Uma Proposta de Reúso de Interface Gráfica com o Usuário Baseada no Padrão Arquitetural MVC

Vinícius H. S. Durelli[1], Matheus C. Viana[2], Rosângela A. D. Penteado

Departamento de Computação (DC) - Universidade Federal de São Carlos (UFSCar)
Caixa Postal 676 – 13565-905 – São Carlos – SP – Brasil

{vinicius_durelli, matheus_viana, rosangel}@dc.ufscar.br

Abstract: *Reuse is one of the software concerns of developers and it can be obtained using software patterns or framework. This paper proposes the reuse of graphical user interfaces, developed using the Model-View-Controller pattern with some adaptations in order to appropriately divide the system layers. It presents two versions of an information system built based on different techniques: a pattern language and a framework, both belonging to the business resource management domain.*

Resumo: *Reúso é uma das preocupações dos desenvolvedores de software e pode ser obtido usando padrões de software ou framework. Este trabalho propõe a reutilização de interfaces gráficas com o usuário, desenvolvidas usando o padrão Model-View-Controller, com adaptações, para possibilitar a separação entre as camadas de sistemas de informação. Para exemplificar tal reúso são apresentadas duas versões de um sistema de informação desenvolvidas com técnicas distintas: com a Linguagem de Padrões para Gestão de Recursos de Negócios (GRN) e com o framework de Gestão de REcursos de Negócios em Java (GRENJ).*

1. Introdução

A qualidade de um sistema é um dos pontos que tem merecido a atenção de desenvolvedores. O desenvolvimento de um sistema, algumas vezes, é realizado a partir da reutilização de soluções e código bem sucedidos em outros sistemas. O reúso de artefatos existentes com o objetivo de criar um novo artefato [Krueger 1992] [Frakes e Kyo 2005] é empregado com os seguintes propósitos: economia de custos, aumento na produtividade e maior garantia da qualidade dos artefatos produzidos [Shiva e Shala 2007]. Dentre as técnicas de reúso pode-se citar: padrões, linguagens de padrões e frameworks de software orientados a objetos.

Padrões, em sua essência, comunicam uma solução de sucesso para determinado problema em um contexto específico [Gamma *et al.* 1995]. Existem padrões em vários níveis de abstração, como por exemplo, análise, projeto, implementação [Shalloway e Trott 2001]. Os padrões são apresentados em um formato que apresenta claramente o problema, sua solução, casos em que foram utilizados e outros padrões relacionados a esse, criando um vocabulário de mais alto-nível [Fowler 2003] [Wirfsbrock 2006].

[1] Apoio financeiro CAPES.
[2] Apoio financeiro CNPq.

Uma coleção estruturada de padrões que reflete e documenta a experiência sobre um domínio específico é denominada linguagem de padrões [Schmidt *et al.* 1996]. Estão em um nível mais alto de abstração em relação aos padrões de projeto [Gamma *et al.* 1995]. Uma linguagem de padrões representa a seqüência temporal de decisões que levam ao projeto completo de uma aplicação em um domínio, de forma a tornar-se um guia para o processo de desenvolvimento. Geralmente são organizadas em árvores ou grafos [Brugali *et al*., 2000].

Framework de software orientado a objetos é outra técnica de reúso, específica de um domínio, composto por um conjunto de classes tanto abstratas quanto concretas e o modo que suas instâncias interagem [Fayad e Schmidt 1997].

Este trabalho faz uso da linguagem de padrões Gestão de Recursos de Negócios (GRN) [Braga *et al.* 1999] e do framework Gestão de REcursos de Negócios em Java (GRENJ) [Durelli 2007]. O GRENJ, implementado em linguagem Java, é produto da reengenharia que está sendo realizada com o framework Gestão de REcursos de Negócios (GREN), que foi desenvolvido em linguagem Smalltalk com base na GRN [Braga 2002]. O objetivo deste trabalho é propor uma abordagem que reutilize interfaces gráficas criadas com base no padrão *Model-View-Controller* (MVC) e facilite o desenvolvimento de sistemas de informação no domínio de gestão de recursos de negócios. Dessa forma, o trabalho dos desenvolvedores é facilitado tanto na construção quanto na manutenção de tais sistemas, além de aprimorar o nível de reúso obtido.

