código demonstra a especificação de dois tipos de componente: **compositeType** (linhas 4-8) e **cameraType** (linhas 10-12). Na especificação destes dois componentes nota-se o uso da operação **createFctItfType** onde são definidos parâmetros como nome (identificador), tipo, cliente ou servidor, obrigatoriedade e cardinalidade. Em seguida, temos a criação da instância dos dois componentes (linhas 14 e 15) e operações que tornam serviços da câmera C1 disponível por duas portas do componente composto (linhas 17 e 18).

Diferente do Fractal, o OpenCOM não oferece suporte nativo ao mapeamento de serviços internos através de diferentes interfaces externas do componente composto. O seu método responsável pelo *binding* vertical, **expose_interface**, possui dois argumentos: o tipo de interface que será exposto e o subcomponente que possui o serviço, permitindo apenas a externalização de

PUC-Rio - Certificação Digital Nº 1012620/CA

uma interface através de uma única interface do componente composto. Esta restrição no mapeamento é decorrente da própria estrutura de um componente primitivo que tem como regra não possuir duas interfaces de mesmo tipo.

```
1  Component boot = Fractal.getBootstrapComponent();
2  TypeFactory tf = Fractal.getTypeFactory(boot);
3
4  ComponentType compositeType = tf.createFcType(new InterfaceType
       [] {
5    tf.createFcItfType("ALL_CAM", "cas.ICamera", false, false,
         false),
6    tf.createFcItfType("LEFT_CAM", "cas.ICamera", false, false,
         false)
7    tf.createFcItfType("RIGHT_CAM", "cas.ICamera", false, false,
         false)
8  });
9  ComponentType cameraType = tf.createFcType(new InterfaceType[] {
10   tf.createFcItfType("CAM", "cas.ICamera", false, false, false)
11 });
12
13 GenericFactory cf = Fractal.getGenericFactory(boot);
14 Component composite = cf.newFcInstance(compositeType,"composite"
       , null);
15 Component c1 = cf.newFcInstance(cameraType, "primitive", "cas.
       Camera");
16 Fractal.getContentController(composite).addFcSubComponent(c1);
17 Fractal.getBindingController(composite).bindFc("ALL_CAM", c1.
       getFcInterface("CAM"));
18 Fractal.getBindingController(composite).bindFc("LEFT_CAM", c1.
       getFcInterface("CAM"));
```

Código 2.5: Exemplo de serviços internos de um componente composto sendo disponibilizados por interfaces diferentes.

Como decisão de projeto para o SCS, optamos por adicionar os três tipos de mapeamento de serviços (1-1, 1-n e n-1) de um subcomponente através de interfaces do componente composto. Esta decisão é baseada na necessidade de nosso exemplo de uso, o CAS, e de outras aplicações SCS que podem ser beneficiadas.

2.3.3 Mapeamento de dependências de subcomponentes como dependências do componente composto

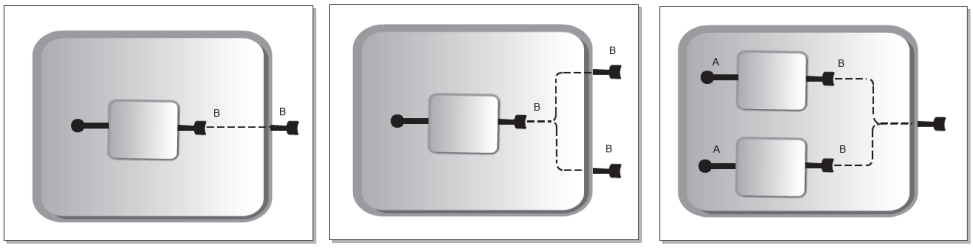
Esta característica diz respeito ao processo de externalização de dependências dos subcomponentes. A partir deste recurso é possível o componente composto requisitar dependências para seus subcomponentes. A Figura

PUC-Rio - Certificação Digital Nº 1012620/CA

2.6 ilustra os três tipos de mapeamentos de dependências identificados em nosso trabalho:

