Frameworks de Aplicações Orientadas a Objetos – Uma Abordagem Iterativa e Incremental

Cristiane Marise Pérez da Silva Carneiro,
Empresa Brasileira de Correios e Telégrafos – Diretoria Regional da Bahia
cristiane@correios.com.br

Manoel Gomes de Mendonça Neto
Núcleo de Pesquisa em Redes de Computadores – Universidade Salvador
mgmn@unifacs.br

Resumo

Reutilização é uma das abordagens mais usadas para melhorar a qualidade e reduzir o custo e tempo de desenvolvimento de *software*. Programação orientada a objetos (OO) é freqüentemente citada como um dos meios para se atingir reusabilidade de software, e *frameworks* orientados a objetos podem ser usados para promover reutilização tanto ao nível de implementação como ao nível arquitetural. Todavia, existem várias dificuldades associadas ao desenvolvimento e uso de *frameworks* OO. As mais importantes são a complexidade de projeto e a dificuldade de compreensão e reutilização de *frameworks*.

Este artigo apresenta uma abordagem iterativa e incremental para o desenvolvimento de *frameworks* orientados a objetos. Esta abordagem cria uma família de *frameworks* OO com crescente grau de sofisticação, onde cada membro da família é por si só um *framework* completo e utilizável que contém mais funcionalidades que o anterior. A abordagem proposta é testada em um estudo de caso *in-vitro* onde uma família de três *frameworks* e três aplicações, uma para cada *framework*, são derivadas.

Palavras-chave: *Framework*, reutilização, orientação a objetos, desenvolvimento.

Abstract

Software reuse is one of the most important approaches used to improve software quality and to reduce software costs and development time. Object-oriented (OO) programming is often cited as a means to achieve software reuse, and object-oriented frameworks can be used to promote software reuse both at implementation and architectural levels. However, there are several difficulties associated with OO framework development and use. The most notable ones are the framework design complexity and difficulties to understand and reuse.

This paper presents an iteractive and incremental approach for object-oriented framework development. The approach creates a family of OO frameworks with increasing degree of sophistication, where each member of the family is a complete and reusable framework that has more functionality than the previous ones. The proposed approach is tested in a in-vitro case study where a family of three frameworks and applications - one for each framework - are derived.

Keywords: Framework, reuse, object-oriented, development.

1. Introdução

Na área de engenharia de *software* várias abordagens buscam melhorar a qualidade dos produtos gerados, bem como diminuir o tempo e o esforço necessários para produzi-los. A reutilização representa um dos fatores que levam ao aumento da qualidade de *software.*

Com o uso da tecnologia orientada a objetos destaca-se a importância da flexibilidade para garantir a reutilização de *software*. Com este objetivo, as técnicas de orientação a objetos utilizam mecanismos tais como herança, polimorfismo e associações dinâmicas.

Entretanto apenas os mecanismos de orientação a objetos por si próprios estão longe de resolver o problema, pois a reutilização não é característica inerente da orientação a objetos, mas é obtida a partir do uso de técnicas que produzam *software*s reutilizáveis.

Para ajudar no processo de reutilização a abordagem de *frameworks* de aplicações orientadas a objetos foi introduzida. Um *framework* consiste de código de *software* que é parcialmente abstrato e cujas partes abstratas podem ser especializadas para produzir aplicações.

Um *framework* deve ser simples o bastante para ser aprendido pelo desenvolvedor de aplicações, deve prover características suficientes que possam ser reutilizadas e ter métodos flexíveis para que estas características possam ser modificadas.

Projetar *frameworks* verdadeiramente flexíveis e reutilizáveis é difícil e trabalhoso, pois há vários problemas durante o seu desenvolvimento e utilização. Em especial citamos:

- Detecção de *hot spots* (pontos de flexibilidade): é difícil determinar quais de suas propriedades devem ser flexíveis.
- Implementação e validação: é difícil escolher a maneira adequada de implementar os *hot spots* e verificar o comportamento de um *framework*;
- Evolução: não é simples evoluir uma *framework* sem tornar sua estrutura (interfaces, hierarquia de classes e outros) complexa de gerenciar e entender.
- Aplicabilidade: é difícil determinar quando o *framework* é apropriado como um todo.
- Complexidade e curva de aprendizagem: *frameworks* freqüentemente se tornam complexos e difíceis de entender.

O objetivo deste trabalho é propor um processo para o desenvolvimento de *frameworks* de aplicações orientadas a objetos que utilizará o mecanismo de desenvolvimento iterativo e incremental para as etapas de coleta de requisitos, análise e detecção de *hot spots*, projeto do *framework*, sua implementação e elaboração de seu roteiro de utilização [2].

A utilização dessa abordagem apresenta como principal vantagem o fato de manter a complexidade da construção do *framework* sob controle através da geração de vários produtos intermediários, que evoluem de forma controlada a cada ciclo de desenvolvimento. Isso permite criar uma família de *frameworks* que pode ser utilizada pelo desenvolvedor para escolher o *framework* que melhor se adapte às necessidades de sua aplicação, considerando fatores como funcionalidade e dificuldade de reuso.

2. *Framework* Orientado a Objetos - Conceitos

Framework é um conjunto de objetos reutilizáveis que engloba conhecimento de determinadas áreas e se aplica a um domínio específico. Uma aplicação completa ou parte significante dela pode ser especializada dessa estrutura fazendo-se as adaptações necessárias ou adicionando-se novas características ao *framework* [15].

Um *framework* determina a arquitetura da aplicação e predefine parâmetros de projeto, permitindo ao projetista concentrar-se nos detalhes específicos da aplicação.

Ele é definido também como um projeto de reutilização que descreve como o sistema está decomposto em um conjunto de objetos que interagem entre si - o sistema pode ser uma aplicação inteira ou apenas um subsistema. O *framework* descreve a interface de cada objeto e o fluxo de controle entre eles [8].

2.1. Hot Spots e Utilização de Frameworks

As partes do *framework* que são abertas à extensão e especialização são denominadas *hot spots* - áreas variáveis ou pontos de flexibilização [15].

Hot spots tornam o *framework* flexível. Eles são uma espécie de lacuna no domínio da aplicação, sendo tarefa do desenvolvedor da aplicação preenchê-las com uma solução própria.

