A busca de generalidade, flexibilidade e extensibilidade no processo de desenvolvimento de frameworks orientados a objetos

Ricardo Pereira e Silva Eng., M.Sc.* Roberto Tom Price Eng., M.Sc., D.Phil

Universidade Federal do Rio Grande do Sul - Instituto de informática
Caixa postal 15064 - Porto Alegre - RS - Brasil

* Universidade Federal de Santa Catarina - Depto. de Informática e de Estatística
Caixa Postal 476 - Florianópolis - SC - Brasil

e-mail: {ricardo, tomprice}@inf.ufrgs.br

Abstract

Features of the framework development process are discussed, focusing on the importance of flexibility and extensibility of the design. The process of class structuring is analysed, regarding the generalization of domain concepts, and the use of design patterns and metapatterns in the development and in the refinement of frameworks. A board-game framework is used as an example to illustrate the steps followed to generate a framework class structure.

Keywords: software reuse; object-oriented software development methodologies; object-oriented frameworks; design patterns, metapatterns.

1 - Introdução

Um framework orientado a objetos é uma estrutura de classes interrelacionadas, que constitui uma implementação inacabada, para um conjunto de aplicações de um domínio. Além de permitir a reutilização de um conjunto de classes, um framework minimiza o esforço de desenvolvimento de aplicações, por portar a definição da arquitetura das aplicações geradas a partir dele, como também por ter predefinido o fluxo de controle destas aplicações [15]. A figura abaixo ilustra uma aplicação desenvolvida a partir de um framework. A parte sombreada corresponde ao framework - uma estrutura de classes que é reutilizada - e o restante é o que é produzido pelo usuário do framework ao desenvolver uma aplicação específica.

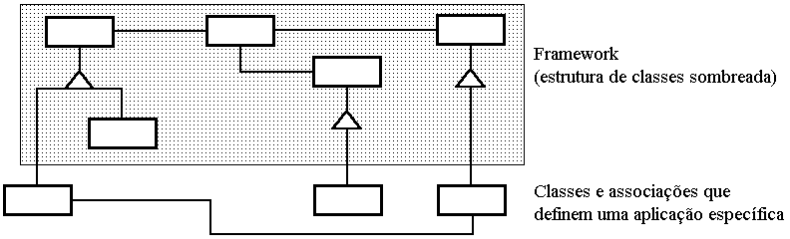

Figura 1 - aplicação desenvolvida utilizando um framework

Existem atualmente poucas metodologias de desenvolvimento de frameworks. A metodologia "Projeto dirigido por exemplo" (*Example-driven design*) [6], é caracterizada pela ênfase à análise do maior número possível de aplicações já desenvolvidas

do domínio tratado, para a obtenção de conhecimento do domínio. Estabelece a seguinte seqüência de etapas para a produção de um framework:

1 - Análise do domínio (as aplicações existentes são a principal fonte de informações);

2 - Definição de uma hierarquia de classes que possa ser especializada para abranger as aplicações analisadas na primeira etapa (um framework);

3 - Teste do framework através do seu uso no desenvolvimento de exemplos (implementar, testar e avaliar cada exemplo usado na primeira etapa, utilizando para isto, o framework desenvolvido).

A metodologia, "projeto dirigido por pontos de flexibilização" (*hot spot driven design*) [10], enfatiza a busca de *hot spots. Hot spots* são as partes da estrutura de classes do framework (classes, métodos) que devem ser mantidas flexíveis, para possibilitar sua adaptação a diferentes aplicações do domínio. A metodologia estabelece basicamente a mesma seqüência de etapas da metodologia "projeto dirigido por exemplo", porém sua ênfase é a identificação das partes da estrutura que variam em diferentes aplicações, e a seleção de soluções de projeto adequadas a estes casos.

As metodologias de desenvolvimento de frameworks existentes estabelecem o processo em linhas gerais, sem se ater à definição de técnicas de modelagem ou detalhar o processo de desenvolvimento. Este trabalho é voltado a aprofundar a discussão em torno do processo de desenvolvimento[1]. Inicialmente é discutida a necessidade de buscar generalidade, flexibilidade e extensibilidade[2] durante o processo de desenvolvimento de um framework, e é apresentada a abordagem de padrões (*patterns*), que contribui para que se atinjam estas metas. É descrita a estrutura do framework de jogos de tabuleiro desenvolvido, FraG, e a seguir são apresentadas algumas situações do projeto deste framework, que ilustram os passos do processo de desenvolvimento de frameworks.

2 - Prioridades no processo de desenvolvimento de frameworks

Um framework é uma abstração de um domínio de aplicações, a ser especializada em aplicações deste domínio. A principal característica buscada ao desenvolver um framework é a generalidade em relação a conceitos e funcionalidades do domínio tratado[3]. Além disto, é fundamental que a estrutura produzida apresente as características flexibilidade e extensibilidade.

