Um Processo Iterativo de Refatoração de Sistemas OO Utilizando Padrões de Projeto de Software

Leide R. C. Rapeli
Departamento de Computação, Universidade Federal de São Carlos
São Carlos, São Paulo, Brasil
Desenvolvimento de Software, S&V Consultoria e Tecnologia, Rua Dom Pedro II, 1330
São Carlos, São Paulo, Brasil
leide_rachel@yahoo.com.br

e

Rosângela A. D. Penteado
Departamento de Computação, Universidade Federal de São Carlos
São Carlos, São Paulo, Brasil
rosangel@dc.ufscar.br

Abstract

OO systems, when not properly designed, may present low reusability and impair their maintainability. An iterative refactoring process using software design patterns is here proposed for these systems. Is has been devised in a prospective way from seven case studies. The process is composed of three stages: 1) Understand the system; 2) Refactor it using design patterns and 3) Verify if the system functionality is preserved after refactoring. Guidelines specific for each design pattern have been prepared for step 2 and exemplified by *Singleton* pattern, presented. A case study of the process and guidelines application is also included.

Keywords: design patterns, refactoring, maintainability, reuse.

Resumo

Sistemas OO, quando não projetados adequadamente, podem apresentar baixa reusabilidade e comprometer sua manutenibilidade. Um processo iterativo de refatoração usando padrões de projeto de software é aqui proposto para esses sistemas. Ele foi concebido de modo prospectivo a partir de sete estudos de caso. O processo é composto de três etapas: 1) Entender o sistema; 2) Refatorá-lo usando padrões de projeto e 3) Verificar se a funcionalidade do sistema permanece inalterada após a refatoração. Diretrizes específicas de cada padrão de projeto foram preparadas para a Etapa 2 e exemplificadas pelo padrão *Singleton*, apresentado. Um estudo de caso de aplicação do processo e diretrizes é também incluído.

Palavras chave: padrões de projeto, refatoração, manutenibilidade, reuso.

1 INTRODUÇÃO

Sistemas de software quando liberados para serem utilizados pelos usuários finais, devem oferecer facilidade de operação e de evolução. Após um período em uso, pode ocorrer que o sistema não mais atenda totalmente os objetivos e à satisfação do cliente e, assim, deve sofrer correções e/ou modificações. Essas atividades fazem parte da manutenção do software e envolvem não somente correção de erros, mas também adaptação a outras tecnologias, adição de nova funcionalidade ou alterações para que manutenções futuras possam ser facilitadas.

Para que a manutenção seja realizada adequadamente, o software deve ser flexível, prevendo alterações futuras quando de sua construção. Com o objetivo de facilitar o desenvolvimento de sistemas de maior manutenibilidade são utilizadas diversas abordagens e linguagens de programação como, por exemplo, programação orientada a objetos, linguagem Java, estruturação por meio de padrões de projeto, etc. Entretanto, mesmo com o uso dessas técnicas nem sempre as modificações ocorrem de forma fácil, organizada e satisfatória.

Um modo de tornar sistemas orientados a objetos, construídos sem a utilização de padrões de projeto, mais flexíveis, manuteníveis e com melhor estruturação é utilizar técnicas de refatoração [4] juntamente com padrões de projeto[1] [5]. Dessa forma, identificam-se situações nas quais esses padrões se aplicam e fazem-se alterações de acordo com a solução proposta por eles.

Este artigo propõe um processo de refatoração de sistemas orientados a objetos implementados em Java, incluindo padrões de projeto no sistema. Para cada padrão de projeto são apresentadas diretrizes de seu reconhecimento no código-fonte, a funcionalidade do sistema é mantida. O artigo está organizado da seguinte maneira: na seção 2 alguns dos trabalhos relacionados a esse são apresentados. Na seção 3, o processo é apresentado para um padrão de projeto, as diretrizes de refatoração são apresentadas. Um exemplo da aplicação do processo, das diretrizes exibidas na seção anterior e a comparação entre os sistemas existente e refatorado são apresentados na seção 4. As considerações finais encontram-se na seção 5.

2 TRABALHOS RELACIONADOS

Com o objetivo de melhorar a qualidade e o projeto de sistemas existentes utilizando o paradigma orientado a objetos, é proposto o conceito de refatoração [4], que aperfeiçoa a estrutura interna de um sistema sem modificar o seu comportamento externo. As refatorações propostas em [4] são formadas por passos simples, cuja realização de forma cumulativa torna o sistema mais fácil de ser entendido e de manutenção menos dispendiosa. Ou seja, refatorar é um modo disciplinado de limpar o código, modificar e simplificar o projeto interno, minimizando as chances de introdução de defeitos.