Além desta Seção, as demais estão organizadas da seguinte forma: a Seção 2 apresenta a linguagem de padrões GRN e os frameworks GREN e GRENJ; a Seção 3 define uma abordagem de criação de uma interface gráfica que possa ser reutilizada por sistemas, com os mesmos requisitos, desenvolvidos com base na GRN ou instanciados com o GRENJ; a Seção 4 apresenta um estudo de caso em que um sistema para uma locadora de DVDs foi desenvolvido primeiramente com base na GRN e posteriormente com base no GRENJ, mas utilizando a mesma interface gráfica; a Seção 5 aborda alguns trabalhos relacionados e, por fim, a Seção 6 apresenta as considerações finais e trabalhos futuros.

2. GRN, GREN e GRENJ

A linguagem de padrões para Gestão de recursos de Negócios (GRN) é composta por quinze padrões de análise divididos em três grupos de acordo com os seus propósitos, sendo alguns deles aplicações ou extensões de outros existentes na literatura. Cada padrão além de sua descrição textual é representado por um modelo de classes em nível de análise. O modelo de classes da aplicação a ser desenvolvida é obtido pelo conjunto de padrões utilizados.

Os padrões do primeiro grupo estão relacionados à identificação do recurso de negócio. Os do segundo grupo abordam as transações envolvendo os recursos e, por fim, os do terceiro grupo estão relacionados aos detalhes das transações [Braga *et al.* 1999] [Braga e Masiero 2002]. Com a GRN é possível desenvolver sistemas de informação, mais especificamente aqueles nos quais seja necessário registrar transações de aluguel, comercialização e manutenção dos recursos de negócio [Braga 2002]. A Figura 1 ilustra a GRN com os seus respectivos padrões, na forma de grafo, explicitando a ordem na qual eles podem ser aplicados. Os principais padrões da GRN são *Locar o Recurso*, *Comercializar o Recurso* e *Manter o Recurso*.

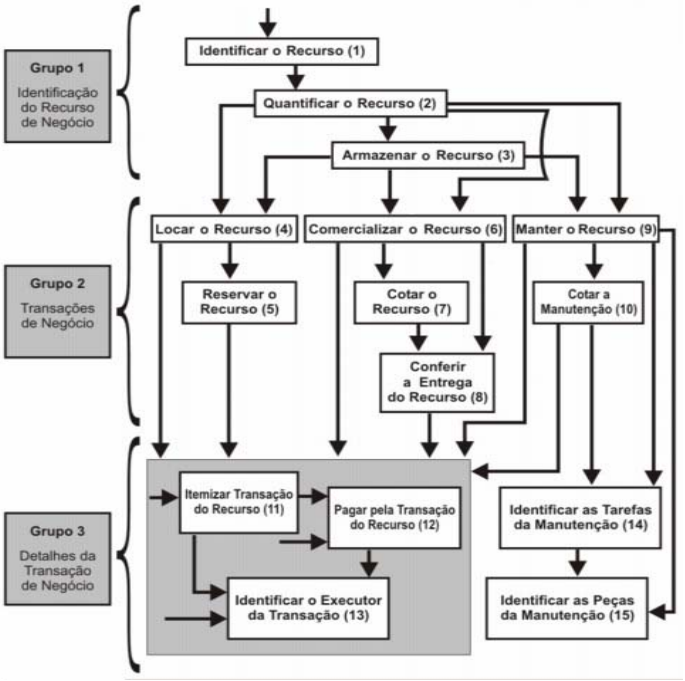

Figura 1. Grafo de fluxo de aplicação da GRN (adaptado de [Braga 2002]).

O framework GREN foi desenvolvido, seguindo o modelo de processo incremental, com base na linguagem de padrões GRN [Braga 2002], com o intuito de facilitar o desenvolvimento de sistemas e proporcionar tanto reúso de projeto quanto de código [Johnson 1997]. Foi implementado na linguagem Smalltalk, utilizando o banco de dados relacional MySQL [MySQL 2007]. A arquitetura do GREN é composta de três camadas: persistência, negócios e interface gráfica com o usuário. A camada de persistência contém as classes relacionadas à conexão com o banco de dados e persistência de forma geral. Os padrões da GRN encontram-se implementados na camada de negócios [Braga 2002]. Na camada de interface gráfica com o usuário estão localizados os formulários, janelas, menus, caixas de diálogos, entre outros responsáveis pela interação com o usuário final.

