1
Introdução

Resumo

Esta introdução descreve a motivação para o desenvolvimento de uma arquitetura para auxiliar o desenvolvimento de soluções baseadas em componentes de software e apresenta uma breve visão geral da abordagem proposta. Ainda neste capítulo, são apresentados alguns trabalhos relacionados. Finalmente, é elaborado um guia do leitor contendo a estrutura deste documento.

Freqüentemente, ao desenvolver novas aplicações é possível reutilizar componentes de software. Uma das vantagens da utilização de componentes é a provável redução no custo total do desenvolvimento de novas soluções. Porém, muitas vezes o custo relativo e a dificuldade de produzir adaptações, "*wrappers*" ou similares, pode comprometer, ou até mesmo inviabilizar os benefícios auferidos pela utilização de componentes. Em parte, este problema é conseqüência da maneira pela qual o componente é projetado, da forma como ele está documentado e da não utilização de técnicas que efetivamente dêem suporte à sua reutilização.

1.1.
Abordagem

Contribuindo para auxiliar principalmente o projeto de soluções baseadas em componentes de software e visando facilitar o contínuo processo de adaptações em soluções, tão recorrente hoje em dia, é que parte da complexidade do desenvolvimento de software pode ser limitada ao estudo e tratamento adequado de questões relativas a conceitos inerentes ao processo de desenvolvimento de software (Tarr et al., 1999).

A reutilização de unidades de software neste trabalho deve ser verbatim, isto é, não deve ser necessário aplicar alterações na unidade para a sua adequação ao contexto tratado em uma aplicação. Neste sentido, procura-se separar os conceitos de composição e de coordenação de componentes para contribuir com o uso verbatim de unidades de software já existentes, em várias aplicações.

Esta abordagem se justifica porque para gerar qualquer solução utilizando componentes é necessário compor os artefatos já existentes, e para atender adequadamente aos requisitos do problema é preciso coordenar as interdependências entre as unidades compostas.

Nesta dissertação é proposta a Arquitetura para a Coordenação e a Composição de Artefatos de Software (ACCA). ACCA estrutura os conceitos de artefatos de software, composição e coordenação de componentes em camadas seguindo o padrão arquitetural *Layer* (Buschmann, 1996). As camadas desta arquitetura foram utilizadas para reunir abstrações específicas de cada conceito, procurando isolar e restringir ao máximo possíveis alterações a uma única camada.

1.2. Motivação

Proclama-se que vivemos uma crise na indústria de software já há algum tempo (Brooks, 1975; McIlroy, 1968). Crise que na prática não existe, uma vez que programas de software de pequeno e grande porte são desenvolvidos e bilhões de pessoas se beneficiam das melhorias proporcionadas por eles (Glass, 2002).

Atualmente, o que está ocorrendo é um constante aumento na complexidade, na abrangência do tratamento e na convivência de um grande número de tecnologias, métodos e técnicas dentro de uma área ainda em formação, a Engenharia de Software.

É da natureza do ser humano questionar, criticar e propor novas maneiras para abordar e resolver problemas. O surgimento de novas tecnologias e o aprimoramento de métodos para o desenvolvimento de software são resultados deste questionamento. Porém, se este questionamento não for feito de forma organizada, ele terá como conseqüência a reinvenção ou o não aproveitamento de experiências e do conhecimento desenvolvidos até o momento. Dentro de Engenharia de Software, este pensamento é, em parte, realizado por meio de um conjunto de técnicas que visam reutilizar artefatos de software gerados a partir de soluções já existentes.

Entende-se por reutilização, o processo de criação de sistemas a partir de unidades de software já existentes ao invés de criá-las do zero (Sametinger, 1997).

Entretanto, existe um espaço enorme entre o discurso pregado por tantos pesquisadores e as técnicas e abordagens existentes. Muitas destas técnicas pouco auxiliam a reutilização de componentes de software, tal qual são disponibilizados. Diferentes adaptações em componentes de software ou na forma de uso destas unidades são necessárias em muitos casos, porém pouco suporte existe para restringir o impacto que estas adaptações ou evoluções geram.

