5
Exemplo de Uso

Neste capítulo é apresentado o nosso exemplo de uso: o CAS (*Capture and Access System*) (Portella et al., 2008). A modelagem do CAS utiliza o conceito de componente composto. Especificamente, com o CAS poderemos validar os recursos de gerência de subcomponentes, mapeamento de serviços de subcomponentes e uso de conectores exógenos. Outras características de componentes compostos como mapeamento de dependências não são exercitados. Como trabalhos futuros, vamos fazer um levantamento de aplicações distribuídas que necessitem deste recurso para poder validar fazer uma avaliação de todos os recursos do SCS-*Composite*. Na Seção 5.1 é apresentada a modelagem do CAS. Na Seção 5.2, entramos em detalhes sobre a implementação do CAS utilizando o *middleware* SCS que não oferece suporte a componentes compostos e mecanismos utilizados pelos desenvolvedores para simular um componente composto. Na Seção 5.3, é demonstrada a nova implementação utilizando o SCS-*Composite*; e, por fim, é feita uma análise do uso do SCS-*Composite* neste sistema.

5.1
Modelagem do CAS

O CAS é uma aplicação desenvolvida pelo laboratório TecGraf/PUC-Rio em parceria com a *Microsoft Research* e Petrobras que propõe-se a capturar mídias de uma experiência ao vivo (reuniões, seminários, aulas, ...) com o objetivo de disponibilizar, posteriormente, estas mídias na forma de um documento multimídia. Portella et al. (2008) descreve alguns cenários para uso do CAS como por exemplo salas de aula, salas de reunião e de propósito geral.

Especificamente, neste capítulo iremos ilustrar o cenário de captura em uma sala de aula. Neste contexto, temos a captura de diversos tipos de dispositivos (projetores, câmeras, microfones, lousas eletrônicas, ...) no momento das atividades conduzidas por um professor. Posteriormente, toda captura de mídia será disponibilizada em um documento multimídia onde o aluno pode revisar o material de forma estruturada e com todas as mídias sincronizadas.

A arquitetura do CAS considera como contexto para seu uso a captura de experiências ao vivo realizadas em um ambiente específico e bem delimitado. Esse ambiente é representado por um componente **Room** que encapsula diversos subcomponentes tornando possível o funcionamento do CAS. A Figura 5.1 apresenta

os tipos de componentes presentes:

- *Room*: agrega diferentes componentes. É responsável por disponibilizar as facetas dos subcomponentes encapsulados e implementa a interface *IRoom* (Código C.1) que oferece operações **set/get** sobre propriedades do ambiente.
- *SpeedCar* (*SpEcializED CApture driverRS*): funciona como um *driver* para dispositivos de captura de mídia e oferece suas funcionalidades através das seguintes interfaces:
  - *IRecord* : disponibiliza funcionalidades genéricas para um dispositivo que captura mídia. Possui métodos para iniciar e parar uma gravação e retornar o status do dispositivo (Código C.2).
  - *IDataTransfer*: define métodos para transferência de dados após término da gravação de uma mídia (Código C.3).
  - *IConfigurable*: disponibiliza métodos **set/get** das propriedades específicas do componente que a implementa (Código C.4).
- *Post-Processors*: desempenha tarefas de pós-processamento através da interface *IPostProcessor* (Código C.5).
- *RoomConfigurator*: desempenha um importante papel de configuração, adaptação, e manipulação sobre os componentes encapsulados por uma sala.
- *Control Panel*: oferece suporte para o gerenciamento do sistema. Possui uma interface gráfica para monitoramento e registro de eventos bem como facilidades de configuração.

O CAS utiliza o *OpenBus* (Tecgraf/PUC-Rio) para auxiliar na integração de seus componentes. O *OpenBus* é um *middleware* para integração de aplicações distribuídas baseadas em componentes. Basicamente, esta integração é feita através do uso de um barramento de comunicação, onde os componentes de uma aplicação realizam ofertas e requisições de serviços presentes no *OpenBus*. Por fim, existe um módulo chamado **Web Viewer** utilizado para acessar os documentos multimídias gerados pela aplicação.