Segundo Lajoie e Keller [9] os *frameworks* não são bibliotecas de classes. Um *framework* é o projeto de um conjunto de classes que colaboram para realizar um conjunto de responsabilidades. Enquanto os componentes das bibliotecas de classes são usados individualmente, classes no *framework* são reutilizadas como um todo para resolver uma instância específica de um certo problema.

Da perspectiva do desenvolvedor de aplicações, a maior diferença entre um *framework* e uma biblioteca de classes está no conhecimento necessário para utilizá-los. Os usuários da biblioteca de classes precisam somente entender a interface externa das classes e definir toda a estrutura de sua aplicação. Em contraste, usuários de *frameworks* devem entender o projeto abstrato do *framework* bem como a estrutura de suas classes, de forma a adaptá-las ou estendê-las. As Figuras 2.1 e 2.2 ilustram essa diferença (a parte sombreada representa classes e associações que são reutilizadas).

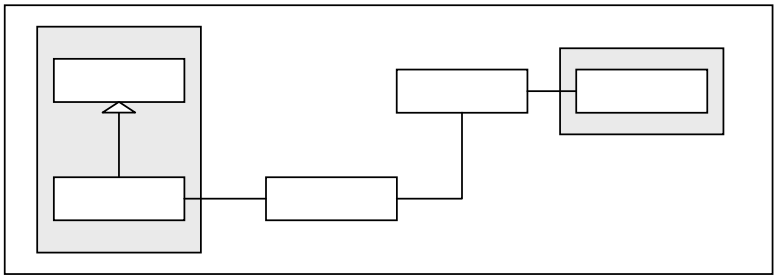

Figura 2.1 – Aplicação desenvolvida reutilizando bibliotecas de classes.

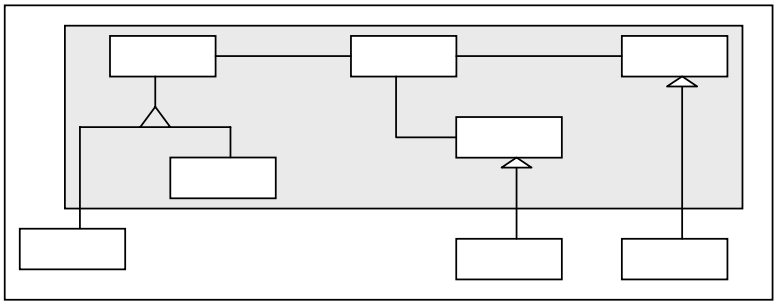

Figura 2.2 – Aplicação desenvolvida utilizando *framework*.

A reutilização promovida pela abordagem de *frameworks* se situa num patamar de granularidade superior à reutilização de classes, por reusar classes interligadas ao invés de classes isoladas [14]. Em geral, é mais difícil aprender a usar *frameworks* que bibliotecas de classes, contudo, seu potencial para reutilização excede bastante o potencial de reutilização da biblioteca de classes, contribuindo mais significativamente para o aumento de qualidade e produtividade no desenvolvimento de *software*.

2.2. Camadas do *Framework*

Os objetos do *framework* são na sua maioria descritos com classes abstratas. O projeto inclui a interface do componente e geralmente um núcleo para implementação. Por exemplo, uma classe abstrata pode definir uma estrutura para um algoritmo. Cada passo do algoritmo é definido como uma chamada para um método abstrato na mesma classe ou em outra classe.

Um *framework* pode especificar as implementações padrões para estes métodos ou deixá-los sem a implementação.

Uma aplicação derivada do *framework* pode usar diretamente as implementações padrões ou especificar novas classes concretas que herdam as classes abstratas e implementa seus métodos abstratos.

É uma boa prática é separar no *framework* as interfaces e a implementação abstrata. Isto resulta em um *framework* com camadas estruturadas claras. As camadas mais altas (mais abstratas) são independentes das camadas mais baixas (mais concretas). Assim as camadas mais baixas podem ser substituídas sem requerer a mudança das mais altas.

O modelo a seguir (Figura 2.3) divide as camadas do *framework* em três: camada de interface, camada de implementação núcleo e camada padrão [15].

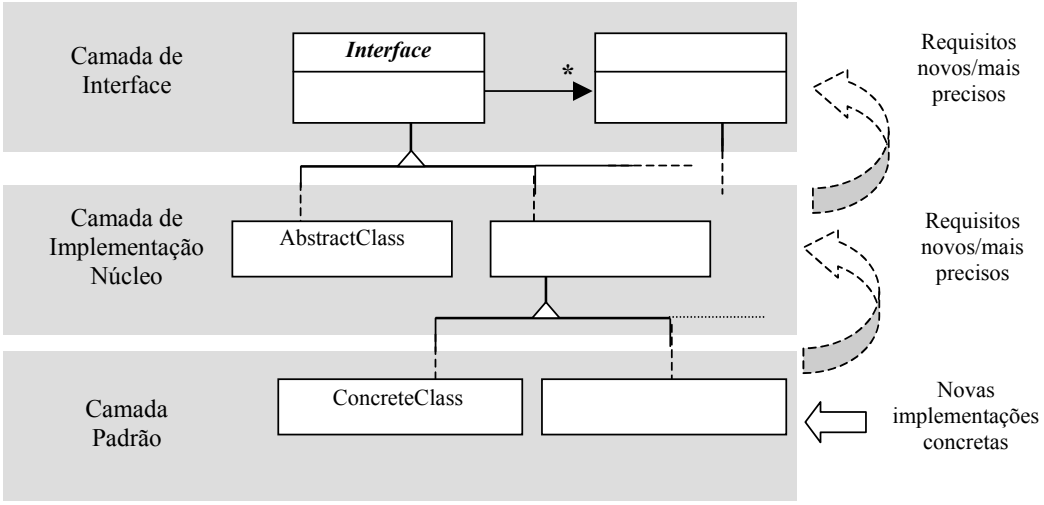

Figura 2.3 – Divisão do *framework* em camadas.

A camada de interface consiste de interfaces (totalmente abstratas) que definem os componentes básicos do *framework*, seus serviços e seus relacionamentos através das assinaturas dos métodos. A camada de interface é totalmente independente da camada de baixo, por exemplo, a implementação da interface.