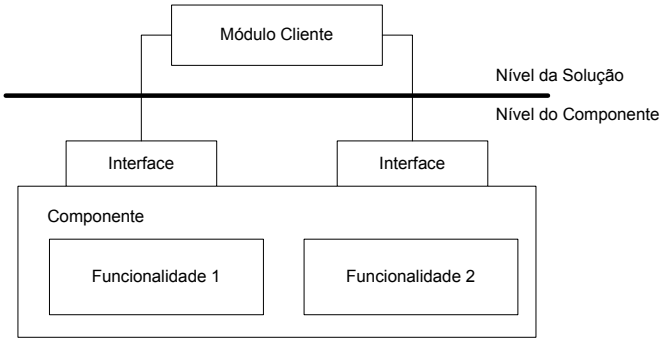

Figura 1 – Níveis de abstração: solução x componente

A utilização de componentes de software implica em pelo menos dois níveis de abstração (Figura 1): o nível do componente e o nível da solução. Embora não sejam responsabilidades dos componentes, as variações ou evoluções da solução geralmente terão algum impacto sobre eles se não houver algum critério para estruturar o desenvolvimento e a aplicação destas alterações. Porém, se as unidades de software forem tão suscetíveis à evolução, todo o investimento feito para gerar um artefato de qualidade precisa ser reaplicado, aumentando os custos finais do desenvolvimento da solução. Caso o custo total de adaptação na unidade torne esta alternativa proibitiva, ou melhor, caso seja mais barato desenvolver novas soluções a partir do zero, a aplicação da abordagem baseada em componentes torna-se pouco interessante.

Para contornar este problema, é preciso analisar a forma como as aplicações baseadas em componentes são desenvolvidas. Isolar questões estáveis de questões menos estáveis pode ser a chave para o sucesso de uma Engenharia de Software Baseada em Componentes. Neste sentido, parte da complexidade do desenvolvimento de software pode ser limitada ao estudo e tratamento adequado de questões relativas a conceitos inerentes ao processo de desenvolvimento (Tarr et al., 1999). Como já mencionado, a composição e a coordenação de componentes são exemplos de conceitos que precisam ser considerados neste processo.

Entende-se por composição, o processo recursivo de reunião e utilização de um conjunto de componentes de software. A coordenação pode ser definida como a gerência de interdependências entre as atividades executadas por um grupo de entidades para atingir um objetivo (Malone & Crowston, 1994). O estudo do conceito de coordenação pode auxiliar a análise da solução corrente e a elaboração de suas futuras versões, de seus cenários de uso e de abordagens alternativas para a solução de problemas.

Finalmente, pode-se concluir que a principal motivação deste trabalho é promover a reutilização verbatim[1] de componentes de software já existentes, empregando-os em uma variedade de aplicações. Neste sentido, o desafio deste trabalho é propor uma abordagem geral que permita a utilização integrada de diferentes modelos, técnicas e propostas para o auxílio à composição e à coordenação de componentes de software reutilizáveis. Além disto, esta abordagem apresenta entidades abstratas relevantes para auxiliar o entendimento das questões envolvidas neste processo.

1.3. Trabalhos Relacionados

Os diferentes trabalhos relatados nesta seção visam identificar algum ponto de interseção com a abordagem proposta neste documento.

1.3.1. Arquiteturas de Software

Model-Driven Architecture (MDA, 2003) é uma abordagem desenvolvida para atender a todo o ciclo de vida de integração de sistemas compostos por software, hardware, pessoas e práticas de negócio. Esta proposta provê uma abordagem sistemática para entender, projetar, operar, e evoluir diversos aspectos destes sistemas, utilizando para isto técnicas e ferramentas. MDA se baseia na modelagem de diferentes aspectos e níveis de abstração de um sistema, e na exploração dos inter-relacionamentos entre estes modelos. Assim como em ACCA, a arquitetura tem um papel central em MDA. Uma arquitetura é utilizada para orientar o projeto de sistemas de software. Mais do que um trabalho

[1] Reutilização verbatim: reutilização "*as is*", sem modificação.

relacionado, acredita-se que a abordagem de MDA pode ser utilizada, em um futuro breve, para estruturar a evolução de ACCA.

