UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
CENTRO DE CIÊNCIAS AGRÁRIAS
DEPARTAMENTO DE COMPUTAÇÃO

LEANDRO FERNANDES DOS SANTOS

EVOLUÇÃO DA FERRAMENTA DE AUXÍLIO À PROTOTIPAÇÃO DE INTERFACES PROFACE

ALEGRE – ES
2014

LEANDRO FERNANDES DOS SANTOS

EVOLUÇÃO DA FERRAMENTA DE AUXÍLIO À PROTOTIPAÇÃO DE INTERFACES PROFACE

Trabalho de Conclusão de Curso apresentado ao Departamento de Computação do Centro de Ciências Agrárias da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.
Orientador.: Profª. M.Sc. Juliana Pinheiro Campos Pirovani

ALEGRE - ES
2014

LEANDRO FERNANDES DOS SANTOS

EVOLUÇÃO DA FERRAMENTA DE AUXÍLIO À PROTOTIPAÇÃO DE INTERFACES PROFACE

Trabalho apresentado ao Departamento de Computação do Centro de Ciências Agrárias da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Aprovado em 10 de Dezembro de 2014.

BANCA EXAMINADORA

Profª. M.Sc. Juliana Pinheiro Campos Pirovani
Universidade Federal do Espírito Santo
Orientadora

Prof. Dr. Bruno Vilela Oliveira
Universidade Federal do Espírito Santo

Prof. M.Sc. Edmar Hell Kampke
Universidade Federal do Espírito Santo

AGRADECIMENTOS

Agradeço a Deus pela vida e por manter em mim viva a certeza de que a caminhada seria longa, mas a vitória certa.

Aos meus pais, Getulio Fernandes dos Santos e Suely Aparecida Vieira Fernandes, meus maiores exemplos de caráter e boa conduta, e toda a minha família. Vocês são presentes de valor incalculável em minha vida. Obrigado por todo apoio material e emocional proporcionados durante todo o curso, sem vocês eu não chegaria até aqui.

À minha orientadora, Profª. M.Sc. Juliana, por todo apoio e empenho para que este trabalho se tornasse concreto.

Aos professores do Departamento de Computação (DCOMP). Todo conhecimento transmitido foi muito além das dimensões da sala de aula, serei eternamente grato.

À minha amada namorada e futura esposa Ludimila Pimenta Alves, pela paciência e compreensão, e por todo o zelo que tens pelo nosso Amor. Eu Te Amo.

Aos amigos de república e demais amigos da UFES, pelos momentos de descontração, pelas muitas horas de estudo e o conhecimento compartilhado.

RESUMO

A identificação e validação de requisitos são fases críticas no processo de desenvolvimento de *software*. É necessário um excelente planejamento e uma constante troca de informações entre os envolvidos para evitar que erros sejam introduzidos no sistema. Os protótipos são fortes aliados nessas fases do desenvolvimento. Eles facilitam a escolha e validação dos requisitos, dando suporte a uma tomada de decisão mais efetiva nas etapas seguintes do desenvolvimento. O objetivo do presente trabalho é a evolução do prototipador de interfaces Proface 3.0, visando tornar possível a criação de protótipos no estilo *Windows* 7. Inicialmente, foram feitas algumas refatorações no código atual da ferramenta, melhorando sua estrutura interna e possibilitando que a evolução fosse realizada mais facilmente.

Palavras chave: Proface, Prototipação, Refatoração.

SUMÁRIO

LISTA DE FIGURAS i
1 INTRODUÇÃO 8
1.1 O PROBLEMA E SUA IMPORTÂNCIA 9
1.2 OBJETIVOS 10
1.2.1 OBJETIVO GERAL 10
1.2.2 OBJETIVOS ESPECÍFICOS 10
2 REVISÃO DE LITERATURA 11
2.1 PROTOTIPAÇÃO DE *SOFTWARE* 11
2.2 REFATORAÇÃO DE *SOFTWARE* 14
2.3 O PROTOTIPADOR DE INTERFACES PROFACE 16
2.4 TRABALHOS CORRELATOS 20
2.4.1 PENCIL PROJECT 21
2.4.2 MOCKINGBIRD 23
2.4.3 CACOO 26
2.4.4 AXURE RP 28
2.4.5 SÍNTESE DOS TRABALHOS CORRELATOS 31
3 METODOLOGIA 33
4 RESULTADOS OBTIDOS 36
4.1 REFATORAÇÃO DO CÓDIGO DO PROFACE 36
4.2 IMPLEMENTAÇÃO DA NOVA PALETA DE COMPONENTES 43
5 CONCLUSÃO 52
6 REFERÊNCIAS 53
APÊNDICE A - Conjunto de testes realizados antes e após o processo de refatoração 55
APÊNDICE B - Conjunto de testes realizados após a evolução do Proface 56
APÊNDICE C – Classe Componente com todos os seus atributos e métodos 57

LISTA DE FIGURAS

Figura 1: Processo de desenvolvimento de protótipo 12
Figura 2: Interface Proface 3.0 17
Figura 3: Protótipo de tela estilo Genérico - Proface 3.0 18
Figura 4: Protótipo de tela estilo Windows 98 - Proface 3.0 19
Figura 5: Protótipo de tela estilo Windows XP - Proface 3.0 19
Figura 6: Protótipo modelo 21
Figura 7: Interface Pencil Project 22
Figura 8: Modelo protótipo elaborado no Pencil Project 23
Figura 9: Interface Mockingbird. 24
Figura 10: Modelo de protótipo elaborado no Mockingbird 25
Figura 11: Interface Cacoo 27
Figura 12: Modelo de protótipo elaborado no Cacoo 28
Figura 13: Interface Axure RP Pro 29
Figura 14: Modelo de protótipo elaborado no Axure 30
Figura 15: Diagrama de atividades usado para guiar o processo de refatoração. 33
Figura 16: Trecho de código duplicado dentro do método selectColor() 36
Figura 17: Trecho do método selectColor() na classe BorderColorEditor 37
Figura 18: Aplicação do *Extract Method* na classe ColorEditor 37
Figura 19: Método selectColor() da classe BorderColorEditor após refatoração 37
Figura 20: Constantes da classe utilitária Utils 38
Figura 21: Método accept(File f) da classe ImageFilter 39
Figura 22: Arrays e campos criados na classe Utils 39
Figura 23: "Método accept(File f)" da classe "ImageFilter" após a classe "Utils" ser refatorada 40
Figura 24: Parte do método mouseReleased(MouseEvent event) antes da aplicação da refatoração Consolidade *Conditional Expression* 40
Figura 25: Método criado na classe PainelPrototipacao para a solução do *bad smell* encontrado 41
Figura 26: Método getIcon(File f) da classe ImageFileView. 41
Figura 27: Métodos criados na classe ImageFileView para a aplicação da refatoração *Decompose Conditional* 42

Figura 28: Método getIcon(File f) da classe ImageFileView após refatoração *Decompose Conditional* 42
Figura 29: Diagrama de classes dos components.....44
Figura 30: Parte do arquivo "*.xml" usado para a geração da nova paleta no estilo *Windows* 7.....45
Figura 31: Parte principal da rotina implementada para efetuar a leitura do arquivo .xml.....46
Figura 32: Parte do método paintComponent(Graphics gp) da classe PBotaoWin7 46
Figura 33: Métodos da classe PJanelaWin7 47
Figura 34: Parte do método insereComponente(Componente object) da classe PainelPrototipacao 48
Figura 35: Tela principal Proface 4.0.....50
Figura 36: Protótipo de tela criado no Proface 4.0 com componentes estilo *Windows* 7 50
Figura 37: Protótipo janela de transferência de arquivos estilo *Windows* 7 criado no Proface 4.0.....51
Figura 38: Classe componente com todos os atributos e métodos 57

1 INTRODUÇÃO

O contexto do processo de desenvolvimento de *software* é altamente dinâmico. Em meio a esse dinamismo, os envolvidos nesse processo, também chamados de *stakeholders*, lidam com uma série de questões que irão determinar o sucesso ou não na conclusão do produto final. Para alcançar o objetivo na construção de um *software* e obter sucesso, deve-se atender a um conjunto de requisitos. Segundo Sommerville (2003), os requisitos são as funcionalidades que um sistema de *software* deve possuir e sobre quais restrições ele deve operar.