A camada de implementação núcleo define o comportamento padrão do *framework* por implementar parcialmente a camada de interface com as classes abstratas. A implementação abstrata tem a intenção de ser usada na maioria dos casos de desenvolvimento de aplicações. Se o desenvolvedor por alguma razão for desenvolver sua própria implementação através da implementação direta de algumas das interfaces da camada de interface, sua carga de trabalho aumenta bastante.

A camada padrão contém implementações completas (classes concretas) para circunstâncias comumente recorrentes [15].

2.3. Relacionamento entre *Frameworks* e Padrões de Projeto

Frameworks são freqüentemente entendidos como sendo padrões de larga escala. Embora isto não seja verdade, o relacionamento entre padrões e *frameworks* é importante: padrões tipicamente têm o foco nos aspectos de flexibilidade de *software* e flexibilidade é exatamente o que é necessário em *frameworks* para possibilitar sua especialização nas

aplicações reais. Atualmente é amplamente aceito que padrões de projeto são bem adequados para projetar e documentar *frameworks.*

Para o desenvolvedor de aplicações, os padrões oferecem a possibilidade para melhor compreender os passos necessários para utilização - criação de subclasses ou configuração de classes - do *framework*. Como resultado, o desenvolvedor vê as classes de um *framework* numa perspectiva de mais alto nível. Padrões permitem que o *framework* possa ser entendido, sem exigir o conhecimento pleno do seu código fonte [15].

3. Desenvolvimento de *Frameworks*

No processo de desenvolvimento de um *framework* deve-se produzir uma estrutura de classes com a capacidade de adaptar-se a um conjunto de diferentes aplicações. Entre as principais características buscadas ao se desenvolver um *framework* estão a generalidade em relação ao domínio tratado e a flexibilidade (alterabilidade e extensabilidade) da estrutura produzida.

O desenvolvimento de um *framework* é diferente do desenvolvimento de uma aplicação padrão. A distinção mais importante é que *frameworks* tem que cobrir vários conceitos relevantes do domínio enquanto uma aplicação se preocupa somente com os conceitos mencionados nos requisitos da aplicação [1].

O desenvolvimento de um framework é então complexo pelos seguintes fatores [14]:

- Necessidade de considerar os requisitos de um conjunto significativo de aplicações de modo a torná-lo genérico.
- Necessidade de ciclos de evolução voltados a dotar a estrutura de classes do *framework* de alterabilidade (capacidade de alterar suas funcionalidades sem conseqüências imprevistas sobre o conjunto da estrutura) e extensibilidade (capacidade de ampliar a funcionalidade presente sem conseqüências imprevistas sobre o conjunto da estrutura).

Em termos práticos, dotar um *framework* de generabilidade, alterabilidade e extensibilidade requerem uma cuidadosa identificação das partes que devem ser flexíveis e a seleção de soluções de projetos de modo a produzir uma arquitetura bem estruturada [14].

3.1. Desenvolvimento Iterativo e Incremental

Um ciclo de vida iterativo se baseia no aumento e refinamento sucessivo de um sistema através de múltiplos ciclos de desenvolvimento - análise, projeto, implementação e teste. Ao contrário do clássico ciclo de vida em cascata, onde cada atividade é executada uma única vez para o conjunto inteiro de requisitos do sistema, o ciclo de vida iterativo está baseado no aperfeiçoamento sucessivo de um sistema. Através dos vários ciclos, o sistema é refinado, ajustado e são adicionados novos requisitos. Cada ciclo trata de um conjunto relativamente pequeno de requisitos [11].

O objetivo do desenvolvimento incremental é adicionar funcionalidades a um sistema durante vários ciclos de liberação de produto. Uma família de produtos pode ser composta de múltiplos ciclos de desenvolvimento iterativo onde cada produto membro da família contém mais funcionalidades que aquele gerado anteriormente.

Desenvolvimento baseado em *frameworks* apresenta vantagens em relação a abordagens tradicionais de desenvolvimento de *software*, mas há obstáculos que dificultam sua aplicação. Silva [14] identificou que os principais problemas são:

- A complexidade de desenvolvimento, que diz respeito à necessidade de produzir uma abstração de um domínio de aplicações adequada a diferentes aplicações.

- A complexidade de uso, que diz respeito ao esforço requerido para aprender a desenvolver aplicações a partir de um *framework*.
- A flexibilidade que o *framework* deve fornecer aqueles que querem utilizá-lo.

O processo de desenvolvimento do *framework* ocorre normalmente de forma iterativa. Sua estrutura de classes é refinada ciclicamente em função da aquisição de informações ou busca de soluções de projeto mais adequadas.

Este trabalho propõe a utilização de uma abordagem de desenvolvimento iterativo e incremental aliado ao uso de técnicas de análise e projeto orientado a objetos para o desenvolvimento de uma família de *frameworks*. Inicialmente, produz-se *frameworks* mais simples e fáceis de entender, para se lidar melhor com os problemas de complexidade do desenvolvimento e da dificuldade de compreensão dos *frameworks*. *Frameworks* mais simples e a realimentação gerada por seu uso no desenvolvimento de aplicações são então usadas nos ciclos de desenvolvimento seguintes para produzir *frameworks* mais sofisticados.

O processo iterativo e incremental ajuda a resolver algumas das questões relativas ao desenvolvimento de *frameworks* e contribui para a qualidade do mesmo, pois a cada ciclo de desenvolvimento ele limita o domínio abordado, simplifica o desenvolvimento e antecipa a utilização empírica do produto final. O conjunto de produtos desenvolvidos forma uma família de *frameworks* com crescente sofisticação dentro de um dado domínio.