Além deste trabalho, outras arquiteturas de software poderiam ser desenvolvidas ou alguns padrões arquiteturais (Buschman, 1996; Schmidt et al., 2001) poderiam ser utilizados alternativamente para auxiliar a atender, parcial ou completamente, o objetivo deste trabalho.

Um exemplo de uma abordagem que poderia ser utilizada para auxiliar a resolução do objetivo deste trabalho é R-RIO (Sztajnberg et al., 1999). R-RIO, que estende o conceito tradicional de configuração, descreve a arquitetura de aplicações a partir de componentes e suas interações, e adicionalmente, de aspectos não funcionais, dentre eles a coordenação.

1.3.2. Contratos como Especificações Prescritivas

Nesta seção é descrito um resumo sobre contratos, obtido a partir do estudo de um bom número de definições relevantes para este trabalho (Helm et al., 1990; Holland, 1992; Fiadeiro & Lopes, 1997; Beugnard et al, 1999; Staa, 2000; Meyer 2000; Szyperski, 2000). A sua relação com ACCA está na possibilidade da utilização de contratos para a documentação das especificações de abstrações presentes nas camadas arquiteturais, úteis ao processo de geração de soluções.

Em Helm et al. (1990), contratos são vistos como construções de alto nível utilizadas para explicitamente especificar interações entre grupos de objetos. Em uma extensão deste trabalho, Holland (1992) define alguns refinamentos e formas de reutilização destas especificações. Estes contratos especificam composições comportamentais em termos dos objetos participantes, das obrigações de cada um, das invariantes que precisam ser mantidas, e dos métodos que o instanciam.

Em geral, contratos de reutilização podem ser vistos como especificações contendo propriedades em que um usuário, um cliente, ou um objeto podem confiar, pois qualquer implementação deste componente deve satisfazê-las (Fiadeiro & Lopes, 1997).

A partir do estudo de diferentes propostas, pode-se perceber que contratos não precisam e não devem ser utilizados para solucionar todos os problemas relativos ao processo de desenvolvimento de aplicações. Deve-se ter em mente que esta abordagem deve ser utilizada para simplificar o processo de geração de

soluções. Caso isto não seja considerado, a complexidade tradicional do desenvolvimento de soluções é simplesmente repassada para os contratos, o que deve ser evitado.

1.3.3. Abordagens de Separação de Conceitos

Com objetivo de contribuir para a separação de conceitos (Dijkstra, 1976), o problema "da tirania da decomposição dominante" (Tarr et al., 1999) também é abordado em ACCA. Em ACCA, esta questão é resolvida em parte ao prever uma camada específica para tratar da composição de artefatos.

Grande parte das abordagens de separação de conceitos existentes propõe diferentes paradigmas para realizar a decomposição de um sistema em unidades e a sua reunião por meio de modelos, métodos e técnicas próprias (Kiczales et al, 1997). Neste trabalho, em específico, é detalhada a relação entre programação orientada a aspectos (POA; AOSD, 2003) e ACCA.

O principal objetivo de programação orientada a aspectos (AOSD, 2003) é prover métodos e técnicas para decompor problemas em um conjunto de componentes funcionais e aspectos. Estes componentes e aspectos são compostos para obter implementações de sistemas (Czarnecki & Eisenecker, 2000). Ainda segundo Czarnecki e Eisenecker (2000), para resolver problemas com esta abordagem, além de identificar as abstrações básicas, é necessário responder a três perguntas. Destas três perguntas, duas são de suma relevância para o entendimento da relação entre POA e ACCA: (1) Quais são as questões relevantes que precisam ser separadas? e (2) Quais são os mecanismos de composição que podem ser utilizados?

