Universidade Federal de Campina Grande - UFCG
Centro de Engenharia Elétrica e Informática - CEEI
Coordenação de Pós-Graduação em Informática - COPIN

PROJETO DE SOFTWARE ORIENTADO A OBJETO

RESUMO - Inversion of Control Containers and the Dependency Injection pattern

Abril de 2008

João Arthur Brunet Monteiro
Mestrado em Ciências da Computação - CEEI/UFCG
Campina Grande, Abril de 2008

1 Resumo

Este documento é uma compilação das idéias contidas no artigo *Inversion of Control Containers and the Dependency Injection Pattern* [1], bem como das minhas opiniões a respeito do assunto abordado.

No artigo em questão, Fowler discute o uso do padrão "Dependency Injection", também conhecido genericamente como "Inversion of Control" e contrasta-o com uma alternativa a este padrão nomeada "Service Locator". Injeção de Dependência visa interligar componentes de diferentes projetos em uma aplicação coesa. Em particular, Fowler trata o problema usando a linguagem Java, onde existe uma tendência interligar componentes de diferentes projetos. No entanto, o autor lembra que a discussão pode ser extendida para outros ambientes orientados a objeto.

O autor deixa bem claro que o problema discutido é como interligar componentes que foram concebidos em projetos diferentes dos quais se tem pouco conhecimento. Em alternativa a complexidade presente na biblioteca padrão de Java (J2EE), vários *frameworks* vêm sendo propostos para endereçar o problema. Estes *frameworks* apresentam princípios interessantes de *design* que na visão do autor merecem ser discutidos.

Com o intuito de descrever melhor o problema e apresentar a solução, Fowler descreve um exemplo simples onde um método necessita de todos os filmes presentes na base de dados para retornar os filmes dirigidos por determinado diretor. A questão está em permitir que o método adquira a informação desejada não importando o modo como ela foi armazenada. Para isso, é definida uma interface *MovieFinder* e uma implementação *MovieFinderImpl*. O problema é que a classe que usa o serviço *findAll()* presente na interface *MovieFinder*, é dependente tanto da interface como da implementação, uma vez que ela necessita criar uma instância do tipo para invocar o método. Este cenário requer muito esforço caso haja mudanaça na maneira como os dados são armazenados.

Fowler cita que uma boa saída é conectar as classes em tempo de execução, e não de compilação. Isto é feito através de um arquivo de propriedades do sistema. O problema é como fazer a ligação entre as implementações de modo que a classe *MovieLister* não conheça implementação mas consiga usar o serviços.

O autor considera o termo "Inversão de Controle" muito genérico para descrever o problema abordado no artigo, uma vez que refere-se a inversão do controle da aplicação para o *framework*. No entanto, neste caso está sendo estudada uma maneira de ligar componentes visto que há pouco conhecimento entre eles. Por isso, o nome escolhido foi "Injeção de Dependência".

A idéia por trás de Injeção de Dependência é adicioanr um objeto (*as-*

sembler) que faça a ligação entre o cliente e a implementação apropriada, ou seja, entre *MovieLister* e uma implementação de *MovieFinder*. Três tipos de Injeção de Dependência são apresentados:

- Injeção via construtor visa ligar inserir um *MovieFinder* na construção do objeto *Movie Lister*. A ligação destes componentes é feita em um código separado podendo usar também arquivos de configuração para afetua-la.

- Injeção via método *set* visa prover um método para setar o *MovieFinder* de um *MovieLister*. A ligação é feita através de um arquivo XML de configuração.

- Injeção via interface, por sua vez, visa usar e definir interfaces para a injeção de dependência.

Fowler apresenta outro modo de tornar os componentes independentes, o *Service Locator*. Este padrão consiste em criar um entidade *ServiceLocator* que é capaz de prover os serviços que uma aplicação necessita. Neste caso, esta entidade seria responsável por retornar o *MovieFinder* apropriado. Fowler descreve *ServiceLocator* como um *Singleton* [2] onde os serviços podem ser registrados (configuração) e recuperados no programa. Além disso, apresenta várias formas de se implementar um *ServiceLocator*, na minha opinião a mais interessante é a que registra os serviços em um mapa e provê um método genérico que retorna o serviço de acordo com o parâmetro passado.

Uma discussão contendo as principais diferenças entre as duas abordagens é conduzida pelo autor. Em suma, Fowler destaca que comumente se escolhe Injeção de Dependência por permitir que *mocks* sejam criados sem muito esforço, facilitando a atividade de teste. No entando, o autor acredita que isso acontece nos projetos em que o *Service Locator* não é implementado de maneira apropriada que permita troca-lo facilmente. De fato, ambas as abordagens, se implementadas corretamente, são fáceis de serem testadas. Outro ponto defendido pelo autor é preferível usar injeção via construtor do que via método *set*, por questões de entendimento do código.

Como conclusão, Fowler acredita que mais importante do que a decisão entre user Injeção de Dependência e Service Locator, é a decisão de separar o uso de serviços da configuração dos mesmos. Neste ponto, concordo plenamente com o autor. Existem duas abordagens adequadas para separar a implementação do uso do serviços, o fato de usar uma das duas é mais importante do que a decisão de qual usar.

Referências

[1] Inversion of control containers and the dependency injection pattern. http://www.martinfowler.com/articles/injection.html.

[2] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1995.