Fowler [4] define também uma maneira de refatorar o sistema examinando sua funcionalidade e seu código. De acordo com determinadas características (*code smells*), deve-se decidir se o código deve ser refatorado e qual refatoração deve ser aplicada. O trabalho aqui apresentado é similar ao de Fowler, porém aplica padrões de projeto na refatoração. Dessa forma, é necessário que o engenheiro

[1] No decorrer do texto, os termos padrões de projeto e padrões são utilizados como sinônimos.

de software tenha conhecimento suficiente dos padrões e, a partir da lista de indícios criada, consiga identificá-los no código fonte de forma correta. Por exemplo, o padrão *Singleton* é utilizado para a instanciação única de um objeto. Um engenheiro de software, ao consultar o código, deve procurar por objetos que atendam a essa característica. Quando esses objetos são encontrados, de acordo com o mostrado na lista de indícios, devem ser realizados os procedimentos indicados nas diretrizes de refatoração para o padrão *Singleton.*

Um processo semelhante ao aqui apresentado, desenvolvido para sete padrões de projeto, foi proposto por Cinnéide [2], sendo que a aplicação desses padrões ocorre somente quando a flexibilidade provida por eles é realmente necessária. Uma abordagem de refatoração utilizando padrões de projeto em Java, com a definição regras de inferência e estratégias de refatoração é especificada por Jeon et al. [6]. Há a utilização de predicados na linguagem de programação Prolog para representar o comportamento e a estrutura do sistema.

O trabalho desenvolvido por Kerievsky [7] introduz padrões de projeto em código Java utilizando algumas das refatorações propostas por Fowler [4], porém não abrange a identificação do padrão de projeto no sistema existente.

JIAD (*Java Based Intent Aspects Detector*) [11] é uma ferramenta que examina o código-fonte de sistemas existentes desenvolvidos em linguagem Java e indica quais padrões podem ser aplicados e quais são as classes a serem refatoradas. A tarefa de refatoração não é apoiada por essa ferramenta, sendo que o engenheiro de software deve realizá-la manualmente ou utilizando outro software de apoio.

Tsantalis et al. [13] propõem um método de identificação da aplicabilidade de padrões de projeto em sistemas existentes utilizando a análise de diagramas de classe UML [14]. Essa abordagem é válida para 20 dos 23 padrões de projeto de Gamma et al. [5], mas não detecta padrões após a realização de refatorações no sistema.

Como comentado acima, há vários trabalhos na literatura técnica que propõem métodos e ferramentas para identificação de padrões de e refatoração ([2], [6], [7], [11] e [13]). Nesses trabalhos também foram evidenciados problemas, como complexidade, abordagens que não abrangem todos os padrões de projeto de Gamma [5] ou falta de procedimentos de refatoração, após a identificação do padrão. O diferencial com o processo aqui proposto é a sua completeza, pois permite a identificação de padrões no sistema existente por meio da lista de indícios e fornece as diretrizes de refatoração para os 23 padrões de projeto[5].

A lista de indícios apresenta as situações mais comuns de reconhecimento da aplicabilidade dos padrões. Com a identificação do padrão e posterior refatoração, de acordo com as respectivas diretrizes definidas, nova documentação do sistema é gerada. O processo é bem controlado, definido e conciso, pois o produto de cada etapa é utilizado na etapa posterior. O código resultante desse processo tem sua estruturação melhorada com baixo acoplamento e alta coesão o que, conseqüentemente, gera maior manutenibilidade.

3 PROCESSO DE REFATORAÇÃO COM PADRÕES DE PROJETO

A definição deste processo ocorreu a partir da realização de estudos de caso envolvendo sistemas de informação, implementados em linguagem de programação Java, que não utilizavam padrões de projeto em seu código e que foram obtidos via Internet. Após a refatoração, o sistema permanece com as mesmas características do original (paradigma, linguagem e funcionalidade), porém apresenta padrões de projeto em sua estrutura e têm a sua documentação atualizada. O esboço geral do processo, composto por três etapas, é apresentado na Figura 1.

Todos os passos da Etapa 1 e todas as demais etapas são realizadas seqüencialmente na primeira iteração. A partir da segunda iteração, até a *n*-ésima, ou seja, enquanto for possível reconhecer indícios no código-fonte da aplicabilidade de padrões de projeto, somente o Passo 3 da Etapa 1 e as Etapas 2 e 3 devem ser realizados. Esse é o significado para a representação da flecha rotulada por 5 na Figura 1.

