Refatorando o SimGrIP: um estudo de caso acerca da aplicação de técnicas de refatoração de software

Carlos Lima[1]
Gabriela Souza
Luiz Chaves
Frederico Pereira
Marcelo Siqueira
Centro Federal de Educação Tecnológica da Paraíba (CEFET-PB)
diegoquirino@gmail.com
gabriguedes@gmail.com
lucachaves@gmail.com
marcelojose@gmail.com
siqueira@gmail.com
fred@cefetpb.edu.br

Resumo: *A demanda dos clientes pela integração de mais funcionalidades ao software adquirido implica no aumento do tamanho e da complexidade do código fonte do programa de computador, prejudicando sua legibilidade e deteriorando sua estrutura inicial. Os padrões de projeto e as técnicas de refatoração servem de guia para o desenvolvedor na sua tarefa de manutenção do código. Com a finalidade de melhorar a estrutura interna do software SimGrIP, realizou-se a refatoração parcial de seu código, reestruturando uma de suas funcionalidades, a fim de se justificar a utilização desta técnica ao se comparar o código final e inicial, com base em critérios como legibilidade e acoplamento, constatando sua eficiência no processo de revitalização do software.*

Palavras-chave: *refatoração de código, padrões de projeto, manutenção de código, SimGrIP.*

Abstract: *The customers request for the integration of more functionalities to the acquired software, results in the growth of size and complexity of the font code of the computer program, damnifying its legibility and damaging its initial structure. Design patterns and refactoring techniques serve as guides for the developers in their code maintenance task. With the purpose to increase the internal structure of the SimGrIP software code, a partial refactoring of its code, restructuring one of the functionalities, intending to justify the use this technique by comparing the final and initial code, based in criteria such as legibility and linkage, confirming its efficiency in the process of software revitalization.*

Key-words: *code refactoring, design patterns, code maintenance, SimGrIP.*

[1] Autor a quem toda correspondência deverá ser endereçada

1. Introdução

A área de desenvolvimento de software apresenta avanços substanciais, que na maioria das vezes, se apóiam no surgimento de novas tecnologias e metodologias de desenvolvimento empregadas para dar o suporte necessário à adequação das ferramentas existentes no mercado às vontades e anseios dos clientes de software.

Nesta perspectiva, a **refatoração de código** surge como um agente corretivo, capaz de ocasionar um melhoramento na estrutura do código de uma aplicação de maneira tal que a mesma possa ter condições de ser modificada tantas vezes quantas o cliente altere suas regras de negócio, sem que se altere a estrutura visual do programa e de maneira menos custosa ao programador.

Utilizar **padrões de projetos** é fundamental. A estrutura codificada do programa necessita seguir um padrão de construção, de chamada de métodos, de regras fixas para comunicação entre objetos, assim como persistir corretamente os dados manipulados.

Tantos fundamentos levaram ao estudo da estrutura de código do **SimGrIP**, um software de caráter educativo cujo objetivo é simular redes IP para fins didáticos de ensino de conceitos chaves de roteamento IP. Este software está parcialmente implementado, o que inclui a construção gráfica e de negócio de hosts, roteadores e links, além da apresentação para o usuário (janelas, botões, entre outros). Porém, a simulação do encaminhamento dos pacotes não estava previamente concluída e diante deste fato foi feito um levantamento inicial dos principais pontos críticos da estrutura de código existentes na ferramenta nas classes e interfaces, ora implementadas para só então decidir-se sobre o estudo de caso (funcionalidade) a ser refatorado.