O processo de levantamento e análise de requisitos é uma das fases da engenharia de requisitos que constitui um dos cernes do processo de desenvolvimento de *software*. É particularmente importante evitar erros nessa fase, uma vez que estes podem se propagar por todo o processo e causar problemas posteriores no projeto e implementação do sistema.

Sommerville (2003) descreve que, na atividade de extração de requisitos, os analistas devem trabalhar diretamente com o cliente e usuários finais do sistema para descobrir mais informações sobre o domínio da aplicação, serviços que o sistema deve ou não fornecer, desempenho exigido do sistema, restrições de *hardware*, etc. Alguns riscos relacionados a essa atividade são: requisitos não identificados, requisitos que não são bem compreendidos e são especificados de forma errada, alterações nos requisitos, etc. Esses podem levar a atrasos em cronogramas e erros na avaliação de custos.

A técnica de prototipação de *software* pode ser usada para auxiliar a fase de levantamento de requisitos e minimizar os riscos associados a essa atividade. Ela consiste na construção de modelos para testar o sistema a ser implementado, visando levantar e validar requisitos em seu processo de desenvolvimento. A prototipação de interfaces permite, por exemplo, que os usuários tenham uma ideia inicial de como será a interface do *software* desenvolvido. Dessa forma, eles podem dizer se as funcionalidades necessárias estão contempladas no modelo apresentado, podem falar sobre a sua interação com os protótipos e se os passos

executados para realizar uma atividade estão corretos. Pressman (2002) destaca que a prototipação auxilia o engenheiro de *software* e o cliente a compreenderem melhor o que precisa ser construído quando os requisitos ainda não estão muito claros.

Algumas ferramentas CASE (*Computer Aided Software Engineering*) auxiliam a criação de protótipos. Ferramentas denominadas CASE são aquelas usadas para auxiliar gerentes e profissionais de engenharia de *software* em toda a atividade associada ao processo de desenvolvimento de *software* (Pressman, 2002).

1.1 O PROBLEMA E SUA IMPORTÂNCIA

Para permitir um desenvolvimento rápido do protótipo, a ferramenta utilizada deve ter uma interface simples e objetiva. Existem muitas ferramentas para prototipação no mercado como Cacoo[1], Mockingbird[2], Axure RP[3] e Pencil[4]. Muitas das vezes, para que o usuário tenha acesso a todos os recursos disponibilizados por estas ferramentas é cobrado um valor. Sendo assim nem todos os usuários têm condições de adquiri-las para acessar os seus recursos na íntegra.

A ferramenta de prototipação Proface[5] é um *software* disponibilizado de forma inteiramente gratuita, possibilitando que todos tenham acesso à totalidade de seus recursos, inclusive estudantes e profissionais de tecnologia. Ela foi desenvolvida na Universidade Federal de Viçosa (UFV) e atualmente está na versão 3.0. O *software* se encontra desatualizado possibilitando somente a construção de interfaces no estilo Windows 98, Windows XP e modo genérico.
Uma das motivações para a evolução dessa ferramenta foi o ótimo retorno obtido dos usuários quanto a sua usabilidade e efetividade no apoio ao processo de

[1]https://cacoo.com/

[2]https://gomockingbird.com/

[3]http://www.axure.com/

[4]http://pencil.evolus.vn/

[5]http://sourceforge.net/projects/proface/

prototipação. Outro motivo são os pedidos dos usuários por uma nova versão da ferramenta.

Além disso, eram necessárias algumas refatorações no código atual da ferramenta. Refatorar consiste em aplicar modificações na estrutura interna do código do programa sem que essas alterem seu comportamento, permitindo que novas modificações possam ser feitas mais facilmente (OPDYKE, 1992). Vários programadores contribuíram no desenvolvimento do Proface e, por isso, não existe um padrão na sua implementação, o que dificulta o entendimento do código. Assim, para a inclusão dos novos recursos propostos, foram realizadas primeiramente algumas refatorações.

1.2 OBJETIVOS

1.2.1 OBJETIVO GERAL

De modo geral, o objetivo do presente trabalho foi a evolução da ferramenta de prototipação de interfaces Proface.

1.2.2 OBJETIVOS ESPECÍFICOS

Os objetivos específicos para a evolução da ferramenta foram:

- Realizar algumas refatorações no código da versão atual do Proface com o objetivo de facilitar as alterações que seriam realizadas nesse trabalho e as modificações futuras;
- Implementar componentes que possibilitem a elaboração de protótipos de interface utilizando telas no estilo do sistema operacional *Windows* 7;
- Testar os recursos implementados.

2 REVISÃO DE LITERATURA

2.1 PROTOTIPAÇÃO DE *SOFTWARE*

Na concepção e desenvolvimento de um novo produto é necessário que as ideias acerca deste sejam abstraídas, de modo a tornar possível a construção de modelos que contemplem suas características essenciais. Essa abstração pode ser traduzida por meio de um protótipo.

O protótipo é um modelo que permite visualizar as características relevantes do sistema antes de implementá-lo. Para Sommerville (2003, p. 145), o "protótipo é uma versão inicial de um sistema de *software* que é utilizada para mostrar conceitos, experimentar opções de projeto e, em geral, conhecer mais sobre os problemas e suas possíveis soluções".

Segundo Preece et al. (2005), um protótipo pode ser qualquer coisa, desde um *storyboard* de papel a uma parte complexa de um *software*, e de uma maquete de cartolina a um pedaço de metal moldado e prensado. O protótipo torna possível a interação dos *stakeholders* com um determinado produto, imaginando e visando adquirir experiências a respeito de como utilizá-lo em um ambiente real, explorando assim os recursos para ele imaginados.

Um protótipo de interface é um desenho da tela, com campos, botões, imagens, *links* e outros componentes, representando como a interface pode vir a ser ou se parecer no *software* final. Eles são úteis, pois podem responder a várias questões durante o processo de desenvolvimento. Ao interagir com um protótipo, os usuários podem dar a sua opinião a respeito das ideias nele apresentadas, podem identificar funcionalidades erradas e esclarecer seu funcionamento, podem perceber que algumas funcionalidades foram esquecidas e proporcionar a melhoria da qualidade do projeto. "Geralmente se diz que os usuários não conseguem transmitir o que querem, mas se vêem algo e começam a utilizá-lo, logo sabem o que não querem" (PREECE *et al.* 2005, p. 260).

Além disso, os protótipos podem ser usados posteriormente para compor a documentação do *software*.

Um modelo proposto para o desenvolvimento de um protótipo de *software* pode ser visto na Figura 1. Nela, são descritas as atividades necessárias no desenvolvimento do protótipo e os artefatos produzidos em cada atividade. O modelo é proposto por Sommerville (2003).

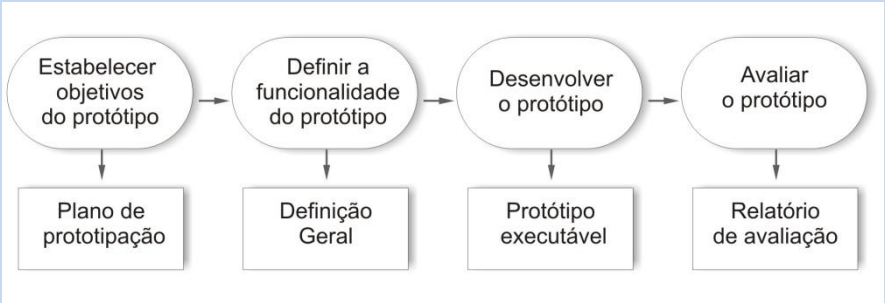

Figura 1: Processo de desenvolvimento de protótipo

Fonte: Sommerville (2003)

Um protótipo pode ser classificado em termos de sua fidelidade. A fidelidade é uma medida relativa do quanto um protótipo se assemelha ao produto que será construído em termos de interação, detalhes, aparência visual, etc. Os protótipos podem ser classificados em baixa-fidelidade, alta-fidelidade e média-fidelidade.

Os protótipos de baixa-fidelidade são desenvolvidos de modo rudimentar e, por isso, não se parecem muito com o produto final. Sua construção envolve materiais que provavelmente não irão compor o produto final como papel e cartolina, em vez de telas eletrônicas e metal, já que sua finalidade é apenas exploratória (PREECE *et al.*, 2005). Eles são úteis porque são simples, baratos, sua produção é rápida e podem ser rapidamente modificados.