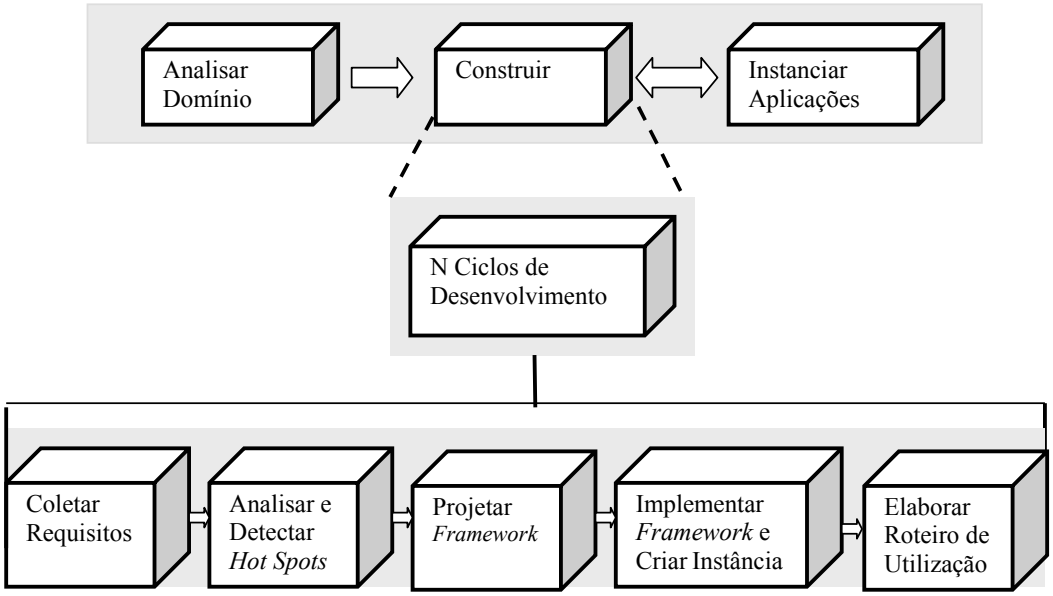

Figura 3.1 – Processo de desenvolvimento iterativo e incremental

3.2. Etapas do Processo Proposto para o Desenvolvimento Iterativo e Incremental

O processo de desenvolvimento proposto na Figura 3.1 inicia com a análise de domínio, onde passa para fase de construção, em seguida são feitos vários ciclos de desenvolvimento. A cada um destes ciclos, requisitos são adicionados ou refinados, é feita uma análise, *hot spots* são avaliados e o projeto é alterado para refletir os novos requisitos ou refinamentos. A cada ciclo é então implementado um *framework* e criado uma instância exemplo. Deve-se elaborar um roteiro ao final de cada ciclo para facilitar o entendimento e a instanciação de aplicações a serem geradas a partir do *framework* produzido. As subseções seguintes descrevem os passos do processo em maior detalhe.

3.2.1. Passo 1 - Analisar Domínio

A análise de domínio fornece a entrada para o desenvolvimento do *framework*. Trata-se de um passo através do qual a informação a ser usada no desenvolvimento do sistema é identificada, capturada e organizada. Ele pode ser visto como uma análise ampla que tenta obter os requisitos do domínio do problema, incluindo requisitos futuros.

Há dois documentos que são importantes neste passo e devem ser o resultado da análise de domínio: o escopo do domínio e um modelo estático contendo objetos (conceitos) importantes do domínio.

É importante salientar que se deve delimitar o escopo do domínio de um *framework*. Um grande escopo torna o *framework* reutilizável para uma maior variedade de aplicações, mas ao mesmo tempo o *framework* torna-se grande e difícil de gerenciar. Além disso, a maioria das características de um *framework* muito grande são pouco utilizadas. Por outro lado, um escopo muito pequeno limita a utilização do *framework* e pode não justificar o esforço gasto para o seu desenvolvimento.

3.2.2. Passo 2 - Construir *Framework*

Este passo tem por finalidade desenvolver um *framework* completo. É composto de vários ciclos de desenvolvimento que são descritos nos tópicos a seguir.

3.2.2.1. Passo 2.1 – Coletar Requisitos

Requisitos são descrições das necessidades ou dos desejos para um produto. O objetivo básico da fase de requisitos é identificar e documentar o que realmente é necessário, em uma forma que comunica claramente essa informação ao cliente e aos membros da equipe de desenvolvimento [11].

Os requisitos definidos por apenas uma pessoa não refletirão as reais necessidades que deverão ser atendidas pelo *framework*, assim é interessante a composição de uma equipe formada por pessoas da área de desenvolvimento, área usuária e com experiência no domínio. Uma estratégia proposta por Landin e Niklasson [10] é listar separadamente os requisitos de várias aplicações relacionadas, identificar os requisitos gerais das aplicações e colocá-los na lista de requisitos do *framework*. Caso não sejam identificadas aplicações relacionadas para se extrair os requisitos do framework, estes poderão ser abstraídos da análise do domínio em que residirá o framework.

Os ciclos de desenvolvimento podem ser organizados em torno dos casos de uso, os quais são divididos em casos de uso concretos e abstratos. Os caso de uso concretos são iniciados por um ator para produzir um resultado e os abstratos contém uma seqüência de ações que são compartilhadas por outros casos de uso. Em se tratando de *frameworks*, cujo uso não será por usuários finais, o mais adequado durante o seu desenvolvimento é a criação de casos de uso abstratos, já que estes não são utilizados por atores, mas por casos de uso concretos (da aplicação) ou outros casos de uso abstratos. Na abordagem iterativa proposta, a lista de casos de uso abstratos do ciclo anterior deve ser estendida com novos casos de uso que reflitam o tipo de funcionalidade requisitada do novo framework a ser derivado.

3.2.2.2. Passo 2.2 – Análise e Detecção de *Hot Spots*

Nessa etapa será delineado um modelo que preencha os requisitos levantados na etapa anterior bem como será coletado e descrito os aspectos variáveis do domínio do *framework*. No primeiro ciclo, o modelo estático preliminar da fase de análise de domínio servirá de entrada para gerar o modelo de objetos da análise (diagrama de classes). O refinamento fará

com que novas classes possam ser acrescentadas ou classes já existentes sejam removidas. A cada ciclo subseqüente, esse modelo gerado será novamente refinado, em função dos requisitos levantados na etapa de coleta de requisitos.

A diferença entre a atividade de análise do *framework* e de uma aplicação está no fato de que os conceitos do *framework* são generalizações dos conceitos das aplicações, ou seja, os conceitos que devem ser comuns nas aplicações são trazidos para o *framework*. É importante ressaltar que a introdução de novas abstrações deve estar dentro do domínio do *framework*.