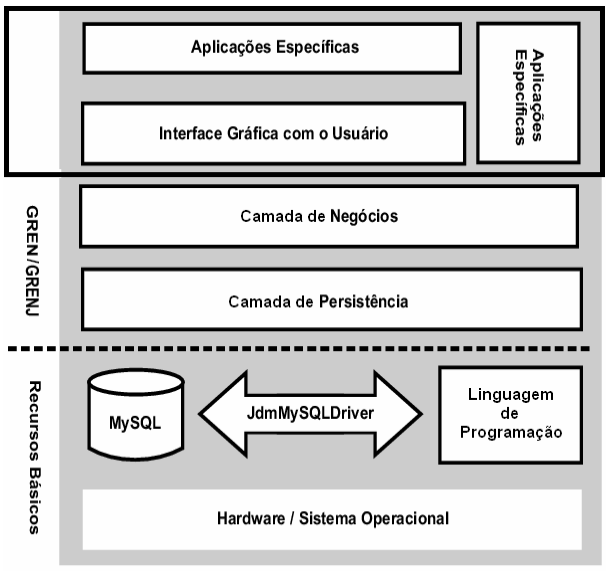

Figura 2. Arquitetura do framework GREN (adaptada de [Braga 2002]).

Considerando que a linguagem Smalltalk não é amplamente utilizada na indústria, mas somente em ambiente acadêmico, optou-se por realizar um processo iterativo de reengenharia para produzir uma nova versão do framework GREN na linguagem Java. Essa nova versão é denominada GRENJ e sua implementação possui, atualmente, mais de cinqüenta por cento dos padrões da GRN implementados. Das três camadas do framework GREN somente as camadas de persistência e negócios vêm passando por reengenharia. A arquitetura do GRENJ é a mesma do GREN, Figura 2, exceto camadas superiores; Aplicações Específicas e Interface Gráfica com o Usuário.

3. Criação de uma Interface Gráfica Reutilizável Utilizando o Padrão MVC

A GRN fornece ao desenvolvedor uma série de diagramas de classe em nível de análise. Esses diagramas facilitam a implementação da camada de negócios dos sistemas de informação pertencentes ao domínio por ela abrangido, propondo formas de se estruturar as classes e seus relacionamentos e fornecendo reúso de experiência. Entretanto, o desenvolvedor que utiliza a GRN fica responsável pela criação do mecanismo de persistência e da interface gráfica com o usuário.

Ao se utilizar o framework GRENJ, o desenvolvedor obtém tanto reúso de projeto quanto de código, dado que os padrões da GRN já se encontram implementados na camada de negócio do framework, bem como o mecanismo de persistência na sua devida camada. Não obstante, é necessário um nível intermediário de conhecimento da arquitetura do GRENJ e de seus pontos variáveis para que se possa instanciar uma aplicação. Todavia, conforme mencionado na seção anterior, o framework ainda não trata da elaboração da camada de interface gráfica com o usuário, tarefa atribuída ao desenvolvedor.

Model-View-Controller (MVC) é um padrão arquitetural que surgiu originalmente como um artefato idiomático na linguagem Smalltalk [Harrison *et al*. 2007]. Emprega os seguintes padrões de projeto: *Observer*, *Strategy* e *Composite* [Gamma *et al.* 1995] e com o seu uso o sistema é dividido em três camadas: *Model*, *View* e *Controller*. A camada *Model* mantém as informações relacionadas ao domínio e os objetos que implementam a funcionalidade principal do sistema de informação. Na camada *View* encontram-se os objetos relacionados à apresentação, ou seja, objetos que representam a interface gráfica com o usuário. A camada *Controller* define a maneira como a interface gráfica (*View*) deve agir, a partir das informações fornecidas pelo usuário, além de atualizar as informações e o estado dos objetos da camada *Model*. Assim, a camada *Controller* atua como um mediador, tendo acesso às classes da camada *Model* para realizar as tarefas do sistema, como por exemplo, registrar os dados de um cliente fornecidos através da interface gráfica. A realização dessa tarefa ocorre com a camada de interface gráfica com o usuário (*View*) coletando os dados do usuário e repassando-os para a camada *Controller* que, com os dados fornecidos, instancia um objeto da classe *Cliente* e persiste as informações desse objeto no banco de dados.

Os passos de cada tarefa podem diferir de uma versão de um sistema para outra, dependendo das classes da camada *Model* e das regras de negócio. Mas, a relação de tarefas deve ser a mesma para existir o reúso total da interface gráfica. Portanto, com o propósito de se obter maior nível de reutilização, propõe-se a criação de uma interface ou classe abstrata, denominada *Identificadora de Tarefas* que tem a finalidade de definir a comunicação existente entre as camadas *Controller* e *View* (interface gráfica) do

sistema de informação. Cada operação dessa interface (*Identificadora de Tarefas)* representa e identifica uma tarefa. A Figura 3 mostra o diagrama de classes que contém as camadas *Model*, *View (*com a interface *Identificadora de Tarefas)* e *Controller*, e o diagrama de seqüência que ilustra a troca de mensagens entre as classes dessas três camadas, respectivamente, *Model_Class*, *GUI_Class* e *Controller*.