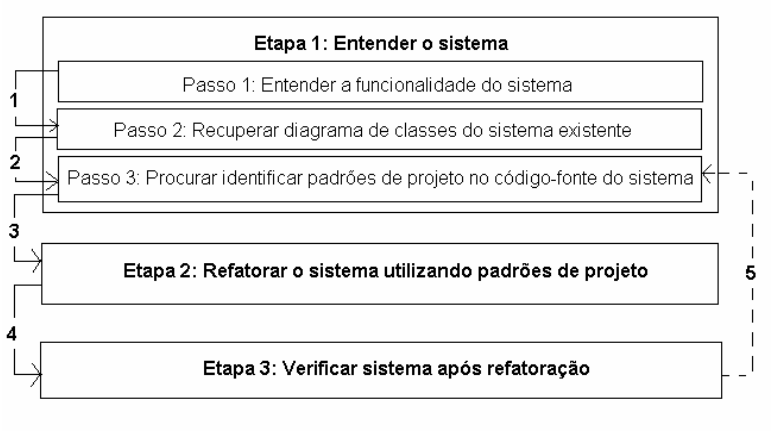

Figura 1: Processo de refatoração com padrões de projeto proposto.

3.1 Etapa 1: Entender o sistema

Seus três passos são realizados seqüencialmente com o objetivo de identificar as funções do sistema e o padrão de projeto que pode ser aplicado na respectiva iteração. A apresentação dos passos dessa etapa segue o formato especificado a seguir:

Passo i / Ação: define o passo *i* da etapa, seguida pela ação que deve ser executada.

Objetivo: descreve o propósito e a justificativa da Ação.

Solução: descreve o quê deve ser feito e como para que o Objetivo seja atingido.

Passo 1: Entender a funcionalidade do sistema.

Objetivo: Compreender a funcionalidade do sistema por meio da sua execução com armazenamento das entradas e saídas.

Solução: Gerar uma tabela contendo as informações, como mostrado na Tabela 1:

- Número da interação: números seqüenciais.
- Interação do usuário com o sistema (Entrada): identificar todas as interações do usuário quando o sistema é executado.

- Resposta do sistema (Saída): listar as saídas fornecidas, geradas pela interação do usuário com o sistema, tanto as exibidas na tela do usuário quanto as apresentadas no console do programador. Listar os possíveis arquivos gerados e seu conteúdo.

Tabela 1: Dados obtidos com a execução do sistema

Número da interação	Interação do usuário com o sistema (Entrada)	Resposta do sistema (Saída)		
		Na tela do usuário	No console do programador	Arquivos gerados
...	...	...	...	...

Passo 2: Recuperar diagrama de classes do sistema existente.

Objetivo: Utilizar/construir o diagrama de classes do sistema atual a partir do código-fonte para possível aplicação de padrões.

Solução:

a) Caso já exista o diagrama de classes do sistema, deve-se avaliá-lo quanto à real representação das informações (classes, métodos, atributos e relacionamentos) nele contidas, em relação ao que existe no código-fonte.

b) Caso não exista, pode-se utilizar uma ferramenta computadorizada que recupere automaticamente esse diagrama, ou fazê-lo manualmente. Essa etapa é a de engenharia reversa ([1], [9]) cujos detalhes não são comentados aqui.

Passo 3: Procurar identificar padrões de projeto no código-fonte do sistema.

Objetivo: A partir do entendimento da funcionalidade do sistema (Passo 1) e do conhecimento de sua arquitetura (Passo 2), o projetista deve procurar identificar trechos de código compatíveis com os apresentados na Tabela 2, que representam indícios da aplicabilidade do padrão. Esse passo é realizado a cada iteração, pois a cada vez somente indícios da aplicabilidade de um dos vinte e três padrões podem ser reconhecidos.

Solução:

a) Se o desenvolvedor conhecer a categoria do problema de projeto (estrutural, comportamental ou de criação), deve procurar identificar os indícios para essa categoria, como apresentados na Tabela 2; ou

b) Se o desenvolvedor não conhecer a categoria do problema, pode começar por qualquer uma delas ou percorrer seqüencialmente a Tabela 2, procurando identificar os indícios para obter a solução desejada.

A Tabela 2 apresenta apenas três padrões de projeto para exemplificar os indícios da possibilidade de aplicar tal padrão em um sistema implementado em Java. A lista completa de indícios para todos os padrões de projeto pode ser encontrada em Rapeli [10].

3.2 Etapa 2: Refatorar o sistema utilizando padrões de projeto

As diretrizes de re-implementação para os 23 padrões de projeto [4] foram descritas com o seguinte formato [10]:

Problema: descreve o problema de projeto que o padrão resolve, como apresentado em Gamma [5].