A detecção de *hot spots* será efetuada verificando-se nos modelos (de domínio e de requisitos) ou nas aplicações já existentes, que aspectos fixos podem ser flexibilizados nas aplicações a serem desenvolvidas a partir do *framework* proposto. Para cada flexibilidade identificada devem ser preenchidos os dados abaixo. Estes dados formarão um catálogo de *hot spots*:

1. Número
2. Nome do *hot spot*
3. Descrição
4. Possíveis exemplos de utilizações do aspecto variável

Tabela 3.1 – Tipos de adaptação para um *hot spot*

Tipo	Descrição
1	Habilitar uma característica
2	Desabilitar uma característica
3	Substituir uma característica
4	Aumentar uma característica
5	Adicionar uma característica

Há vários tipos de *hot spots* em um *framework*. A Tabela 3.1 mostra cinco diferentes maneiras pela qual um *hot spot* pode ser adaptado para produzir aplicações específicas [4,5]. Os dois tipos mais comuns são habilitar uma característica e adicionar uma característica. Habilitar uma característica consiste em ativar características que fazem parte do *framework* mas podem não fazer parte da sua implementação padrão. Ao contrário de habilitar uma característica onde o desenvolvedor está usando serviços existentes, provavelmente de uma forma nova, adicionar uma característica envolve adicionar algo que o *framework* não estava capacitado antes. As adições são freqüentemente feitas através da extensão de classes existentes ou adição de novas classes.

Em relação ao grau de apoio fornecido pelo *framework* ao desenvolvedor existem três tipos:

1) Opcional: o desenvolvedor seleciona um componente pré-definido disponível numa biblioteca.

2) Suporte Padrão: as adaptações estão associadas a instanciações pré-definidas que dependem de informações fornecidas pelo usuário da aplicação.

3) Ilimitado: as adaptações são realizadas sem apoio do *framework*. O desenvolvedor tem liberdade para alterar o *framework* adicionando características não previstas no *framework* original.

Para completar a descrição do *hot spot* deve ser verificado em cada um deles o(s) tipo(s) de adaptação(es) necessária(s), com base na Tabela 3.1 e o grau de apoio fornecido, incluindo tais informações no catálogo do *hot spot*.

No primeiro ciclo de desenvolvimento são identificadas e analisadas as possíveis variabilidades que estão fixas nos artefatos gerados. Nos ciclos subseqüentes, à medida que

novas funcionalidades e/ou variabilidades são introduzidas, novas análises devem ser efetuadas e deve ser determinado quando incluir um ou vários *hot spots* em um ciclo.

3.2.2.3. Passo 2.3 – Projetar *Framework*

Nessa fase, o modelo estático de objetos criado na análise deve ser detalhado e os *hot spots* incluídos para gerar o diagrama de classe de projeto. Tanto os pontos de flexibilidade descritos no catálogo de *hot spots* são modelados quanto as partes fixas. Como mencionada na Seção 2.2, para separar o projeto do código as classes deverão ser estruturadas seguindo o modelo de camadas.

A finalidade da fase de projeto é fornecer uma base para uma implementação genérica que servirá a várias aplicações. Ou seja, deve prover reusabilidade. A maneira mais prática de aumentar a reusabilidade é através da utilização de padrões de projeto. O projeto do *framework* pode ser baseado em um ou mais padrões de projeto (vide Seção 2.3). Para isso os padrões devem ser investigados na literatura, como por exemplo no catálogo de padrões do livro de Gamma et al. [6]. Se eles se aplicam ao problema, este deve ser resolvido de acordo com a proposta do padrão de projeto.

3.2.2.4. Passo 2.4 – Implementação e Instanciação Exemplo

Com a finalização do diagrama de classes de projeto para o ciclo corrente de desenvolvimento há detalhes suficientes para gerar o código do *framework*. Segundo Larman [11] em termos realistas, durante a programação serão feitas muitas mudanças e serão descobertos e resolvidos problemas com os detalhes.

Uma das forças de um processo de desenvolvimento iterativo e incremental está no fato que os resultados de um ciclo anterior realimentam o início do ciclo seguinte. Assim, resultados subseqüentes de coleta de requisitos, análise e de projeto estão sendo continuamente refinados, beneficiando-se do trabalho de implementação anterior.

A implementação em uma linguagem de programação orientada a objetos requer que se escreva código fonte para as classes e os métodos. A forma para melhor fazer a transição do projeto para a implementação é através da utilização do diagrama de classes da fase de projeto. Este já contém o nome das classes, superclasses, assinaturas de métodos e os atributos simples de uma classe.

A instanciação de exemplo é feita gerando uma ou mais aplicações com o *framework* implementado. Através dessa instanciação é possível identificar se o *framework* realmente atende aos requisitos definidos e fornece a flexibilidade desejada.

3.2.2.5. Passo 2.5 – Elaborar Roteiro

A elaboração de um roteiro ajudará o desenvolvedor de aplicações a criar aplicações a partir do *framework*. Assim, o roteiro deverá conter informações suficientes para permitir o entendimento do *framework* e sua forma de trabalho.

Um modelo proposto é o seguinte:

1) Fazer uma descrição do objetivo do *framework*.
2) Listar o que é necessário para a aplicação utilizar o *framework.*
3) Explicar cada elemento básico do *framework* que a aplicação poderá utilizar em termos de classes concretas e/ou abstratas.
4) Mostrar exemplos que demonstrem a utilização dos elementos do item 3.
5) No caso de *framework* caixa branca (o usuário constrói classes a partir das classes disponíveis) explicar como e onde o desenvolvedor poderá estender as funcionalidades do *framework* e inserir seu próprio código.

6) No caso de *framework* caixa preta (o usuário tem que escolher uma das classes fornecidas) explicar como os elementos do *framework* são acoplados à aplicação.

É fundamental que o roteiro forneça uma visão geral do *framework* e especifique quais classes concretas estão disponíveis para uso, quais podem ser estendidas, o que pode ser implementado, quais classes abstratas são usadas, entre outros aspectos.

3.2.3. Passo 3 - Instanciar Aplicações

Há várias formas de usar um *framework*, algumas podem requerer um profundo conhecimento do *framework*. Todas as formas são diferentes da maneira tradicional de desenvolver *software* usando tecnologia orientada a objetos, já que todas forçam a aplicação a se encaixar no *framework*.