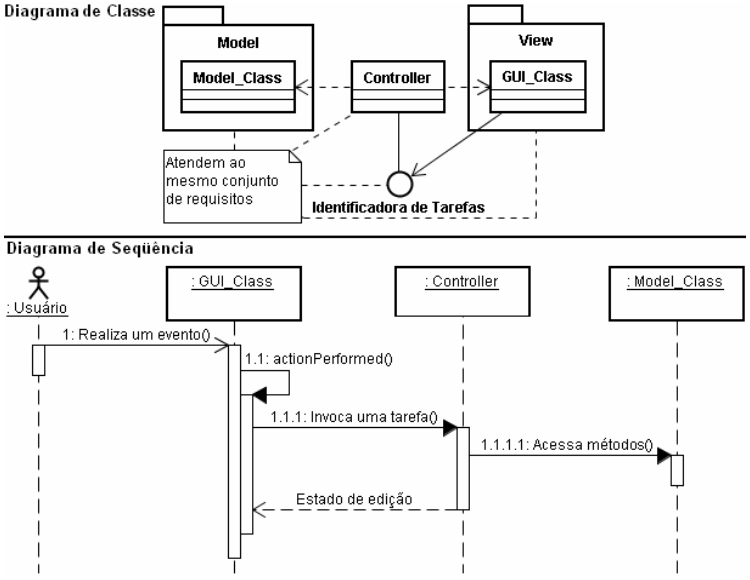

Figura 3. Diagramas de classe e seqüência do padrão arquitetural MVC.

4. Estudo de Caso

Para ilustrar a abordagem proposta foram desenvolvidas duas versões para um sistema de informação fictício que atende aos requisitos, Tabela 1, de uma locadora de DVDs. A primeira versão do sistema foi desenvolvida utilizando os padrões da GRN e o padrão arquitetural MVC para a construção da interface gráfica. A segunda versão, que atende aos mesmos requisitos, foi instanciada usando o framework GRENJ e o padrão MVC com a reutilização da camada *View*, elaborada durante o desenvolvimento da primeira versão.

Tabela 1. Conjunto de requisitos do sistema de informação da locadora de DVDs.

#	Descrição
1	A locadora realiza o aluguel de DVDs de filmes que podem ter uma ou mais cópias (DVDs).
2	Cada filme possui um código, título e ano.
3	Cada DVD possui código que identifica sua posição na prateleira, informação que indica se está disponível ou alugado e o título do filme nele contido.
4	Os filmes são classificados por categoria, que indica o valor diário do aluguel desse DVD.
5	Os filmes também são classificados por gênero (comédia, terror, ação, drama, etc.).
6	Os DVDs são alugados para os clientes cadastrados da locadora. As informações que o sistema deve manter sobre o cliente são: código, nome, telefone e CPF.
7	As informações de locação são: código, a data de locação, a data de devolução prevista, código do cliente, DVDs alugados, data de devolução efetiva e o valor. Um cliente pode alugar mais de um DVD em uma mesma locação e deve devolver todos no mesmo instante. A data de devolução parevista é de um dia para cada DVD alugado em relação à data de locação. O valor da locação varia de acordo com a soma dos valores de cada DVD alugado.
8	Se os DVDs não forem devolvidos na data de devolução prevista, o cliente deve pagar multa.O valor da multa é um valor fixo multiplicado pelos dias de atraso na devolução dos DVDs.

4.1. Versão 1: Sistema da Locadora de DVD Utilizando os Padrões da GRN

Como mencionado na Seção 2, a GRN estabelece uma ordem para a aplicação dos seus padrões que resulta em um modelo de classes da aplicação em nível de análise. A Figura 4 ilustra o modelo de classes para o sistema de locadora de DVDs, utilizando notas UML para ilustrar o nome original das classes de cada um dos padrões usados.

O primeiro padrão aborda como identificar o recurso e suas possíveis classificações de acordo com os requisitos do sistema em desenvolvimento. Analisando os requisitos da Tabela 1, um único recurso foi identificado e definido pela classe *Filme*, relacionada com as classes *Categoria* e *Genero*, que representam classificações do recurso. O segundo padrão determina a quantificação do recurso. Como para um *Filme* podem existir várias cópias, a classe *DVD* foi criada para representá-las. O terceiro padrão é opcional e não foi aplicado neste estudo de caso.