Solução: apresenta detalhadamente as diretrizes para refatorar o código do sistema original, de acordo com o padrão de projeto identificado.

Na solução de refatoração, as diretrizes informam se classes devem ser criadas, transformadas, removidas, etc. e a ordem em que essas atividades devem ocorrer. Assim, havendo necessidade de criar novas classes, devido ao uso do padrão identificado, essas terão a extensão _cp. Já as classes existentes que sofrerem alterações durante a implementação do padrão identificado terão a extensão _ap acrescida a seus nomes. A mesma nomenclatura é utilizada para os métodos criados e alterados.

O código apresentado nas diretrizes de refatoração é denominado de gabarito ou não alterado. Por gabarito entende-se que o código será alterado pelo desenvolvedor, de acordo com a aplicação. Devem ser substituídos apenas nomes e tipos de parâmetros, nomes de métodos e de classes devem ser mantidos como apresentados nas diretrizes. O código não alterado, como indica o próprio nome, deve ser utilizado tal como é apresentado, ou seja, sem modificações.

Para exemplificar as diretrizes de refatoração de um padrão, são reproduzidas as definidas para o padrão *Singleton*, definidas em Rapeli [10].

Tabela 2: Fragmento da lista de indícios

Padrão	Indícios
Categoria: Criação	

Singleton	O código deve apresentar trechos de controle de um recurso, que deve ser utilizado por um cliente de cada vez. Exemplo:
	`ConexaoBancoDados conexao = new ConexaoBancoDados();`
	`if(banco não está ativo){`
	`  conexao.conectar();`
	`  //declarações utilizando o banco de dados`
	`  conexao.desconectar();`
	`}else if(banco está ativo){`
	`  Sysout.println("Não foi possível conectar ao banco");`
	`}`

Categoria: Estrutural	
Decorator	O código deve apresentar agregação de funcionalidade(s) a um ou mais objetos, em tempo de execução. Exemplo: `if(cliente é homem){ cliente.AdicionaClienteSorteio(); }`
Categoria: Comportamental	

Visitor	O código deve realizar uma ou mais operações em objetos ou ainda, em um grande número de objetos. Exemplo: geração do objeto relatório, que deve recolher determinados dados de todos os objetos relacionados a ele.
	`//objetos é o vetor que contém todos os objetos`
	`//a serem considerados na operação`
	`for(int i = 0; i < objetos.size(); i++)`
	`  soma = soma + objetos.getTotalGasto();`

3.2.1. Refatoração com o Padrão Singleton

Problema: Garantir que uma classe tenha somente uma instância e fornecer um ponto global de acesso para a mesma.

Solução:

1. Para os passos de **a** a **c**, apresentados a seguir, atualizar o diagrama de classes.

   a) Criar a classe `Singleton_cp`, cujo código deve ser idêntico ao da Figura 2. A instanciação do objeto referente à classe `Singleton_cp` ocorrerá no Passo **c** (Figura 3).

b) Renomear a classe que contém o trecho de código identificado com a aplicabilidade do padrão *Singleton* (classe `X`, por exemplo) no Passo 3 da Etapa 1, adicionando ao seu final a extensão `_ap` (classe `X_ap`, por exemplo).

```
1   public class Singleton_cp {
2         static boolean instance_flag = false;
3    public Singleton(){
4      if (instance_flag) throw new RuntimeException();
5      else instance_flag = true;
     }
6    public void finalize(){ instance_flag = false; }
}
```

Figura 2: Classe `Singleton_cp`.

c) Retirar, da classe `X_ap`, o código que faz o controle do recurso e substituí-lo pelo código apresentado na Figura 3.

i. Na linha 2 é declarado e instanciado o objeto do tipo *Singleton*.

ii. Após a utilização do recurso, ele é liberado para novo uso, linha 3.

```
1 public class X_ap{
2   private Singleton singleton = new Singleton();
      //utilização do recurso
3     singleton.finalize();//liberação do recurso
  ...
}
```

Figura 3: Classe `X_ap`.

3.3. Etapa 3: Verificar sistema após refatoração

A Etapa 3 do processo é definida pelo mesmo formato utilizado na Etapa 1 e possui apenas um passo.

Passo 1: Verificar a funcionalidade do sistema.

Objetivo: Reavaliar o sistema após a refatoração quanto ao seu comportamento, a fim de constatar se sua funcionalidade permanece inalterada.