- **Mapeamento 1-1**: uma dependência de um subcomponente é mapeada como uma dependência do componente composto (Figura 2.6(a)). Tanto o Fractal quanto OpenCOM oferecem este tipo de mapeamento.
- **Mapeamento 1-n**: uma dependência de um subcomponente é mapeada como uma ou mais dependências do componente composto (Figura 2.6(b)). Este recurso é oferecido apenas pelo Fractal.
- **Mapeamento n-1**: dependências de diferentes subcomponentes são mapeadas como apenas uma dependência do componente composto (Figura 2.6(c)). Este recurso é oferecido apenas pelo Fractal.

Figura 2.6: Mapeamentos de dependências dos subcomponentes através de interfaces externas do componente composto.



2.6(a): Mapeamento 1-1 - Uma dependência de um subcomponente é mapeada como uma dependência do componente composto.

2.6(b): Mapeamento 1-n - Uma dependência de um subcomponente é mapeada como uma ou mais dependências do componente composto.

2.6(c): Mapeamento n-1 - As dependências de mesmo tipo de diferentes componentes são mapeadas como apenas uma dependência do componente composto.

Assim como no mapeamento de serviços, o OpenCOM oferece apenas o mapeamento com aridade 1-1 para os receptáculos. De forma análoga ao mapeamento dos serviços, esta restrição é decorrente de um componente não possuir dois receptáculos de mesmo tipo.

Como cenário da Figura 2.6(b), podemos citar uma aplicação onde uma dependência de um subcomponente é representada como dependências diferentes do componente composto. Desta forma, é possível o usuário da aplicação adicionar comportamento diferente para cada dependência externalizada do componente composto.

Já para o cenário da Figura 2.6(c), podemos imaginar o processo de implantação de uma aplicação formada por um componente composto. Um componente composto pode encapsular diversos subcomponentes que necessitam das mesmas dependências. Neste exemplo, o componente composto pode funcionar como um injetor de dependências, recebendo a dependência necessária

PUC-Rio - Certificação Digital Nº 1012620/CA

pelos subcomponentes e, em seguida, repassando as dependências para todos os subcomponentes.

Para o SCS optamos por oferecer os três tipos de mapeamentos. Entendemos que o recurso de tornar possível interações mais complexas a partir de operações do componente composto trazem maior facilidade para o usuário da API desenvolver sua aplicação.

2.3.4 Conectores para Binding Vertical

Esta dimensão define se o modelo de componentes possui um elemento mediador responsável pelo fluxo de comunicação entre as interfaces do componente composto e dos subcomponentes (*binding* vertical). Lau et al. (2006) descreve este elemento mediador pela terminologia de conector exógeno. Através do uso de conectores exógenos, o controle da aplicação é desacoplado do componente composto e transferido para uma outra entidade. Um conector pode ser caracterizado como elemento de primeira ordem a medida que pode ser construído, instanciado e atribuído a uma variável.

Diversos padrões de fluxo de comunicação já existem, muitos deles análogos a padrões bem conhecidos do paradigma de orientação a objetos vistos em (Russell et al., 2006) e (Gamma et al., 1995). Nestes dois trabalhos são citados desde simples padrões como o *Proxy* até padrões mais complexos como *Adaptador*, *Facade*, *Decorator*, entre outros.

No padrão *Proxy*, o conector repassa as requisições do componente composto para os subcomponentes e, caso seja necessário, após o retorno das requisições dos subcomponentes, realiza uma operação de unificação sobre as respostas retornando como resultado o valor desta unificação ao componente composto. O padrão *Proxy* atende bem as necessidades encontradas em nosso exemplo de uso, o CAS. Basicamente, no CAS as interfaces do componente composto que representam serviços dos subcomponentes apenas repassam chamadas para os subcomponentes e aguardam se a requisição foi realizada com sucesso pelos subcomponentes.

