

PADRÕES E FRAMEWORKS PARA A CONSTRUÇÃO DE SISTEMAS

Giovane de Jesus Oliveira Freitas <giovanefreitas@yahoo.com.br>

Faculdade de Viçosa – Departamento de Ciências Exatas e Tecnológicas – Curso de Bacharelado em Sistemas de Informação

R. Dr. Milton Bandeira, 380- Centro- CEP 36570-000, Viçosa, MG

RESUMO

Padrões de projetos e *frameworks* são duas palavras muito utilizadas atualmente nos grupos de discussão, livros que se destinam a desenvolvedores e projetistas de software. Este artigo tem por objetivo apresentar ao leitor uma breve explicação sobre estes dois temas, apresentando suas definições, características e uma comparação entre eles.

INTRODUÇÃO

A reutilização é um tema muito estudado pela engenharia de *software*. Os desenvolvedores sempre buscam reutilizar *softwares*, trechos de códigos e análises já realizadas. O objetivo é utilizar o trabalho já feito para construção de sistemas futuramente, reduzindo o trabalho, tempo gasto, custos e facilitando a manutenção futura. A idéia básica é que os sistemas sejam projetados em forma de componentes independentes, para que possam ser aplicados aos mais diversos tipos de sistemas. Em busca deste ideal os desenvolvedores na década de 60 criaram diversas bibliotecas de funções que auxiliavam na construção de sistemas, mas estas bibliotecas sempre tratavam de domínios específicos [MALDONADO 2001a].

Com o surgimento da abordagem de desenvolvimento de *software* orientada a objetos, aumentaram-se também as motivações para a elaboração e implementação de sistemas modulares e bibliotecas reutilizáveis. Visando este objetivo, nas últimas décadas, foram realizadas diversas pesquisas sobre padrões de *softwares*, buscando definir padrões para análise, projeto, arquitetura e processo de desenvolvimento de sistemas [MALDONADO 2001a].

O surgimento do paradigma de orientação a objeto e a definição destes padrões propiciaram o aumento do reuso de *software* e contribuíram para o desenvolvimento de *frameworks* que permitem aumentar o reuso e reduzir o tempo e custos de desenvolvimento.

2. PADRÕES

Os padrões de projetos tiveram sua origem no trabalho feito pelo arquiteto Christopher Alexander no final dos anos 70. Ele escreveu dois livros: “A Pattern Language” e “A Timeless Way of Building”. [MALDONADO 2001a]

“Os padrões permaneceram esquecidos por um tempo, até que ressurgiram na conferência sobre programação orientada a objetos (OOPSLA) de 1987, mais especificamente, no Workshop sobre Especificação e Projeto para Programação Orientada a Objetos. Nesse trabalho, Beck e Cunningham falam sobre uma linguagem de padrões para projetar janelas em Smalltalk. A partir de então, muitos artigos, revistas e livros têm aparecido abordando padrões de software, que descrevem soluções para problemas que ocorrem freqüentemente no desenvolvimento de software e que podem ser reusadas por outros desenvolvedores. Um dos primeiros trabalhos estabelecendo padrões foi o de Coad, no qual são descritos sete padrões de análise. Nesse mesmo ano Coplien publicou um livro definindo inúmeros “idiomas”, que são padrões de programação específicos para a linguagem C++. Em 1993, Gamma e outros introduzem os primeiros de seus vinte e três padrões de projeto, que seriam publicados em 1995. Esses foram os trabalhos pioneiros na área de padrões, seguidos por muitos outros nos anos seguintes” (MALDONADO, 2001a).

Os padrões podem ser aplicados em diversos níveis de abstração, eles podem ser de arquitetura, projeto, análise, implementação, etc. Os padrões são organizados em pares “problema/solução” onde diversos tipos de problemas são explicitados e para cada um é definida uma estrutura que pode ser aplicada para solucioná-lo. Isto auxilia os desenvolvedores, uma vez que eles não precisam descobrir qual a melhor forma para solucionar um problema, basta encontrar o padrão com problema equivalente e aplicara solução.

A criação de uma “linguagem de padrões” (BRAGA), além de auxiliar no desenvolvimento propriamente dito, ajuda a melhorar a comunicação entre os membros da equipe de desenvolvimento, uma vez que ao citar o nome de um padrão é transmitido de forma indireta um grande volume de informações, que definem o padrão e como ele deve ser utilizado.