Na prototipação de alta-fidelidade são utilizados materiais que irão compor o produto final. Esse tipo de protótipo utiliza alguma linguagem de programação ou *software* auxiliar para a criação de versões executáveis do sistema. Assim, eles apresentam características reais do *software* tais como botões, cores e ícones e, por isso, é o que mais se aproxima do produto final. Os protótipos de alta-fidelidade possibilitam

que testes de usabilidade sejam feitos. Aguiar (2007) destaca que os usuários podem simular o comportamento da interface realizando várias operações (entradas de dados, preenchimento de formulários e seleção de ícones) assim como fariam no sistema final, fornecendo assim ótimos *feedbacks* para os projetistas.

A Tabela 1, obtida em Preece (2005), apresenta algumas vantagens e desvantagens da prototipação de alta-fidelidade em relação à prototipação de baixa-fidelidade.

Tabela 1: **Eficácia de protótipos de baixa-fidelidade *vs.* alta-fidelidade**

Tipo	Vantagens	Desvantagens
Protótipo de baixa-fidelidade	• Custo mais baixo de desenvolvimento • Avalia múltiplos conceitos de *design*. • Instrumento de comunicação útil. • Aborda questões de leiaute de tela. • Útil para identificação de requisitos de mercado. • *Proof-of-concept* (demonstrações de que o conceito funciona).	• Verificação limitada de erros. • Especificação pobre em detalhe para codificação • "Uso" conduzido pelo facilitador. • Utilidade limitada após o estabelecimento de requisitos. • Utilidade limitada para testes de usabilidade. • Limitações de fluxo e navegação.
Protótipo de alta-fidelidade	• Funcionalidade completa. • Totalmente interativo. • Uso conduzido pelo usuário. • Define claramente o esquema de navegação. • Uso para exploração e teste. • Mesmo *look and feel* do produto final. • Serve como uma especificação viva. • Ferramenta de venda de *marketing*.	• Desenvolvimento mais caro. • Sua criação demanda tempo. • Ineficiente para *designs proof-of-concept* (demonstrações de que o conceito funciona). • Não serve para coleta de requisitos.

Fonte: Preece (2005).

O conceito de prototipação de média-fidelidade no contexto de desenvolvimento de *software* ainda não é muito difundido na literatura. Aguiar (2007) diz que um protótipo de média-fidelidade consiste na implementação computadorizada de uma aplicação limitada funcionalmente, contendo apenas as funções essenciais para avaliar alguns cenários específicos. Nesse tipo de prototipação aliam-se as vantagens da prototipação de baixa-fidelidade e alta-fidelidade e excluem-se as desvantagens de ambas, criando assim uma alternativa intermediária. Os protótipos de média-fidelidade são uma ótima escolha para testar ideias que seriam testadas

em fases mais tardias do desenvolvimento, trazendo assim um rápido *feedback* dos usuários e tornando possível descobrir erros ainda no começo do projeto.

Um tipo de prototipação muito utilizada é o *wireframe*, segundo Silva e Savoine (2010, p. 47), este tipo "[...] é desenvolvido como um documento que apresenta a estrutura e o conteúdo da interface, indicando peso e relevância de cada elemento do *layout* e sua relação com os demais elementos que formam o sistema." O *wireframe* também pode ser classificado em relação a sua fidelidade e é bastante usado no desenvolvimento *web*. A maioria das ferramentas disponíveis no mercado possibilitam a construção de protótipos deste tipo.

2.2 REFATORAÇÃO DE *SOFTWARE*

Frequentemente, desenvolvedores de *software* têm a necessidade de fazer alterações no código, seja para implementação de novos requisitos encontrados ou para manutenções futuras após a entrega do sistema. É nesse contexto de constantes modificações que surge o conceito de refatoração de *software*.

Segundo Fowler (1999, p. 09), "a refatoração é o processo de alteração de um sistema de *software* de modo que o comportamento externo do código não mude, mas que sua estrutura interna seja melhorada". A refatoração está diretamente ligada a características do *software* como modularidade, eficiência, clareza e simplicidade. Ela busca melhorar a legibilidade e compreensão do código e torná-lo mais reutilizável, o que facilita a manutenção posterior evitando também que defeitos sejam incluídos.

As raízes da refatoração como um processo sistematizado surgiram na década de 80, por meio da linguagem de programação *Smalltalk*. Opdyke (1992) propôs em sua tese de doutorado algumas técnicas para tornar a evolução do *software* no qual trabalhava mais simples e menos custosa, concretizando assim o primeiro trabalho formal sobre o tema. São apresentadas em seu trabalho 26 refatorações (23

primitivas e 3 compostas), enfatizando como essas preservam o comportamento do sistema após serem aplicadas.

As refatorações primitivas são descritas em quatro categorias, sendo estas:

- Criação de entidades: criar uma classe vazia, variável ou função;
- Deleção ou remoção de entidades: remover classes, variáveis ou funções que não sejam referenciadas;
- Alteração de entidades: alterar nomes de classes, variáveis, parâmetros de funções, reordenar argumentos de funções, etc;
- Movimentação de variáveis: Mover variáveis para superclasses e mover variáveis para subclasses.

Já as refatorações compostas são definidas em relação ao acesso a uma variável, conversão de trechos de códigos em funções e movimentação de classes.

Roberts et al. (1996) desenvolveram a primeira ferramenta com finalidade de auxiliar a refatoração, o *Refactoring Browser*. Roberts (1999) abordou a refatoração com foco especificamente em linguagens orientadas a objetos e inseriu o conceito de pré-condições e pós-condições. Segundo ele, pré-condições são asserções que o programa deve possuir para que a refatoração seja aplicada, e as pós-condições especificam como as asserções são transformadas pela refatoração.

Fowler (1999), considerado uma referência na área, descreve os passos para se realizar uma refatoração efetiva. Ele aborda alguns conceitos que considera imprescindíveis na hora de refatorar como, por exemplo, o de possuir um conjunto de testes sólidos para cada seção de código a fim de garantir que as modificações sejam únicas e não alterem o comportamento do código anterior. Os testes devem ser auto-verificáveis e servem como *feedback* dos resultados da refatoração. Eles dizem se erros estão sendo introduzidos.

Fowler (1999) também apresenta um catálogo com 72 refatorações, sendo que cada uma é descrita com base nestas cinco partes:

- Nome: Nome dado a refatoração, usado para identificá-la no contexto;

- Resumo: Descrição do que a refatoração faz e em qual situação ela é necessária;
- Motivação: Motivo para se usar a refatoração e em que circunstâncias ela não deve ser usada;
- Mecânica: Passo a passo descrevendo como a refatoração funciona;
- Exemplos: São abordados exemplos para ilustrar o uso da refatoração.

A refatoração é muito usada e constitui uma das técnicas mais importantes para algumas metodologias de desenvolvimento. Ela compõe uma das bases da *Extreme Programming*, uma metodologia ágil conhecida como XP (BECK; FOWLER, 2000). Alguns ambientes integrados de desenvolvimento conhecidos como IDE's (*Integrated Development Enviroment*) fornecem meios para facilitar o processo de refatoração. As IDE's *NetBeans*[6] e *Eclipse*[7], ambas para desenvolvimento na linguagem *Java*, possuem esse tipo de recurso.

A refatoração em sua essência não é algo tão novo, pois aplicar melhorias e reestruturar o código é uma tarefa contínua e constante no processo de desenvolvimento, sendo usada às vezes de modo informal ou implícito. O que existe atualmente é uma formalização e documentação de técnicas que possibilitam aplicar a refatoração de forma segura, fazendo com que o comportamento externo do *software* mantenha-se inalterado após a refatoração.

2.3 O PROTOTIPADOR DE INTERFACES PROFACE

O prototipador de interfaces Proface é uma ferramenta para auxílio à prototipação na fase de extração e validação de requisitos. O objetivo da ferramenta é permitir a criação de interfaces com botões, campos de texto, imagens, etc., de forma simples e rápida. Sua primeira versão foi produzida por Kaabah Di Lacerda em 2006. Ele o desenvolveu como projeto final de curso no Bacharelado em Ciência da Computação do Departamento de Informática da Universidade Federal de Viçosa

[6]https://netbeans.org

[7]http://www.eclipse.org

(UFV), sob orientação do Professor Dr. José Luis Braga. Ao longo do tempo outras versões foram produzidas em trabalhos acadêmicos na UFV e, atualmente, o *software* se encontra na versão 3.0.

