1
Introdução

Engenharia de *Software* baseada em Componentes (ESBC) é uma área bem estabelecida em Engenharia de *Software*. Entre as técnicas e tecnologias que formam sua base podemos destacar o paradigma de orientação a objetos, arquitetura de *software*, linguagens de descrição de arquitetura (ADL), *middleware*, e outras abordagens mais tradicionais como desenvolvimento modular e estrutural.

De acordo com Meyer (2000) desde seu início tem sido difícil obter uma definição precisa e comum para o conceito de *componentes de software*. Uma definição bem aceita atualmente é dada por Szyperski (2002):

> Um componente de *software* é uma *unidade de composição* com, ao menos, *interfaces* contratualmente especificadas e *dependências de contexto* explícitas. Um componente de *software* pode ser implantado (do inglês *deployed*) independentemente e está sujeito a composição por terceiros.

Crnkovic et al. (2010) realizou um estudo que tem como importante contribuição um *framework* de classificação de modelos de componentes. Neste *framework* o cenário de um sistema baseado em componentes é constituído por uma plataforma de execução, componentes e *bindings* horizontais. A plataforma de execução diz respeito ao sistema operacional e recursos computacionais disponíveis.

Um componente é criado para interagir com outros componentes de acordo com regras pré-definidas. Em outras palavras, um componente é um módulo de *software* formado por um código de execução e metadados legíveis por uma máquina (tipicamente incluindo uma assinatura de interface) que explicitamente descrevem os serviços que o componente oferece e os serviços necessários de outros componentes e da plataforma de execução.

Um componente é especificado por um conjunto de propriedades. Propriedades, aqui, são utilizadas levando em conta o seu significado definido por dicionários padrões: uma construção em que indivíduos podem ser distinguidos. Existem diferentes classificações de propriedades. Uma classificação usada ge-

ralmente é distinguir entre propriedades funcionais e extra-funcionais. Propriedades funcionais dizem respeito a serviços e dependências de um componente. Por outro lado, propriedades extra-funcionais dizem respeito a características como atributos de qualidade de um serviço (tempo de resposta, intervalo de confiança, nível de segurança, entre outros).

Os *bindings* horizontais definem conexões através de metadados dos componentes. A Figura 1.1 ilustra os dois tipos de *bindings* horizontais. Em 1 é ilustrado o *binding* entre componentes (que habilita a interação entre os componentes) e, em 2, o *binding* entre os componentes e a plataforma operacional (que torna possível a implantação de componentes em um sistema). Dois componentes C1 e C2 conectados por suas interfaces representam um *assembly* ou uma *composição*. Diversos modelos possuem poder de expressividade para definir a partir destas *composições* novos componentes.

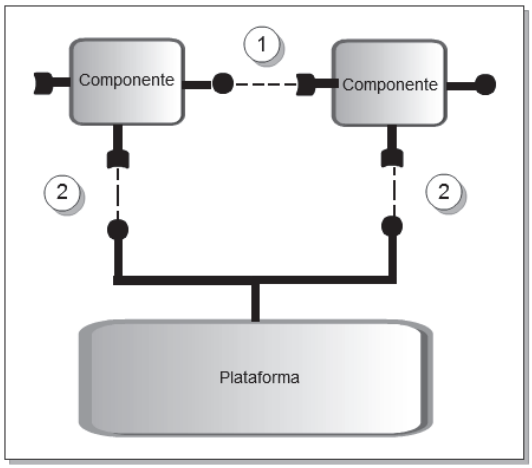


Figura 1.1: Sistema baseado em componentes. Em 1 é ilustrado o *binding* entre dois componentes; e em 2, é ilustrado *binding* entre um componente e a plataforma operacional.

Levando-se em consideração que novos componentes podem ser definidos a partir de uma composição, os componentes podem ser classificados em dois tipos: primitivos e compostos (Vide Figura 1.2). Nos modelos de componentes atuais baseado no paradigma de orientação a objetos como Fractal (Bruneton et al., 2006) e OpenCOM (Coulson et al., 2008), um componente primitivo é representado como um objeto e seus serviços e dependências como interfaces. O componente composto é o resultado de uma *composição* encapsulada por uma membrana. Esta membrana é uma camada que oferece serviços de introspecção e reconfiguração sobre seus subcomponentes.