A flexibilidade reflete a capacidade do framework de alterar suas funcionalidades, em função da necessidade de uma aplicação específica, o que é operacionalizado através de uma identificação adequada das partes da estrutura que diferem em aplicações distintas do mesmo domínio - os *hot spots*, usando a nomenclatura de Pree [10]. Isto equivale a uma diferenciação entre os conceitos gerais do domínio, e os conceitos específicos das aplicações. Uma vez obtida esta diferenciação, é necessária a seleção de soluções de projeto adequadas

---

1 A discussão em torno da documentação de frameworks é tema de outros artigos [8] [13].

2 Flexibilidade se refere à capacidade alterar a funcionalidade presente, sem conseqüências imprevistas sobre o conjunto da estrutura; e extensibilidade se refere à capacidade ampliar a funcionalidade presente, sem conseqüências imprevistas sobre o conjunto da estrutura [2]. Apesar de se ter usado a terminologia de Fayad e Cline, esta questão já foi tratada anteriormente por Meyer [7] e por outros autores.

3 Capaz de constituir uma abstração geral do domínio tratado, que possa ser especializada com o mínimo esforço possível do usuário do framework para a obtenção de aplicações específicas - maximizando assim, a reutilização de software.

para modelagem dos *hot spots*, de modo que a estrutura de classes resultante apresente a flexibilidade requerida.

A questão da extensibilidade se refere à manutenibilidade do framework. Um framework possui uma estrutura de classes mais complexa que a estrutura de uma aplicação do seu domínio. Esta estrutura é construída em ciclos iterativos. A evolução da estrutura do framework se estende por toda a sua vida útil, pois à medida em que é utilizado, novas funcionalidades podem ser agregadas. Assim, na definição de abstrações, deve-se ter a preocupação de prever futuras utilizações para o framework, inclusive a possibilidade de estender os limites do domínio tratado.

Para construir um framework capaz de se adaptar a um conjunto de aplicações diferentes, é fundamental que se disponha de modelagens de um conjunto significativo de aplicações do domínio. Este conjunto pode se referir a aplicações previamente desenvolvidas, como preconiza a metodologia "Projeto dirigido por exemplo"[4] [6], ou a aplicações que se deseja produzir a partir do framework. A ótica de diferentes aplicações é o que dá ao desenvolvedor do framework a capacidade de diferenciar conceitos gerais de conceitos específicos. Em termos práticos, dotar um framework de generalidade, flexibilidade e extensibilidade requer uma cuidadosa identificação de *hot spots*, e a seleção de soluções de projeto de modo a produzir uma arquitetura bem estruturada.

A pesquisa sobre identificação de *padrões* fornece estruturas de projeto semiprontas que podem ser reutilizadas, contribuindo no desenvolvimento de frameworks, na obtençao de uma organização de classes com as caracerísticas de flexibilidade e extensibilidade. *Padrões* contribuem com o uso adequado de herança e com o desenvolvimento de projetos em que o acoplamento entre classes é minimizado [10] [6].

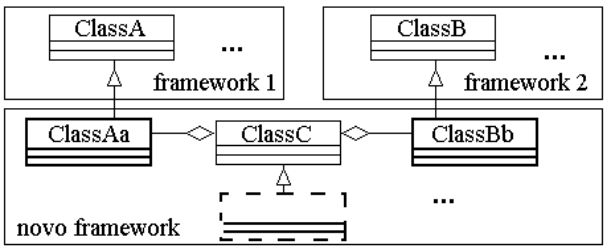

Figura 2 - combinação de frameworks

O processo de desenvolvimento de frameworks pode envolver a tarefa de combinar frameworks para uso conjunto. O framework de jogos de tabuleiro - FraG - usado como exemplo neste trabalho, utiliza o framework MVC [9] (vide figura 3). Neste caso FraG foi desenvolvido de modo a respeitar as restrições de MVC. Uma outra situação seria a utilização de mais de um framework - por exemplo, MVC e um framework para sonorização, suportando um framework de jogos. Neste caso, além de respeito às restrições dos frameworks reutilizados, o framework de jogos teria a responsabilidade de compatibilizar os protocolos de controle dos outros dois frameworks. A figura 2 ilustra o uso de composição de objetos para a combinação de objetos responsáveis pela lógica de controle.

O desenvolvimento de frameworks pode se valer do uso de componentes - estruturas de software desenvolvidas segundo a abordagem *component-oriented programming* [14]. Componentes podem ser utilizados como objetos acoplados ao framework, sem relação com sua hierarquia de herança (por agregação). A compatibilização do framework com a interface do componente usado, é responsabilidade do desenvolvedor do framework. Funcionalidades como a sonorização de uma aplicação, por exemplo, poderiam ser desempenhadas por

---

4 Para Johnson o desenvolvimento de um framework deve ser posterior ao desenvolvimento de aplicações do respectivo domínio - pelo menos três aplicações, que sejam representativas em relação ao conjunto de aplicações do domínio [6].

componentes. Em contrapartida, frameworks, podem ser usados para construir componentes. Neste caso, o usuário do framework (componente) não teria acesso à sua estrutura de classes (uso como caixa-preta), mas apenas a uma interface - para parametrização e acesso às funcionalidades. Por outro lado, os desenvolvedores de componentes podem se valer da flexibilidade e extensibilidade dos frameworks para maximizar a reutilização de software, no desenvolvimento de diferentes componentes.

3 - Padrões

Padrões (*Patterns*) constituem uma abordagem bastante recente em termos de reutilização de projeto, no contexto do desenvolvimento de software orientado a objetos. A principal questão tratada é como proceder para registrar - para poder reutilizar - a experiência de projeto de software adquirida ao longo dos desenvolvimentos. As abordagens "padrões de projeto" (*design patterns*) [3] e "metapadrões" (*metapatterns*) [10] constituem as contribuições mais relevantes, em termos de geração de subsídios para o desenvolvimento de frameworks.