Ao responder estas perguntas procura-se comparar POA com ACCA. No caso de POA, artefatos de software são componentes funcionais e aspectos, que por sua vez podem ser compostos e coordenados de alguma forma. (1) Nesta versão de ACCA, questões de coordenação e composição são separadas de artefatos de software. Como aspectos são artefatos de software, eles podem continuar a serem identificados independentemente de ACCA. A responsabilidade de ACCA está em oferecer mecanismos para viabilizar a composição destes artefatos e prover formas de gerenciar os relacionamentos entre os componentes funcionais e os aspectos. (2) A resposta à segunda pergunta está na realização da

camada de Composição. Como composição é um conceito intrínseco à reunião de artefatos, ao criar uma camada específica para este conceito, espera-se que o engenheiro de software entenda melhor o conjunto de mecanismos de composição que são passíveis de utilização. Além disto, como esta camada é sujeita a alterações pelo arquiteto de software, para atender a novas necessidades é possível estudar a inclusão de novos mecanismos de composição.

1.3.4. Darwin

Darwin (Magee et al., 1995) é uma linguagem declarativa de amarração que pode ser utilizada para definir hierarquicamente composições de componentes. A princípio, esta linguagem suporta a especificação de estruturas dinâmicas e de estruturas estáticas que podem ser evoluir durante a sua execução. As duas abstrações principais abordadas em Darwin são componentes e serviços. Serviços são as formas pelas quais os componentes interagem. Neste sentido, componentes podem prover serviços como saída e ao mesmo tempo requerer serviços como entrada. Por meio destas informações é possível determinar as características associadas à amarração de componentes que formam as chamadas composições.

A relação de Darwin com a proposta apresentada neste documento está no processo de especificação da amarração. Este processo pode ser visto como o estabelecimento de interdependências entre os componentes de software disponíveis. Em ACCA isto é feito na camada de Coordenação. No entanto, a proposta apresentada por (Magee et al., 1995) não se preocupa com a integração de diferentes tecnologias de componentes de software, questão esta que em ACCA é resolvida pela camada de Composição.

1.4. Guia do Leitor

O segundo capítulo desta dissertação apresenta alguns conceitos básicos para a sua elaboração. No segundo capítulo são abordadas algumas definições para componentes de software, além da conceituação de composição e de coordenação de componentes de software. Ainda no segundo capítulo, são discutidos alguns padrões de composição e padrões de coordenação, que são exemplos de abstrações aplicadas no desenvolvimento deste trabalho. Finalmente,

conceitua-se o que é uma arquitetura de software, além de explicar em mais detalhes o padrão arquitetural *Layer* (Buschman, 1996).

O terceiro capítulo apresenta a arquitetura ACCA, descrevendo em detalhes a sua organização em camadas. Neste sentido, são detalhadas abstrações para a camada de artefatos de software, para a camada de composição e para a camada de coordenação propostas em ACCA. Após este detalhamento, esta proposta é avaliada conceitualmente.

O quarto capítulo apresenta uma das possíveis realizações de ACCA. Este capítulo visa demonstrar a viabilidade desta proposta. Ainda neste capítulo, a realização da camada de coordenação de ACCA é exemplificada com a proposta de contratos de coordenação de Lano et al. (2002). Em seguida, é apresentado um *framework*, desenvolvido no escopo desta dissertação, para a realização da camada de composição de ACCA. Finalmente, é descrita a utilização de ferramentas de apoio ao armazenamento e à catalogação de artefatos de software para exemplificar a realização da camada Artefatos de Software de ACCA.

No quinto capítulo, após introduzir a arquitetura ACCA, é apresentado um método de reificação[2], que pode ser utilizado para a concretização de ACCA. Além disto, é descrito um pequeno processo de desenvolvimento que pode ser utilizado para a geração de soluções a partir de componentes de software já existentes.

O sexto capítulo discorre sobre o estudo de caso de serviços de busca de publicações acadêmicas e relata os experimentos com o desenvolvimento de alguns padrões de concorrência (Grand, 1998).

Finalmente, no sétimo capítulo são apresentados as conclusões, a lista de contribuições e os possíveis trabalhos futuros que poderão ser desenvolvidos.

[2] Método de tornar a proposta abstrata de ACCA em uma estrutura concreta, pronta para ser utilizada.