O componente composto possibilita duas formas de visões. A primeira é uma visão simples onde os subcomponentes encapsulados são abstraídos. Desta forma, o componente passa a ser visto externamente apenas como um

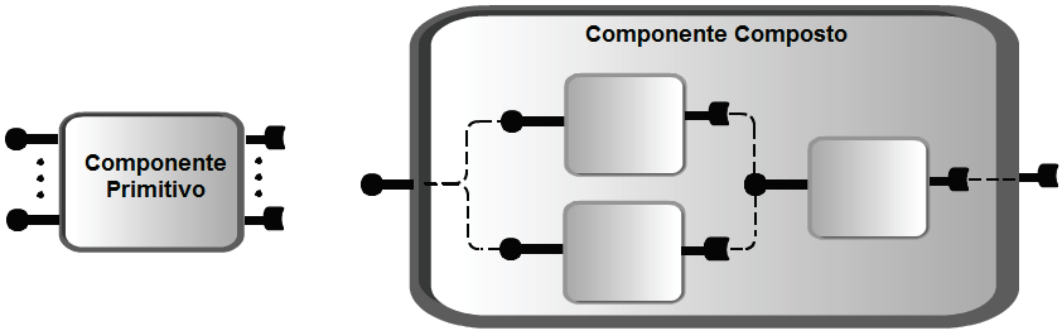
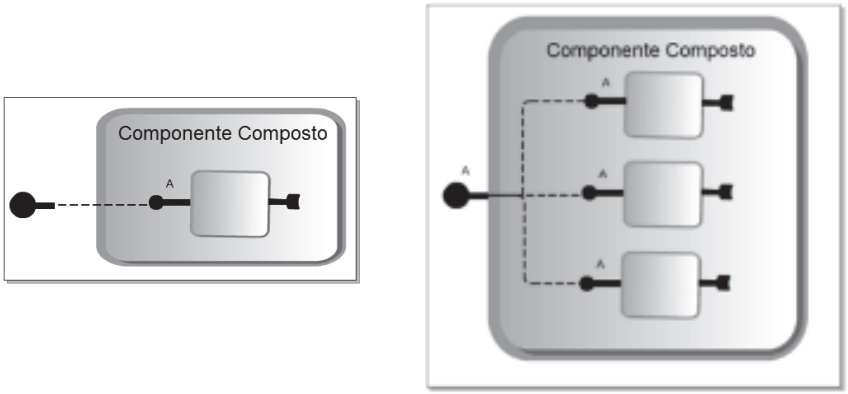


Figura 1.2: Componente primitivo e composto.

componente primitivo. A segunda visão é mais complexa, pois explicita os subcomponentes e um conjunto de operações de configuração e introspecção. É importante destacar que um componente composto pode encapsular outros componentes compostos.

Os componentes compostos apresentam um novo tipo de *binding* conhecido como vertical. Este tipo de *binding* diz respeito à conexão das interfaces dos subcomponentes encapsulados com as interfaces do componente composto (Vide Figura 1.3). Através deste *binding* os subcomponentes tornam acessíveis seus serviços e dependências fora do componente composto.

Figura 1.3: Representação dos dois tipos de *bindings* verticais



1.3(a): Um subcomponente externaliza uma interface A.

1.3(b): Mais de um subcomponente externaliza uma interface de mesmo tipo através de uma única interface da membrana.

Os *bindings* verticais são classificados de acordo com sua aridade (Vide Figura 1.3). A Figura 1.3(a) ilustra um *binding* vertical do tipo 1-1, onde apenas um subcomponente torna acessível sua interface A para fora do componente composto. Na Figura 1.3(b) temos o *binding* vertical do tipo

1-n, onde dois ou mais subcomponentes tornam acessíveis suas interfaces do tipo A através de uma única interface do componente composto.

Segundo Lau et al. (2007), a idéia de componentes compostos é reconhecida como uma boa prática para sistemas baseados em componentes em função de encapsular estruturas complexas. Este encapsulamenteo promove o reuso dessas estruturas como um único componente. Esta prática é adotada por diversos modelos de componentes como Fractal (Bruneton et al., 2006), OpenCOM (Coulson et al., 2008), Koala (van Ommering et al., 2000), Kobra (Atkinson et al., 2001) e SaveCCM (Hansson et al., 2004).

Embora seja uma característica presente em modelos bem estabelecidos, ainda não existe uma definição bem aceita em vários aspectos do conceito de componentes compostos, tais como o mapeamento de serviços e dependências de subcomponentes através de interfaces de um componente composto, compartilhamento de subcomponentes e mecanismos de introspecção e reconfiguração.

