Frameworks Conceitos e Aplicações

Article · November 2021

CITATIONS
0

READS
119

2 authors:

Semíramis Assis
Universidade Católica do Salvador
3 PUBLICATIONS **1** CITATION
SEE PROFILE

Rita Suzana Pitangueira Maciel
Universidade Federal da Bahia
55 PUBLICATIONS **368** CITATIONS
SEE PROFILE

Frameworks
Conceitos e Aplicações

Semíramis Ribeiro de Assis

Aluna do Curso de Ciência da Computação da Faculdade Ruy Barbosa, sob orientação da professora Rita Suzana.

sema@lognet.com.br

Resumo

O conceito de reuso de código está sendo cada vez mais explorado pelos desenvolvedores de software para diminuir tempo e esforço de programação. Com esta finalidade, os *frameworks* estão sendo desenvolvidos e pesquisados, permitindo que partes ou todo um sistema possa ser reutilizado. Por meio de uma pesquisa bibliográfica, este artigo conceitua e apresenta os *frameworks* de diversas categorias, considerando as vantagens e desvantagens da sua utilização. Conclui-se que a utilização de *frameworks* tem vantagens e desvantagens, e que já existem muitos *frameworks* sendo utilizados.

Palavras-chave

Frameworks, *Patterns*, orientação a objeto.

1. Introdução

O reuso de código é uma das principais metas para os desenvolvedores de software. O problema do reuso é que apenas pequenas partes do código podem normalmente ser reaproveitadas. Uma proposta para que uma parcela maior de código seja reutilizada, diminuindo, assim, o tempo de programação, são os *frameworks* orientados a objetos [1].

Um *framework* tem por objetivo a reutilização de código em aplicações que estejam em um mesmo domínio, isto é, a construção de um *framework* visa atender uma

determinada classe de aplicações [2]. Ele descreve como será a interação entre as classes e objetos do sistema, e como será a decomposição do sistema em objetos [3].

A usabilidade dos *frameworks* se estende por uma variedade de áreas de *design* de *software* como, por exemplo, interfaces de busca e catalogação, protocolo de rede, sistemas operacionais, etc [4].

Por meio de uma pesquisa bibliográfica, busca-se conceituar *frameworks*, apresentar sua aplicabilidade e alguns exemplos existentes.Primeiro será explicada toda a parte conceitual de *frameworks* (seções 2, 3, 4, 5, 6). Em seguida, a aplicabilidade e usabilidade de *frameworks* serão apresentadas (seção 7), seguindo-se de um exemplo de *framework* já existente (seção 8). Por último, a conclusão final (seção 9).

2. *Frameworks* – Conceituação

Um *framework* se baseia nas principais características da programação orientada a objetos: abstração de dados, polimorfismo e herança. Estas características são as responsáveis por tornar o *framework* possível de ser feito objetivando a reutilização, pois permitem a construção de classes genéricas [3].

A abstração acontece no projeto de um *framework* no momento que se define quais as funções possíveis de serem implementadas na aplicação específica, ou seja, quais as características que não serão acopladas ao código do *framework*.

Em *framework,* a herança ocorre quando classes que são esqueletos para a implementação de características específicas do problema são implementadas, herdando suas características da superclasse correspondente no *framework*.

O polimorfismo se faz presente a partir do momento em que classes existentes no projeto do *framework* são sobrescritas para atender às especificidades da aplicação em desenvolvimento.

Cada objeto num *framework* é uma classe abstrata, ou seja, eles são apenas esqueletos que podem ser utilizados como modelo para a implementação de componentes. Quando esses esqueletos são utilizados, o que resta ao programador é incluir o código específico à sua aplicação [2].

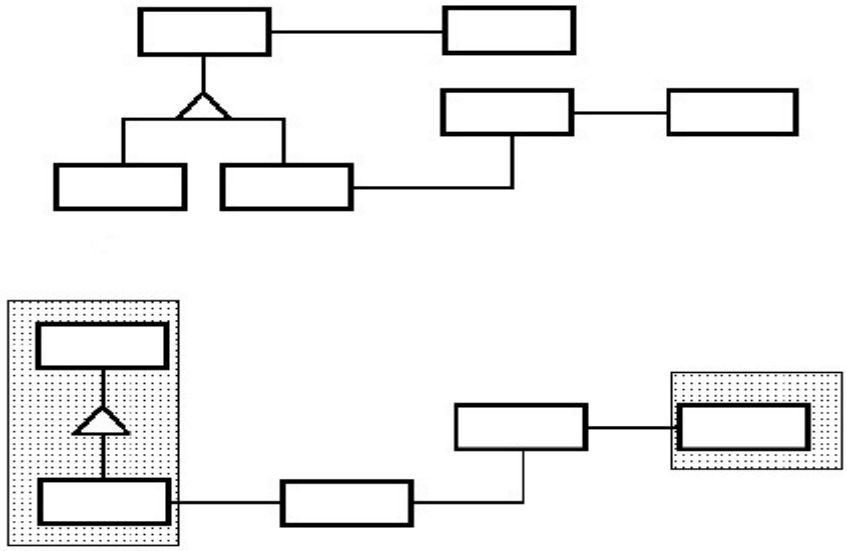

Figura 1 – Comparação entre um sistema desenvolvido sem e com reutilização de código.

Fonte:SILVA, 2000. [5]

A figura 1 ilustra uma implementação de um sistema que, primeiramente, não faz reuso de código, em comparação com um sistema que toma proveito do reuso (a área mais escura da figura representa as classes reutilizadas). No primeiro caso, todas as classes devem ser implementadas, ocasionando um maior gasto de tempo e esforço do programador, além de propiciar uma maior existência de erros. No segundo caso, apenas algumas classes devem ser desenvolvidas, reduzindo as probabilidades de erro e o tempo de desenvolvimento.