Com base em suas experiências em identificação de padrões em estruturas de classes, Gamma, Helm, Johnson e Vlissides elaboraram um catálogo com vinte e três padrões de projeto, de uso geral [3]. Cada padrão de projeto descreve uma situação de projeto e a solução proposta, ou seja, a definição de uma microarquitetura[5] (organização de classes) que se aplica ao problema, na qual são definidas as classes envolvidas, suas responsabilidades e a forma como cooperam.

Pree [10] propôs metapadrões (*metapatterns*), como complemento ao enfoque do catálogo de padrões de projeto. Os metapadrões também são soluções de projeto de uso geral, porém, menos específicas, se comparadas aos padrões de projeto. A ênfase é a alocação de métodos *template* e *hook*[6] nas classes, com vista a obter determinadas soluções, em termos de flexibilidade da implementação.

Comparando as duas abordagens, observa-se que os metapadrões são mais abstratos. Os padrões de projeto, apesar de constituirem soluções de projeto de uso geral, são direcionados à solução de determinados tipos de problema. Os metapadrões não são voltados a solucionar problemas de projeto específicos, mas à definição de estruturas de implementação flexíveis.

Os padrões de projeto apresentam estruturas de maior granularidade. São constituídos por estruturas de classes, com métodos e atributos definidos. Os metapadrões apenas relacionam dois métodos (template e hook), que podem ser de uma mesma classe ou de classes distintas (relacionadas ou não por herança).

Os padrões de projeto podem ser usados na definição da estrutura de classes do framework, ou para refinar a estrutura de classes ao longo do projeto [3]. Usar padrões de projeto demanda conhecê-los - não apenas os do catálogo, que são de uso geral, mas também padrões de domínios específicos à medida em que se tornam disponíveis e padrões arquitetônicos, como MVC.

---

5 A expressão microarquitetura significa uma organização de classes, abrangendo um subconjunto das classes do projeto global. O termo contrasta com a expressão arquitetura, que se refere à organização de todo o conjunto de classes.

6 Método template, é um método que demanda a cooperação de outros métodos (métodos hook) para completar seu processamento [10].

Metapadrões podem ser usados sempre que se identificar a necessidade de manter a implementação de um algoritmo flexível. Podem inclusive complementar a abordagem de padrões de projeto, a partir da flexibilização da implementação de padrões de projeto selecionados. Assim como no caso dos padrões de projeto, o uso de metapadrões pode ocorrer tanto nas fases preliminares de definição da estrutura de classes, como no refinamento da estrutura.

A utilização de padrões em desenvolvimento de projeto está relacionada à busca de generalidade dos conceitos do domínio. A identificação de um conceito geral leva freqüentemente à identificação de variações em torno deste conceito, verificadas em diferentes aplicações. Isto equivale a identificar *hot spots* na estrutura de classes, o que demanda soluções de projeto adequadas aos problemas de projeto verificados.

4 - FraG, um framework para jogos de tabuleiro

Está sendo desenvolvido um framework para a geração de jogos de tabuleiro, FraG, construído como uma ferramenta didática, para o ensino de aspectos referentes a desenvolvimento e utilização de frameworks. FraG é voltado a generalizar um conjunto de jogos que tem em comum o uso de tabuleiro, podendo também utilizar dados, peões e outros elementos, caracterizando um domínio de aplicações[7]. A atual versão foi implementada em Smalltalk, sob o framework MVC, que dá suporte aos aspectos de geração de interfaces gráficas, bem como à interação do usuário com a aplicação através destas interfaces.