Desenvolver uma aplicação usando *frameworks* produzidos com a abordagem descrita neste trabalho envolve a escolha do *framework* adequado para a aplicação dentro da família de *frameworks*. A escolha deve ser baseada nos requisitos levantados para a aplicação a ser gerada, na documentação e no roteiro dos *frameworks* membros da família, a fim de verificar em qual deles a aplicação se encaixa melhor.

A maneira mais fácil de utilizar o *framework* selecionado é como um *framework* caixa preta. Neste caso, a aplicação reutiliza as interfaces do *framework* e as regras para conectar os componentes. Programadores de aplicação somente devem saber que objetos do tipo A são conectados a objetos do tipo B; eles não têm que conhecer as especificações exatas de A e B. Algumas vezes o novo uso do *framework* vai requerer novas subclasses. Esta forma de utilização que exige mais conhecimento, pois se precisa estendê-lo através das classes abstratas que formam o núcleo do *framework*. Embora mais árdua, esta é uma forma mais poderosa de se utilizar o *framework*.

É possível que a construção de aplicações sob um *framework* leve a obtenção de novos conhecimentos do domínio tratado, que talvez não estivessem disponíveis durante a construção do *framework*. Estas novas informações podem levar a necessidade de alterar o *framework*, causando a sua evolução. Neste caso, se for verificado que as novas características poderão ser utilizadas por uma larga faixa de aplicações, deve-se partir para um novo ciclo de desenvolvimento.

É importante notar que haverá um custo benefício entre a seleção de *framework* dentro da família disponibilizada. *Framework* mais simples serão mais fáceis de entender e selecionar, mas a depender da aplicação eles precisaram ser mais estendidos (tratados mais freqüentemente como frameworks caixa branca) que *framework* mais complexos. A documentação e roteiros de utilização dos *frameworks* membros da família são peças fundamentais no processo de seleção de qual *framework* será utilizado para desenvolver a aplicação. A estrutura incremental facilitará a compreensão destes *frameworks*.

Este passo conclui a descrição do processo interativo e incremental proposto. Uma descrição mais detalhada deste processo pode ser encontrada em [2]. Lá, se apresentam os critérios de entrada e de saída de cada um destes passos e os produtos resultantes deles.

4. Estudo de Caso

Nesta seção é apresentado um exemplo de desenvolvimento *in-vitro* de uma família de três *frameworks* utilizando o processo de desenvolvimento proposto [2]. Será utilizado como modelo base o *JHotDraw* versão 5.1, que é um *framework* gráfico implementado em Java para editores de aplicações baseados em objetos. *JHotDraw* foi escrito por Erich Gamma e pode ser encontrado em [7].

O *JHotDraw* foi reduzido a um conjunto mínimo inicial. Foram efetuados três ciclos de desenvolvimento seguindo os passos mostrados na Seção 3 (Figura 3.1) e gerados três *frameworks* (Figura 4.1). Cada *framework* da família pode ser utilizado para produzir um conjunto de aplicações que tenham os requisitos cobertos por ele. Toda documentação destes frameworks, incluindo os seus diagramas UML, podem ser encontrados no Capítulo 4 de [2]. Para comprovar a utilidade dos *frameworks* gerados, uma aplicação foi instanciada para cada um deles (Figuras 4.2, 4.3 e 4.4).

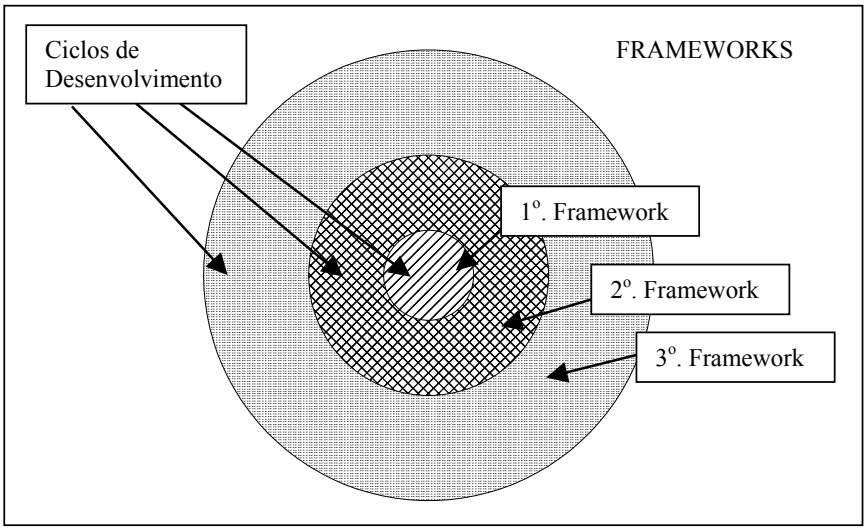

Figura 4.1 – *Framework* gerado a cada ciclo

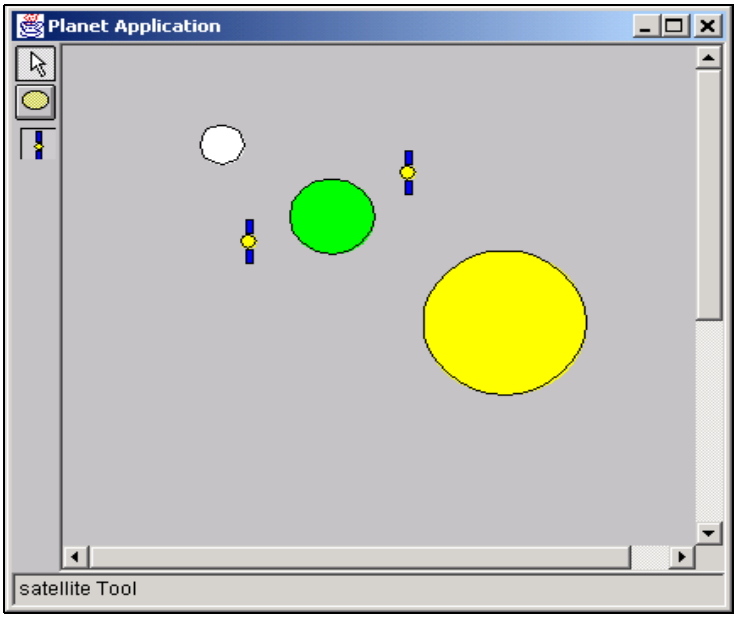

Figura 4.2 – *Interface* de *planet application*.