Como exemplo de uso de outros padrões mais complexos, podemos citar o padrão *Adaptador*. O padrão *Adaptador* pode ser útil para manter a compatibilidade de versões entre as interfaces do componente composto e dos subcomponentes. Neste padrão, o conector converte a interface do componente composto em uma outra interface esperada pelo subcomponente. Através do padrão *Facade*, é possível o componente composto oferecer uma interface de tipo que é resultado de um conjunto de interfaces de diferentes tipos dos seus subcomponentes. Também, temos o padrão *Deferred Choice*

PUC-Rio - Certificação Digital Nº 1012620/CA

onde o conector realiza uma avaliação para quais subcomponentes irá repassar requisições do componente composto levando em consideração aspectos do ambiente operacional ou de acordo com um plano de execução definido pelo usuário.

O Fractal oferece suporte ao uso de conectores exógenos. Caso o usuário queira desenvolver o fluxo de comunicação entre o componente composto e os subcomponentes, deve criar um componente que oferece a interface *BindingController* (Vide Código 2.3) e, em seguida, implementar o fluxo de comunicação através das operações desta interface.

Em contraste, o OpenCOM apresenta a composição endógena. A composição endógena refere-se ao *binding* sem a presença de uma entidade para realizar a intermediação das interfaces do componente composto e dos subcomponentes. Neste tipo de *binding*, ao ser realizada uma requisição à uma interface do componente composto que representa um serviço de um subcomponente, a interface do componente composto funciona apenas como um *proxy* que repassa a requisição para o subcomponente.

Em um primeiro momento, optamos por oferecer o suporte nativo (composição endógena) para os três tipos de mapeamentos (1-1, 1-n e n-1). Entretanto, identificamos que esta decisão iria contra o nosso objetivo de oferecer uma API simples à medida que para o mapeamento 1-n teríamos que implementar a lógica de interação entre a interface do componente composto e o serviços dos subcomponentes, sendo que os tipos de interações são inúmeros como descritos anteriormente.

Para contornar o problema de sobrecarregar a implementação do nosso componente composto e oferecer uma API simples, optamos por oferecer o suporte ao uso de conectores exógenos para o mapemanto 1-n. Para os mapeamentos com aridade 1-1 e n-1 oferecemos a composição endógena (sem um mediador). A composição endógena será realizada através de operações do nosso modelo proposto. Para a composição exógena, 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 e, também, seja possível customizar conectores para sua própria aplicação.

2.3.5 Relacionamento Serviço/Componente

A navegação entre serviços é um instrumento de introspecção do paradigma de orientação a componentes. Através deste recurso é possível a partir de um serviço S implementado por um componente C obter os outros serviços deste componente.

PUC-Rio - Certificação Digital Nº 1012620/CA

No Fractal, a navegação entre serviços, é possível através da interface servidora *Interface* (Vide Código 2.6). Esta interface especifica quatro operações para retornar o nome da interface, o tipo da interface, o componente que a implementa e verifica se é uma interface interna ou externa. Por exemplo, se *a* é um referência para um serviço *Account* de um componente, o *Component* deste serviço pode ser acessado com o código *((Interface)a).getFcItfOwner()).* O resultado desta operação torna acessível qualquer outro serviço do componente.

```
1 package org.objectweb.fractal.api;
2
3 interface Interface {
4    string getFcItfName ();
5    Type getFcItfType ();
6    Component getFcItfOwner ();
7    boolean isFcInternalItf ();
8 }
```

Código 2.6: Interface para auxiliar a navegação entre facetas no Fractal.

O OpenCOM oferece a interface *IUnknown* (Vide Código 2.7) para suportar a navegação entre serviços. Através do método **QueryInterface** dada uma interface é possível retornar a referência do objeto que a implementa. Todo componente OpenCOM deve oferecer a interface *IUnknown* para permitir a navegação entre serviços.

```
1 package OpenCOM;
2
3 import java.lang.String.*;
4 import java.lang.reflect.*;
5
6 public interface IUnknown {
7
8     Object QueryInterface(String interfaceName);
9
10 }
```

Código 2.7: Interface para auxiliar a navegação entre facetas no OpenCOM.