O embasamento teórico desta pesquisa e as soluções vitais para o melhoramento da infra-estrutura de código do SimGrIP levam em consideração o conceito de refatoração e os exemplos propostos por Martin Fowler, em seu livro intitulado "*Refactoring: Improving the Design of Existing Code*", assim como a aplicação de padrões de projetos propostos por Erich Gamma, Richard Helm, Ralph Johnson e John Vlissides, no livro "*Design Patterns: Elements of Reusable Object-oriented Software*" e por Eric e Elisabeth Freeman, no livro "*Head First Design Patterns*". Outras fontes bibliográficas, como Willian C. Wake e consultas informativas à internet também foram utilizados, com o intuito de esclarecer e pormenorizar os estudos propostos, tudo isto para apontar os erros na estrutura de código do SimGrIP e corrigir as falhas identificadas no software em análise.

2. Metodologia

O SimGrIP foi desenvolvido pelos estudantes Ricardo do Amaral Nóbrega, Bárbara de Assis Xavier, Juan Damasceno e Ramon Borges, orientados pelo professor Frederico Costa Guedes Pereira do Centro Federal de Educação Tecnológica da Paraíba.

Recentes necessidades de expansão encontraram impasse e dificuldade de serem operacionalizadas. Isso evidenciou necessidade de refatoração. Inicialmente, foi feita uma análise do código e a equipe levantou as informações necessárias para discutir mais detalhadamente a melhor forma de começar a reestruturar a estrutura de código do sistema.

Na fase seguinte, feito o levantamento dos pontos críticos da ferramenta e, escolhido o estudo de caso, iniciaram-se as aplicações das técnicas de refatoração e da utilização de padrões de projeto para revitalizar uma funcionalidade da estrutura de código do SimGrIP: **a interface de apresentação para o usuário** (ou seja, a janela principal do sistema). Esta etapa foi realizada em laboratório, utilizando o ambiente de programação do Eclipse, versão 3.2.2 e a plataforma de desenvolvimento J2SE 6.0 (JAVA), além de um repositório de dados (CVS), hospedado nos servidores do *Projeto Java.net*, o que possibilitava interação permanente entre a equipe de refatoração.

Atualmente, o sistema se encontra com uma funcionalidade refatorada e com outros pontos de refatoração detectados. Executadas as devidas correções, poder-se-á inicializar a fase de testes para validação. Após esta etapa, o SimGrIP finalmente poderá ser utilizado pelos clientes.

3. Conceitos básicos

3.1. Refatoração de código

Refatoração de código é a arte de transformar um código já existente e problemático, às vezes caótico, em algo legível e de fácil manutenção sem, contudo, alterar a aparência e o funcionamento externo do software.

A última consideração da definição anterior é bastante frisada por Fowler (1999) em sua afirmação "[...] refatoração não muda o comportamento observável do software. O software ainda realiza a mesma função que realizava antes."

Refatorar significa dar nova vida ao código, adequá-lo aos padrões. Quando um código é bem estruturado, é mais fácil de manipulá-lo, de alterá-lo

e de se encontrarem possíveis falhas. E foi com esse objetivo que surgiram as técnicas de refatoração.

3.2. Técnicas de refatoração

Até agora foram citados os benefícios da refatoração, entretanto pode ser perigosa se praticada precipitadamente, sem antes fazer-se um levantamento geral dos principais problemas do código a ser alterado.

Primeiro, deve-se olhar o código como um todo, procurar os principais problemas e trabalhar um de cada vez. A refatoração deve ser feita, através de pequenas mudanças e, a cada mudança, testes devem ser feitos com o software para garantir que os resultados obtidos sejam sempre os mesmos. Fowler (1999) fala que *"[...] o efeito cumulativo dessas pequenas mudanças pode melhorar radicalmente o projeto."*

Existem diversos livros, manuais e tutoriais que ajudam o programador a encontrar os principais erros de um código. Além disso, a maioria deles fornece um passo a passo sobre como proceder com a refatoração para cada tipo de erro. As técnicas usadas neste estudo de caso serão citadas a diante.