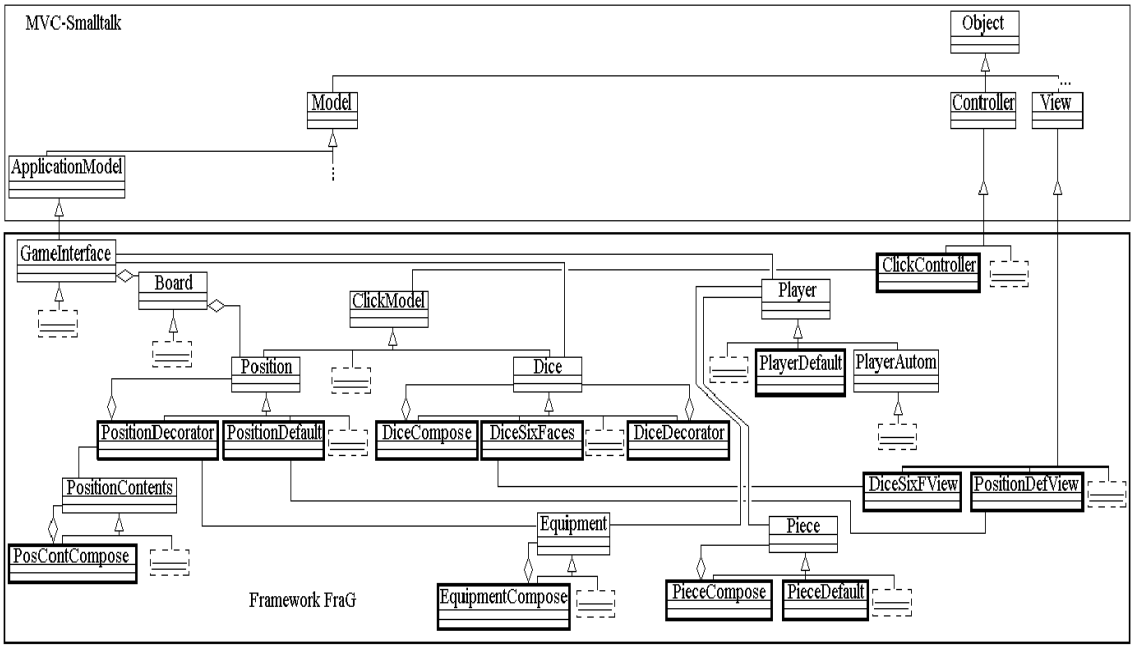

Figura 3 - modelo de objetos do framework FraG

A figura 3 apresenta um modelo de objetos que contém as principais classes do framework de jogos de tabuleiro[8]. A figura 4 apresenta a estrutura de classes de um exemplo

---

7 Exemplos de jogos: Jogo da Velha, Dama, Ludo Real, Gamão, Banco Imobiliário, RPG etc.

8 Foram omitidas as relações de atributos e métodos, e informações relativas às associações (tipo, cardinalidade etc.). Classes do framework sem superclasse explicitada, são subclasses de Model. Os nomes de algumas classes se referem a elementos do domínio: Dice (dado), Board (tabuleiro), Player (jogador), Piece (peão), Position (casa).

de aplicação desenvolvida sob o framework - um Jogo da Velha, que demanda a criação de apenas duas classes. A notação usada [12] diferencia classes abstratas[9] (contorno com traço fino), classes concretas (contorno com traço gosso) e classes a serem criadas pelo usuário do framework na geração de aplicações (contorno tracejado).

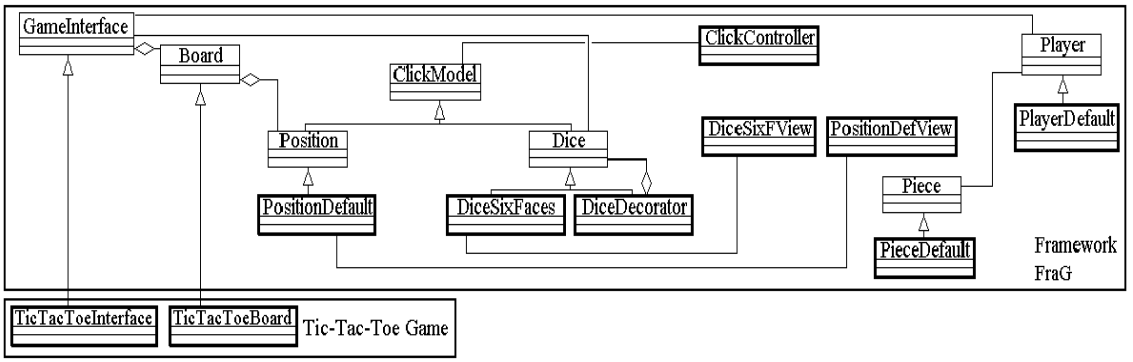

Figura 4 - modelo de objetos do Jogo da Velha, desenvolvido sob o framework FraG

5 - As etapas do processo de desenvolvimento de frameworks

A construção de um framework passa por um conjunto de atividades, não-seqüenciais, e que se repetem até a obtenção de uma estrutura de classes que satisfaça os requisitos estabelecidos de generalidade, flexibilidade e extensibilidade. Para maior clareza, pode-se dividir o processo de desenvolvimento da estrutura de classes de um framework nas seguintes etapas:

- etapa de generalização;
- etapa de flexibilização;
- etapa de aplicação de metapadrões;
- etapa de aplicação de padrões de projeto;
- etapa de aplicação de princípios práticos de orientação a objetos.

A seguir são apresentadas cada uma destas etapas, com a preocupação de ilustrá-las através de situações de projeto enfrentadas ao longo do desenvolvimento do framework FraG.

5.1 - Etapa de generalização

A etapa de generalização consiste na identificação de estruturas idênticas nas aplicações analisadas, e na fatoração de estruturas em que se identificam semelhanças, de modo a obter uma estrutura de classes que generalize o domínio tratado. Um primeiro aspecto na busca de generalidade da estrutura do framework, é a identificação das classes que representam elementos e conceitos do domínio tratado. Isto é obtido a partir da confrontação de estruturas de classes de diferentes aplicações (implementadas ou não). Nesta etapa também são buscados elementos prontos que podem reutilizados, como outros frameworks e componentes. No caso do framework FraG, foi utilizado o framework MVC (Smalltalk). No início do seu desenvolvimento havia disponível a implementação de um jogo ("Corrida") em Smalltalk, e o respectivo projeto, cuja estrutura de classes está ilustrada na figura 5 [11].

---

9 Consideram-se classes abstradas as classes que possuem pelo menos um método abstrato (com a assinatura definida, mas sem implementação) [5].