O primeiro framework disponibiliza uma área para o desenho de figuras, um conjunto de figuras, e as funcionalidades para: (1) desenhar linhas, (2) desenhar formas básicas, (3) agrupar figuras; (4) remover linhas e formas básicas; (5) alterar tamanho da figura e (6) mover figura. A partir deste framework foi criada uma aplicação para desenhar planetas e satélites em duas dimensões (Figura 4.2) [3].

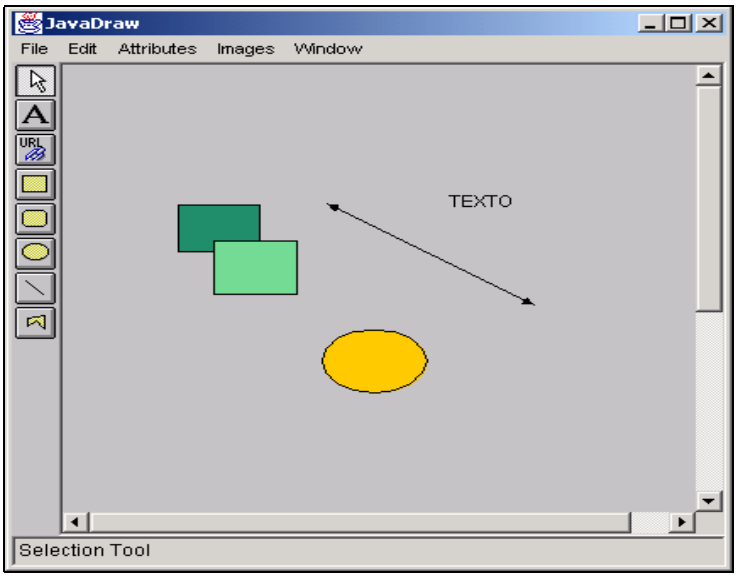

Figura 4.3 – *Interface* de *JavaDraw*.

O segundo framework disponibiliza todas as funcionalidades do primeiro, e adiciona funcionalidades para: (1) criação de menus; (2) inserção de texto com formatação de fonte estilo e tamanho; (3) execução de comandos de edição gráfica (recortar, colar, copiar). A partir deste framework foi criada uma aplicação para um editor gráfico de figuras geométricas (Figura 4.3) [7].

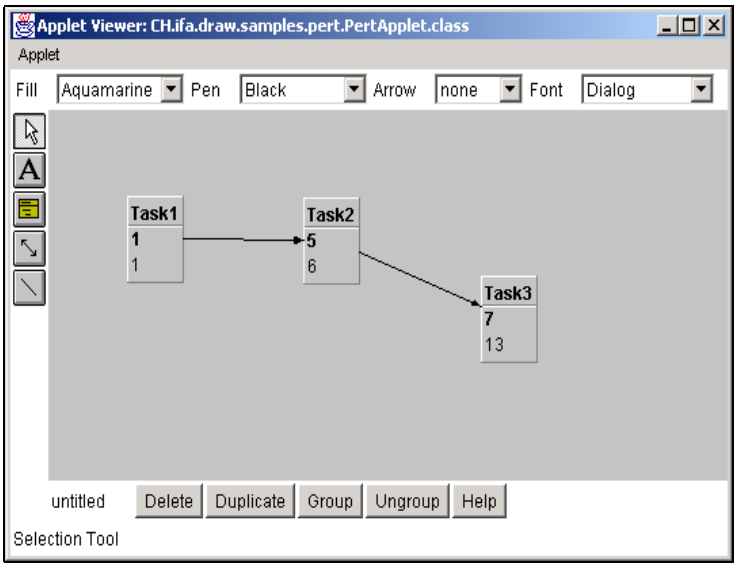

Figura 4.4 – *Interface* de *PertApplet*.

O terceiro framework disponibiliza todas as funcionalidades do segundo, e adiciona funcionalidades para: (1) criar conectores entre figuras; (2) permitir a decoração das figuras com bordas; (3) execução em ambiente Web. A partir deste framework foi criada aplicação para o desenho de redes Pert (Figura 4.4) [7].

5. Discussão do Trabalho Realizado

O desenvolvimento de *frameworks* orientados a objetos é uma tarefa complexa. Há vários problemas em se projetar e utilizar um *framework,* em especial:

- Definição de quais propriedades podem ser flexíveis (configuráveis, extensíveis ou reimplementáveis).
- Implementação dos *hot spots* e validação de suas instâncias.
- Controle da evolução do *framework*.
- Entendimento do domínio do *framework* e sua aplicabilidade.
- Controle da complexidade e dificuldade de aprendizado para sua utilização.

Para minimizar as dificuldades e orientar o desenvolvedor foi proposto um processo de desenvolvimento de *framework* utilizando a abordagem iterativa e incremental. Um estudo de caso foi realizado seguindo a abordagem proposta. No estudo utilizamos o *framework JHotDraw* como ponto de partida e desenvolvemos três *frameworks* incrementalmente mais complexos a cada ciclo de desenvolvimento.

O estudo mostra que é possível seguindo o processo proposto, produzir *frameworks* de maneira cíclica e incremental, criando uma família de *frameworks* com crescente grau de sofisticação.

Também indica que nossa abordagem é factível e pode ser usada para mitigar problemas como: (1) complexidade; (2) identificação e evolução de pontos de flexibilização; (3) entendimento de domínio; e (4) diminuição de dificuldade de aprendizado de *frameworks*. O estudo de caso, todavia, é limitado. Ele se baseia em um estudo *in-vitro* que parte da minimização de uma *framework* já existente para criação da família de *frameworks*.

6. Considerações Finais

Nos últimos anos a demanda por *software*s mais complexos e com maior qualidade. Para satisfazer a esses requisitos a reutilização é a forma encontrada, pois em contraposição ao desenvolvimento de todas as partes do sistema, reutilizar recursos previamente produzidos, torna possível produzir *software*s em menos tempo, com menos esforço e com mais qualidade.