5.2 Implementação com Middleware SCS

Esta seção descreve a implementação do CAS utilizando o *middleware* SCS. É importante destacar a diferença entre a modelagem (Figura 5.1) e implementação (Figura 5.2) do CAS ao utilizar o *middleware* SCS. Esta diferença existe em função do SCS em sua versão atual não possuir um poder de expressividade que abrange componentes compostos.

Nesta versão, o componente composto **Room** além de implementar sua faceta **IRoom** deve implementar as facetas dos **SpeedCars**: *IRecord*, *IDataTransfer* e *IConfigurable*. Estas facetas funcionam como um *proxy* que repassa requisições para

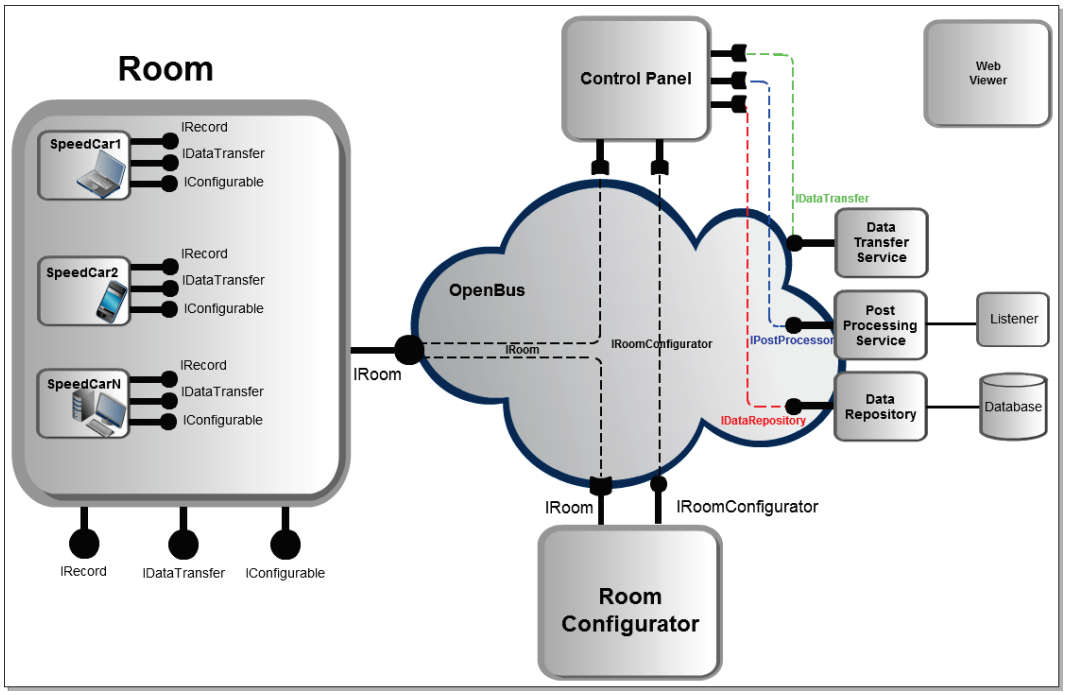

Figura 5.1: Modelagem realizada para arquitetura atual do CAS.