3.3. Padrões de Projeto

Em seu livro Gamma, Helm, Johnson e Vlissides (1995) fazem a seguinte pergunta: "Quantas vezes você teve um *déjà-vu de um projeto – aquele sentimento de que você resolveu o problema antes, mas não sabe exatamente onde ou como?"*

Foi para permitir que os desenvolvedores se lembrem de uma solução e possam reutilizá-la, que existem os padrões de projetos. Os padrões de projetos descrevem os problemas e suas soluções, para que estas possam ser utilizadas sempre que necessário.

Em geral, dois aspectos preocupam o desenvolvedor (ou a equipe de desenvolvedores) quanto à escolha do padrão de projeto adequado: a *metodologia*, para o desenvolvimento de sistemas e a *linguagem de modelagem*, para o projeto de software orientado a objetos. É plausível que a *dificuldade de combinação* de todos os elementos que fazem parte do projeto de um sistema seja pertinente, mas não deve ser fruto da não escolha ou escolha de uma metodologia de desenvolvimento de sistemas inadequada e da má utilização da linguagem de modelagem.

Assim sendo, **padrões de projeto** constituem estruturas recorrentes no projeto de software orientado a objetos com a finalidade de guiar a metodologia de desenvolvimento, criando padronizações que devem ser obedecidas por todos os integrantes da equipe de desenvolvimento. Eles *nomeiam*, *abstraem* e *identificam* aspectos chave de uma estrutura de projeto, tornando-a útil e adaptável toda a estrutura de código.

4. Importância da refatoração de código

A importância da refatoração consiste em, cada vez, mais agilizar o processo de manutenção e facilidade de extensão. Como o desenvolvimento de software está associado a uma grande variabilidade de concretização, então podem ser encontradas muitas soluções, e em grande parte, soluções ruins. E isso, às vezes, acaba tornando-se uma constante em equipes de desenvolvimento de software ainda inexperientes, que geralmente são cobradas pelo tempo e custo, o que acaba acarretando falta de modelagem e planejamento de boas maneiras de implementação dos requisitos do sistema. E esse fato é atestado por Fowler, que cita: *"Um número significativo de projetos inadequados de programas têm sido criado por desenvolvedores menos experientes, resultando em aplicações que são ineficientes e de difícil manutenção e extensão"*.

Para evitar isso, a refatoração é composta de técnicas, no geral, são simples, pois uma pequena modificação no sistema, que não altere o seu comportamento funcional, já garante uma grande melhoria. E elas se fortalecem, principalmente, se forem baseadas em algumas mudanças de qualidade não-funcionais, ou seja, simplicidade, flexibilidade, clareza, desempenho. Alguns dos seus exemplos são:

A mudança de trechos de códigos para outras áreas; a exclusão de trechos repetitivos ou redundante; a mudança de alguns trechos para um escopo de algum método, ou de um único método; o deslocamento de certas características para uma hierarquia maior (generalização). Fazer um detalhamento dos requisitos do sistema a fim de se obter uma melhor modelagem do código, pois com a introdução dos requisitos é que se parte para o problema; renomear algumas variáveis, métodos ou classes com o objetivo de manter uma maior correspondência com seu sentido real; aplicar padrões de projeto para garantir-lhe modularização.

Assim o seu sucesso dessas aplicações chegará a proporções que garantem bons resultados em regras de projetos de desenvolvimento como a do XP, que sempre garante um aperfeiçoamento da estruturação do código com o grande detalhamento de seus requisitos e designer.

5. Estudo de caso: refatoração da interface de apresentação ao usuário

5.1. Análise preliminar

Analisando o código do software SimGrIP e seu diagrama de classes, notaram-se alguns pontos que necessitavam de refatoração. Dentre eles pode-se citar:

A classe Application, que sozinha montava, controlava e tratava os elementos de interface. Isso a tornava uma estrutura monolítica; uma classe interna à classe Application para manipular arquivos *xml*.
Nomes de classes não condizentes com seus papéis (ver Figura 1).