O inter-relacionamento de classes num *framework* é o que o diferencia da reusabilidade de biblioteca de classes, ou seja, no último a interligação entre as classes deve ser determinada pelo desenvolvedor da aplicação, enquanto que esta interligação é parte constituinte do primeiro [5]. Esta diferença é ilustrada na figura 2.

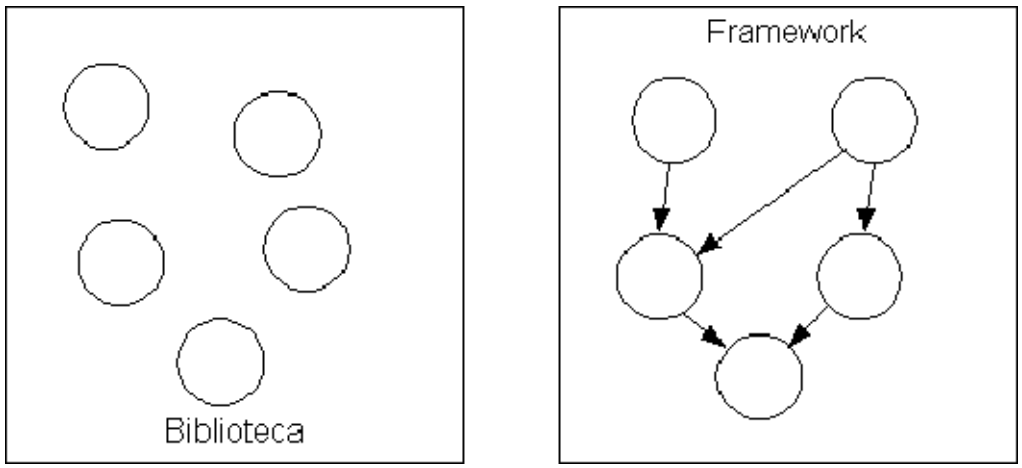

Figura 2 – Comparação entre Biblioteca de classes e um *Framework*

Fonte: Sauvê, 1999. [2]

Num *framework* há uma inversão de controle, ou seja, não é o programador quem irá determinar o momento certo para invocação de métodos e procedimentos, mas o próprio *framework* se encarrega disso e de manter o fluxo de execução na aplicação [3].

Uma característica importante de um *framework* é a confiabilidade, pois parte de seu código já está pronta e testada, diminuindo o esforço com manutenção [1].

Além do reuso de código, já explicado acima, um *framework* também reutiliza outras partes de um projeto, como *design* e análise. Com um *framework* é possível particionar um sistema em pequenos componentes, descrevendo a interface entre eles. Esta característica torna possível construir um número grande de sistemas com poucos componentes existentes. O reuso da análise se dá devido ao *framework* ditar quais são os componentes necessários a um domínio de problema específico e descrever uma linguagem para este domínio, ocasionando numa melhor comunicação entre os projetistas de um mesmo *framework* [3].

Os benefícios do uso de *frameworks*, além do reuso, da facilidade de manutenção e da inversão de controle, são a capacidade de extensão e a modularidade. A modularidade é atingida quando os detalhes de implementação são encapsulados pelo *framework* em classes fixas. Já sua extensibilidade se dá devido ao *framework* disponibilizar "esqueletos" de métodos, permitindo que as classes possam ser estendidas. A extensão de um *framework* é uma característica fundamental para economizar tempo de programação [3].

3. Classificação dos *Frameworks*

Segundo Silva [5], a classificação de *frameworks* pode se dá a partir do modo de como ele será empregado, que pode ser voltado para dados ou para a arquitetura. Se um *framework* é voltado para os dados, a construção de aplicações se dá pelas várias maneiras de

combinar a instanciação das classes já existentes. Já os voltados para a arquitetura possuem subclasses construídas tendo como base as classes pré-existentes do *framework*.

Os *frameworks* podem ser classificados de acordo com seu escopo nas seguintes categorias:

1 ***System Infrastructure frameworks*:** são os *frameworks* utilizados na construção de sistemas operacionais e interface com usuário, por exemplo. Sua função é simplificar a construção da infraestrutura destes tipos de sistemas, buscando a portabilidade e eficiência dos mesmos.Também conhecidos como *frameworks* horizontais, segundo Sauvê [2].

2 ***Middleware Integration frameworks*:** são os *frameworks* responsáveis pela comunicação em ambientes e aplicações distribuídas. Como exemplo, podemos citar o *framework Object Request Broker* (ORB) e os bancos de dados transacionais.

3 ***Enterprise Application frameworks*:** são os *frameworks* voltados para aplicações comerciais e para a área dos negócios. Este tipo de *framework* tem a capacidade de construir aplicações para o usuário final, resultando numa melhor relação investimento/retorno, porém são mais caros que os dois tipos anteriores [3]. Também conhecidos como *frameworks* verticais, de acordo com Sauvê [2].

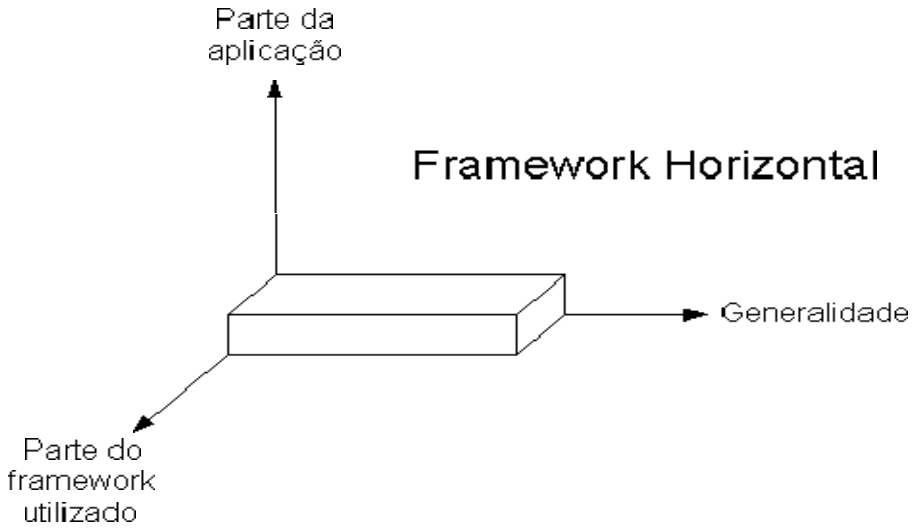

Figura 3 – Exemplo de *framework* de aplicação.