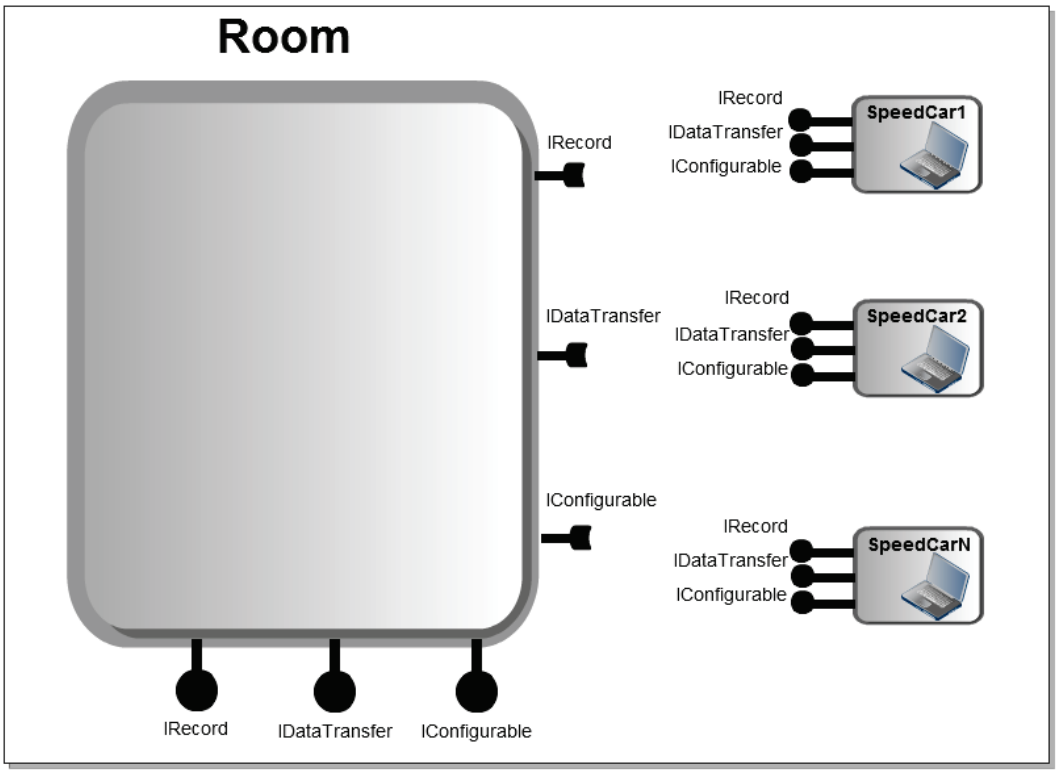

Figura 5.2: Implementação do CAS com o SCS sem suporte a componentes compostos.

os subcomponentes. Através desses *proxies* é feito o mapeamento de serviços dos subcomponentes através de um componente composto.

Basicamente, o *bootstrap* do CAS consiste na instanciação dos componentes **Room** e **RoomConfigurator** com um nome que os identificará no OpenBus, como por exemplo **SALA1**. Após registro no OpenBus de ambos componentes, um **SpeedCar** instanciado com o mesmo identificador **SALA1** tem acesso ao **RoomConfigurator** através da faceta *IRoomConfigurator* (Código C.6). O **SpeedCar** solicita a conexão ao ambiente através da operação **connectComponent**. A partir desta operação o **RoomConfigurator** verifica se o **SpeedCar** pode ser conectado ao componente **Room**. Caso a conexão tenha sido realizada com sucesso é retornado um identificador para o **SpeedCar** utilizado para removê-lo através da operação **disconnectComponent**.

O componente **Room** possui cinco estados: "not configured", "ready", "recording", "transferring", "postprocessing" e "finished". O seu estado inicial é "not configured". À medida que os **SpeedCars** são conectados o **RoomConfigurator** verifica se a configuração mínima foi atendida. Após a configuração necessária ser atingida o componente *Room* passa para o estado "ready". Desta forma, é possível iniciar a captura de determinado evento. Ao término de captura de um evento, os dados são transferidos para um diretório definido onde é realizado um pós-processamento e sincronização dos dados. Ao final de todo procedimento é disponibilizado pelo *WebViewer* o evento registrado pelo CAS.