Nos dois modelos a interface de um componente composto exportada de um subcomponente funciona como uma interface implementada pelo próprio componente composto. Para este comportamento ser possível no processo de *binding* vertical é necessário definir que a interface externalizada deve retornar o componente composto para navegação adicionando uma indireção no serviço oferecido pelo subcomponente.

PUC-Rio - Certificação Digital Nº 1012620/CA

O SCS com suporte a componentes compostos irá seguir a mesma abordagem dos dois modelos onde um serviço externalizado tem o mesmo comportamento de um serviço implementado pelo componente composto oferecendo, assim, a navegação entre serviços.

2.3.6 Compartilhamento de subcomponentes

Esta dimensão define se uma instância de um componente pode ser adicionada em mais de um componente composto. Esta característica é presente no modelo Fractal, que argumenta ser uma funcionalidade que preserva o encapsulamento de componentes e oferece economia de recursos. O OpenCOM não oferece esta funcionalidade.

A Figura 2.7 demonstra um exemplo de compartilhamento de subcomponentes. O cenário apresenta dois componentes compostos **Menu** e **Toolbar** que encapsulam o componente **Undo**. Podemos representar o componente **Undo** como um subcomponente compartilhado entre os componentes **Menu** e **Toolbar**. É esperado que o componente **Undo** apresente informações consistentes do seu estado para os componentes **Menu** e **Toolbar**. Por exemplo, o estado de sua propriedade **status** (*enabled*/*disabled*) deve ser o mesmo quando os componentes *Menu* e **Toolbar** consultarem o componente **Undo**.

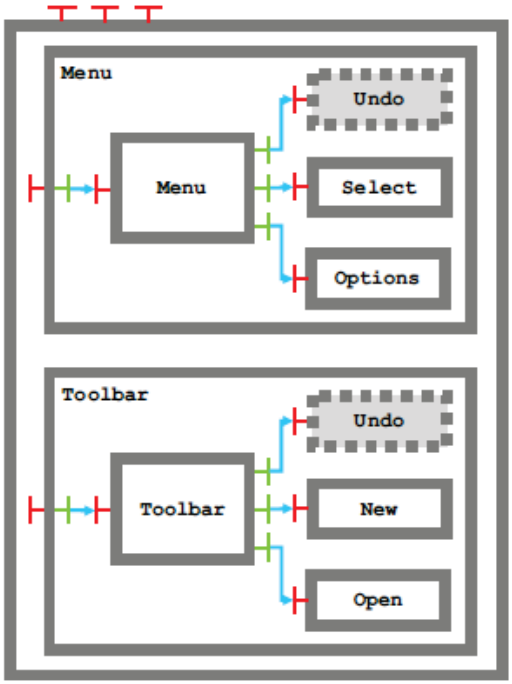


Figura 2.7: Arquitetura de uma interface gráfica com compartilhamento de subcomponentes.

O benefício de encapsulamento é uma boa prática de programação e

PUC-Rio - Certificação Digital Nº 1012620/CA

optamos por adicionar esta funcionalidade ao SCS com suporte a componentes compostos. Entretanto, esta funcionalidade pode nos levar a um problema de ambiguidade entre dependências de um subcomponente compartilhado que pode ser visualizado na Figura 2.8. O cenário da Figura 2.8 é composto por:

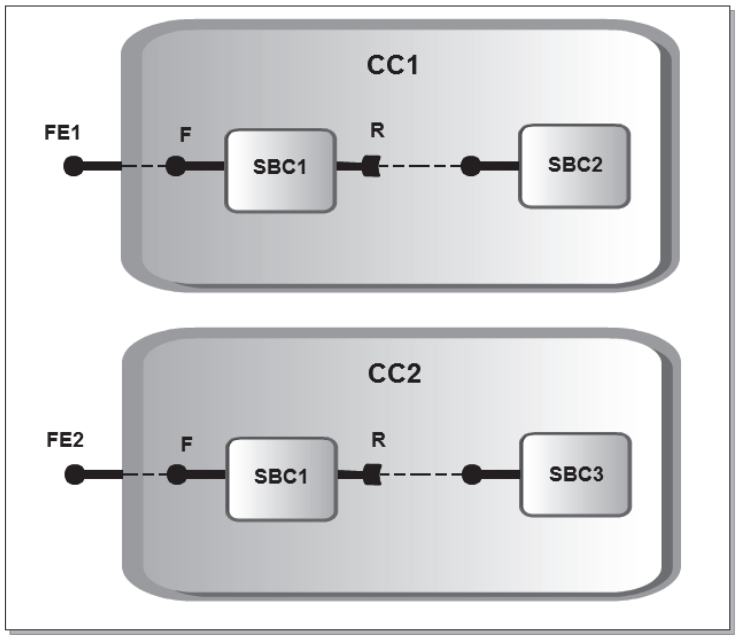


Figura 2.8: Cenário onde o compartilhamento de subcomponentes nos leva a uma situação de ambiguidade quanto as dependências do componente compartilhado.

- Dois componentes compostos: CC1 e CC2;
- Um subcomponente SBC1 compartilhado entre CC1 e CC2, que possui uma faceta do tipo F e um receptáculo do tipo R;
- Dois subcomponentes SBC2 e SBC3 encapsulados por CC1 e CC2, respectivamente. Ambos subcomponentes estão conectados ao receptáculo R do tipo R de SBC1;
- Bindings entre SCB1 - SBC2 e SCB1 - SBC3;
- Faceta F de SCB1 disponibilizada externamente por CC1 e CC2 através de FE1 e FE2.

Neste cenário, caso haja uma requisição sobre as facetas FE1 surge a questão: Qual dependência deve ser requisitada por SBC1? Em função deste tipo de ambiguidade decidimos utilizar o recurso de compartilhamento com a restrição de permitir esta funcionalidade apenas para componentes sem receptáculos.

PUC-Rio - Certificação Digital Nº 1012620/CA

2.3.7
Navegação entre Subcomponente e Componente Composto

Esta característica garante mecanismos de introspecção em uma aplicação. O Fractal oferece este recurso através das interface *SuperController* (Vide Código 2.2) e *ContentController* (Vide Código 2.1). A primeira permite a navegação no sentido "filho" =▷ "pai" e a segunda a navegação "pai" =▷ "filho". O OpenCOM só permite a navegação no sentindo "pai" =▷ "filho" através da interface *ICFMetaInterface* descrita na subseção 2.1.2.

Para o SCS, assim como no Fractal, optamos por oferecer a navegação nos dois sentidos. Essa funcionalidade é importante para auxiliar em mecanismos de introspecção. Por exemplo, podemos citar a possibilidade de levantar a descrição hierárquica de uma arquitetura em tempo de execução.

2.3.8
Mecanismos de Reconfiguração

Uma característica importante de sistemas baseados em componentes consiste em oferecer suporte a reconfiguração dinâmica de forma confiável. Os modelos Fractal e OpenCOM oferecem suporte a reconfiguração dinâmica de baixo nível tornando complexo programar reconfigurações e garantir confiabilidade.

Para o Fractal existem trabalhos como FPath e FScript (David et al., 2009) que têm como objetivo diminuir a complexidade de reconfiguração dinâmica do Fractal. FPath consiste em um DSL (*Domain Specific Language*) que oferece uma forma de navegar em aplicações modeladas pelo Fractal e realizar consultas em sua arquitetura. FSCript é uma linguagem de *script* que engloba FPath e torna possível a definição de reconfigurações complexas. FScript garante a confiabilidade destas reconfigurações com um poderoso controle em tempo de execução que provê semântica ACID (Atomicidade, Consistência, Isolamento e Durabilidade) para estas reconfigurações.