O presente trabalho foi direcionado para a área de desenvolvimento de *framework* de aplicações orientadas a objetos entre as abordagens de reutilização existentes. O desenvolvimento baseado em *frameworks* apresenta vantagens em relação a abordagens tradicionais por promover a reutilização da arquitetura e do código, porém, a carência de processos de desenvolvimento voltados para a construção de *frameworks* dificulta a sua adoção.

Diante desse cenário, foram apresentados nas seções anteriores os conceitos de *framework* de aplicações orientadas a objetos, mostradas as principais dificuldades e problemas no seu desenvolvimento e proposto um processo iterativo e incremental. O processo proposto tem potencialmente as seguintes vantagens:

- Controle da complexidade, dado cria uma família de framework progressivamente mais sofisticadas.

- Facilita e identificação e evolução de *hot spots* com padrões de projeto e validação de suas instâncias através da geração de instâncias exemplo a cada ciclo.
- Controla a evolução do *framework* através do acompanhamento das etapas dos ciclos de desenvolvimento, contribuindo para a qualidade do mesmo.
- Facilita o entendimento progressivo do domínio do *framewor*k e sua aplicabilidade através da documentação gerada a cada ciclo.
- Diminui a dificuldade de aprendizado para utilização dos *frameworks* através dos roteiros progressivamente mais sofisticados que são gerados a cada ciclo.

O estudo de caso mostrou que a abordagem é factível através da criação de uma família de três *frameworks* incrementalmente mais sofisticadas. O estudo de caso, todavia, foi limitado e se baseou em um estudo *in-vitro* que partiu da minimização de uma *framework* já existente para criação da família de *frameworks*. Um estudo mais complexo e *in-vivo* está sendo planejado para criação de uma família de *frameworks* para mineração visual de dados [12]. Os trabalhos futuros planejados são:

- Extensão do processo de desenvolvimento para incluir etapas de teste e manutenção de *framework.*
- Estudo de técnicas de documentação específicas para esse tipo de família de *framework.*
- Estudo de como escolher a melhor forma para se evoluir um *framework.*
- Avaliação de custo/benefício para evolução do *framework.*

Referências Bibliográficas

[1] BOSH, J.; MOLIN, P.; MATTSON, M.; BENGTSSON, P.; FAYAD, M. **Framework Problems and Experience**. In Fayad, M.E.; Johnson, R.E; Schmidt, D.C. Building Application Frameworks: Object-Oriented Foundations of *Framework* Design, John Wiley & Sons.pp.55-82, 1999.

[2] Carneiro, C. **Frameworks de Aplicações Orientadas a Objetos – Uma abordagem Iterativa e Incremental**. 2003, 110 f. Dissertação de Mestrado. UNIFACS - Universidade Salvador, Salvador, Bahia, Brasil.

[3] DOUGLAS, KIRK. **Pratical 4 Solutions**, Department of Computer Science, University of Strathclyde, 2002. Disponível em : <https://www.cis.strath.ac.uk/teaching/ug/classes/52.440/prac4Sol-2002.html>. Acesso em: 07 de dez. 2002.

[4] FROEHLICH, G.; HOOVER, H; LIU, L; SORENSON, P. **Hooking into Object-Oriented Application Frameworks**; Proceedings of International Conference on Software Engineering; 1997.

[5] FROEHLICH, G.; HOOVER, H; LIU, L; SORENSON, P. **Reusing Hooks**; In Fayad, M.E.,Schmidt, Douglas C; Building Application Frameworks*:* Object Oriented Foundations of *Framework* Design, 1999; pp 219-236.

[6] GAMMA, E. R. HELM; JOHNSON, R.; VLISSIDES J. **Design Patterns: Elements of Reusable Object-Oriented Software**. Reading, MA: Addison Wesley, 1995.

[7] GAMMA, E. **JHotDraw Framework**, 2001 Disponível em: <http://members.pingnet.ch/gamma/JHD-5.1.zip>. Acesso em: 05 de abr. 2002.

[8] JOHNSON, RALPH. E.; FOOTE, B. Designing Reusable Classes. **Journal of Object-Oriented Programming**, 1988. Disponível em: <ftp://st.cs.uiuc.edu/pub/papers/frameworks/designing-reusable-classes.ps>. Acesso em: 10 de maio 2001.

[9] LAJOIE, RICHARD; KELLER, RUDOLF K. **Design and Reuse in Object-Oriented Frameworks: Patterns, Contracts and Motifs in Concert**; In: Proceedings of the 62nd Congress of the Association Canadienne Francaise pour l'Avancement des Sciences, Montreal, Canada, 1994 Disponível em: <ftp://st.cs.uiuc.edu/pub/patterns/papers/acfas.ps>. Acesso em: 15 de mai. 2001.

[10] LANDIN, NIKLAS; NIKLASSON, AXEL. **Development of Object Oriented Frameworks**. 1995, 154 f. Master Thesis. Department of Communication Systems, Lund University, Sweden.

[11] LARMAN, CRAIG. **Utilizando UML e Padrões Uma Introdução a Análise e ao Projeto Orientado a Objetos**. Editora BookMan, 1999.

[12] MENDONÇA, MANOEL; ALMEIDA, MÁRCIO. **Uso de Interfaces Abundantes em Informação para Exploração Visual de Dados**. In: Brazilian Workshop on Human Factors in Computer Systems (IHC´2001),. Brazilian Workshop On Human Factors in Computer Systems, P. 256-268, 2001, Florianópolis.

[13] PREE, WOLFGANG. **Design Patterns for Object-Oriented Software Development**. Reading, MA:Addison-Wesley, 1995.

[14] SILVA, RICARDO PEREIRA. **Suporte ao Desenvolvimento e Uso de Frameworks e Componentes**. 2000. 262 f. Tese (Doutorado em Ciência da Computação) - Universidade Federal do Rio Grande do Sul, Instituto de Informática, Rio Grande do Sul.

[15] VILJAMAA, ANTTI. **Pattern-Based Framework Annotation and Adaptation – A Systematic Approach**. 2001. 118 f. Thesis (Licenciate Thesis in Computer Science), University of Helsinki, Finland.