O Proface foi desenvolvido para ser distribuído de forma gratuita. A versão atual está disponível no repositório do *sourceforge*[8] para *download*. A ferramenta foi codificada em *Java* e, após o download, é possível executá-la.

A interface do Proface é bem simples. Seus recursos estão bem dispostos na tela principal (Figura 2). Do lado esquerdo ficam as paletas que são usadas para a criação do protótipo. Ele possibilita a criação de componentes de interface em três diferentes estilos: Genérico, *Windows* 98 e *Windows* XP. O estilo Genérico contém componentes de interface que são visualmente independentes de plataforma. À direita encontram-se dois painéis, sendo o primeiro relativo aos arquivos do projeto que se encontra em edição e o segundo apresenta as propriedades do componente de interface que estiver selecionado no momento. Ao centro encontra-se a área de prototipação, onde os protótipos são construídos e podem ser visualizados.

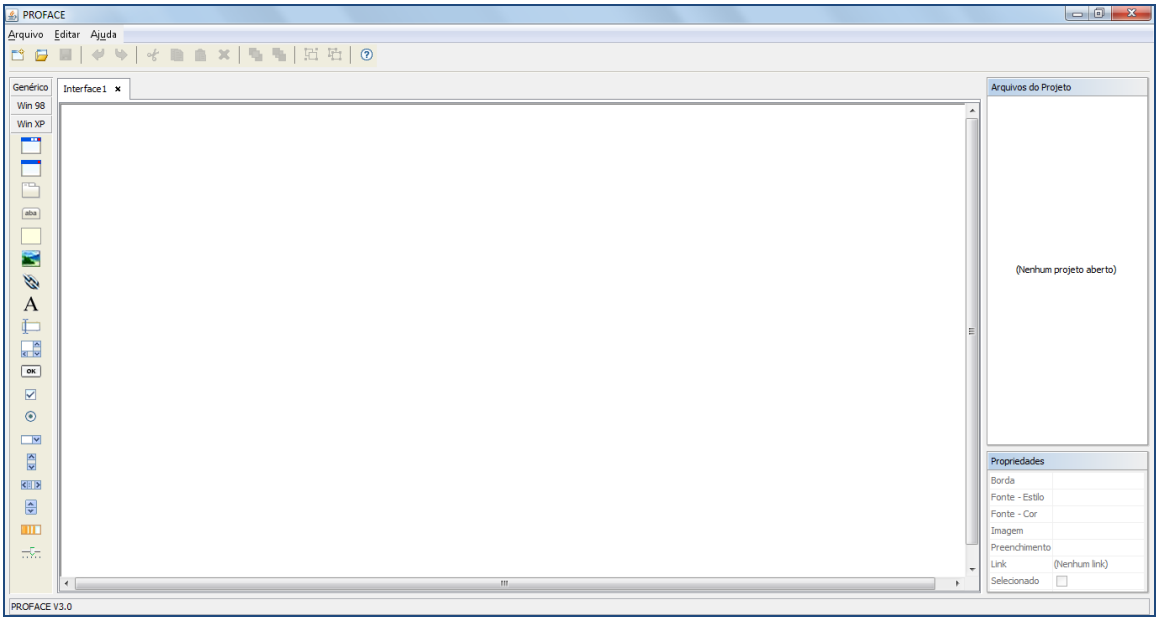

Figura 2: Interface Proface 3.0

[8]http://sourceforge.net/projects/proface/

O Proface permite exportar os protótipos para 3 diferentes tipos de arquivos, sendo estes .jpg, .png e .html. Ao exportar os protótipos de um determinado projeto para .html, uma documentação é criada e é possível visualizar no *browser* os protótipos exportados. Outro recurso interessante é que, ao adicionar um componente “Botão” ou um componente “*link*”, pode-se associar a eles *link* para outros protótipos que se encontram no mesmo projeto, permitindo assim a navegação entre os protótipos.

Para ilustrar o uso da ferramenta foi criado um protótipo usando cada estilo disponível na ferramenta. As figuras abaixo mostram o resultado da construção do protótipo:

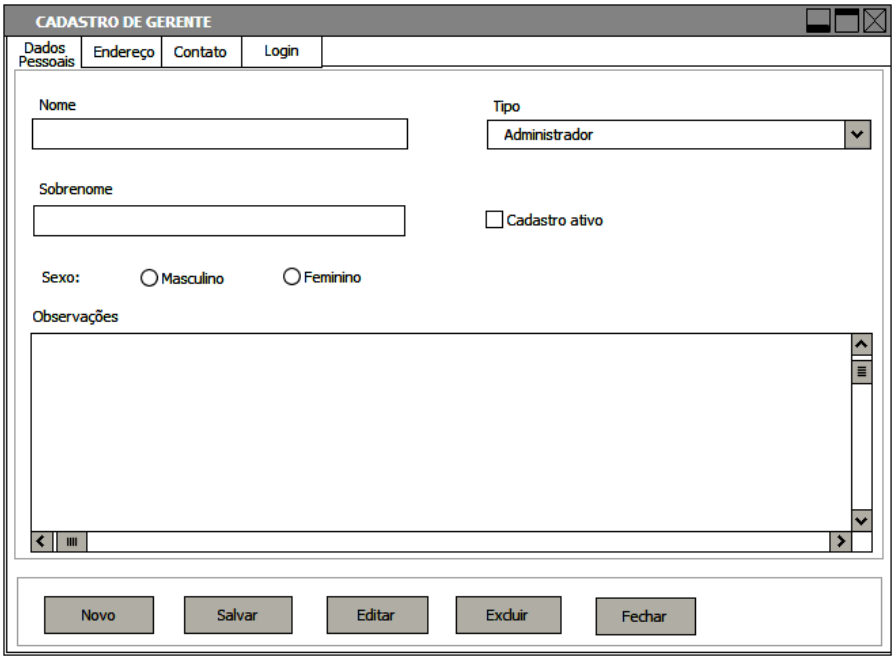

Figura 3: Protótipo de tela estilo Genérico - Proface 3.0

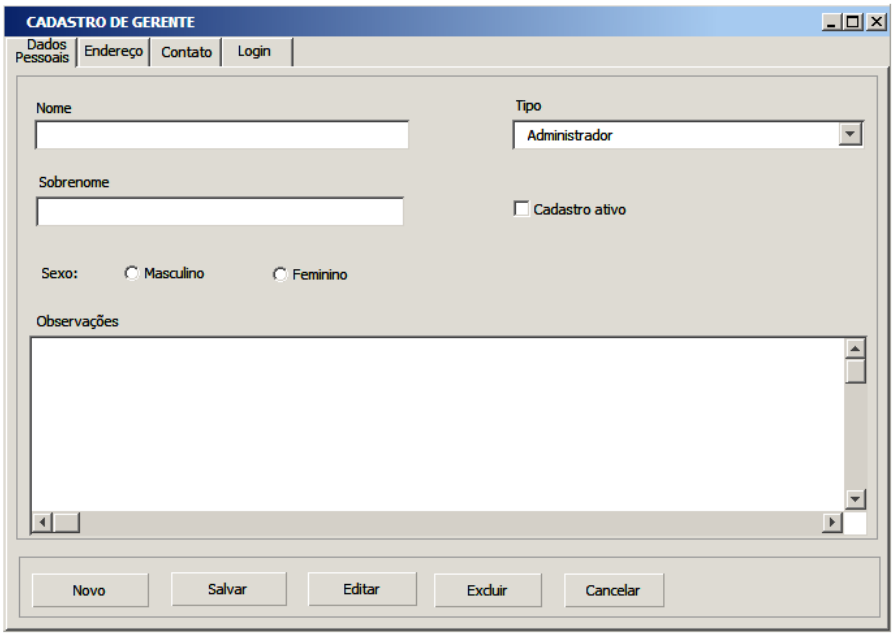

Figura 4: Protótipo de tela estilo Windows 98 - Proface 3.0

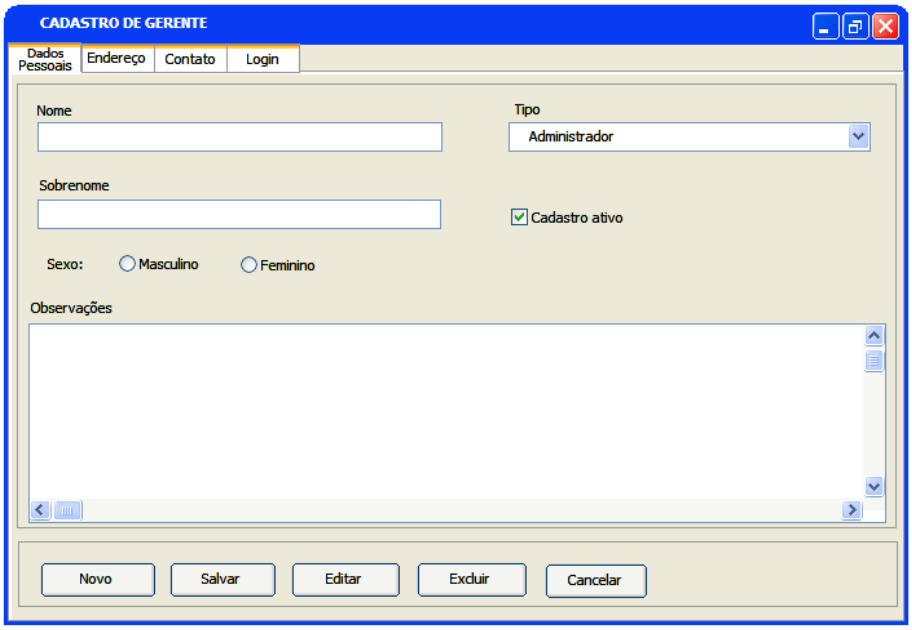

Figura 5: Protótipo de tela estilo Windows XP - Proface 3.0

Com estes exemplos foi possível explorar muitos recursos que a ferramenta já oferece e mostrar a riqueza de detalhes nos protótipos que podem ser construídos. Quanto mais detalhes, mais próximo do produto final o protótipo será, possibilitando que os usuários o validem de uma maneira mais abrangente e que os requisitos sejam especificados de forma mais completa.

2.4 TRABALHOS CORRELATOS

Algumas ferramentas usadas para auxiliar a prototipação serão apresentadas a seguir, destacando algumas de suas características e funcionalidades. Todas as ferramentas apresentadas geram protótipos de média-fidelidade, assim como o Proface 3.0.

Uma tabela será apresentada posteriormente contendo as seguintes características de cada ferramenta:

- Tipo de Licença: Se a ferramenta é paga ou gratuita ou parcialmente gratuita;
- Componentes avaliados: Se a ferramenta possui os componentes básicos que compõem a interface da maioria dos protótipos. Os componentes analisados foram: Janela, Caixa de diálogo, Painel tabulado, Imagem, *Links*, Texto, Campo de texto, Caixa de texto, Botão, *Ckeck Box*, *Radio Button*, Menu tipo *drop down*, Barras de rolagem e barra de progresso. Quando a ferramenta continha todos os componentes o valor assumido foi “Possui todos” ou ”Não possui todos“ caso contrário.

- Idiomas: Idiomas suportados pela ferramenta: “Inglês” ou “Português e Inglês”;

- Exportação HTML do protótipo: Se a ferramenta permite que o protótipo seja exportado para arquivo .html (“Sim” ou “Não”);

- Ligação entre telas: Se a ferramenta possui o recurso de criação de *links* entre telas (“Sim” ou “Não”).

Para exemplificar o uso de cada ferramenta, foi elaborado um protótipo modelo como o da Figura 6.

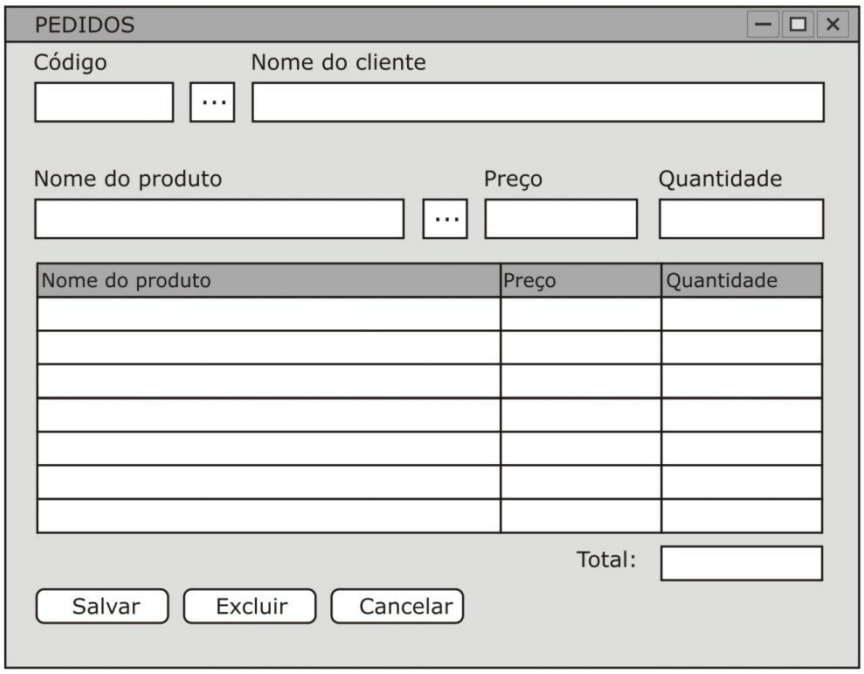

Figura 6: Protótipo modelo

2.4.1 PENCIL PROJECT

O *Pencil Project*[9] é uma ferramenta de prototipação gratuita *open-source* para sistemas GNU/Linux, *Windows* e Mac. Ela está disponível em duas formas: como aplicação *Desktop* e como extensão para o *Firefox*[10]. A Figura 7 exibe a interface da ferramenta.

[9] http://pencil.evolus.vn/

[10] https://www.mozilla.org/pt-BR/firefox/new/

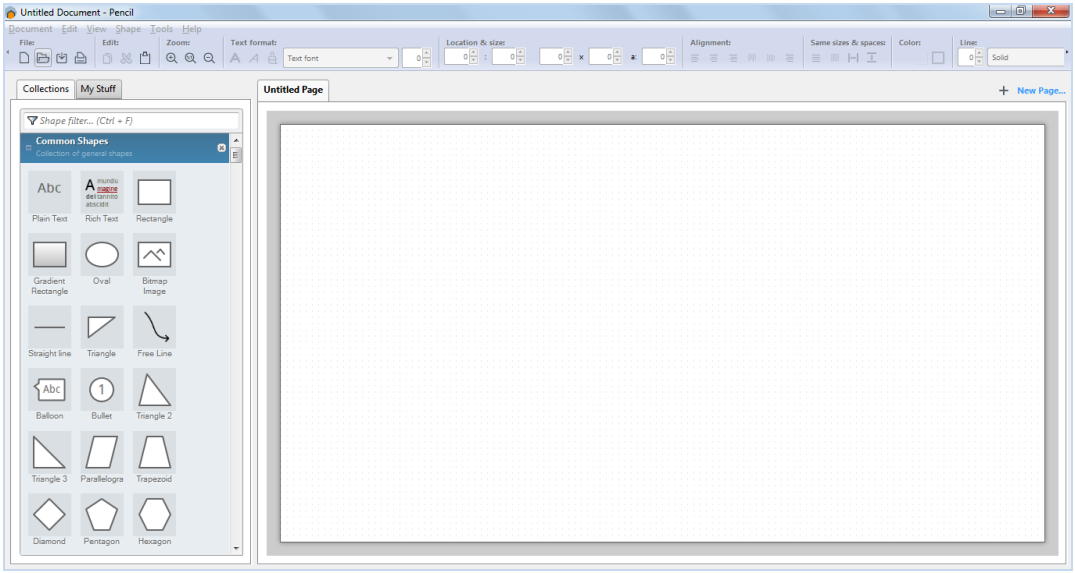

Figura 7: Interface *Pencil Project*

Fonte: http://pencil.evolus.vn/

A ferramenta possui recursos bem dispostos em sua interface. Do lado esquerdo da interface está a aba principal denominada *Collections* (coleções), onde se encontram as categorias de componentes de interface possíveis de serem prototipados. Exemplos de categorias disponíveis são: *Commom Shapes* (Formas simples), *Basic Web Elements* (Elementos básicos para Web), *Desktop Sketchy* GUI (Esboço para interfaces Desktop), etc. Também é possível instalar coleções disponíveis no site da ferramenta para personalizar a aba.