Por exemplo, para o mapeamento de serviços de subcomponentes como serviços de um componente composto (*binding* vertical), tanto o SaveCCM quanto Koala estabelecem alguns padrões de interações. O Fractal deixa possível o usuário da API implementar seus próprios padrões de comunicação em sua aplicação. Em contrapartida, o OpenCOM possibilita apenas o *binding* vertical do tipo 1-1.

Quanto ao quesito de navegação entre subcomponentes e componentes compostos, o Fractal oferece a navegação no sentido "filho" $\Rightarrow\triangleright$ "pai" e "pai" $\Rightarrow\triangleright$ "filho". Já o OpenCOM só permite a navegação no sentido "pai" $\Rightarrow\triangleright$ "filho". Por fim, alguns modelos como o Fractal oferecem o compartilhamento de subcomponentes. Como argumento o Fractal defende que o compartilhamento é uma funcionalidade útil por preservar o encapsulamento de componentes e economizar recursos. Entretanto, o OpenCOM não oferece este recurso e justifica que pode adicionar inconsistências em uma aplicação.

1.1 Objetivos e Contribuições

Para preencher a falta de uma definição bem aceita entre os modelos que incorporam o conceito de componentes compostos este trabalho tem como objetivo principal um estudo experimental sobre componentes compostos através da extensão de um modelo pré-existente: o modelo SCS (Augusto et al., 2009).

O primeiro passo do nosso estudo experimental é o levantamento sobre o modelos Fractal e OpenCOM, que possuem o conceito de componentes

compostos em sua semântica. Com este levantamento, esperamos enumerar as principais características de componentes compostos. O entendimento de como cada modelo incorpora tais características será importante para apoiar decisões em nosso estudo experimental.

Em seguida, será realizado um estudo e a especificação do modelo SCS. Este estudo aprofundado sobre o SCS, é importante para avaliarmos a melhor forma de adicionar o suporte a componentes compostos de maneira flexível e pouco intrusiva.

E, por fim, será feita a especificação e implementação do modelo SCS-*Composite* (SCS com componentes compostos). A partir da implementação do nosso modelo proposto, poderemos validar nossa extensão em uma aplicação: o CAS (*Capture and Access System*) (Portella et al., 2008). Esperamos com este trabalho realizar duas contribuições, a saber:

1. Um levantamento comparativo sobre como os modelos Fractal e OpenCOM incorporam o conceito de componentes compostos.

2. A formalização do modelo SCS atual e de sua extensão com suporte a componentes compostos.

1.2 Estrutura do Documento

Este trabalho está organizado em seis capítulos. O Capítulo 2 descreve o estudo e comparação de trabalhos relacionados. Especificamente, vamos focar nos modelos Fractal e OpenCOM. Dois motivos nos levaram a escolha destes modelos para nosso estudo:

– Ambos os modelos são de propósito geral, assim como o SCS. Estes modelos oferecem mecanismos básicos de especificação e composição de componentes através de suposições sobre formas de interação, suporte a sistemas distribuídos, compilação ou implantação em tempo de execução.

– Documentação e acesso ao código fonte das implementações dos modelos que nos permite realizar exemplos práticos.

No Capítulo 3 apresentamos a extensão do modelo SCS proposta para oferecer suporte a componentes compostos. Primeiro, entraremos em detalhes sobre o modelo SCS que, atualmente, possui suporte a apenas componentes primitivos. Em seguida, será descrito o nosso modelo com suporte a componentes compostos: *SCS-Composite*. No capítulo 4, será demonstrada a API sugerida com base no estudo do Capítulo 3 e a sua implementação para componentes locais e distribuídos com exemplos básicos.

No Capítulo 5 é apresentado o nosso exemplo de uso: o CAS (*Capture and Access Service*). Esta aplicação foi desenvolvida utilizando o modelo SCS com suporte a apenas componentes primitivos. Será apresentada a implementação da nova versão do CAS utilizando o SCS-*Composite* e uma comparação entre as duas arquiteturas avaliando benefícios e dificuldades em modelar a arquitetura de uma aplicação utilizando um modelo de componentes com suporte a componentes compostos. Por fim, no Capítulo 6, são apresentadas as conclusões e possíveis trabalhos futuros.