Solução: Utilizar as mesmas interações realizadas anteriormente no Passo 1 da Etapa 1 (Entender a funcionalidade do sistema), que formaram a Tabela 1, para analisar se as saídas após a refatoração permanecem inalteradas. Caso haja alterações no comportamento do sistema, há indício de que o padrão não foi utilizado adequadamente. O engenheiro de software deve retornar ao início do processo, identificar o ponto de falha e realizar as correções adequadas.

4 ESTUDO DE CASO

O estudo de caso aqui apresentado envolve a refatoração do sistema de videolocadora utilizando o padrão *Singleton*. Esse sistema tem 1883 linhas de código, foi obtido via Internet não possuia documentação além do código-fonte, com 11 classes: uma para banco de dados e as demais misturam tratamento de interface com o usuário e acesso ao banco (*MS-Access*).

4.1 Refatoração com o Padrão *Singleton*

4.1.1 Etapa 1: Entender o Sistema

Nesta subseção são descritos os três passos realizados na Etapa 1.

Passo 1: Entender a funcionalidade do sistema

O entendimento do sistema ocorreu a partir de sua execução, preenchendo-se os campos da Tabela 1, como mostrado parcialmente na Tabela 3. As seguintes convenções foram utilizadas: a) nomes de campos e botões: estão em fonte *courier new em itálico*; b) informações inseridas durante a execução do sistema pelo usuário estão entre aspas. Essa tabela difere da Tabela 1 quanto à coluna "arquivos gerados", pois nenhum arquivo foi gerado.

Tabela 3: Interações do usuário com o sistema de videolocadora

Nº Int.	Interação do usuário com o sistema (Entrada)	Resposta do sistema (Saída)	
		Tela do usuário	Console do programador
1	Executar o sistema.	A tela de *login* é apresentada contendo os campos *usuário, senha*; os botões *Ok* e *Cancelar* e a mensagem "Informe o usuário e a senha".	Nenhuma mensagem é exibida.
2	Na tela de *login*, inserir o nome "Admin" e o usuário "root". Pressionar o botão *Ok*.	O menu principal do sistema é exibido com as opções: *Arquivo-Sair*; *Cadastro-Cliente*; *Cadastro-Filme*; *Movimentação-Aluguel*; *Movimentação-Devolução*; *Movimentação-Mostra de Filme*; *Relatórios-Locações*; *Auxílio-Sobre*.	Nenhuma mensagem é exibida.
...			
14	Inserir os valores "40", "João", "11111" e "01/01/2000" nos respectivos campos. Pressionar o botão *Gravar*.	Quando o valor da matrícula é inserido, os botões *Gravar* e *Excluir* são habilitados. Quando o botão *Gravar* é pressionado, todos os campos são ocultados.	Nenhuma mensagem é exibida.

Passo 2: Recuperar diagrama de classes do sistema existente

O diagrama de classes foi gerado automaticamente pela ferramenta de software Omondo [8], executada como um *plugin* na plataforma de desenvolvimento Eclipse [3]. A Figura 4 apresenta o diagrama de classes do sistema de videolocadora original.

Passo 3: Procurar identificar padrões de projeto no código-fonte do sistema

A funcionalidade do sistema foi obtida por meio de sua execução (Passo 1), daí a lista de indícios (Tabela 2) foi percorrida conforme especificado no item ***b*** da solução exibida na seção 3.1, Passo 3. Dessa forma, verificou-se que o sistema apresenta apenas uma tela ao usuário de cada vez, que o impossibilita de ter acesso a mais de uma função do menu ao mesmo tempo. Isso é um indício da aplicabilidade do padrão *Singleton* nesse sistema e o recurso a ser controlado pelo padrão são as opções disponíveis no menu. A aplicação desse padrão ocorreu em todo trecho de código que realiza essa tarefa, em todas as classes que apresentam as telas do menu ao usuário, denominadas genericamente por `X`.

Para exemplificar, a Figura 5 exibe um trecho de código semelhante ao que foi identificado nas classes do sistema, que indicam que o padrão *Singleton* pode ser aplicado (implementado).