Após criar o protótipo, é possível alinhar componentes dentro da página, mudar sua cor, tamanho, posição, etc.

Os recursos de mais destaque na ferramenta são:

- Acesso a uma base de dados com coleções de formas para auxílio à criação de gráficos, telas para *Android,* etc*;*
- Suporte à criação de diversos diagramas, tais como de processos e gráficos de fluxo.
- Possibilidade de exportar documentos para os seguintes formatos de arquivos: .png, .html, .pdf e .odt;
- Capacidade de criação de *links* entre telas prototipadas.

Para criar o protótipo modelo foi usada a versão 2.0.5 de 2012, a mais atual disponível no site da ferramenta, e a categoria *Desktop – Sketchy* GUI (Esboço de interfaces gráficas para *desktop*). Não houve muita dificuldade na criação, porém o uso do componente tabela não é muito intuitivo. A interface na qual se configura as colunas e linhas é confusa e nada trivial. Também convém destacar que para alinhar um texto em um componente *"*Campo de texto*"* é preciso acessar as propriedades deste, o que pode ser complicado para usuários iniciantes. O protótipo modelo criado na ferramenta é apresentado na Figura 8.

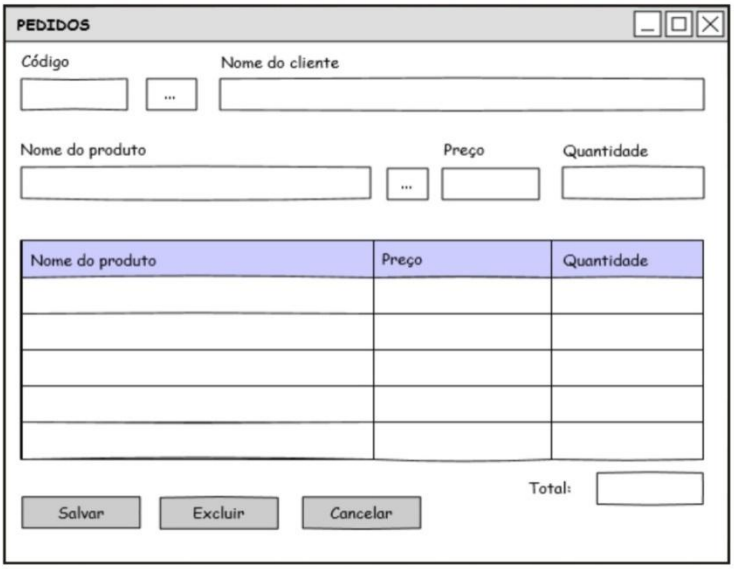

Figura 8: Modelo protótipo elaborado no Pencil Project

2.4.2 MOCKINGBIRD

Ferramenta totalmente online que permite a criação de protótipos de forma rápida e fácil. É essencialmente voltada para a prototipação de *websites,* tornando possível a produção de protótipos do tipo *wireframe*, que é um tipo de ilustração, uma representação da estrutura de um *site* e os relacionamentos (*links*) entre as páginas.

É possível experimentar a ferramenta gratuitamente pela opção "*Try it now*" disponível na página *web*[11]. Para utilizar o serviço na íntegra e salvar os projetos

[11]https://gomockingbird.com/mockingbird/

desenvolvidos, é necessário criar uma conta e aderir a um plano. Os planos são classificados em 5 formas diferentes:

- *Free plan*: Pode-se criar somente um projeto com um limite de 10 páginas e não é possível salvá-lo.
- *Umlimited*: Neste plano o número de projetos ativos e projetos que podem ser arquivados é ilimitado.
- *Pro*: Neste plano é possível manter até 25 projetos ativos e o número de projetos que podem ser arquivados é ilimitado.
- *Team*: Pode-se manter 10 projetos ativos e o número de projetos que podem ser arquivados é ilimitado.
- *Personal*: Neste plano é possível manter somente 2 projetos ativos e o número de projetos que podem ser arquivados é ilimitado.

A interface da ferramenta pode ser vista na Figura 9.

Figura 9: Interface *Mockingbird*.

Fonte: https://gomockingbird.com/mockingbird/

A interface da ferramenta é bem didática, possibilitando uma rápida navegação. A tela principal é dividida em duas partes, sendo a primeira onde ficam os componentes, identificada como *widgets*, e a segunda onde o protótipo é construído. As características e recursos que mais se destacam na ferramenta são:

- Facilidade de uso e rapidez na criação dos protótipos;

- Possibilidade de manter ilimitados projetos ativos, arquivados (somente disponível na versão paga);
- Possibilidade de ter ilimitados colaboradores nos projetos (somente disponível na versão paga);
- Criação de *links* entre telas de um mesmo projeto.

Na elaboração do protótipo modelo, foi observado que a criação de tabelas e personalização dessas não é trivial, pois precisa-se alterar parâmetros no elemento chamado *Data Grid* para que se tenha a configuração desejada. A página web da ferramenta afirma que também é possível criar protótipos para *Desktop,* mas o número de componentes disponíveis e os estilos parecem não permitir construir protótipos nesse modelo. Outro ponto a destacar é que a ferramenta permite que os projetos sejam exportados somente nos formatos .png e .pdf. A Figura 10 mostra o modelo proposto construído usando a ferramenta.

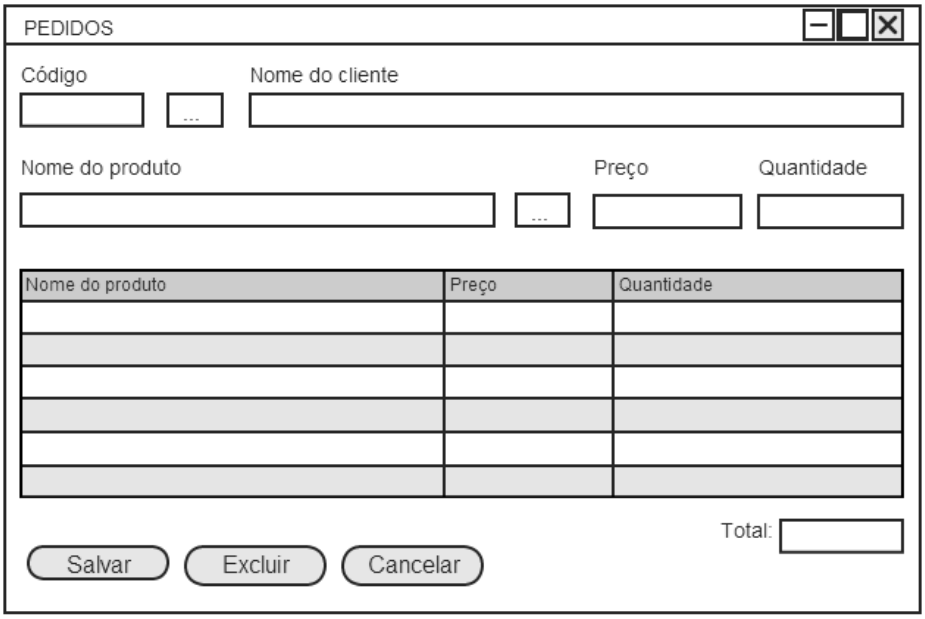

Figura 10: Modelo de protótipo elaborado no *Mockingbird*

2.4.3 CACOO

Cacoo[12] consiste também de uma ferramenta totalmente *online.* Ela possibilita criar protótipos do tipo *wireframe e* vários outros, como gráficos de fluxos, diagramas UML, diagramas de rede, etc.

Para utilizá-la também é necessário criar uma conta e aderir a um plano de uso, sendo possível usar a ferramenta de forma gratuita ou paga. Os planos disponíveis são:

- *Team*: Esse plano é indicado para times de desenvolvedores e é subdividido em 3 categorias:
    - *Basic:* Permite utilizar todos os recursos da ferramenta com até 10 usuários;
    - *Premium:* Permite utilizar todos os recursos da ferramenta com até 30 usuários;
    - *Max:* Permite utilizar todos os recursos da ferramenta com até 100 usuários;
- *Plus:* Plano indicado para um único usuário no qual é possível usar quase todos os recursos da ferramenta;
- *Free:* Plano indicado somente para quem deseja experimentar a ferramenta e explorar alguns de seus recursos.