3. FRAMEWORKS

Um *framework* é o projeto de um conjunto de objetos que colaboram entre si para execução de um conjunto de responsabilidades. Um *framework* reusa análise, projeto e código (MALDONADO, 2001b). Ele reutiliza análise porque divide um problema maior em problemas menores, possibilitando que sistemas complexos sejam construindo apenas estabelecendo relações entre *frameworks* que resolvem problemas menores. Ele reutiliza projeto porque define a interface de programação que o desenvolvedor irá utilizar para desenvolver aplicações futuras. Ele reusa projeto porque contém algoritmos abstratos e descreve a interface que o programador deve implementar e as restrições a serem satisfeitas pela implementação. E finalmente, reusa código porque o desenvolvedor poderá herdar e especializar as classes disponibilizadas pelo *framework*.

3.1. Inversão de controle

Uma característica importante dos *frameworks* é a inversão de controle, esta inversão se dá pelo fato de que não é mais o desenvolvedor que define o curso da aplicação e sim o *framework* utilizado. Quando uma aplicação é desenvolvida tendo como base um *framework*, é necessário apenas implementar as estruturas estabelecidas por ele, a aplicação não irá definir o comportamento padrão, apenas irá responder às requisições feitas pelo *framework*.

O *framework* geralmente faz o papel de programa principal, coordenando e seqüenciado as atividades da aplicação (MALDONADO, 2001b).

3.2. Frameworks “caixa branca” x “caixa preta”

Um *framework* caixa branca é aquele em que a integração é feita através de implementação de classes que herdam de classes do *framework* ou que implementem interfaces pré-definidas. Para trabalhar com a estrutura caixa branca o desenvolvedor deve conhecer detalhes de implementação do *framework*, esta estrutura possui a vantagem de ser mais flexível, por permitir implementações personalizadas, porém exigem mais conhecimento sobre a estrutura do *framework*, dificultando o processo de aprendizado.

Já um *framework* caixa preta é aquele onde a integração é feita através da ligação de objetos do aplicativo a objetos do *framework*. Esta estrutura apresenta uma interface de comunicação com desenvolvedor relativamente simples, não é necessário conhecer a estrutura do *framework*, basta conhecer sua API e como utilizá-la. Este tipo de *framework* possui a vantagem de apresentar menos custo de aprendizado, porém pode ser menos flexível que a caixa branca.

Uma terceira estrutura de *framework* foi definida a partir destas duas, que é a estrutura caixa cinza, um meio termo, permitindo tanto a ligação de objetos como a herança de classes e implementação de interfaces. Esta estrutura pode apresentar um custo de aprendizado equivalente ao do *framework* caixa preta e a flexibilidade do *framework* caixa branca.

4. UMA COMPARAÇÃO ENTRE PADRÕES E FRAMEWORKS

Fazendo uma comparação entre padrões e *frameworks*, Johnson diz que padrões são mais abstratos do que *frameworks*, porque um *framework* é um *software* executável, enquanto um padrão representa conhecimento e experiência sobre *software* (pode-se dizer que *frameworks* têm natureza física enquanto que padrões têm natureza lógica). (MALDONADO, 2001a)

Os padrões são menores que os *frameworks* uma vez que um padrão é composto por poucas classes. Já um *framework* é uma estrutura muito mais complexa, constituída por diversas classes e podendo englobar

diversos padrões. Os padrões geralmente são criados para solucionar problemas de diversos tipos de aplicações, enquanto que os *frameworks* são criados para solucionar problemas específicos.

Este trabalho visa demonstrar alguns padrões e como podem ser aplicados, não é objetivo gerar uma documentação completa sobre os padrões. Para obter mais detalhes sobre os padrões é recomendada a leitura dos materiais citados na bibliografia.

Um grupo de padrões muito conhecido é o “Gangue dos quatro” também conhecido como GoF (*Gang of Four*). Este grupo de padrões lista 23 padrões divididos nas seguintes categorias:

1. De Criação: padrões para controlar a criação de objetos.
2. Comportamentais: padrões que coordenam a interação entre os objetos.
3. Estruturais: padrões que gerenciam os relacionamentos entre os objetos.
4. Sistema: padrões que gerenciam as interações em nível sistêmico.

5. CONCLUSÃO

A utilização de padrões de projeto e *frameworks* podem contribuir muito para o desenvolvimento de um sistema, principalmente por apresentar soluções prontas para os problemas enfrentados no processo de desenvolvimento, desde que seja feito um uso racional.