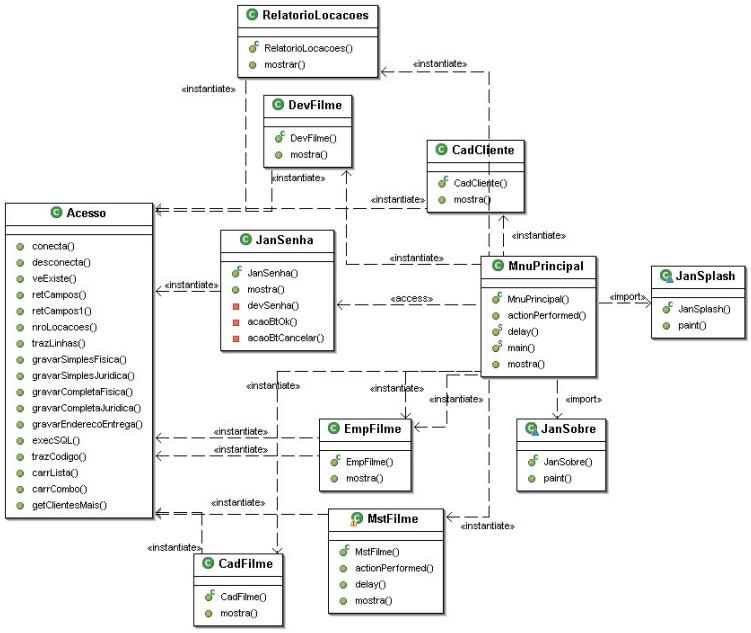

Figura 4: Diagrama de classes do sistema de videolocadora original.

```
1 class X extends JDialog{
...
2   this.setModal(true);
...
```

Figura 5: Código-fonte identificado para aplicação do padrão *Singleton.*

4.1.2 Etapa 2: Refatorar o sistema utilizando padrões de projeto

A seguir são descritas as atividades realizadas na solução da refatoração utilizando o padrão *Singleton*, aplicadas ao sistema de videolocadora:

1. O diagrama de classes foi gerado após a aplicação das diretrizes de re-implementação, pela ferramenta de software Omondo [8], utilizando a plataforma de desenvolvimento Eclipse [3]. O diagrama de classes da Figura 7 foi obtido após a refatoração, realizada para o sistema do estudo de caso, com o padrão *Singleton*. As classes criadas e/ou alteradas pelo processo de refatoração são marcadas no diagrama com um asterisco (`RelatorioLocacoes_ap`, `CadCliente_ap`, `Singleton_cp`, `EmpFilme_ap`, `DevFilme_ap`, `MstFilme_ap`, `CadFilme_ap`, `MnuPrincipal_ap`).

a) Foi criada a classe `Singleton_cp` como definido no Passo **a** (Solução), seção 3.2.1.

b) Foram encontradas as classes que tratam da interface com o usuário e que contêm código como apresentado na Figura 3. Foi alterado o nome dessas classes inserindo-se a extensão _ap (linha 1 da Figura 6).

c) Foi retirado o código que faz o controle do recurso (apresentado na Figura 5) de todas as classes identificadas no Passo **b** e substituído pelo indicado no processo de refatoração. Uma dessas classes identificadas como controladoras do recurso é `MstFilme_ap` (Figura 6), em que o código das linhas 3, 6 e 12 foi inserido.

```
1 public class MstFilme_ap extends JFrame implements ActionListener{
        //declaração de componentes visuais da tela
        ...
2       private Acesso ac = new Acesso();
3       private Singleton_cp singleton;
4       private MstFilmeBD_cp mstFilme = new MstFilmeBD_cp();
5       public MstFilme_ap() {
6          singleton = new Singleton_cp();
7          try {
8             mostra();
9          } catch (Exception ex) { ex.printStackTrace(); }
        }
10      public void mostra() throws Exception { ... }
        ...
11      private void aoFechar(WindowEvent e) {
12        singleton.finalize();
13        dispose();
        }
}
```

Figura 6: Classe alterada com o padrão *Singleton.*

4.1.3 Etapa 3: Verificar sistema após refatoração

As interações apresentadas na Tabela 3 foram novamente realizadas, constatando-se que o comportamento do sistema permaneceu inalterado.

4.2 Análise do Sistema Refatorado Comparado com o Sistema Inicialmente Existente

No diagrama de classes do sistema refatorado, Figura 7, as classes marcadas com asterisco são as criadas ou alteradas pelo processo de refatoração. Ao analisar os diagramas de classes do sistema de videolocadora original, Figura 4, e do refatorado, Figura 7, observa-se que o sistema refatorado possui uma classe adicional (`Singleton_cp`) e, conseqüentemente, mais relacionamentos.

O padrão *Singleton* tornou o sistema mais robusto, visto que há uma classe (`Singleton_cp`) dedicada exclusivamente a controlar a exibição das telas, e não há mais uma linha de código (Figura 5) inserida em todas as classes que tratam da interface com o usuário, pouco intuitiva ao mantenedor do sistema.

A nomenclatura adotada para nomear/renomear as classes facilitou a identificação das que foram criadas devido à aplicação de padrões, já que os nomes são significativos. Isso pode ser verificado no diagrama de classes do sistema refatorado, Figura 7. Esse fator também contribui para que manutenções futuras sejam facilitadas, bem como o entendimento do sistema.