A interface da ferramenta pode ser vista na Figura 11.

[12]https://cacoo.com/

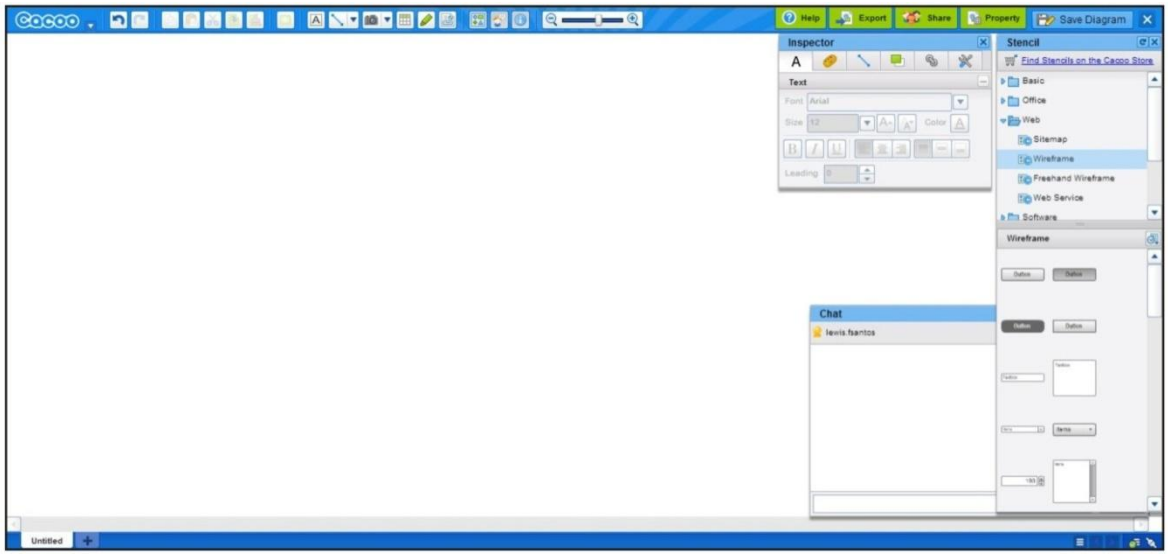

Figura 11: Interface Cacoo

Fonte: https://cacoo.com

A interface da ferramenta é bem organizada, diferenciando das demais na disposição dos elementos na tela. A tela com os componentes para prototipação, por exemplo, fica localizada do lado direito, onde é possível modificar atributos de cor, alinhamento, posição, atribuição de *links*, entre outros.

Do lado direito, na aba "*Stencil",* os tipos de protótipos estão organizados dentro de pastas, onde é possível abri-las e escolher um modelo a ser utilizado, como por exemplo, *wireframe*. Também do lado direito encontra-se uma janela de bate-papo, permitindo a interação instantânea com outros envolvidos no projeto.

Alguns recursos são diferenciados nessa ferramenta, como a capacidade de importar arquivos do computador do usuário para o projeto instantaneamente e tirar foto da tela de um site, onde é necessário informar a URL deste e a foto é feita automaticamente. Também é possível tirar *screenshots* de qualquer lugar da tela da ferramenta. Os recursos que merecem destaque na ferramenta são:

- Compartilhamento do projeto e pastas com outros usuários. Limitado a uma pasta no plano *Free* e ilimitado nos outros planos. O número de usuários é de até 15 por diagrama no plano *Free* e ilimitado nos outros planos;

- Possibilidade de comunicação instantânea com outros usuários colaboradores por meio do recurso de *chat.* Disponível em todos os planos;
- Possibilidade de importação de outros arquivos para o projeto e *screenshot* instantâneo de telas. Disponível em todos os planos;
- Exportação dos protótipos nos formatos .pdf, .ppt, .ps, .svg e .png. No plano *Free*, é possível exportar somente no formato .png.

A construção do protótipo modelo se deu de forma rápida e bem intuitiva, pois os componentes são de fácil manuseio. Convém destacar que não foi possível construir o protótipo no estilo *Desktop*, uma vez que não existe essa opção, sendo possível criá-lo somente no modo *wireframe*. O modelo do protótipo elaborado é mostrado na Figura 12.

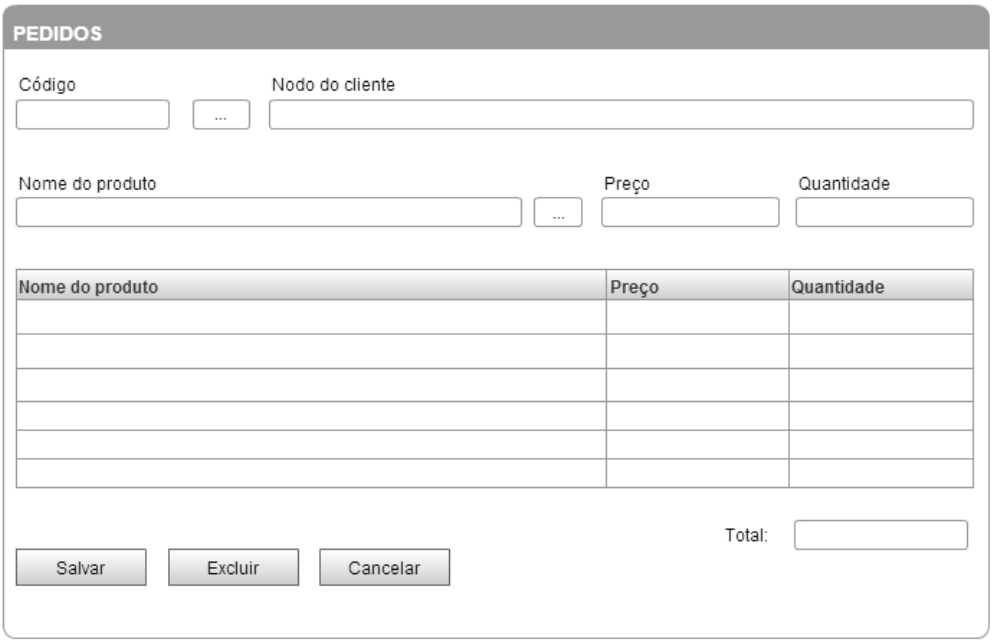

Figura 12: Modelo de protótipo elaborado no Cacoo

2.4.4 AXURE RP

O *Axure* RP[13] consiste de uma ferramenta *Desktop de* prototipação disponível para plataformas *Windows* e MAC. Permite criação de protótipos do tipo *wireframes* para *websites* e também a documentação desses protótipos criando uma espécie de

[13]http://www.axure.com/

histórico de documentação para aplicações. Os protótipos desenvolvidos possuem uma ótima qualidade.

A ferramenta é paga e possui dois tipos de licença: *Axure RP Standard* e *Axure RP Pro.* É possível utilizar um dos dois modos (*Standard ou Pro*) de maneira gratuita por 30 dias. Para isso, é necessário fazer o *download* do arquivo de instalação diretamente na página *web*[14]. A versão *Standard* pode ser adquirida com o pagamento de uma taxa. Ambas as licenças são definitivas, ou seja, uma vez adquiridas não é necessário que sejam renovadas. Na versão *Pro* é possível utilizar os recursos de documentação dos protótipos e adicionar colaboradores aos projetos, o que não é possível na versão *Standard*.

Após a instalação da versão escolhida, a elaboração de protótipos na ferramenta pode ser feita de maneira bastante intuitiva. A interface da ferramenta pode ser vista na Figura abaixo.