O **RoomConfigurator** realiza a gerência sobre os subcomponentes mantendo estruturas que informam qual o estado da aplicação. Por exemplo, após adição de um **SpeedCar**, o **RoomConfigurator** garante que o componente **Room** conhece todos os **SpeedCars** encapsulados e vice-versa, bem como mantem estruturas responsáveis pelo mapeamento de serviços internos para fora do componente **Room**. Tais tarefas foram implementadas pelo desenvolvedor da aplicação. Esta sobrecarga, por conta da falta de expressividade do SCS, aumentou os esforços da equipe de desenvolvimento do CAS na criação de uma série de políticas para simular o componente **Room** como um componente composto enquanto poderia estar se dedicando em tarefas da lógica de negócio.

Como mecanismo para simular o comportamento de um componente composto podemos citar o mapeamento de serviços dos subcomponentes (Figura 5.2). As facetas do tipo *IRecord* dos subcomponentes **SpeedCar1**, **SpeedCar2** e **SpeedCarN** são conectadas ao receptáculo de tipo *IRecord* do componente **Room** e, a partir da faceta *IRecord* de **Room** é possível acessar as facetas *IRecord* dos subcomponentes. Para cada novo mapeamento necessário na aplicação o componente **Room** deve criar um receptáculo e uma faceta. Esta regra implica em maior trabalho na manutenção do CAS já que, como visto na Seção 2.3.2 sobre mapeamentos de serviços internos, muitas vezes é necessário realizar o mapeamento interno por diferentes interfaces do componente composto.

Um cenário que ilustra a necessidade recorrente de um novo mapeamento

de serviço, é a captura de uma apresentação realizada em uma sala através de quatro câmeras: duas do lado esquerdo (C1 e C2) e duas do lado direito (C3 e C4) (Figura 2.5). Nesta aplicação muitas vezes é necessário realizar operações sobre as quatro câmeras (iniciar, pausar, parar, verificar status,...) de uma só vez. Entretanto, em algumas situações será necessário realizar operações específicas sobre câmeras levando em consideração sua localização. A implementação desta demanda, com a versão atual do CAS, exige a adição de duas facetas do tipo *IRecord* (*LeftRecord* e *RightRecord*). Esta exigência torna o trabalho de manutenção do CAS bastante burocrático e pouco modular, pois o componente **Room** deverá ser recompilado cada vez que for necessário um novo mapeamento de serviços internos.

5.3 Implementação com Middleware SCS-Composite

A utilização do SCS-*Composite* no CAS tem como objetivo concentrar os esforços de implementação da equipe de desenvolvimento apenas com tarefas de lógica da aplicação. Com esta nova versão o componente **Room** é um componente composto e oferece introspecção e configuração sobre os componentes encapsulados. As próximas subseções descrevem aspectos como gerência sobre componentes, mapeamento de serviços internos e visualização da aplicação que atendem as necessidades da modelagem do CAS.

5.3.1 Gerência sobre Componentes

O controle sobre a adição e remoção de subcomponentes é realizada através das operações **addSubComponent** e **removeSubComponent**. Depois de adicionados, esses componentes podem ser acessados através da operação **getSubComponents**. Basicamente, o desenvolvedor CAS não precisa mais manter uma estrutura com a informação de quais subcomponentes são encapsulados por determinado componente **Room**.

Na versão atual, o desenvolvedor CAS precisa se preocupar em manter sempre consistente o estado de um ambiente informando quais subcomponentes estão encapsulados. Com o CAS composto, a operação **addSubComponent** garante que todos componentes mantenham esta informação atualizada. Desta forma, após adicionar um componente **SpeedCar** de áudio em um ambiente **Room** através da operação **getSuperComponent** da faceta *ISuperComponent* do **SpeedCar** é possível acessar o componente **Room** que o encapsula. Da mesma forma, o componente **Room** através da operação **findComponent** ou **getSubComponents** tem acesso ao componente **SpeedCar**.