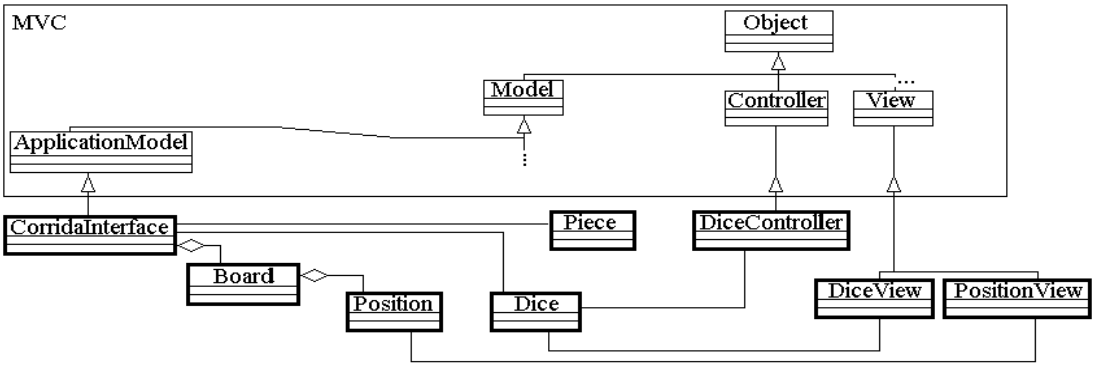

Figura 5 - modelo de objetos do jogo "Corrida"

O confronto da estrutura de classes do jogo "Corrida", com estruturas de classes desenvolvidas para outros jogos[10], levaram:

- ao reconhecimento das classes Board, Position e Dice, como gerais para o domínio, e
- à necessidade de definir uma classe "Player" para o framework, originada da fatoração da classe Piece (do jogo "Corrida") em duas classes: Piece e Player. No jogo Corrida a figura do jogador se confunde com a figura de seu peão. Em jogos com vários peões por jogador esta abstração não é adequada.

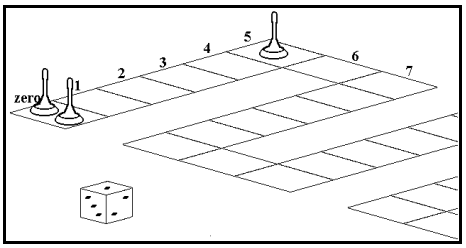

Figura 6 - ilustração do jogo "Corrida"

5.2 - Etapa de flexibilização

Nesta etapa o objetivo é identificar o que deve ser flexibilizado na estrutura de classes que generaliza o domínio, de modo que a estrutura possa ser especializada, gerando diferentes aplicações. É um processo de localização de *hot spots* na estrutura de classes e de escolha de soluções de projeto para modelá-los. A identificação de *hot spots* ocorre quando se determinam situações de processamento comuns às aplicações do domínio, porém, com variações de uma aplicação específica para outra. Considerando o exemplo do framework FraG, a ação do usuário sobre os elementos do jogo para o procedimento de lances, é uma situação de processamento comum às aplicações do domínio. Durante um lance de uma partida, a ação do usuário pode se dar sobre dados (lançamento), casas (colocar, remover peões) ou outros elementos que venham a ser definidos - considerando inclusive a possibilidade de ocorrência de várias ações a cada lance. No jogo inicialmente implementado (Corrida), a única ação do jogador era o lançamento do dado - e por isso a presença da classe DiceController na estrutura de classes deste jogo[11]. Em um Jogo da Velha, por exemplo, a ação do usuário ocorre sobre as casas - o que demandaria a associação de objetos controladores às casas. Generalizando o tratamento das ações do usuário no framework, foram definidas as classes ClickModel (abstrata) e ClickControler, subclasses de Model e Controller, respectivamente. ClickController é responsável por verificar a ação do mouse, e ClickModel, por conter o protocolo de resposta a esta ação. Este protocolo é

---

10 Desenvolvidas apenas modelagens, sem que fossem produzidas as respectivas implementações.

11 Cada elemento gráfico que deve responder à ação do usuário, necessita de um controlador associado (instância de subclasse da classe Controller). Esta restrição se deve ao uso do framework MVC - para suportar a interface gráfica dos jogos.

herdado pelas classes Dice e Position, do framework - o que ilustra uma situação de uso adequado de herança para reutilização de interfaces. As figuras abaixo ilustram esta evolução da estrutura de classes. Observe-se que esta solução é polimórfica: instâncias de ClickController podem interagir com objetos de classes diferentes.