A utilização, sem critérios, de padrões pode resultar em uma aplicação extremamente complexa e de difícil manutenção, podendo também interferir no desempenho da mesma. Alguns padrões são complexos e exigem muita dedicação para seu entendimento, caso seja utilizado um padrão complexo para elaborar uma solução para um problema simples, isto trará mais malefícios do que benefícios, pois estará complicando algo que poderia ser resolvido de uma forma mais simples e clara.

O sucesso de um projeto utilizando padrões dependerá da capacidade dos projetistas para definir quando um padrão deve ser aplicado ou não, fazendo um uso racional e controlado das estratégias.

A utilização racional de *frameworks* exige decisões mais complexas, pois além de ter de definir quais componentes do sistema serão implementados através de *frameworks*, deve-se escolher um *framework* maduro, com previsão de crescimento e suporte disponível.

Os *frameworks* exigem um grande investimento para o treinamento da equipe de desenvolvimento, mas uma vez treinada, a equipe apresentará uma produtividade muito maior, possibilitando a construção de sistemas em prazos que seriam impraticáveis sem o auxílio destas estruturas. Esta afirmativa é válida, considerado-se que o *framework* escolhido irá simplificar o problema e não o contrário, pois o uso de um *framework* complexo para implementar um software simples não trará benefícios.

REFERÊNCIAS BIBLIOGRÁFICAS

STELTING STEPHE; MAASSEN OLAV. **Applied Java Patterns**. Prentice Hall PTR, 2001. 598 p. ISBN: 0-13-093538-7

PRONUS engenharia de software. **Conceitos Básicos de Controle de Versão de Software**. Disponível em: http://www.pronus.eng.br/artigos_tutoriais/gerencia_configuracao/controle_versao.php. Acesso em: 15 abr. 2006, 14:43

PRONUS engenharia de software. **Subversion**. Disponível em: http://www.pronus.eng.br/artigos_tutoriais/gerencia_configuracao/subversion.php. Acesso em: 15 abr. 2006, 14:43

PRONUS engenharia de software. **Controle de Mudanças com Trac**. Disponível em: http://www.pronus.eng.br/artigos_tutoriais/gerencia_configuracao/trac.php. Acesso em: 15 abr. 2006, 09:33

PRONUS engenharia de software. **O que é Gerência de Configuração?**. http://www.pronus.eng.br/artigos_tutoriais/gerencia_configuracao/gerencia_configuracao.php. Acesso em: 15 abr. 2006, 14:46

DEXTRA. **Controle de Versão de Software com o CVS**. Disponível em: <http://www.dextra.com.br/empresa/artigos/cvs.htm>. Acesso em: 15 abr. 2006, 14:50

MALDONADO, J. C.; BRAGA, R. T. V.; GERMANO, F. S. R.; MASIERO, P. C. **Introdução aos Padrões de Software**. São Carlos: Universidade de São Paulo, 2001. (a)

MALDONADO, J. C.; BRAGA, R. T. V.; GERMANO, F. S. R.; MASIERO, P. C. **Padrões e Frameworks de Software**. Universidade de São Paulo (b)

LARMAN, GRAIG. **Utilizando UML e padrões**. Porto Alegre: Bookman, 2000. 492 p.

Wikipédia. **Framework**. Disponível em: <http://pt.wikipedia.org/wiki/Framework>. Acesso em: 15 abr. 2006, 14:54

Wikipédia. **API**. Disponível em: <http://pt.wikipedia.org/wiki/API>. Acesso em: 11 jun. 2006, 14:37

[BRAGA] BRAGA, ROSANA T.V. . **Um Processo para Construção e Instanciação de Frameworks baseados em uma Linguagem de Padrões para um Domínio Específico**. Universidade de São Paulo.

FOWLER, MARTIN. **Inversion Of Control - Containers de Inversão de Controle e o padrão Dependency Injection**. Disponível em: <http://www.javafree.org/content/view.jf?idContent=1>. Acesso em: 15 jun. 2006, 14:53

BAUER, CHRISTIAN; KING, GAVIN. **Hibernate em Ação**. Rio de Janeiro: Ciência Moderna, 2006. 560p.
Core J2EE Patterns - Data Access Object. Disponível em: <http://java.sun.com/blueprints/corej2eepatterns/Patterns/index.html>