Fonte: Sauvê, 1999. [2]

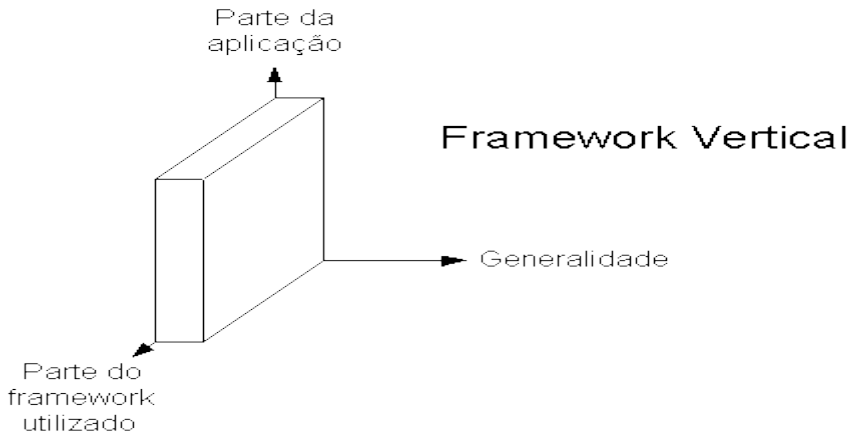

Figura 4 – Exemplo de *framework* de domínio.

Fonte: Sauvê, 1999. [2]

As figuras 3 e 4 ilustram os tipos de *framework* de aplicação (*system infrastructure frameworks*) e de domínio (*enterprise application frameworks*). O enfoque maior no *framework* de aplicação se dá na parte utilizada do *framework* e na generalização deste. Já no *framework* de domínio, a parte da generalidade perde enfoque para a parte da aplicação e a do *framework* que está sendo utilizada.

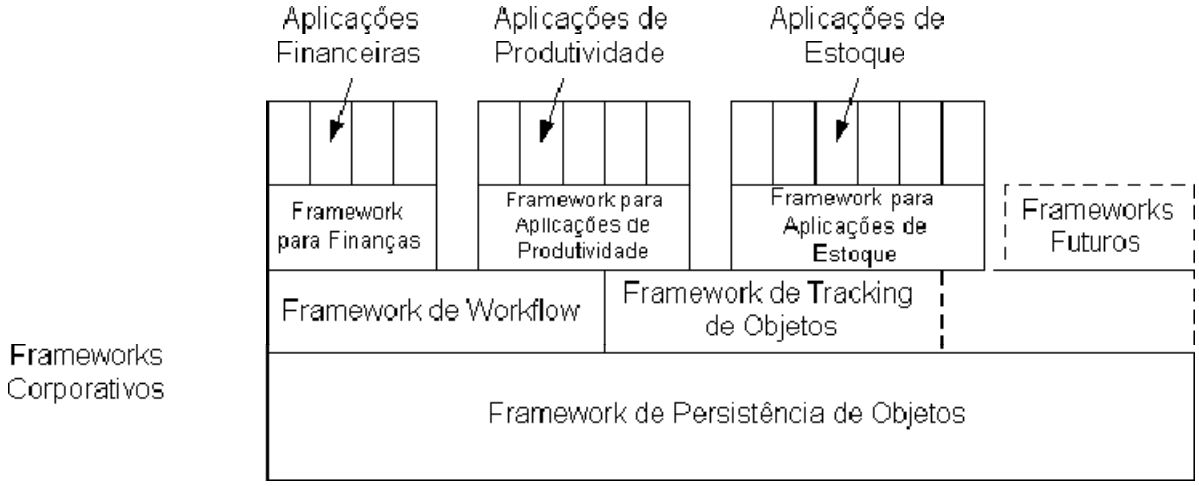

Figura 5 – Exemplo da combinação entre *frameworks* verticais e horizontais.
Fonte: Sauvê, 1999. [2]

A figura 5 ilustra como pode ser feita a combinação entre os *frameworks* verticais e horizontais. Os horizontais servem como uma base sobre a qual se apóiam os verticais, devido ao primeiro ter uma maior preocupação com a generalidade do domínio, enquanto que o segundo busca o detalhe ao nível das aplicações.

A combinação destes tipos de *frameworks* pode, também, não ser utilizada, escolhendo-se um tipo ou outro [2].

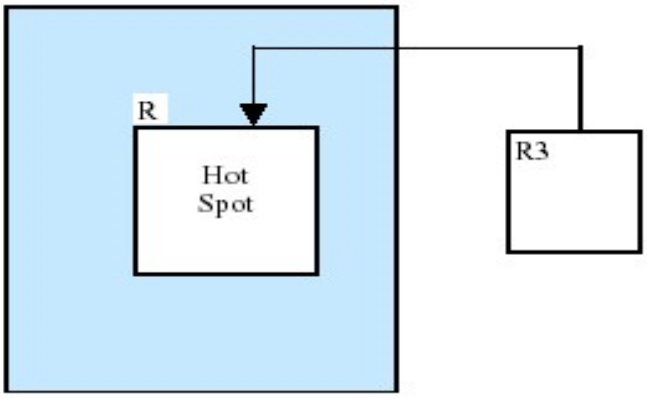

Figura 6 – *Framework* de caixa branca.