O segundo grupo de padrões da GRN é responsável pelas transações do negócio. De acordo com os requisitos apenas o padrão *Locar o Recurso* é necessário, acarretando na implementação das classes *Locacao*, *Cliente* e *Multa*. *Locacao* representa a transação de mesmo nome que é realizada toda vez que clientes, os interessados nas transações, realizam a locação de um DVD. *Multa* trata das tarifas cobradas quando ocorrem atrasos na devolução dos DVDs de uma locação.

Do terceiro grupo de padrões da GRN, apenas o padrão *Itemizar Transação do Recurso* foi aplicado para permitir que uma ou mais instâncias do recurso sejam vinculadas a uma transação. A classe *ItemLocacao* é responsável por encapsular os DVDs relacionados a uma locação. A Figura 4 mostra o diagrama de classes completo criado para o sistema da locadora de DVDs com a utilização dos padrões da GRN citados anteriormente.

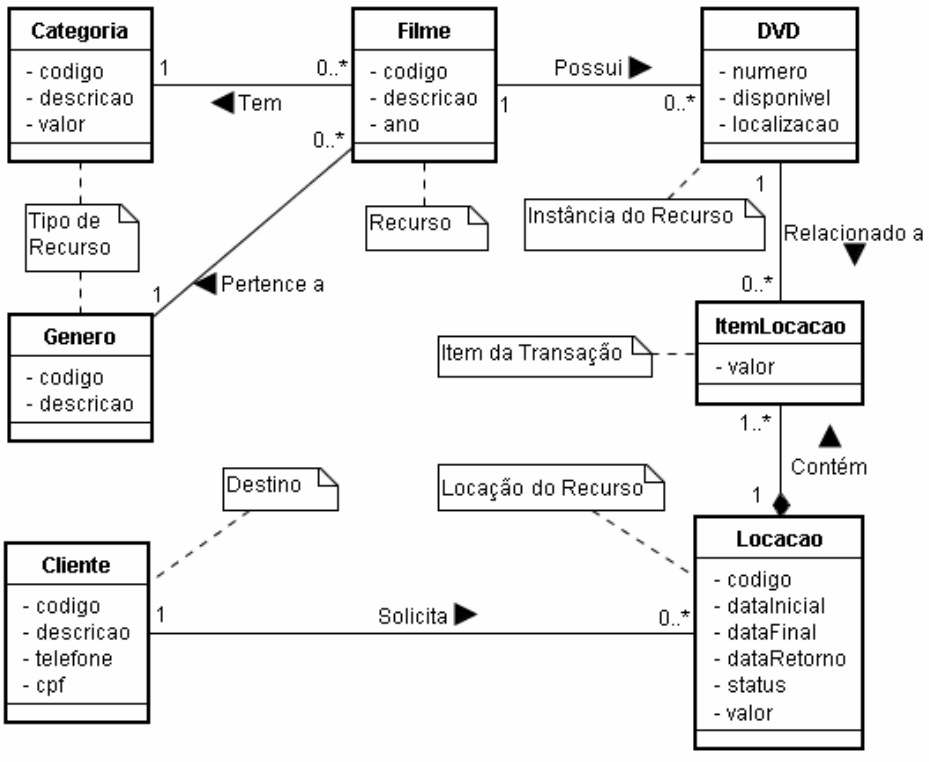

Figura 4. Diagrama de classes do sistema da locadora.

A Tabela 2 mostra na primeira coluna as tarefas referentes à transação de locação e na segunda, os métodos associados à interface *Identificadora de Tarefas* que são implementados pela camada *Controller* utilizando a funcionalidade implementada na camada *Model*.

Tabela 2. Relação de tarefas do sistema da locadora de DVDs.

#	**Tarefa**	**Método da Identificadora de tarefas**
1	Iniciar uma nova locação	iniciarNovaLocacao
2	Iniciar a alteração dos dados de uma locação	alterarLocacao
3	Remover uma locação	apagarLocacao
4	Salvar os dados da locação	salvarLocacao
5	Cancelar a edição (inserção/alteração) de uma locação	cancelarEdicaoLocacao
6	Inserir um item (DVD) na locação em edição	inserirItemLocacao
7	Remover um item (DVD) da locação em edição	removerItemLocacao
8	Realizar a devolução de DVDs (finalização da locação)	finalizarLocacao

4.2. Versão 2: Sistema da Locadora de DVD Utilizando o GRENJ

Como o GRENJ é baseado na GRN, a instanciação da segunda versão do sistema é feita utilizando os mesmos padrões descritos na Seção 4.1. Desse modo, classes equivalentes às classes criadas na Seção anterior foram construídas, por meio de extensões das classes já implementadas no GRENJ que representam as classes dos padrões da GRN.