Como mecanismos de reconfiguração para o OpenCOM de mais alto nível do que a API identificamos o Plastik (Batista et al., 2005). O Plastik consiste da integração de uma ADL (uma extensão de ACME/Armani) com o OpenCOM. A partir do Plastik é possível realizar a especificação, criação e gerência de componentes em tempo de execução e garantir a integridade do sistema após reconfigurações.

Neste trabalho não entramos em maiores detalhes sobre mecanismos de reconfiguração para o nosso modelo proposto. Nós sugerimos que uma linha interessante a ser investigada é oferecer uma biblioteca auxiliar com rotinas de reconfiguração ou uma DSL assim o Fractal e OpenCOM. Um estudo mais

PUC-Rio - Certificação Digital Nº 1012620/CA

aprofundado sobre cenários de reconfiguração necessários em nosso exemplo de uso do Capítulo 5 pode auxiliar na identificação de algumas rotinas básicas. Depois de realizado este levantamento, um segundo passo seria a definição e implementação dessas rotinas com semântica ACID.

2.4 Considerações Finais

Este capítulo apresentou um levantamento sobre dois modelos consolidados que incorporam o conceito de componentes compostos: Fractal e OpenCOM. Este levantamento foi o primeiro passo para nosso estudo experimental de implantação desta funcionalidade em um modelo já existente, o SCS.

No *framework* de classificação para modelos de componentes desenvolvido por Crnkovic et al. (2010) não são descritos maiores detalhes sobre o conceito de componentes compostos. Procuramos preencher essa lacuna com o levantamento feito na seção anterior, onde enumeramos diversas características existentes no Fractal e OpenCOM. A Tabela 2.1 apresenta um resumo de como os modelos estudados implantam as características enumeradas e como esperasse que o SCS ofereça tais recursos.

O Fractal é um modelo que tem a semântica de componentes compostos em sua essência. Por este motivo oferece um poder de expressividade maior do que o OpenCOM para o usuário da API. Características como mapeamento de serviços e dependências internas através de interfaces diferentes do componente composto, tratamento de conectores como elementos de primeira ordem e relacionamento Componente/Subcomponente nas duas direções nos leva, em um primeiro momento, a seguir o Fractal como base para o nosso estudo experimental. Porém, não podemos descartar características importantes do OpenCOM como suporte a desenvolvimento incremental e composição endógena em relação ao mapeamento de dependências internas.

PUC-Rio - Certificação Digital Nº 1012620/CA

Tabela 2.1: Comparação entre os modelos Fractal e OpenCOM sobre características de componentes compostos em conjunto com as decisões realizadas para o modelo SCS.

Item	Suporte ao Desenvolvimento Incremental
Fractal	Não oferece suporte
OpenCOM	Oferece Suporte
SCS	Idem ao OpenCOM
Item	Mapeamento de Serviços dos Subcomponentes
Fractal	Serviços por Interfaces Diferentes
OpenCOM	Serviço Por uma Interface
SCS	Idem ao Fractal
Item	Mapeamento de Dependências
Fractal	1-1 / 1-n / n-1
OpenCOM	1-1
SCS	Idem ao Fractal
Item	Conectores Bindings Verticais
Fractal	Composição Exógena
OpenCOM	Composição Endógena
SCS	Exógena/Endógena
Item	Relacionamento Serviço/Componente
Fractal	Oferece Suporte
OpenCOM	Oferece Suporte
SCS	Oferece Suporte
Item	Compartilhamento de Subcomponentes
Fractal	Oferece Suporte
OpenCOM	Não oferece suporte
SCS	Oferece com restrições
Item	Navegação Subcomponentes/Componente Composto
Fractal	Pai-Filho / Filho-Pai
OpenCOM	Pai-Filho
SCS	Idem ao Fractal
Item	Mecanismos de Reconfiguração
Fractal	Nível API/DSL
OpenCOM	Nível API/DSL
SCS	Nível API

PUC-Rio - Certificação Digital Nº 1012620/CA