Fonte: Maldonado. [6]

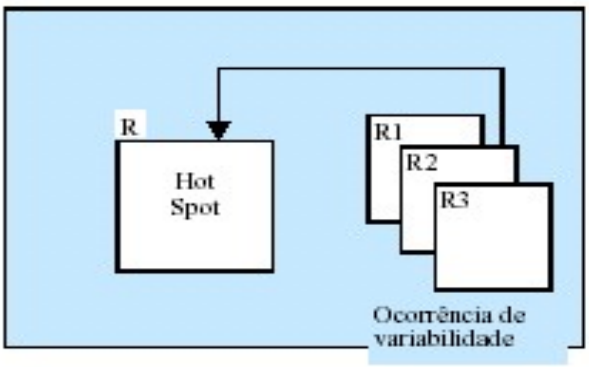

Figura 7 – *Framework* de caixa preta.

Fonte: Maldonado. [6]

Uma outra classificação dos *frameworks* é quanto à maneira como se dá sua extensão. Esta classificação se divide em três: *frameworks* de caixa branca (*whitebox*), de caixa cinza (*graybox*) e de caixa preta (*blackbox*). Os de caixa branca se baseiam nas características da orientação a objetos, como herança e ligação dinâmica, e requisitam dos desenvolvedores um bom conhecimento interno do *framework*. Os de caixa preta se baseiam na composição de objetos, ou seja, eles definem a interface com os componentes de modo que elas sejam conectadas ao *framework* por meio da composição de objetos, tendo maior facilidade de uso e extensão que os de caixa branca. Já os de caixa cinza são um balanceamento entre os de caixa preta e branca, pois ele provê tanto flexibilidade quanto capacidade de extensão, evitando assim as desvantagens destes dois tipos [3].

As áreas que possuem características específicas de uma aplicação num *framework* são denominadas *Hot Spots*. Em aplicações que pertencem a um mesmo domínio, um ou mais *Hot Spots* podem as caracterizar. Já as partes de um *framework* que são fixas para todas as aplicações que o utilizam são denominadas *Frozen Spots* [6].

As figuras 6 e 7 ilustram o funcionamento de *frameworks* de caixa branca e preta, respectivamente. No *framework* de caixa branca, sua utilização é feita com base no conhecimento do código do *Hot Spot* da aplicação, porém não é preciso muito esforço para buscar um método para a implementação deste *Hot Spot*, enquanto que no de caixa preta é necessário que se conheça as alternativas possíveis de implementação para que se possa escolher qual a melhor [6].

4. Vantagens e Desvantagens dos *Frameworks*

Até o momento foram abordadas algumas características dos *frameworks* que os tornam vantajosos como técnica de construção de *software*. Iremos, então, apresentar algumas desvantagens do uso deste método.

O custo de treinamento para o uso de *frameworks* é alto, e a aprendizagem leva um tempo considerável, o que permite concluir que a relação custo/benefício não é tão boa, mesmo considerando que o treinamento perdurará por vários projetos [3].

A construção de um *framework* não é simples, e deve ser bem planejada para que o objetivo principal (reuso de código) seja alcançado [2].

A validação e remoção de erros podem ser complicadas num *framework*. O uso de classes genéricas na composição de um *framework* não só auxilia a abstrair os detalhes da aplicação como dificultam a processo de depuração do código, pois não podem ser depurados separadamente da parte específica da aplicação. Outra "faca de dois gumes" é a característica de inversão de controle num *framework*, pois ao alternar o fluxo de execução entre a parte específica do *framework* e a parte específica da aplicação, evita que os desenvolvedores tenham controle sobre o código do *framework*, dificultando a localização de erros [3].

Outro problema do uso de *frameworks*, concordando com Fayad [3], é relativo à sua implementação em uma linguagem específica. Esta característica limita o uso do *framework*, visto que linguagens de programação orientadas a objeto nem sempre podem ser combinadas, não sendo interessante trabalhar com outra linguagem que não a da construção do *framework*.

A documentação de um *framework* é uma questão muito importante para sua utilização. A deficiência na documentação gera um grande acréscimo na dificuldade de treinamento para o uso do *framework*, o que caracteriza mais uma desvantagem [3].

Figura 8 – Comparação entre o custo e benefício no uso de *frameworks*.

Fonte: Maldonado. [6]

A figura 8 ilustra um resumo das vantagens e desvantagens na utilização de *frameworks*. Mesmo com todas as desvantagens, as vantagens são ainda maiores, o que viabiliza sua utilização pelos programadores.

5. *Frameworks* X Outros métodos de reuso

Uma diferença entre *frameworks* e outros meios de reuso de código é que o primeiro se dá através de programas escritos em uma determinada linguagem, não requerendo uma notação ou ferramentas especiais [3].

Frameworks também podem se comparar aos *Patterns* (padrões) para reuso de código e *design*.Um *Pattern* é a generalização de um problema que ocorre com freqüência em um meio, descrevendo sua solução genérica, o que possibilita que esta seja utilizada inúmeras vezes [9]. O que diferencia os dois é o fato que os *Patterns* ditam apenas o problema a resolver, sua possível solução e o como deve ser o comportamento desta solução, enquanto que os *frameworks* incorporam um conjunto destes padrões [3]. O resultado é que um *framework* pode conter diversos padrões, mas a recíproca não é verdade [2].

Um *Hot Spot* e sua interação com o *Frozen Spot* num *framework* são identificados a partir do padrão que compõe este *framework*. É de responsabilidade do *Pattern* gerenciar a comunicação entre a parte fixa e a variável do *framework* [8].

Uma característica importante para um bom *design* de um *framework* é que este evite o máximo de repetição de código, pois, ao incorporar novas funcionalidades ao

framework, não é interessante ao desenvolvedor voltar sua atenção ao código já existente, temendo numa possível repetição, mas se focar no código que está sendo desenvolvido no momento [7].