O GRENJ proporciona reúso caixa-branca, que é obtido por meio de herança de classes pré-existentes [Fayad e Shmidt 1997] [Cunningham *et al*. 2006], ou seja, subclasses equivalentes às criadas na Seção anterior foram criadas, como mostra a Tabela 3, evitando-se que o desenvolvedor comece do zero (*from scratch*) um novo sistema. As customizações necessárias, adição de novos atributos e métodos para implementar a funcionalidade pretendida são introduzidas nas subclasses criadas. A instanciação de sistemas utilizando frameworks requer que o desenvolvedor conheça a sua arquitetura, principalmente de seus pontos variáveis (*hot spots*), que devem ser customizados.

Tabela 3. Relação das classes da camada de negócio do sistema da locadora.

#	**Classe criada**	**Classe do GRENJ que foi estendida**
1	*Filme*	*Resource*
2	*Categoria*	*SimpleType*
3	*Genero*	*SimpleType*
4	*DVD*	*ResourceInstance*
5	*Locacao*	*ResourceRental*
6	*Cliente*	*DestinationParty*
7	*Multa*	*FineRate*
8	*ItemLocacao*	*TransactionItem*

Devido ao reúso de código proporcionado pelo framework o número de linhas de código necessário para implementar a segunda versão do sistema foi menor que o da primeira. Dessa forma, infere-se que houve redução de esforço despendido para implementação. Com a utilização somente da GRN o desenvolvedor obtém reúso do modelo da camada de negócios (experiência proporcionada pelos padrões), mas fica sob sua responsabilidade a construção das outras camadas, como por exemplo, a de persistência e interface com o usuário, bem como a da arquitetura do sistema que define a comunicação entre essas camadas. Já com o uso do GRENJ o trabalho é minimizado,

considerando o reúso da camada de negócios, de persistência e da comunicação existente entre elas.

4.3. Criação e reutilização da interface gráfica

A Figura 5 apresenta o esquema de acoplamento da interface gráfica para as duas versões do sistema da locadora de DVD desenvolvidas com base na GRN e no GRENJ, respectivamente. A camada de interface com o usuário capta cada evento e chama o método da interface *Identificadora de Tarefas* correspondente, que é implementado por alguma classe na camada *Controller*. A interface gráfica desconhece totalmente o funcionamento interno dos métodos implementados na camada *Controller* e definidos pela *Identificadora de Tarefas*. Para ela só são necessárias as informações que devem ser enviadas e quais terá como resposta. Essa propriedade permite a sua reutilização em qualquer versão de um mesmo sistema atendendo a requisitos comuns, mesmo que utilizando tecnologias diferentes.

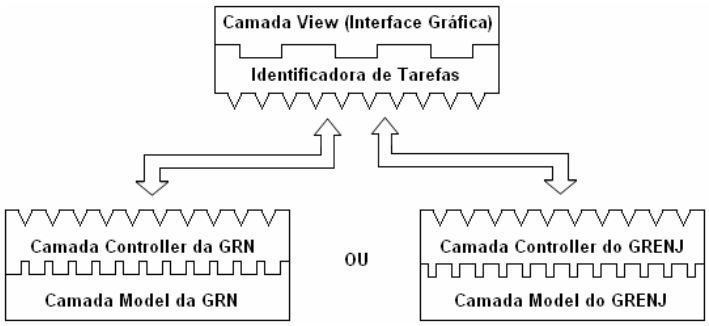

Figura 5. Esquema de acoplamento da interface gráfica com as duas versões do sistema.

Na implementação da arquitetura proposta, classes da camada de interface gráfica com o usuário possuem uma variável de referência do tipo *Identificadora de Tarefas*. Dessa forma, classes da camada *Controller* implementam essa interface, proporcionando a funcionalidade necessária por meio da interação com a camada de negócios (*Model*). Por exemplo, caso a interface *Identificadora de Tarefas* defina um método denominado *inserirCliente*, classes concretas da camada *Controller* que implementam essa interface devem fornecer a implementação da tarefa por meio da comunicação com as classes da camada de negócios, nesse caso a classe *Cliente*. A Figura 6 ilustra alguns trechos de código de uma interface *Identificadora de Tarefas*; a Figura 7 mostra uma classe que implementa essa interface e, a Figura 8 exibe um trecho de código de uma classe localizada na camada de interface gráfica com o usuário.

```
import java.sql.Date;

public interface IdentificadoraDeTarefas {

/**
* Persiste o cliente no banco de dados.
* @return <code>true</code> se o cliente caso cliente
* tenha sido persistido com sucesso, <code>false</code> caso
* contrário
*/
    public boolean inserirCliente( String nome, Date dataNasc, String cpf );

}
```