A subseção anterior mostrou como é realizado o mapeamento de serviços internos através de interfaces externas do componente composto. A forma como se dá o mapeamento torna mais custosa a remoção de um componente com serviços

disponibilizados através do componente composto. A remoção de um subcomponente implica na verificação se o mesmo está conectado em alguma faceta ou receptáculo do componente **Room**. Caso existam conexões, o desenvolvedor CAS deve tomar cuidado em realizar todas desconexões para manter a aplicação em um estado consistente. Mais uma vez, nota-se a necessidade de manter uma estrutura que tenha uma associação de todos os subcomponentes e mapeamentos. Através da operação **removeSubComponent** o desenvolvedor CAS não precisa mais de se preocupar com esta tarefa.

5.3.2 Mapeamento de Serviços Internos

O mapeamento de facetas e receptáculos dos subcomponentes através do componente composto é realizado através das operações **bindFacet**, **unbindFacet**, **bindReceptacle** e **unbindReceptacle**. Desta forma, é possível realizar o mapeamento de serviços internos sem precisar de seguir a regra de que para cada novo mapeamento deve ser criado uma faceta e receptáculo novos para o componente **Room** (Figura 5.2).

Na versão do CAS que usa o SCS original, após o mapeamento de serviços internos, era necessário manter uma estrutura informando todas as facetas e receptáculos envolvidos para o momento de remoção do mapeamento ou do subcomponente. Na nova versão do CAS, não é preciso manter esta estrutura. O desenvolvedor CAS utiliza a operação **bindFacet** para realizar o mapeamento automático de facetas, que retorna um identificador. Este identificador é utilizado para remover o mapeamento através da operação **unbindFacet**. A operação **bindReceptacle** e **unbindReceptacle** funcionam de forma semelhante para o mapeamento de receptáculos.

5.3.3 Binding Vertical com Aridade 1-n

No CAS, é observado a presença do *binding* vertical com aridade 1-n. A Figura 5.3 ilustra um exemplo deste cenário composto por três componentes do tipo **SpeedCar**: **cmpAudio**, **cmpVideo** e **cmpPPT** que desejam externalizar, respectivamente, as facetas **fctIRecord** do tipo *IRecord* através da faceta **fctIRecordRoom** do componente **Room**. Para este tipo de mapeamento o nosso modelo proposto indica o uso de conectores conforme Seção 3.3.2.

Basicamente, para as facetas **fctIRecord** dos subcomponentes serem externalizadas através da faceta **fctIRecordRoom** do componente **Room** a equipe CAS deve implementar um componente do tipo conector (**cmpConnector**). Este conector deve possuir uma faceta e receptáculo múltiplo do tipo *IRecord* que irão intermediar o fluxo de comunicação entre a faceta **fctIRecordRoom** e as facetas dos subcomponentes.

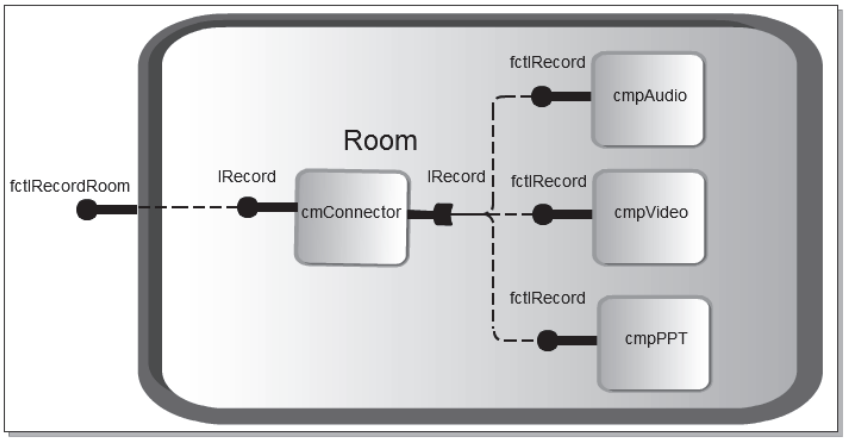

Figura 5.3: Exemplo do uso de conectores no CAS.