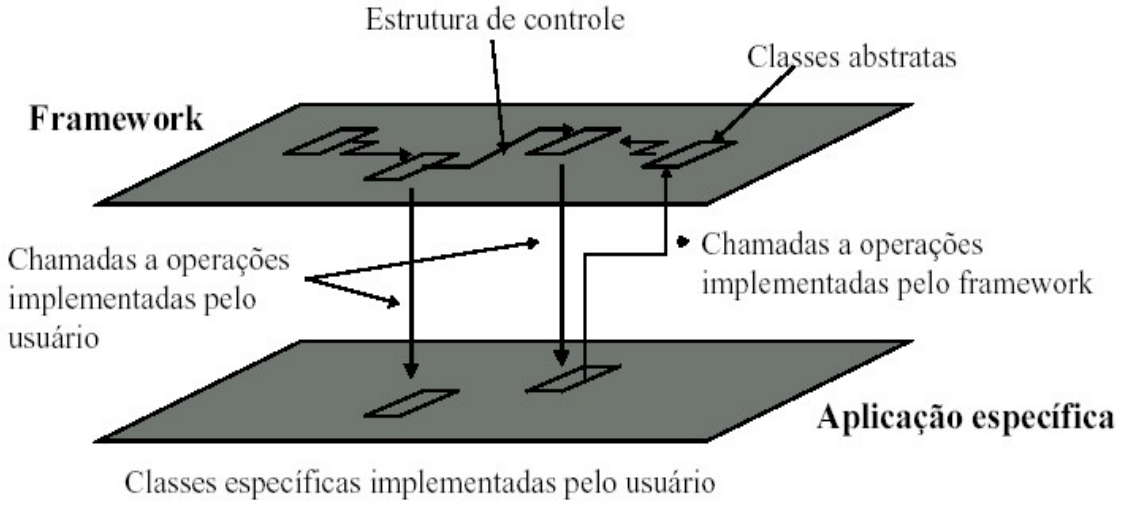

Figura 9 – A estrutura de um *framework*

Fonte: CCUEC ,1999. [10]

A figura 9 ilustra a estrutura interna e externa de controle de um *framework*, além do seu relacionamento com as características específicas de uma aplicação feita com base neste *framework*.

6. Ciclo de Vida de um *Framework*

Esta seção irá apresentar o ciclo de vida de um *framework*, seguindo os princípios de Silva [5].

O ciclo de vida de um *framework* está diretamente relacionado aos componentes que o deram origem, ou que surgiram a partir deste *framework*.

O projetista do *framework* não só tem como responsabilidade a definição do grau de flexibilidade que os usuários poderão ter, como tem importância na manutenibilidade e na estrutura sobre a qual o *framework* é composto.

A construção de um *framework* deve ser direcionada para um determinado domínio de aplicações. Com base nisso, são as aplicações deste domínio que dão suporte à criação do *framework* específico para ele, e, conseqüentemente, um *framework* também pode sofrer alterações devido a algum detalhe - antes desconhecido ou desconsiderado pelo desenvolvedor - pertinente de uma aplicação do domínio de aplicações englobado por este *framework*.

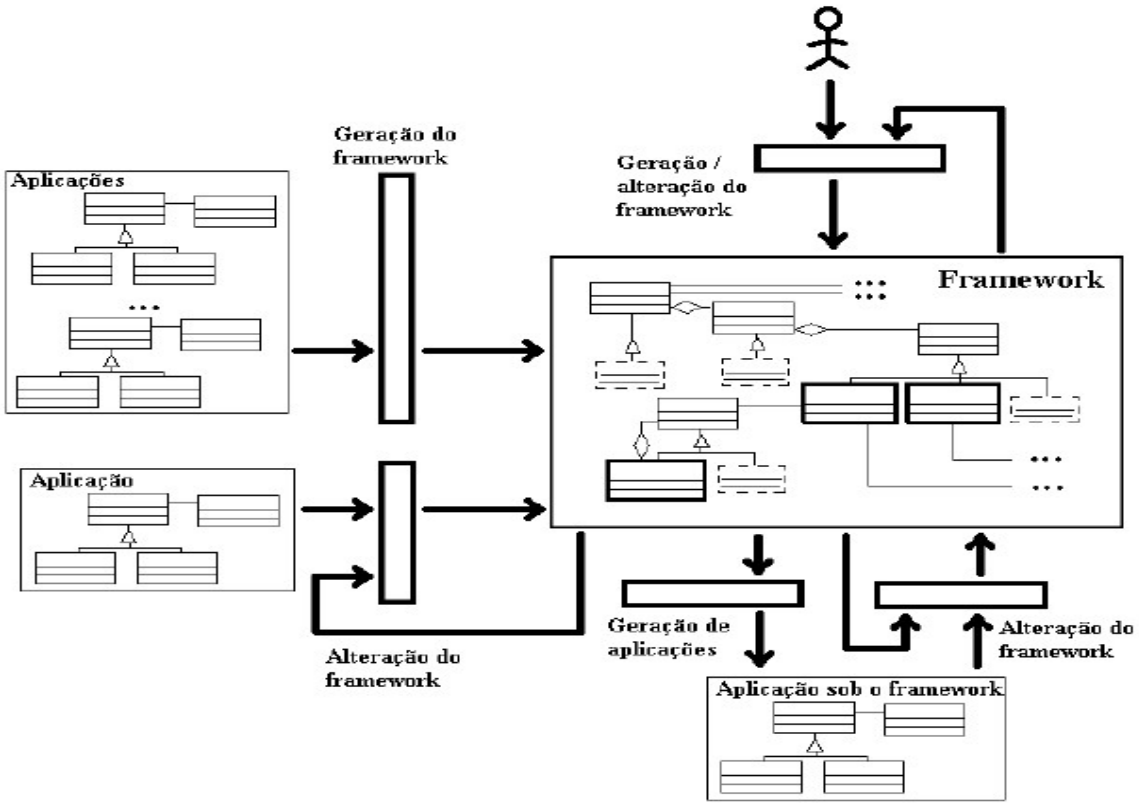

Figura 10 – Ciclo de vida de um *framework*.