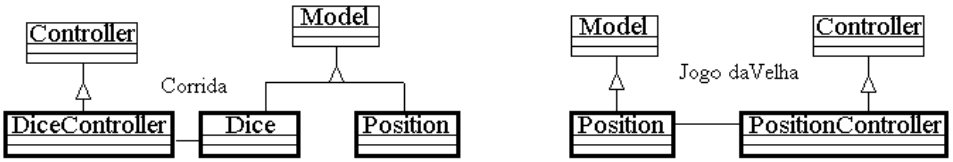

Figura 7 - detalhe das estruturas de classes do Jogo Corrida e do Jogo da Velha

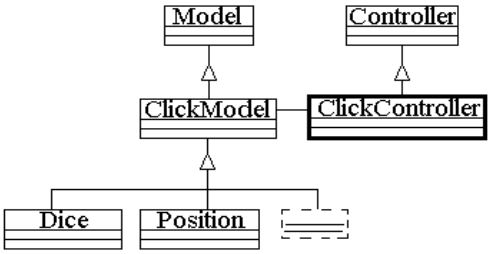

Figura 8 - a generalização do controle de ações do usuário, no framework FraG

5.3 - Etapa de aplicação de metapadrões

Uma vez identificado um *hot spot*, deve-se comparar o requisito de flexibilização por ele imposto com os casos tratados por padrões existentes. Se um padrão é julgado adequado para a solução do problema, ele deve ser incorporado à estrutura de classes do framework. No caso específico dos metapadrões, o uso de um metapadrão consiste em transformar um procedimento geral em um método *template*, cujo comportamento é flexibilizado através da dependência de métodos *hook*, que podem ser trocados. No exemplo apresentado, observa-se que inicializar um jogo é um procedimento comum às aplicações do domínio, porém cada aplicação tem aspectos particulares. Um método *template* pode ser usado para generalizar a estrutura do algoritmo de inicialização, deixando as particularidades de cada caso sob a responsabilidade de métodos *hook*. Isto caracteriza uma situação em que é possível usar um metapadrão. A figura 9 ilustra quatro ocorrências do metapadrão *unificação*, envolvendo o método "initialize" da classe abstrata GameInterface, do framework FraG. Este método é responsável por promover a instanciação e inicialização dos objetos, necessários no início da execução da aplicação. No caso do domínio tratado, o processo de inicialização de um jogo segue o seguinte algoritmo:

- produzir um tabuleiro específico (com a geração de casas, jogadores, peões e dados compatíveis com este tabuleiro);
- proceder a inicialização de atributos dos objetos;
- definir a ordem de procedimento de lances dos jogadores e
- exibir estado inicial da partida e aguardar ação do usuário.

Cada um dos passos que compõem o procedimento acima, que define o método "initialize" (um método *template*), é implementado na forma de um método *hook* (respectivamente, "generateBoard", "initializeObjects", "first:" e "enableStart").

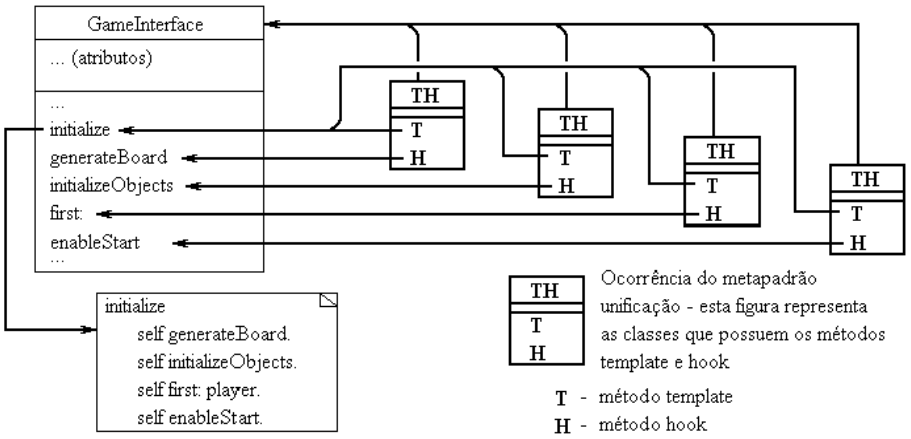

Figura 9 - destaque ao uso do metapadrão unificação

5.4 - Etapa de aplicação de padrões de projeto

A incorporação de padrão de projeto consiste em incluir as classes do padrão na estrutura de classes em desenvolvimento (ou fazer com que classes já existentes assumam as responsabilidades correspondentes às classes do padrão de projeto). A implementação do método "generateBoard" do framework FraG (citado no item anterior) adota o padrão de projeto "Abstract Factory". Em conseqüência disto, a única responsabilidade do método, que deve ser definido na subclasse concreta de GameInterface (específica da aplicação), é chamar o método construtor da classe correspondente ao tabuleiro da aplicação (subclasse concreta de Board). A figura 10 ilustra a aplicação do padrão de projeto "Abstract Factory" no framework. Esta solução produz acoplamento apenas da subclasse de GameInterface a uma subclasse de Board, e não o contrário. Assim, um tabuleiro pode ser reutilizado com diferentes interfaces.

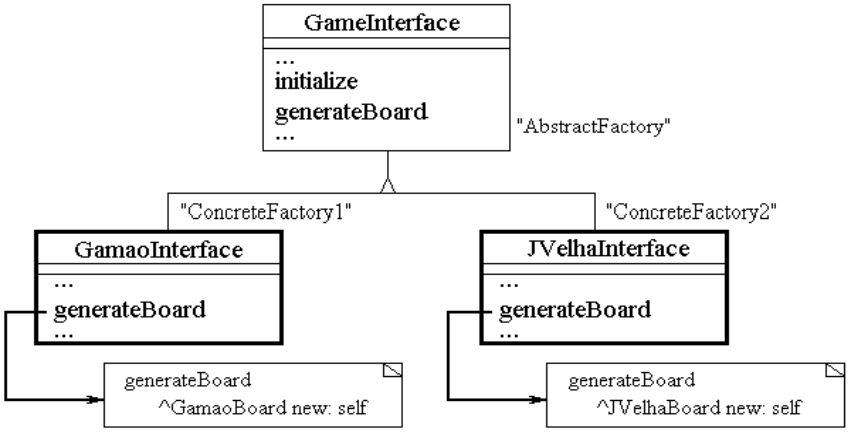

Figura 10 - destaque ao uso do padrão de projeto "Abstract Factory"