Além da repetição de código dentro da classe Application. Tudo isso somado à ausência de comentários e documentação, tornava o SimGrIP em um software difícil de ler e a inserção das outras funcionalidades numa tarefa muito custosa.

Optou-se por fazer a refatoração na classe Application, pois esta, por concentrar um grande número de funções, contava com um número excessivo de classes internas – num total de dez – e toda a criação do frame de apresentação da interface estava contida em apenas um método. O arquivo que continha esta classe era enorme, com mais de mil e quinhentas linhas (ver Figura 2).

5.2 Utilizando padrões

Antes de refatorar o SimGrIP, fez-se uma análise para determinar quais padrões de projeto poderiam ser introduzidos para dar mais flexibilidade ao projeto. Decidiu-se por utilizar os padrões de acordo com o que segue demonstrado na

Tabela 1.

Tabela 1 - Descrição de padrões e seus usos no SimGrIP

Padrão	Utilização
Factory Method	Este padrão foi usado para flexibilizar a criação de componentes de interface, como botões, menus e painéis. Pode-se verificar sua implementação nas classes do pacote *br.edu.cefetpb.simgrip.gui* cujo nome termina com o palavra "Factory".
Command	Padrão utilizado no tratamento de eventos dos componentes da interface. As classes do pacote *br.edu.cefetpb.simgrip.gui.listener* implementam este padrão.
Façade	O padrão Façade foi usado para a criação e manipulação de todo o frame contendo os componentes da interface. A classe ApplicationGuiFacade é a "fachada" da aplicação, ou seja, ela é a interface de acesso aos componentes do frame.

5.3 Aplicando técnicas de refatoração.

Tendo os padrões definidos, foi necessário aplicar em algumas técnicas de refatoração para que se implementassem estes padrões. Estas técnicas estão descritas na

Tabela 2.

Tabela 2 - Descrição de técnicas de refatoração e seus usos no SimGrIP

Técnica de Refatoração	Utilização
Extract Class	Técnica que explana os procedimentos necessários para a extração de uma classe de dentro da outra, utilizada no SimGrIP para extrair as classes internas e fábricas.
Extract Method	Técnica que descreve como extrair métodos menores e concisos de um método extenso que faz várias tarefas. Foi usada para dividir o método *main(String args[])* da antiga classe Application em métodos menores, que foram inseridos nas fábricas, responsáveis por criar componentes específicos.
Move Method	Técnica que explica como mover métodos de uma classe para outra. Usada para mover os métodos de criação, previamente construídos a partir do uso da técnica *Extract Method*, para dentro das fábricas.
Inline Temp	Técnica usada para diminuir a criação de variáveis temporárias. Foi utilizada nas fábricas de painéis e menus, nas chamadas dos métodos de adição de componentes.