A chamada a uma operação da faceta externalizada **fctIRecordRoom** repassa a chamada ao componente **cmpConnector**. A partir da faceta do conector é possível acessar o seu receptáculo, e, consequentemente, as facetas dos subcomponentes. Encontramos um aspecto negativo nesta abordagem quanto à introdução de uma indireção - uma nova chamada feita do conector para o subcomponente. Embora, tenha sido introduzido mais uma indireção para aplicação do CAS este fato é amenizado em função do baixo grau de aninhamento de componentes.

Ademais, adicionamos um novo conceito para o desenvolvedor que tem que se preocupar com a implementação do conector responsável pelo fluxo de comunicação entre o componente composto e subcomponentes. Uma linha interessante para trabalhos futuros seria criar fábricas geradoras de conectores. Estas fábricas poderiam gerar conectores com fluxo de comunicação otimizados.

Por fim, antes toda a responsabilidade de implementação do fluxo de comunicação se encontrava no próprio componente **Room**, assim, qualquer modificação sobre o mapeamento de serviços internos de um componente composto implicava na recompilação de todo o componente **Room**. Agora, com a modularização e código-fonte mais estruturado, é necessário recompilar apenas os conectores.

5.3.4 Gerência de Configuração da Aplicação

Os mecanismos de inspeção oferecidos através das operações **findComponent** e **getSubcomponents** da faceta *IContentController* facilitam a implementação de mecanismo que verifiquem a configuração mínima para execução do CAS em um ambiente.

Estes mecanismos de inspeção, também, são importantes para futuras implementações onde espera-se ter um suporte à reconfiguração dinâmica com semântica ACID. Para início de uma transação de reconfiguração será realizada uma inspeção sobre o estado da aplicação para salvar o último estado consistente antes de serem iniciadas as reconfigurações. Desta forma, caso haja alguma falha na reconfiguração

a aplicação consegue se recuperar.

Também, a partir da faceta *IContentController* é possível acessar a faceta *IComponent* e assim tratar um componente composto como um simples componente primitivo. Como visto no Capítulo 3, a idéia de componentes compostos poderem ser vistos como um componente primitivo é reconhecida como uma boa prática para sistemas baseados em componentes em função de abstrair estruturas complexas e aumentar o reuso.

5.4 Comparação do CAS: SCS Vs. SCS-Composite

Neste capítulo foi apresentada uma implementação do CAS utilizando o SCS-*Composite*. Originalmente, o CAS foi implementado utilizando o SCS sem suporte a componentes compostos. Entretanto, sua modelagem utilizou o conceito de componentes compostos. Para preencher a lacuna de falta de expressividade do conceito de componentes compostos do SCS o desenvolvedor do CAS tem que se preocupar com detalhes além da lógica da aplicação.

Como aspecto positivo, podemos destacar que o nosso modelo proposto atende aos requisitos do CAS, como gerência sobre subcomponentes, mapeamento de serviços de subcomponentes e formas diferentes de inspeção da configuração de uma aplicação. Este suporte torna possível o desenvolvedor CAS se concentrar apenas em tarefas da lógica da aplicação.

Quanto à gerência sobre componentes, o *middleware* mantém consistente a informação de quais subcomponentes pertencem a determinado componente composto e qual componente composto encapsula determinado subcomponente. A partir desta gerência sobre os componentes, o administrador do CAS possui em tempo de execução uma visão arquitetural da aplicação através de mecanismos de introspecção.

A configuração sobre mapeamento de serviços dos subcomponentes também é feita de maneira bastante direta através do uso das operações **bindFacet** e **unbindFacet** da interface *IContentController*. Como visto na Seção 5.1, na versão original do CAS um novo mapeamento de um serviço de um subcomponente exige a recompilação do componente **Room**. Esta exigência torna o trabalho de manutenção do CAS bastante burocrático e pouco modular. O mapeamento oferecido pelo nosso *middleware* pode ser realizado em tempo de execução tornando o desenvolvimento e manutenção do CAS mais ágil e modularizado.