Fonte: Silva, 2000. [5]

A figura 10 ilustra toda a interação existente no ciclo de vida de um *framework*, focando desde sua geração até a criação de aplicações baseadas no *framework*. A figura também ilustra a possibilidade de alteração do *framework* devido à existência dos detalhes não considerados no momento de sua criação, mas que são importantes para o domínio o qual engloba.

Pode-se concluir que o ciclo de vida de um *framework* só acaba quando todos os requisitos exigidos pelo seu domínio de aplicação forem atendidos, ou quando o domínio englobado pelo *framework* deixa de ser utilizado.

Se o domínio para o qual o *framework* foi construído for suficientemente grande, o nível de detalhamento pode ser tanto que o ciclo de vida do *framework* não chegue a um final.

7. Usabilidade dos *Frameworks*

Com o intuito de tornar um *framework* fácil de ser utilizado, é importante que algumas regras sejam seguidas. Estas regras serão apresentadas segundo Fayad [3].

A primeira delas é a simplicidade. No projeto de um *framework*, as classes que não serão utilizadas ou que serão subutilizadas devem ser evitadas, de modo que o projeto

fique mais simples de ser entendido pelos desenvolvedores. Se os métodos num *framework* forem muito extensos, é melhor que o divida em partes menores, podendo-se organizar estas pequenas partes em *frameworks* diferentes que possam se relacionar. Isso se deve ao custo de aprender muitos métodos ser maior que o de aprender poucos métodos em *frameworks* separados.

Um outro fator importante para a usabilidade de um *framework* é aumentar a produtividade de seus usuários através de métodos e classes menores. Isso significa que as super classes e os métodos que as compõem devem ser reduzidos, mas não devem ser tão poucos que proíba os usuários de sobrescrever métodos.

Se as denominações do mundo real forem utilizadas no *framework* proposto, o entendimento do usuário será maior em pouco tempo. Não é muito sugestivo o uso de abreviações no lugar de nomes mais complexos, visto que os últimos são mais fáceis de memorizar que os mnemônicos.

Um padrão para nomes de classes e métodos deve ser adotado. Para um melhor entendimento dos usuários ao utilizarem um *framework,* é interessante que o nome da classe seja compatível com o nome do método que está contido nela.

Ao desenvolvedor cabe, também, a tarefa de separar as funções num *framework*, ou seja, cada método deve ter uma função específica, não podendo existir métodos muito parecidos com outros. Com isso, o projetista do *framework* deve ter a cautela de não permitir que métodos com significados bem-definidos no mundo real possam ser sobrescritos, tendo seu significado alterado.

Evitar que o usuário cometa erros é mais uma tarefa de um *framework.* Para que esta tarefa seja possível, é necessário que as partes importantes do *framework* sejam encapsuladas, não deixando que o usuário esteja com o controle absoluto do código.

A reversibilidade dos resultados é outro fator de importância para a usabilidade de um *framework*. Permitir que os resultados sejam vistos imediatamente após a realização da operação e desfeitos, se necessário, é uma boa alternativa para aumentar o grau de usabilidade deste *framework*.

8. Exemplo de *framework*

Nesta seção, apresentarei um exemplo de *Framework* para Interfaces de Busca e Catalogação de Conteúdo do *ContentNet*, utilizando o padrão *Instructional Management Systems* (IMS), que tem por objetivo padronizar a distribuição e utilização de conteúdos

acadêmicos nos sistemas de aprendizagem. Este *framework* se divide em dois: um para a interface de Busca e outro para a de Catalogação.

- *Framework* para interface de Busca:

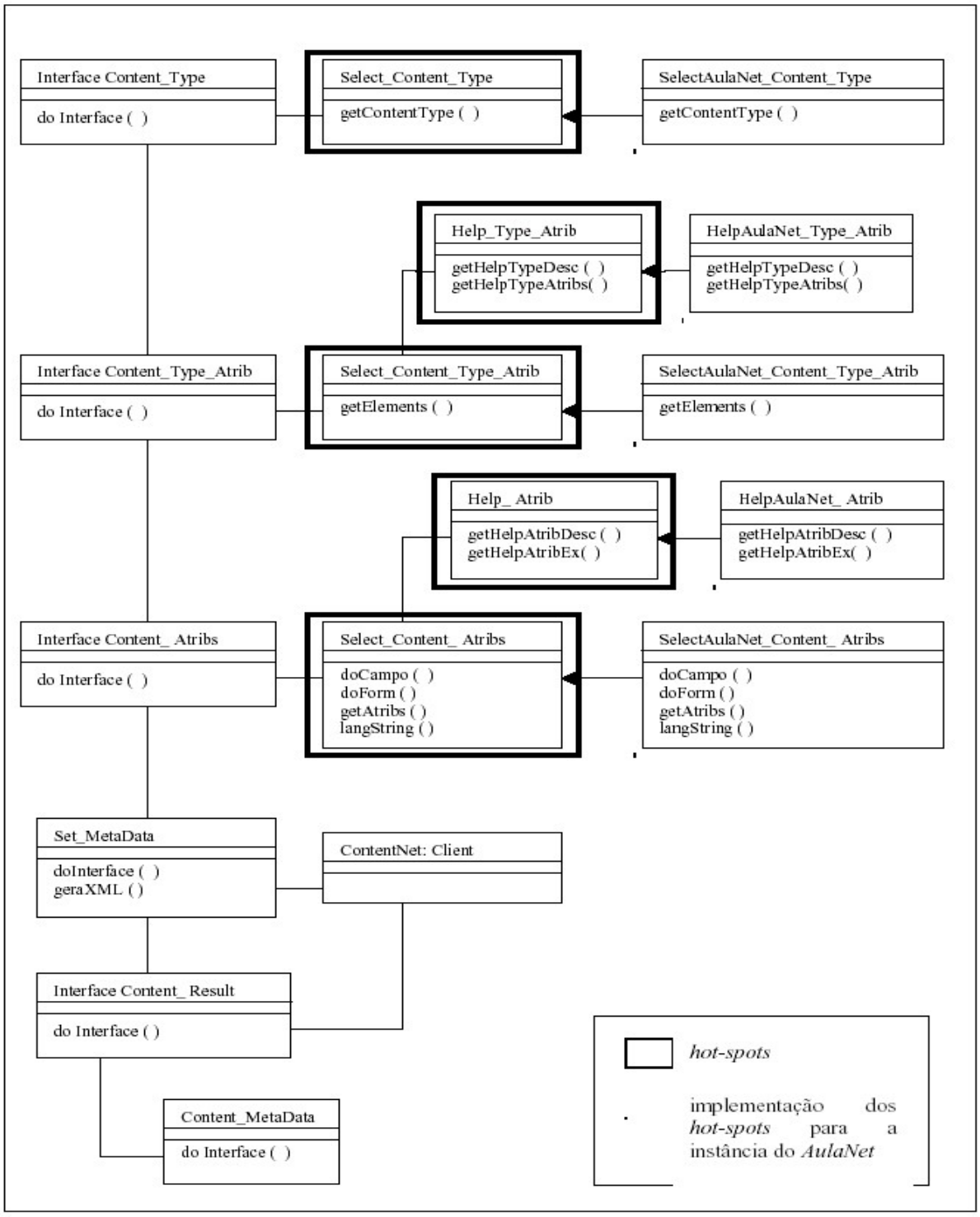

Figura 11 – Diagrama de classes do *framework* para interface de Busca.

Fonte: Lucena, 2000. [11]

Este *framework*, ilustrado na figura 11, tem como objetivo implementar uma interface para Busca através da cooperação entre suas classes. As classes que estão destacadas são os já citados *Hot Spots* do *framework*, e serão estas as classes que poderão ser implementadas de acordo com a necessidade da aplicação específica.

As outras classes são os *Frozen Spots*, e são estas classes que ditam qual será o fluxo de dados no *framework* e qual será seu comportamento.

A primeira classe, a *Interface Content Type*, será executada, e, a depender da resposta que receba, poderá fazer uma chamada a classe *Interface Content Type Atrib* ou a classe *Select Content Type*. Este comportamento é o mesmo para todas as outras classes.

- *Framework* para interface de Catalogação

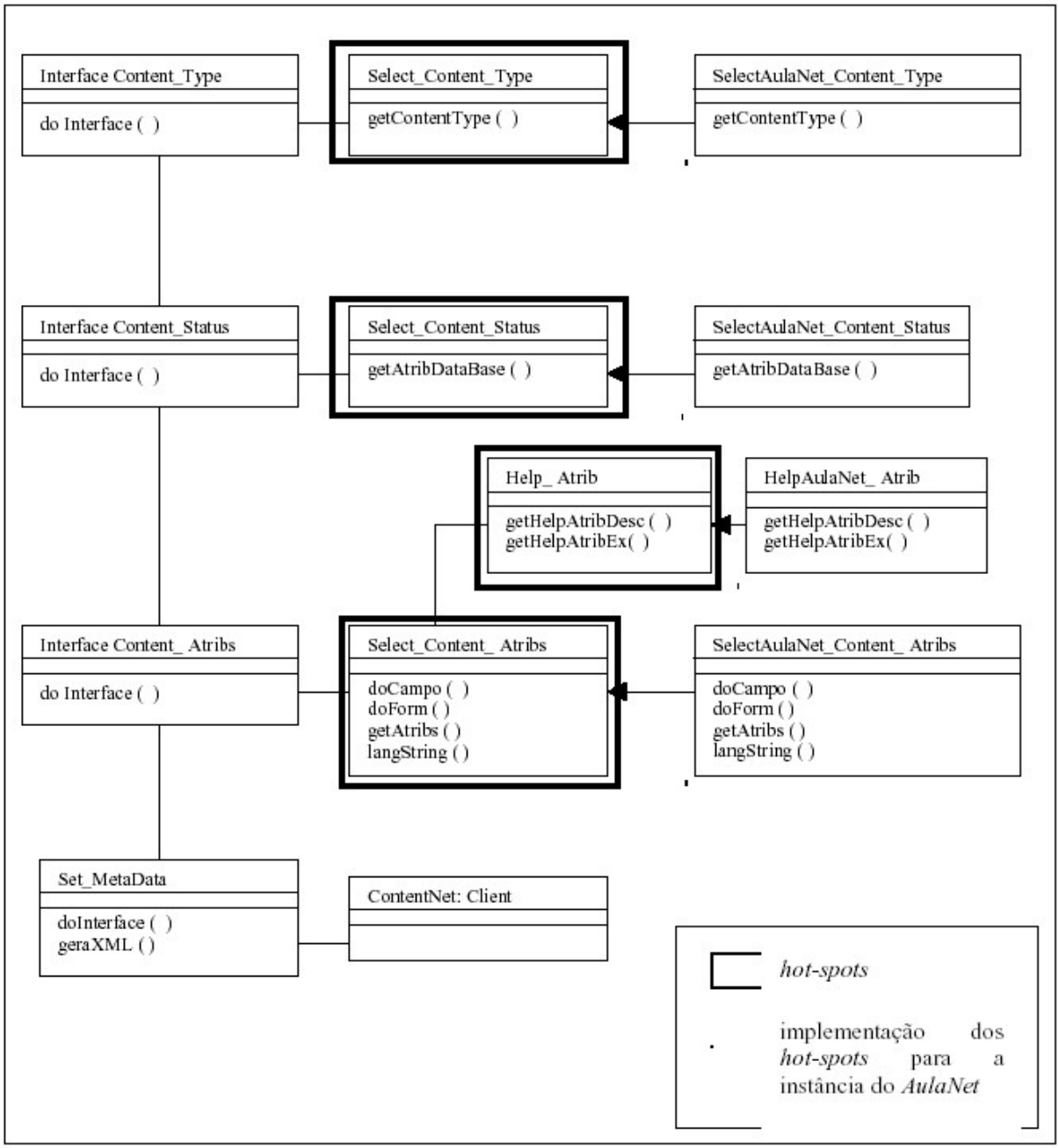

Figura 12 – Diagrama de classes do *framework* para Catalogação.