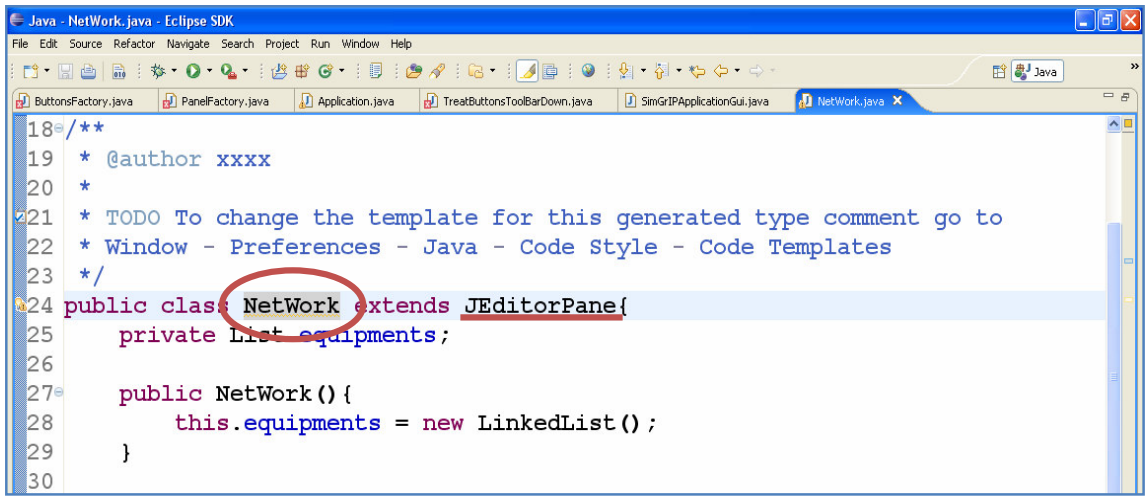

Figura 19 - Nomes de classes devem ser claros nas devidas especificações.

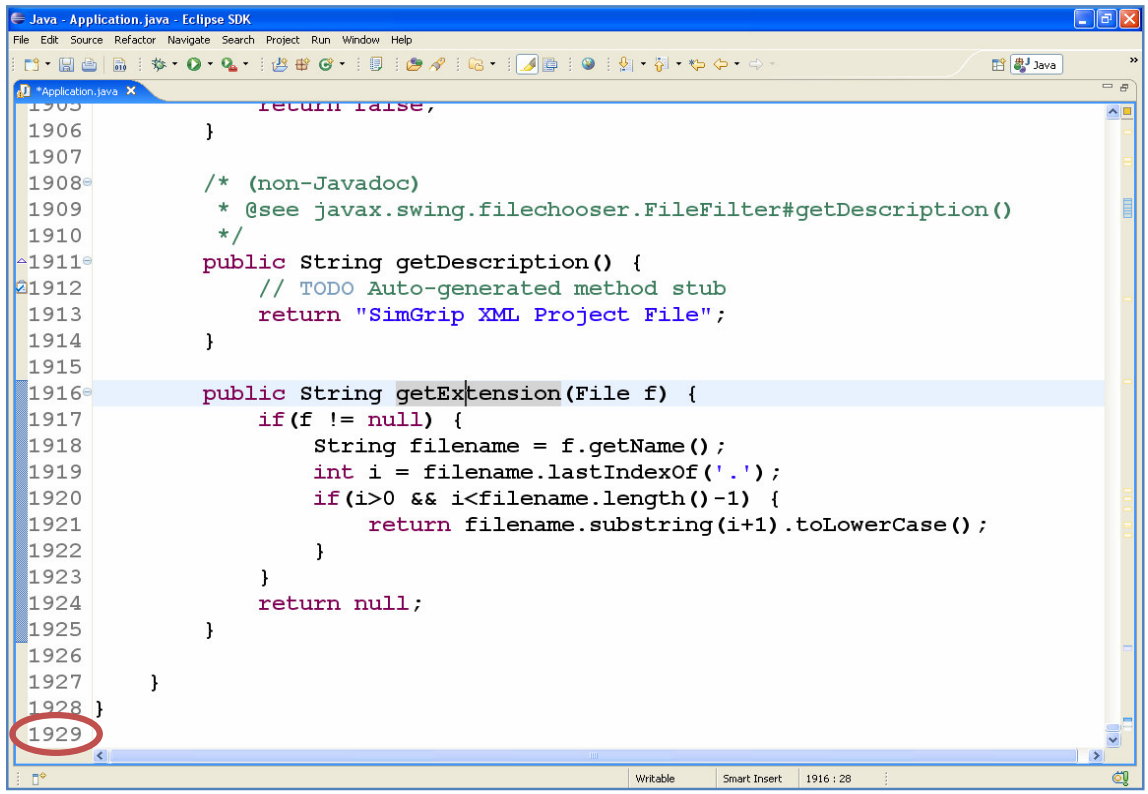

Figura 20 - Trecho de código do SimGrIP

5.4 O produto final

O SimGrIP conta com dois novos pacotes, um para criação da *Graphical User Interface* (GUI), outro para tratá-la. Há uma melhor divisão de tarefas, uma modularização maior (Ver Figura 21).