Também é importante ressaltar que a implementação de mecanismos no CAS para preencher a falta de expressividade do SCS em relação ao conceito de componentes compostos gera um maior esforço por parte dos desenvolvedores. Por exemplo, o processo de mapeamento de uma faceta *IRecord* (Código C.2) de um subcomponente através de um componente composto **Room** seguia três passos. Primeiro, temos a criação de uma faceta e receptáculo de mesmo tipo da faceta que será externalizada no componente **Room** (Código 5.1).

```
local facetDescs = {}
facetDescs.IRecord = {
  name = "recorder",
  interface_name = "IDL:cas/recorder/IRecord:1.0",
  class = Record
}

local receptDescs = {}
receptDescs.IRecordReceptacle  = {
  name = "IRecordReceptacle",
  interface_name = "IDL:cas/recorder/IRecord:1.0",
  is_multiplex = true,
  type = "ListReceptacle"
}
```

Código 5.1: Definição da faceta no componente composto que representará a faceta do subcomponente.

Em seguida, deve ser realizada a implementação de uma faceta no componente **Room** com o mesmo tipo da faceta do subcomponente (*IRecord*). A partir da faceta **recorder** criada é possível acessar o receptáculo **IRecordReceptacle**. Este receptáculo irá realizar conexões com as facetas que serão externalizadas dos subcomponentes (Código 5.2).

```
local recepRoom = room:getFacetByName("IReceptacles")
recepRoom = orb:narrow(recepRoom,"IDL:scs/core/IReceptacles:1.0")

fctSub = subCmp:getFacetByName("recorder")
recepRoom:connect("IRecordReceptacle", fctSub)
```

Código 5.2: Conexão da faceta do subcomponente com o receptáculo do componente composto.

Por fim, o último passo consiste na implementação de um objeto repassador de requisições para os subcomponentes (Código 5.3). Este objeto deve implementar uma função repassadora para cada método da interface *IRecord*.

```
local Record = oo.class{}

function Record:startRecord()

  local context = self.context
  local recp = context.IRecordReceptacle

  for k,v in pairs(recep) do
    v:startRecord()
  end

```

```
12 end
```

Código 5.3: Faceta *IRecord* do componente composto que torna acessível a faceta do subcomponente.

Ao término deste procedimento é possível a partir do componente composto acessar a faceta do subcomponente. No SCS-*Composite* este esforço é minimizado, pois a externalização de uma faceta é realizada através da operação **bindFacet** da interface *IContentController* como pode ser visto na Seção 4.3.1 (Código 4.5).

No CAS, é observado a presença do *binding* vertical com aridade 1-n. Para este tipo de mapeamento o nosso modelo proposto indica o uso de um tipo especial de componentes que oferece a mesma faceta externalizada de um subcomponente e implementa o fluxo de comunicação entre o componente composto e os subcomponentes. Como aspecto negativo temos a introdução de mais uma indireção. Entretanto, no CAS entendemos que essa sobrecarga de desempenho não é tão custosa, dado o pequeno grau de aninhamento de componentes compostos. Como trabalho futuro podemos seguir a linha de criar fábricas geradoras de conectores com fluxo de comunicação otimizada.

Por fim, o *middleware* SCS-*Composite* atendeu aos requisitos de componentes compostos do CAS. Entretanto, seria interessante a implementação em outras aplicações para investigar mecanismos que não foram testados pelo CAS como mapeamento de dependências de um subcomponente. Também é interessante analisar se em outras aplicações existem componentes compostos com alto grau de aninhamento. Caso seja constatado que sim, é necessário criar mecanismos de otimização para os *proxies* criados no componente composto após externalização de um serviço ou dependência.