Fonte: Lucena, 2000. [11]

O *framework* ilustrado na figura 12 segue os mesmos princípios do *framework* já explicado anteriormente, ilustrado na figura 11, ou seja, suas classes seguem uma ordem, controlando o fluxo da execução da aplicação. O resultado da execução de uma classe serve como entrada para a execução de uma nova tarefa em outra classe. Seu objetivo é facilitar o sistema de catalogação pelo Catalogador responsável.

9. Conclusão Final

Reusar código não é uma tarefa fácil, mas é importante para a economia de tempo e linhas de código. Para que o reuso seja utilizado de uma maneira mais proveitosa, surge o desenvolvimento dos *frameworks*, que permitem que não só uma pequena parte do código possa ser reaproveitada, mas todo ele.

Apesar das inúmeras vantagens na utilização de *frameworks*, também é importante citar suas desvantagens, como o custo para seu desenvolvimento e treinamento, sua depuração e correção de erros e sua documentação – esta última deve ser muito bem elaborada para facilitar a utilização e o aprendizado do *framework*. Porém, todas estas desvantagens são superadas pelas vantagens, já que, uma vez que o *framework* se torne familiar, muitos projetos poderão ser desenvolvidos baseados nele, com um bom ganho de performance na equipe desenvolvedora.

O reuso baseado em *frameworks*, quando comparados a outras técnicas de reuso, a exemplo dos *Patterns* e biblioteca de classes, se mostra mais eficiente, pois os fundamentos de um *framework* são bem mais específicos para seu domínio que os *Patterns* (que são estruturas mais genéricas, não tão específicas quanto o *framework*) e que as bibliotecas de classes (onde as classes são reutilizadas, mas com todo o controle da aplicação nas mãos do desenvolvedor, o que ocasiona uma maior propensão a erros que no uso de *frameworks)*.

O desenvolvedor de *frameworks* tem papel fundamental no desempenho deste, podendo torna-lo mais ou menos intuitivo, com muita ou pouca capacidade de extensão, entre outras características importantes. Por este motivo, antes da criação do *framework* deve-se fazer um estudo mais aprofundado do domínio de aplicação que será beneficiado pelo *framework*, quais características são importantes para este domínio, quais partes poderão ser consideradas *Hot Spots* e quais serão os *Frozen Spots*, entre outras características.

Outro personagem importante para a utilização e o desenvolvimento de *frameworks* é o seu usuário. Como o *framework* está direcionado para uma boa utilização por parte do usuário, é importante que os requisitos de usabilidade (adoção de nomes significativos, nomes relacionados com a realidade, etc) sejam seguidos para que um número maior de pessoas possa se beneficiar do *framework*.

Referências Bibliográficas

[1] SILVA, Viviane Torres da. *Frameworks.* Disponível em: http://gft.ucp.br/staff/tavares/topicosII/aula*Framework*.ppt . Data de Acesso:16/10/2002.

[2] SAUVÊ, Jacques Philippe. *FRAMEWORKS*. Paraíba: Universidade Federal da Paraíba, 1999.5 p. (Notas de Aula). Disponível em: http://jacques.dsc.ufpb.br/cursos/map/html/frame/oque.htm . Data de Acesso:12/10/2002.

[3] FAYAD, Mohamed C.; SCHMIDT, Douglas C.; JOHNSON, Ralph E. *Building Application Frameworks – Object-Oriented foundations of frameworks design*. Estados Unidos: Wiley, 1999.

[4] GUIMARÃES, José de Oliveira. *Frameworks* São Paulo, nov. 2000. Disponível em: http://www.dc.ufscar.br/~jose/courses/oc/apostilas-patterns.zip . Data de Acesso: 16/10/2002.

[5] SILVA, Ricardo Pereira e. *Suporte ao desenvolvimento e uso de frameworks e componentes.* Porto Alegre: Universidade Federal do Rio Grande do Sul, 2000. 262 p. (Tese, Doutorado em Ciência da Computação).Disponível em: http://www.inf.ufsc.br/~ricardo/download/tese.pdf Data de Acesso: 19/10/2002.

[6] MALDONADO, José Carlos et al. *Padrões e Frameworks de software.* Local: São Paulo: Universidade de São Paulo, ano de apresentação não disponível. 37 p. (Notas didáticas). Disponível em: http://www.icmc.sc.usp.br/~rtvb/apostila.pdf . Data de acesso: 28/10/2002.

[7] FOWLER, Martin. Avoiding Repetition. *IEEE SOFTWARE*, Califórnia, n. 1,p. 97 –99, jan/fev. 2001

[8] CALDER, Paul; WINN, Tiffany. Is This a Pattern? . *IEEE SOFTWARE*, Califórnia, n. 1, p 59 - 65, jan/fev. 2002.

[9] TAVARES, Carlos. *Design Patterns.*Petrópolis: Universidade Católica de Petrópolis, 2002. 30 p. (Notas de aula). Disponível em: http://www.inf.ucp.br/profs/tavares/01_2002/aulaDP-1.ppt . Data de Acesso: 30/10/2002.

[10] CENTRO DE COMPUTAÇÃO DA UNIVERSIDADE ESTADUAL DE CAMPINAS (CCUEC). *Orientação a Objetos da Teoria à prática em Java.* Campinas, 1999. Disponível em: http://ftp.unicamp.br/pub/apoio/treinamentos/OO/oops.pdf Data de Acesso: 12/10/2002.

[11] LUCENA, Carlos José Pereira de; SILVEIRA,Milene Selbach. *Frameworks para Interfaces de Busca e Catalogação de Conteúdo do ContentNet.* Março 2000. Disponível

em: http://www.inf.puc-rio.br/~milene/publicacoes/mcc13_00.pdf Data de Acesso: 10/10/2002.