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 - EVOLVING FRAMEWORKS

Abril de 2008

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

1 Resumo

O conteúdo deste documento consiste na exposição das idéias contidas no artigo "Evolving Frameworks: A Pattern Language for Developing Object-Oriented Frameworks" [2], bem como das minhas opiniões a respeito do mesmo.

Roberts e Johnson iniciam o artigo definindo que *frameworks* são *designs* reusáveis de sistemas de *software*. Além disso, são lembrados alguns benefícios provenientes do uso de *frameworks* na construção de aplicações. Dentre eles, o principal é a redução do custo desta atividade, visto que o reuso de código é uma característica intrínseca ao uso de *frameworks*.

Mesmo com o fato de *frameworks* apresentarem grandes benefícios ao desenvolvimento de *software*, os autores destacam que construir um bom *framework* é uma atividade complexa e cara, visto que ele necessita ser simples o suficiente para ser entendido, além de prover *features* e permitir flexibilidade para mudanças. Neste ponto, em minha opinião, reside o problema abordado pelos autores, ou seja, **a complexidade de se construir bons frameworks é alta**. Para justificar a construção de um *framework*, é preciso que o tempo economizado em função do reuso provido por ele seja maior que o tempo gasto para a construção do mesmo.

Com o intuito de endereçar o problema exposto, os autores propuseram um conjunto de padrões, nomeado linguagem de padrões, para o desenvolvimento de *frameworks*. Ao todo, são apresentados nove padrões:

- **Three examples**: Para desenvolver um *framework*, desenvolva primeiramente três aplicações no domínio do problema as quais o *framework* ajudaria no processo de desenvolvimento.

  Os autores advogam o uso deste padrão alegando que boas abstrações são geradas a partir da implementação de funcionalidades e quanto mais examplos forem analisados, mais gerais serão os *frameworks*.

- **White-Box Framework**: Desenvolver *frameworks* usando herança ao invés de composição, ou seja, construindo-o para que ele seja caixa-branca.

  Desta maneira, agrega-se uma alta flexibilidade à mudanças de determinadas funcionalidades além de um alto poder de reuso proveniente da herança.

- **Component Library**: Construir uma biblioteca contendo os objetos mais comuns e que poderão ser reusados por quem usa o *framework*.

Embora sejam aplicações diferentes construídas com o auxílio do *framework*, essas aplicações podem fazer uso de objetos semelhantes. Neste caso, também acredito que se houver uma biblioteca que disponibilize objetos comuns às aplicações, mais código pode ser reusado, diminuindo por consequência o tempo de desenvolvimento.

- **Hot Spots**: Hot Spots são as partes do código que se repetem com frequência quando se está desenvolvendo uma aplicação baseada no *framework*. Este padrão visa eliminar estes pontos através da criação de objetos, focando o desenvolvedor a mudar configurar esses objetos ao invés de implementar métodos.

- **Pluggable Objects**: Construir subclasses adaptáveis usando parametrização.

  Neste caso, os autores lembram que criar subclasses para mudar uma pequena propriedade é desnecessário, bastando adicionar parâmetros para diferenciar essas classes.

- **Fine-Grained Objects**: Dividir os objetos enquanto as divisões gerarem objetos com significados diferentes para o problema.

  Este padrão visa evitar a explosão de objetos no sistema, o que aumenta a complexidade do entendimento mesmo.

- **Black-Blox Framework**: Usar herança para organizar os componentes de biblioteca e composição para os componentes de aplicação.

  Conectar objetos sem precisar ter conhecimento sobre o seu funcionamento interno é uma das características de *framework* caixa-preta. Esta característica, na minha visão, é um ponto a ser destacado, uma vez que os componentes podem ser trocados/modificados em tempo de execução.

- **Visual Builder**: Criar um interface gráfica que permita a criação e conexão de objetos.

- **Language Tools**: Criar ferramentas de *debug* e inspeção para a interface gráfica criada no padrão anterior.

Concordo com os autores em relação ao documento e suas idéias sem grandes ressalvas. Acredito que a compilação dos padrões apresentados formam um bom roteiro para o desenvolvimento de um bom *framework*. Muitos deles, incentivam o uso dos padrões de projetos apresentados por Gamma e co-autores[1], o que é reconhecidamente uma boa prática de programação.

Referências

[1] 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.

[2] D. Roberts, R. Johnson, et al. Evolving Frameworks: A Pattern Language for Developing Object-Oriented Frameworks. *Pattern Languages of Program Design*, 3, 1998.