6
Conclusão

Neste trabalho foi proposto adicionar à semântica do modelo SCS o conceito de componentes compostos. Crnkovic et al. (2010) cita diversos modelos de componentes que possuem em sua semântica o conceito de modelos de componentes compostos, entretanto, não existe uma definição bem aceita sobre várias aspectos como, o mapeamento de serviços e dependências dos subcomponentes, navegação entre subcomponentes e componentes compostos, compartilhamento de subcomponentes e mecanismos de reconfiguração.

Para preencher esta lacuna foi realizado um levantamento sobre dois modelos de componentes bem estabelecidos: Fractal e OpenCOM. Estes dois modelos foram selecionados por serem modelos de componentes de propósito geral da mesma forma que o SCS e pela disponibilidade tanto de documentação quanto de código fonte das implementações.

Este levantamento foi o primeiro passo para concepção do modelo proposto, SCS-*Composite*. Em seguida foi realizado um estudo aprofundado e a especificação do modelo SCS. O levantamento sobre Fractal e OpenCOM em conjunto com o estudo sobre o SCS serviram de base para gerar o modelo SCS-*Composite* com suporte a componentes compostos.

Um componente composto é formado por uma membrana que encapsula uma composição. Esta membrana oferece mecanismos de configuração e introspecção sobre os subcomponentes da composição, bem como o mapeamento de serviços e dependências através da faceta *IContentController*. Ademais, um componente composto deve garantir as propriedades de um componente primitivo, como por exemplo, a unicidade dos indentificadores de suas facetas e receptáculos.

Tanto um componente primitivo quanto um componente composto possuem uma faceta obrigatória: *ISuperComponent*. Através desta faceta um subcomponente pode acessar os componentes compostos que o encapsulam. Esta navegação é importante para auxiliar na aplicação das restrições de conexões entre dois componentes e no teste de verificação se um componente pode ser compartilhado, conforme apresentado na Seção 3.3.5. Ele também é um importante mecanismo de introspecção para o SCS-*Composite*.

Foram adicionadas duas regras para validação do *binding* horizontal. Estas regras estabelecem 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 descrito em maiores detalhes na Seção 3.3.1. E, principalmente, permite que o encapsulamento proposto pela abstração de componentes compostos seja garantido.

As alterações na estrutura de um componente e mecanismos de *binding* representam mudanças na base do SCS. Embora nosso objetivo inicial era não realizar este tipo de modificação intrusiva, elas foram necessárias para garantir características de componentes compostos como suporte ao desenvolvimento incremental, relacionamento subcomponente/componente composto e compartilhamento de componentes.

A partir das diretrizes do modelo proposto, nota-se uma forte influência do Fractal em aspectos como mapeamento de serviços e dependências, composição exógena utilizando conectores, navegação do subcomponente para o componente compostos e compartilhamento de subcomponentes. Entretanto, ele também possui características oriundas do OpenCOM para oferecer uma API flexível tornando possível o suporte ao desenvolvimento incremental e oferecendo operações diferentes para o *binding* vertical e horizontal.

Basicamente, o modelo SCS-*Composite* se diferencia do SCS em três aspectos: estrutura do componente primitivo, adição de um novo tipo de componente e mecanismo de *binding*. As modificações na estrutura de um componente primitivo e o novo tipo de componente composto implicaram em alterações na fábrica de componentes.

Foram desenvolvidas duas implementações do modelo SCS-*Composite*: a primeira, baseada na versão 2.0 do SCS para componentes locais e, a segunda, baseada na versão 1.0 para componentes distribuídos em Lua. A introdução do conceito de facetas e receptáculos externalizados implicou na adição de indireções. Estas novas indireções podem representar sobrecarga de desempenho, entretanto, neste trabalho não focamos em como otimizar tais indireções. O nosso objetivo principal foi validar se a API do SCS-*Composite* atendia aos requisitos de um aplicação que utiliza o conceito de componentes compostos.

Também foi apresentado um exemplo de uso aplicando o modelo SCS-*Composite* sobre o CAS. Esta aplicação foi modelada utilizando o SCS sem suporte a componentes compostos. Porém, na essência de sua modelagem existe o conceito de componentes compostos. Em sua versão atual, foi constatado que o desenvolvedor da aplicação tinha trabalho redobrado por ter que implementar mecanismos para simular componentes compostos. Em nosso exemplo de uso, nota-se que o *middleware* SCS-*Composite* atendeu aos requisitos de uma aplicação modelada utilizando o conceito de componentes compostos. E, possibilitou o desenvolvedor do CAS se concentrar apenas em tarefas da lógica da aplicação.

Como trabalhos futuros temos várias frentes de investigação. Dentre estas frentes podemos citar a implementação do SCS-*Composite* para todas as implementações do SCS (versão local em C# e C++ e distribuída em C#, Java e C++).

Em conjunto com estas novas versões é interessante, também, levar este modelo para o ASCS (*Annotated Software Components System*) (Saldanha, 2010) e adicionar novas anotações para suportar o conceito de componentes compostos. Ainda, na linha de facilitar o desenvolvimento de aplicações poderíamos implementar geradores de *stubs* para os conectores de *bindings* verticais.

A API do SCS-*Composite* foi desenvolvida para possuir mecanismos de introspecção de uma aplicação. Um caminho a seguir seria a criação de uma linguagem de descrição de arquitetura que utilizasse estes mecanismos, assim como a linguagem FPath que oferece uma forma de navegar e realizar consultas em arquiteturas modeladas utilizando o Fractal. A criação desta linguagem auxiliaria em mecanismos de reconfiguração dinâmica com semântica ACID sobre componentes compostos.

Quanto a validação dos recursos do SCS-*Composite*, foram atendidos os requisitos da modelagem do CAS. Entretanto, o CAS não contempla todos os recursos do SCS-*Composite*, desta forma, seria interessante a realização de validação em outras aplicações para investigar mecanismos que não foram testados, como por exemplo, o mapeamento de dependências de um subcomponente. Também seria interessante verificar o grau de aninhamento de componentes compostos em outras aplicações. Caso seja constatado que existem aplicações com alto grau de aninhamento, é necessário criar mecanismos de otimização para os *proxies* criados no componente composto após externalização de um serviço ou dependência.

Por fim, um conceito importante introduzido foi o mecanismo de *proxies* criados por um receptáculo externo. Este mecanismo é necessário para um subcomponente aceitar a conexão de uma faceta implementada por um componente externo à sua composição. Entretanto, estes *proxies* podem gerar inconsistências na inspeção de uma arquitetura. É necessário um estudo mais aprofundado sobre como lidar com as incoerências introduzidas pelo mecanismo de *proxy* oferecendo formas de introspecção mais precisas para o usuário de nossa API.