A classe `MstFilme`, após a refatoração, tem relacionamento com a classe `Singleton_cp`, como pode ser visto na Figura 7. Além disso, considerando-se na Figura 5 a classe `X` como sendo `MstFilme`, pode-se observar detalhadamente a modificação de código ocorrida, na Figura 6.

Os métodos da classe do padrão (`Singleton_cp`) podem ser reutilizados na íntegra em qualquer outra aplicação desse padrão. A classe da aplicação, pode haver mais de uma, terá as adaptações necessárias para controle do recurso utilizado (padrão).

Embora somente um padrão tenha sido comentado neste trabalho, outros dez também foram aplicados nesse sistema: *Builder*, *Prototype*, *Visitor*, *Adapter*, *Bridge*, *Decorator*, *Proxy*, *Command*, *Iterator* e *Template Method*, apresentados em Rapeli [10].

Apesar do número de linhas de código ter aumentado, a complexidade do sistema não aumentou, visto que houve controle sobre a classe inserida e as alteradas. O sistema refatorado é mais manutenível e seu entendimento é facilitado, pois as classes são mais coesas, ou seja, têm funcionalidade específica.

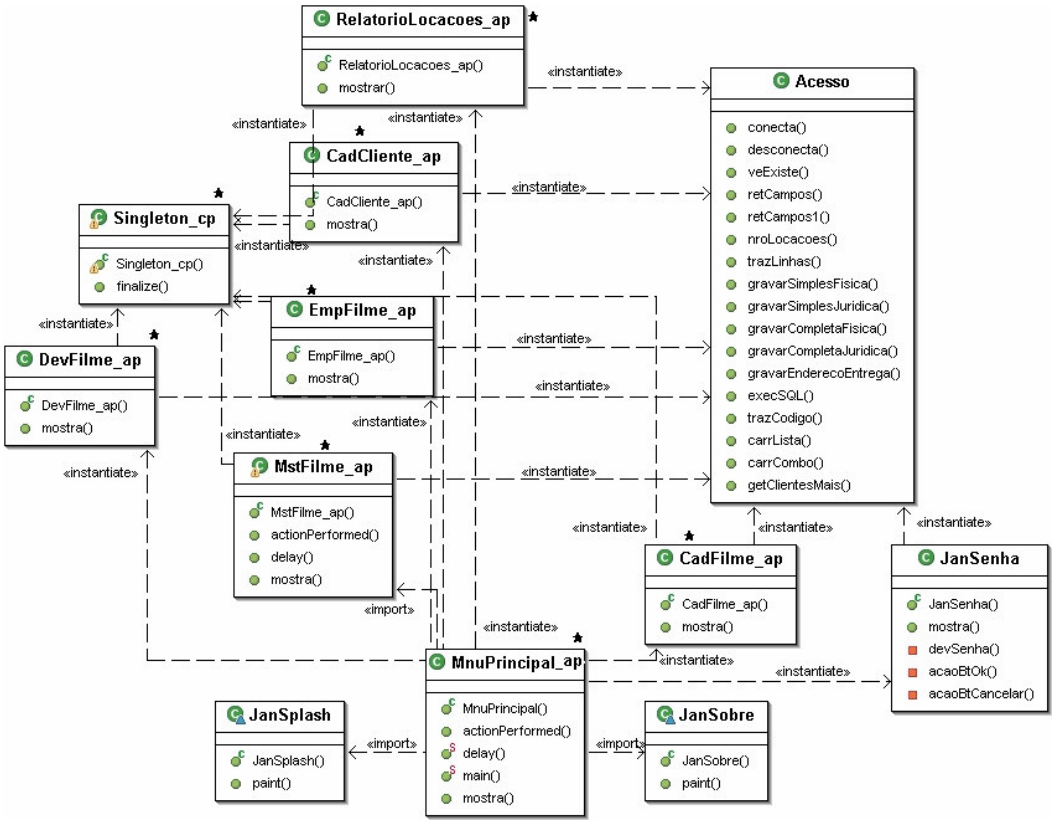

Figura 7: Diagrama de classes do sistema de videolocadora após a refatoração com o padrão *Singleton*.

5 CONSIDERAÇÕES FINAIS

O trabalho proposto é um processo de refatoração que permite identificar a aplicabilidade de padrões de projeto de software [5] em sistemas desenvolvidos com a linguagem de programação Java e sua posterior reestruturação, de acordo com diretrizes específicas para cada padrão de projeto.