Figura 6. Interface Identificadora de Tarefas.

```
import java.sql.Date;

public class AController implements IdentificadoraDeTarefas {

    private static IdentificadoraDeTarefas controller;

    public boolean inserirCliente(String nome, Date dataNasc, String cpf) {
        Cliente cliente = new Cliente( nome, dataNasc, cpf );
        return cliente.save();
    }

}
```

Figura 7. Classe que implementa a interface gráfica com o usuário.

```
puclic class FrameCliente extends JFrame {

//padrão Absract Factory
private IdentificadoraDeTarefas controller = ControllerFactory.getInstance();

...

}
```

Figura 8. Trecho de código de uma classe da camada de interface gráfica com o usuário.

Evita-se o acoplamento entre classes da camada de interface gráfica com o usuário e classes da camada *Controller* por meio da aplicação do padrão de projeto *Abstract Factory* [Gamma *et al.* 1995]. Uma possível implementação desse padrão é ilustrada na Figura 9. Se for necessário alterar o comportamento da camada *Controller*, somente a linha que cria uma instância da classe *AController* é alterada. Assim, deve-se fornecer a classe que realiza a funcionalidade pretendida para que se comunique com as classes da camada de negócios (*Model*).

```
public class ControllerFactory {

    private static IdentificadoraDeTarefas instancia;

    public static synchronized IdentificadoraDeTarefas getInstance() {
        if ( instancia == null ) {
            instancia = new AController();
        }
        return instancia;
    }

}
```

Figura 9. Implementação do padrão Abstract Factory [Gamma *et al*. 1995].

Devido à forma de implementação utilizada, a interface gráfica da primeira versão do sistema da locadora de DVD pôde ser acoplada à segunda versão sem modificação em seu código. A Figura 10 mostra a tela referente à transação de locação de DVDs da interface gráfica das duas versões do sistema da locadora de DVDs.

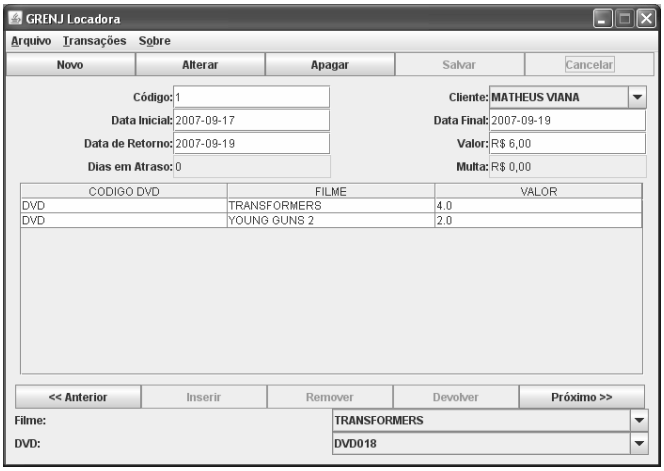

Figura 10. Interface gráfica reutilizada nos dois sistemas de informação.

5. Trabalhos Relacionados

Alguns dos trabalhos encontrados na literatura tratam do desenvolvimento de sistemas de informação com a utilização isolada ou de frameworks ou linguagens de padrões [Brugali e Sycara 2000] [Braga 2002] [Braga e Masiero 2002]. Nesses trabalhos seus autores evidenciam os benefícios em utilizar uma linguagem de padrões de análise, como por exemplo, para que a qualidade do sistema seja garantida. Entretanto, não foram encontrados trabalhos relacionados ao aqui realizado, ou seja, sistemas de informação desenvolvidos com base em linguagem de padrões de análise e frameworks e que ocorra reúso de interfaces gráficas com o usuário.

Os frameworks Struts [] e Spring [] são implementações *open source* do padrão MVC utilizadas por diversas organizações empresariais. Diferentemente do que foi feito neste trabalho, esses frameworks poderiam auxiliar a implementação de sistemas de informação que serão disponibilizados na *web*. Já o framework GRENJ apóia na implementação de sistemas de informação *stand-alone*.

6. Considerações Finais e Trabalhos Futuros

Este artigo mostrou duas versões de um sistema de informação, no domínio de gestão de recursos de negócio, utilizando a linguagem de padrões GRN e o framework GRENJ, com a reutilização de uma mesma interface gráfica com o usuário.