O padrão de projeto "Observer" foi usado para fazer com que o objeto tabuleiro[12] seja afetado pela ação do usário: o objeto tabuleiro é o observador, e as instâncias de subclasses de ClickModel, os sujeitos. Quando um dos sujeitos é submetido a uma ação do usuário, o observador é notificado, devendo produzir uma resposta adequada. A aplicação do padrão de projeto "Observer" na definição da estrutura do framework desenvolvido,

12 Instância de subclasse concreta de Board.

possibilitou que fosse deixada para o usuário do framework apenas a tarefa de implementar os métodos que produzem a resposta do tabuleiro - todo o procedimento de interação com o mouse é reutilizado. Além disto, há um baixo acoplamento entre observador e sujeitos. As classes de observadores não guardam referência dos sujeitos, e a referência do observador mantida pelos sujeitos independe da classe do observador - assim, casas e dados podem ser reutilizados em diferentes tabuleiros. A figura 11 ilustra o uso do padrão de projeto "Observer" no framework.

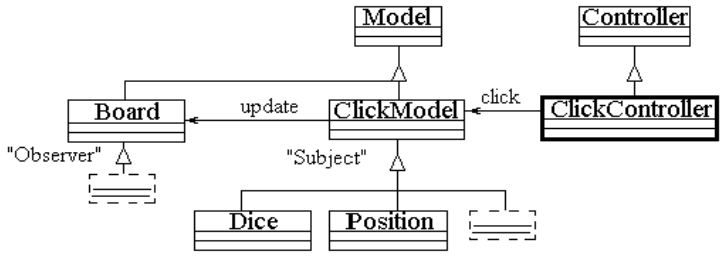

Figura 11 - destaque ao uso do padrão de projeto "Observer"

Adotou-se o padrão de projeto "Decorator" para estender a funcionalidade da classe Position, permitindo a uma casa executar ações sobre os peões que a ocupam[13]. A ação específica é responsabilidade de um objeto *conteúdo*, associado à casa através do objeto decorador[14]. Com isto, as subclasses de Position se tornam menos específicas - e, portando, mais adequadas à reutilização - pois, as mesmas casas podem ser usadas com ou sem ações associadas. O uso do padrão "Decorator" associado à classe Position está descrito na figura 12. Neste caso, o uso do padrão permitiu que a funcionalidade de Position fosse estendida através de composição de objetos.

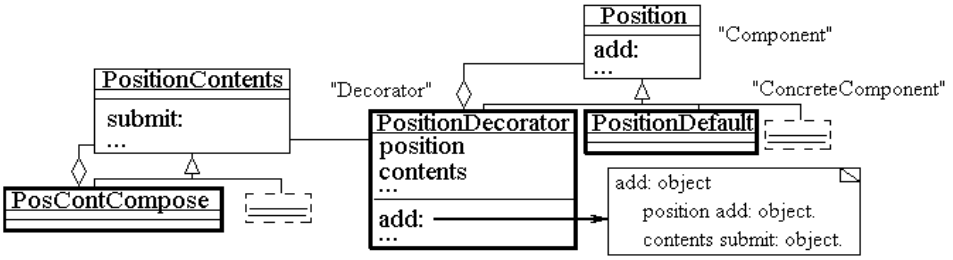

Figura 12 - destaque ao uso do padrão de projeto "Decorator"

O uso de padrões pode ocorrer durante o desenvolvimento ou durante o refinamento de uma estrutura de classes. A figura 13 apresenta o uso do padrão de projeto "Decorator" associado à classe Dice - de forma bastante semelhante ao uso com a classe Position. A diferença é que no caso da classe Dice, a necessidade do padrão foi verificada quando, ao desenvolver uma aplicação, foi necessário bloquear a ação de um dado de seis faces (para impedir seu lançamento). Isto exigiu o refinamento da estrutura inicialmente definida. Uma alternativa possível seria o uso de herança: gerar uma subclasse de DiceSixFaces com esta funcionalidade. Esta solução foi considerada uso inadequado de herança: a funcionalidade de

---

13 Isto é dispensável em jogos como Jogo da Velha ou Damas, mas é necessário em jogos como Banco Imobiliário e RPG.

14 O objeto conteúdo é uma instância de uma subclasse concreta de PositionContents e é responsável por um tipo de ação específica, e o objeto decorador é uma instância da classe PositionDecorator.

bloqueio estaria sendo forçada sobre uma classe para reutilizar sua implementação. O uso do padrão "Decorator" possibilitou bloquear dados a partir de composição de objetos, sem alteração da classe abstrata Dice, ou das suas subclasses concretas, que implementam dados específicos.