O número de classes e pacotes aumentou, conseqüentemente e aumentou também o tamanho do diagrama de classes. Porém, a inserção, modificação e retirada de componentes da interface foi facilitada, pois agora, com o código bem dividido, sabe-se qual trecho lida com tais componentes e qual trecho modificar.

É necessário salientar que a interface do usuário continua a mesma, apesar de toda a modificação na

estrutura interna. E é assim que a refatoração deve ser feita.

5.5 Pontos Críticos de Implementação / Refatoração Futura.

O ponto de partida para a refatoração feita foi uma classe apenas, *Application*, mas existem outras que não estão bem projetadas e devem ser alvos de refatorações futuras, como:

Classes de modelo que deveriam apenas implementar a lógica do negócio, mas que ainda estão acessando e modificando elementos visuais diretamente, quebrando o padrão de design MVC.

Métodos extensos, que implementam várias tarefas, cuja lógica pode ser dividida em um número maior de métodos.

Classes que possuem métodos com comandos *switch* extensos e repetitivos.

Uma vez que estes problemas sejam solucionados, o software estará pronto para a adição das outras funcionalidades previstas anteriormente, como a geração automática de endereços IP para uma rede e o gerenciamento automático de tabelas de roteamento, por exemplo.

6. Conclusões e perspectivas futuras

Dadas as necessidades de expansão, manutenção e legibilidade do código do software SimGrIP, a refatoração aplicada possibilitou uma melhor adequação da funcionalidade modificada aos seus requisitos funcionais. Isso beneficia desenvolvedores e clientes. Os primeiros, por haver mais segurança operacional. O segundo, por poder usufruir das facilidades da ferramenta e ter uma resposta mais eficiente às mudanças necessárias.

O produto da refatoração, embora não afete a interface gráfica com o usuário, ocasiona mudanças de implementação acentuadas, que dão maior confiabilidade a estrutura de código da aplicação.