O desenvolvimento de sistemas de informação utilizando abordagens não apoiadas por frameworks requer do desenvolvedor total responsabilidade quanto às atividades de análise e projeto. Já com o uso do framework GRENJ, essas atividades são apoiadas pelos padrões da GRN. Ou seja, os padrões fornecem os atributos e métodos básicos, ficando a cargo do desenvolvedor somente o que é específico da aplicação e não fornecido pelo framework, o que reduz o tempo de desenvolvimento e a inserção de erros.

Além de proporcionar o reúso, a GRN e o framework GRENJ induzem os desenvolvedores a se preocuparem com o projeto e a arquitetura do sistema, construindo códigos mais organizados, uma vez que torna mais clara a divisão entre as camadas do sistema. Tal divisão facilitou a aplicação do padrão de arquitetura *Model-View-Controller*, o que tornou as camadas mais independentes e reutilizáveis. Essa

abordagem também permite que haja maior facilidade na manutenção de um sistema, uma vez que as camadas podem ser modificadas sem afetar as demais. Outra vantagem, proporcionada pelo uso do padrão MVC, é o melhor entendimento do código por parte dos desenvolvedores, pois apenas a menção do nome de um padrão conhecido já remete ao entendimento dos detalhes mais importantes sobre sua implementação.

A inserção de interfaces gráficas no framework GRENJ e a implementação de outros padrões que compõem a GRN, como os que cuidam de venda, manutenção, são alguns dos trabalhos futuros. Além disso, pretende-se construir um *wizard* para facilitar a instanciação de aplicações de sistemas de informação no GRENJ bem como o estudar a possibilidade de usar um framework transversal em sua camada de persistência. Esses trabalhos visam aumentar a manutenibilidade do GRENJ e permitir que diferentes sistemas gerenciadores de bancos de dados possam ser utilizados para gerar sistemas de informações.

Referências

Braga, R. T. V. (2002). Um processo para construção e instanciação de frameworks baseados em uma linguagem de padrões para um domínio específico. Tese de doutorado, ICMC/USP, São Carlos - SP.

Braga, R. T. V., Germano, F. S. R., e Masiero, P. C. (1999). A Pattern Language for Business Resource Management. Proceedings of the Annual Conference on Pattern Languages of Programs, 6:1–33.

Braga, R. T. V. e Masiero, P. (2002). A Process for Framework Construction Based on a Pattern Language. Computer Software and Applications Conference, 2002. COMPSAC 2002. Proceedings. 26th Annual International.

Brugali, D. e Sycara, K. P. (2000). Frameworks and Pattern Languages: an Intriguing Relationship. ACM Computing Surveys.

Cunningham, H., Liu, Y., e Zhang, C. (2006). Using Classic Problems to Teach Java Framework Design. Science of Computer Programming, 59.

Durelli, V. H. S. (2007). Reengenharia Iterativa do Framework GREN. Documento de qualificacão, Universidade Federal de São Carlos, São Carlos – SP.

Fayad, M. e Schmidt, D. C. (1997). Object-oriented application frameworks. Communications of the ACM, 40(10):32–38.

Fowler, M. (2003). Patterns. Software, IEEE, 20:2–3.

Frakes, W. e Kyo, K. (2005). Software reuse research: status and future. IEEE Transactions on Software Engineering, 31:529–536.

Gamma, E., Helm, R., Johnson, R., e Vlissides, J. (1995). Design Patterns: Elements of

Reusable Object-Oriented Software. Addison-Wesley.

Harrison, N. B., Avgeriou, P., e Zdun, U. (2007). Using Patterns to Capture Architectural Decisions. IEEE Software, 24(4):38–45.

Johnson, R. E. (1997). Frameworks = (components + patterns). Communications of the ACM, 40.

Krueger, W. C. (1992). Software Reuse. ACM Computing Surveys, 24(2):131–183. Manolescu, D., Kozaczynski, W., Miller, A., e Hogg, J. (2007). The Growing Divide in the Patterns World. IEEE Software, 24(4):61–67.

MySQL (2007). “mysql”. http://www.mysql.com/. Acessado 11 de Setembro de 2007.

Schmidt, D. C., Fayad, M., e Johnson, R. E. (1996). Software patterns. Communications of the ACM, 39(10):37–39.

Shalloway, A. e Trott, J. (2001). Design Patterns Explained: A New Perspective On Object-Oriented Design. Addison-Wesley.

Shiva, S. G. e Shala, L. A. (2007). Software Reuse: Research and Pratice. Fourth International Conference on Information Technology, pages 603–609.

Wirfs-Brock, R. (2006). Refreshing Patterns. Software, IEEE, 23:45–47.