A utilização de padrões de projeto não é trivial quando um sistema é desenvolvido e também apresenta dificuldades para o seu reconhecimento em um sistema existente. As diretrizes aqui apresentadas amenizam essa dificuldade, mas exigem do engenheiro de software conhecimento detalhado dos padrões de projeto. Estima-se que o software refatorado dessa forma seja mais fácil de se manter e, conseqüentemente, de sofrer evoluções.

A adoção de uma nomenclatura própria para classes e métodos inseridos ou alterados no sistema, extensões _cp, _ap e nomes de classes que remetem ao nome do padrão, visa à organização do código facilitando a manutenção do sistema, visto que as classes que representam os padrões de projeto são facilmente identificadas pelo desenvolvedor, não sendo misturadas às classes do domínio da aplicação.

Uma das contribuições deste trabalho é a lista de indícios, que fornece ao engenheiro de software características da aplicabilidade de um padrão de projeto a partir do código-fonte existente ou da funcionalidade desse sistema. Embora, para a elaboração da lista de indícios para reconhecimento no código de cada padrão de projeto e das diretrizes de refatoração, somente tenham sido utilizados sistemas de informação simples e com funcionalidade bem conhecida, infere-se que o processo e as diretrizes podem ser aplicados a sistemas em outros domínios. A dificuldade na obtenção de

sistemas com código disponível, que pudessem ser utilizados para validação do processo foi o motivo pelo qual outros tipos de sistemas não foram utilizados. Para esses estudos de caso conduzidos a lista de indícios mostrou-se satisfatória e adequada.

Para a elaboração deste processo e das diretrizes de refatoração foram utilizados outros sete sistemas, nos quais um ou mais padrões foram aplicados isoladamente. Entre os trabalhos futuros pretende-se pesquisar: utilização do processo apresentado com sistemas de médio porte para que também possa ser avaliada a generalidade da lista de indícios e a possibilidade de aplicar mais de um padrão durante uma refatoração; a definição de mecanismos de correções, caso a refatoração não tenha sido executada adequadamente; o refinamento e a ampliação da lista de indícios, facilitando a identificação do padrão no sistema existente e o desenvolvimento de uma ferramenta automatizada para a aplicação do processo.

REFERÊNCIAS

[1] Cagnin, M. PARFAIT: uma contribuição para a reengenharia de software baseada em linguagens de padrões e frameworks. Tese (Doutorado em Ciência da Computação) – Universidade de São Paulo, São Carlos, 2005.

[2] Cinnéide, M. Automated Application of Design Patterns: A Refactoring Approach. Tese (Doutorado em Ciência da Computação) – Trinity College, Dublin, 2000.

[3] Eclipse. Disponível em: www.eclipse.org, acesso em Nov 2005.

[4] Fowler, M. Refactoring: Improving the Design of Existing Code. Boston: Addison-Wesley, 2000.

[5] Gamma, E., Helm, R., Johnson, R., Vlissidees, J. Design patterns – Elements of Reusable Object-Oriented Software, New York: Addison-Wesley, 1995.

[6] Jeon, S., Lee, J., Bae, D. An Automated Refactoring Approach to Design Pattern-Based Program Transformations in Java Programs. In: Proceedings of Asia-Pacific Software Engineering Conference (IEEE), 9., 2002.

[7] Kerievsky, J. Refactoring to Patterns. Addison-Wesley, 2004.

[8] Omondo. Disponível em: www.omondo.com, acesso em Nov 2005.

[9] Pressman, R. Software Engineering, 6. ed., McGraw-Hill, 2005.

[10] Rapeli, L. R. C. Refatoração de sistemas Java utilizando padrões de projeto: um estudo de caso. Dissertação (Mestrado em Ciência da Computação), UFSCar – São Carlos, 2005.

[11] Rajesh, J., Janakiram, D. JIAD: A Tool to Infer Design Patterns in Refactoring. In: Proceedings of International Conference on Principles and Practice of Declarative Programming, 6., Verona, 2004.

[12] Shi, N., Olsson, R. Reverse Engineering of Design Patterns for High Performance Computing. In: Proceedings of Workshop on Patterns in High Performance Computing. Illinois, 2005.

[13] Tsantalis, N., Chatzigeorgiou, A., Halkidis, S. T., Stephanides, G. A Novel Approach to Automated Design Pattern Detection, In: Proceedings of 10th Panhellenic Conference on Informatics (PCI'2005), Volos, Greece, Nov. 11-13, 2005.

[14] UML: Unified Modeling Language, versão 2.0. Disponível em http://www.omg.org/technology/documents/formal/uml.htm, acesso em Jun 2006.