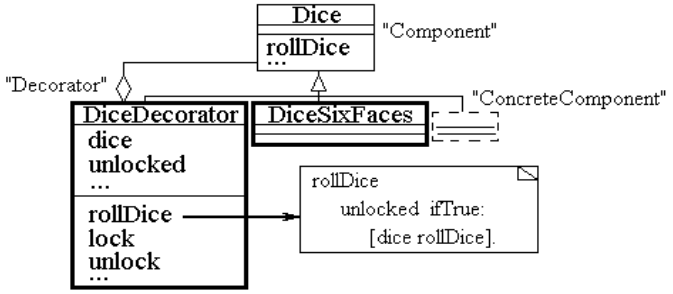

Figura 13 - destaque ao uso do padrão de projeto "Decorator"

5.5 - Etapa de aplicação de princípios práticos de orientação a objetos

Para a obtenção de um framework com uma estrutura de classes flexível e extensível é necessário seguir princípios de projeto orientado a objetos[15], como o uso de herança para reutilização de interfaces, ao invés do uso de herança para reutilização de código (a delegação é mais adequada que o uso indevido de herança); reutilização de código através de composição de objetos; preocupação em promover polimorfismo, na definição das classes e métodos, de modo a possibilitar acoplamento dinâmico, etc. [4] [7]. Nos frameworks o uso de herança tem a finalidade de agrupar as generalidades do domínio em classes abstratas, no topo da hierarquia de classes. Isto promove uso adequado de herança, pois a principal finalidade destas classes abstratas é definir as interfaces a serem herdadas pelas subclasses concretas das aplicações.

O uso do padrão de projeto "Decorator" para possibilitar o bloqueio de um dado no framework FraG (descrito no item 5.4), ilustra uma situação em que a delegação se mostrou uma alternativa mais adequada que herança.

6 - Conclusão

O desenvolvimento de frameworks tem como principal meta produzir uma estrutura de classes capaz de generalizar um domínio de aplicações. Esta característica possibilita a especialização da estrutura para a produção de diferentes aplicações.

Para imprimir o caráter de generalidade à estrutura de classes é necessária a aquisição de conhecimentos do domínio, em que as principais fontes de informação são as modelagens de diferentes aplicações - existentes ou a desenvolver; baseadas ou não em orientação a objetos. O confronto de diferentes estruturas leva ao desenvolvimento de uma estrutura que as generaliza, e à identificação dos *hot spots* - que ressaltam as diferenças e, portanto, as partes do framework que devem ser mantidas flexíveis. O uso de padrões e metapadrões, bem como a observação de princípios de projeto orientado a objetos leva à obtenção de uma organização de classes bem estruturada e mais fácil de ser compreendida, minimizando futuros esforços no uso do framework, bem como na sua extensão.

---

15 O que não é exclusivo do contexto de frameworks, mas também se aplica ao desenvolvimento de aplicações.

Foram descritas e ilustradas com exemplos, as etapas seguidas para a obtenção da estrutura de classes de um framework - etapas de generalização, flexibilização, aplicação de metapadrões e de padrões de projeto e aplicação de princípios práticos de orientação a objetos. Foi mostrado através de exemplos, como as soluções de projeto propostas pela abordagem de padrões podem ser reutilizadas para a obtenção de uma organização de classes dotada de flexibilidade e extensibilidade.

7 - Bibliografia

[1] CAMPO, M. R. **Compreensão visual de frameworks através da introspeção de exemplos**. Porto Alegre: UFRGS/II/CPGCC, mar. 1997. Tese de Doutorado.

[2] FAYAD, M., CLINE, P. Aspects of software adaptability. **Communications of the ACM**. v.39, n.10. oct. 1996.

[3] GAMMA, E. *et al*. **Design patterns**: elements of reusable object-oriented software. Reading: Addison-Wesley, 1994.

[4] JOHNSON, R. E., FOOTE, B. Designing reusable classes. **Journal of object-oriented programming**. p. 22-35, jun./jul. 1988.

[5] JOHNSON, R. E., RUSSO, V. F. **Reusing object-oriented designs**. Urbana: University of Illinois , 1991. Technical Report UIUCDCS91-1696.

[6] JOHNSON, R. E. **How to design frameworks**. In: Object-Oriented Programming Systems, Languages and Applications Conference - OOPSLA, 1993, Washington. **Proceedings...** Washington.

[7] MEYER, B. **Object-oriented software construction**. Englewood Cliffs: Prentice Hall, 1988.

[8] NOWACK, P. **Frameworks - representations and perspectives**. In: Workshop on Language Suport for Design Patterns and frameworks, jun. 1997, Jyväskylä. **Proceedings**... Jyväskylä.

[9] PARCPLACE. **VisualWorks User's Guide**. ParcPlace Systems Inc.1994.

[10] PREE, W. **Design patterns for object oriented software development**. Reading: Addison-Wesley, 1995.

[11] SILVA, R. P. **Avaliação de metodologias de análise e projeto orientadas a objetos voltadas ao desenvolvimento de aplicações, sob a ótica de sua utilização no desenvolvimento de frameworks orientados a objetos**. Porto Alegre: UFRGS/II/CPGCC, jul. 1996. TI 556.

[12] SILVA, R. P., PRICE, R. T. **Técnicas relacionadas ao desenvolvimento de frameworks**. Porto Alegre: UFRGS/II/CPGCC, dez. 1996. EQ 08.

[13] SILVA, R. P. e, PRICE, R. T. **O uso de técnicas de modelagem no projeto de frameworks orientados a objetos**. In: Proceedings of 26th International Conference of the Argentine Computer Science and Operational Research Society (26th JAIIO) / First Argentine Symposium on Object Orientation (ASOO'97), aug. 1997, Buenos Aires. **Proceedings**... Buenos Aires.

[14] SZYPERSKI, C. **Component-oriented programming: a refined variation on object-oriented programming**. In: European Conference on Object-Oriented Programming - ECOOP, 1996, Linz. **Tutorial Notes** Linz.

[15] TALIGENT. **Leveraging object-oriented frameworks**. Taligent Inc. white paper, 1995.