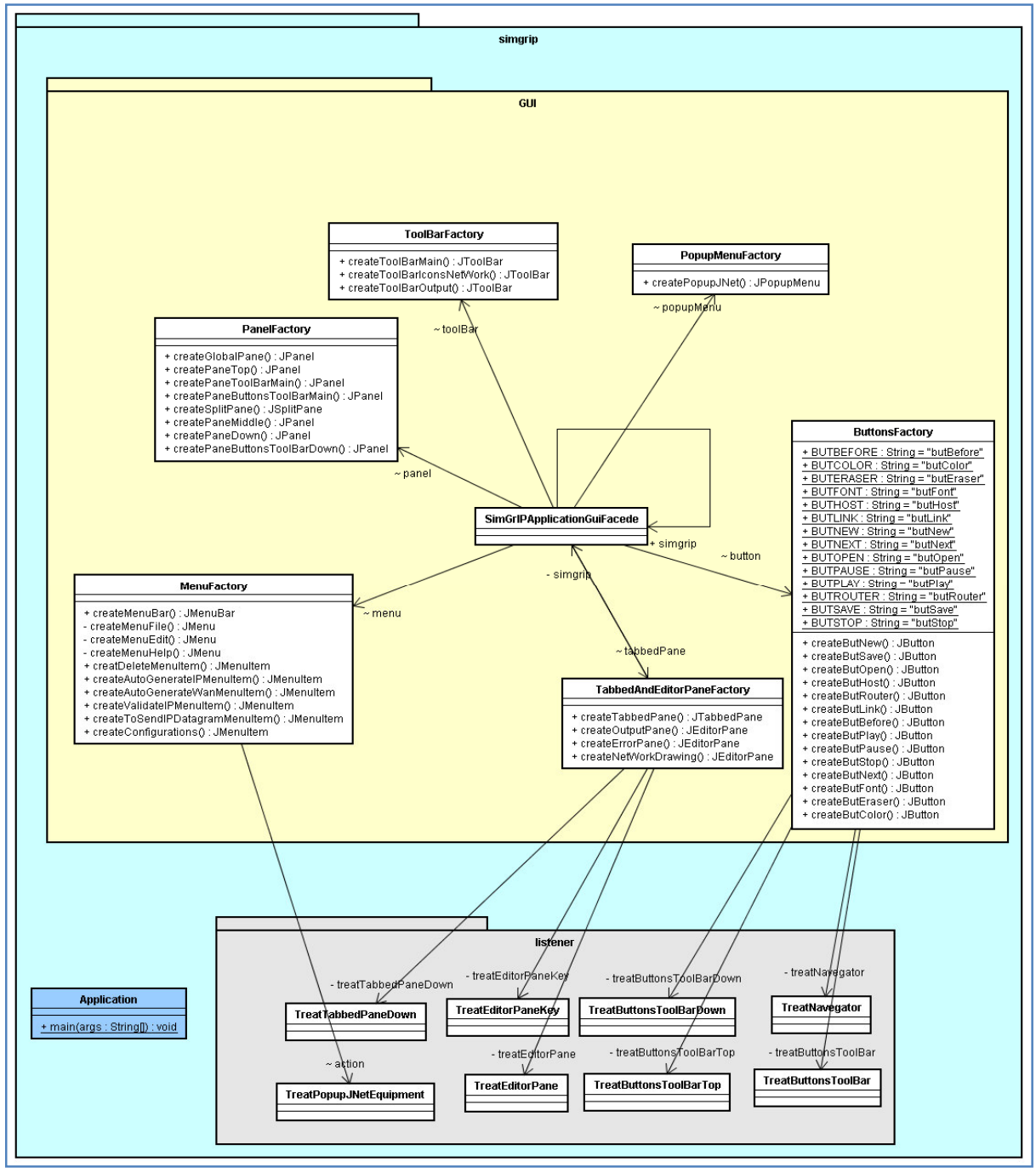

Figura 21 - Novo Diagrama de Classes da Aplicação que envolve a parte refatorada.

Utilizados os conceitos e técnicas de refatoração, obteve-se uma funcionalidade do software mais adequável e manutenível. A mesma poderá ter elementos incluídos com maior facilidade, ou seja, se tornará estendível, atualizável. Esses resultados estimulam a aplicação das técnicas de refatoração nas demais funcionalidades do sistema, onde foram detectados pontos críticos (item 0), passíveis de manutenção.

Portanto, como requisito vital da engenharia de software, a finalização da **refatoração do SimGrIP** contribuirá de maneira substancial à sua sobrevivência e, assim, uma vez que sua estrutura codificada se encontre mais legível, organizada e desacoplada, responder-se-á de maneira mais rápida e eficiente às mudanças de requisitos dos seus clientes.

7. Referências

FOWLER, M.; BECK, K.; BRANT, J.; OPDYKE, W.; ROBERTS, D. **Refactoring: Improving the Design of Existing Code**. Addison Wesley: USA, 2003.

FREEMAN, E; FREEMAN, E. **Head First Design Patterns**. 1ª ed. O'Reilly Media: USA, 2004.

GAMMA, Erich; HELM, Richard; JOHNSON, Ralph & VLISSIDES, John **Design Patterns: Elements of Reusable Object-Oriented Software**. Addison Wesley: USA, 1995.

WAKE, William C. **Refactoring Workbook**. Addison Wesley: USA, 2003.

Responsabilidade de autoria

As informações contidas neste artigo são de inteira responsabilidade de seus autores. As opiniões nele emitidas não representam, necessariamente, pontos de vista da Instituição e/ou do Conselho Editorial.