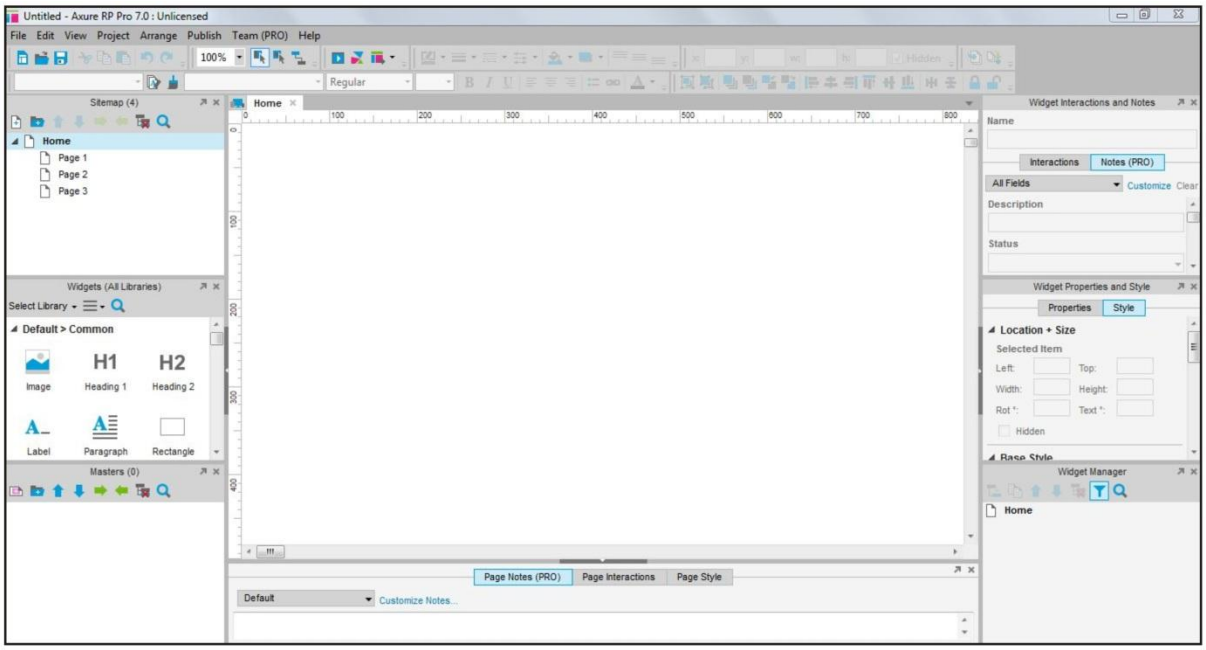

Figura 13: Interface *Axure* RP Pro

Fonte: http://www.axure.com/

Os componentes de prototipagem encontram-se dispostos do lado esquerdo, na janela "*Widgets*". Uma característica não encontrada nas outras ferramentas é que

[14]http://www.axure.com/download

no Axure RP é possível adicionar eventos aos elementos. Por exemplo, em um botão pode-se adicionar o evento *onclick,* assim como criar *links* para outras telas.

Ao concluir a edição de um protótipo, é possível visualizá-lo no navegador web pressionando a tecla F5. No recurso de documentação, denominado "*Notes PRO"*, pode-se adicionar informações importantes ao protótipo, como o seu status, que pode assumir os valores de "Proposto", "Aprovado" e "Incorporado".

Os recursos de maior destaque da ferramenta são:

- Interface amigável;
- Criação de *links* entre interfaces;
- Adição de eventos aos componentes, o que permite maior interatividade ao validar o protótipo;
- Adição de bibliotecas de protótipo, disponível apenas após o pagamento da licença.
- Geração de documentação;
- Adição de usuários colaboradores.

O modelo desenvolvido na ferramenta pode ser visto na Figura 14. Cabe ressaltar que não foi possível encontrar um componente que representasse o tipo janela do protótipo, que foi construído usando a forma básica "*Rectangle*".

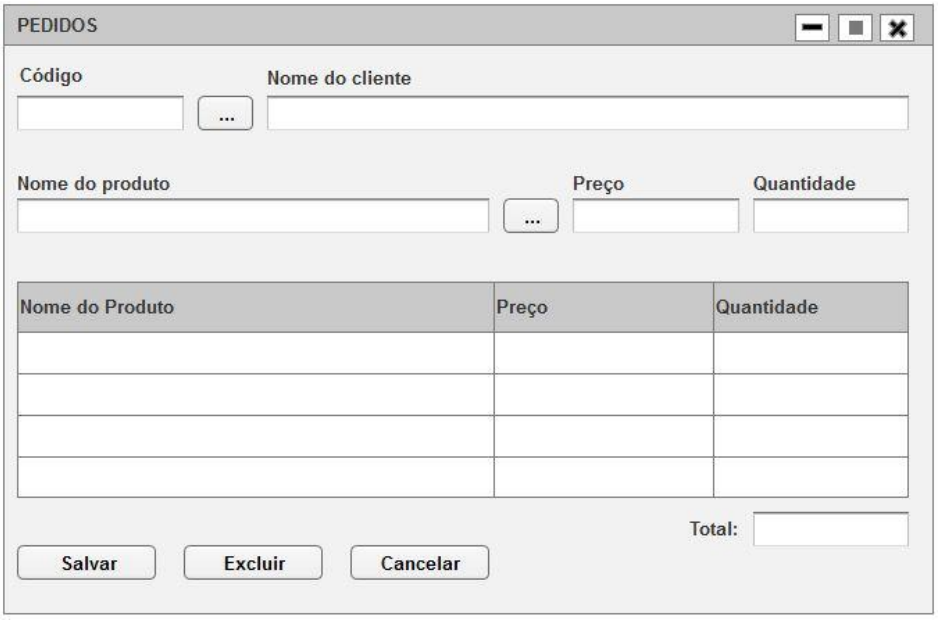

Figura 14: Modelo de protótipo elaborado no Axure

2.4.5 SÍNTESE DOS TRABALHOS CORRELATOS

A Tabela 2 apresenta um resumo das características principais das ferramentas listadas anteriormente.

Tabela 2: Características principais das ferramentas de prototipagem

	Pencil Project	Mockingbird	Cacoo	Axure RP
Tipo de Licença	Gratuita	Parcialmente gratuita	Parcialmente gratuita	Parcialmente gratuita
Componentes avaliados	Possui todos	Possui todos	Possui todos	Não possui todos.
Idiomas	Inglês	Inglês	Inglês	Inglês
Exportação HTML do protótipo	Sim	Não	Não	Sim
Links entre telas	Possui	Possui	Possui	Possui

Ao utilizar as ferramentas na construção do protótipo modelo, foram verificados os prós e contras de cada uma. Um ponto a ser destacado é que todas as ferramentas possuem interface em inglês, o que pode ser uma dificuldade para usuários que não são nativos de língua inglesa.

Considerando essas ferramentas, a possibilidade de criação de designs realistas no estilo *Desktop* é bem restrita. O Pencil Project é o único que pode ser obtido com sua totalidade de recursos de forma gratuita. As demais ferramentas exigem pagamento de taxas de licenciamento para isso.

A ferramenta Proface, cuja evolução é o foco desse trabalho, possui o diferencial de ser gratuita. Assim, todos os seus recursos estão disponíveis a qualquer pessoa, seja um estudante ou profissional. O Proface possibilita a criação de protótipos de interfaces *Desktop* bem próximas das interfaces que serão criadas posteriormente, ou seja, bem realistas. Como se pode ver são poucas as ferramentas que possibilitam a criação de interfaces *Desktop* tão ricas, embora exista uma excelente para criação de interfaces *web* (Axure).

Além disso, o Proface possibilita a criação de protótipos de forma rápida e dinâmica, possibilitando seu uso em reuniões com clientes e proporcionando uma comunicação eficiente entre os *stakeholders*.

3 METODOLOGIA

Inicialmente foi realizada uma revisão bibliográfica para aprofundar os conhecimentos acerca do contexto no qual o trabalho está inserido. A prototipação de software no âmbito de extração e validação de requisitos foi estudada. Também foram investigadas ferramentas semelhantes ao Proface 3.0 a fim de realizar comparações e destacar funcionalidades.

Posteriormente, técnicas para a execução da refatoração foram analisadas. A referência utilizada foi o livro de Martin Fowler (FOWLER, 1999). A escolha desta referência se deu pelo fato dela ser bem reconhecida na comunidade acadêmica e por se tratar de um catálogo de refatorações voltado para aplicações orientadas a objeto.

Para realizar a refatoração do código fonte do Proface 3.0 foi adotada uma metodologia parcialmente baseada em (MENS; TOURWÉ, 2004). A fim de nortear o processo de refatoração, um diagrama de atividades foi elaborado. A Figura 15 apresenta esse diagrama.

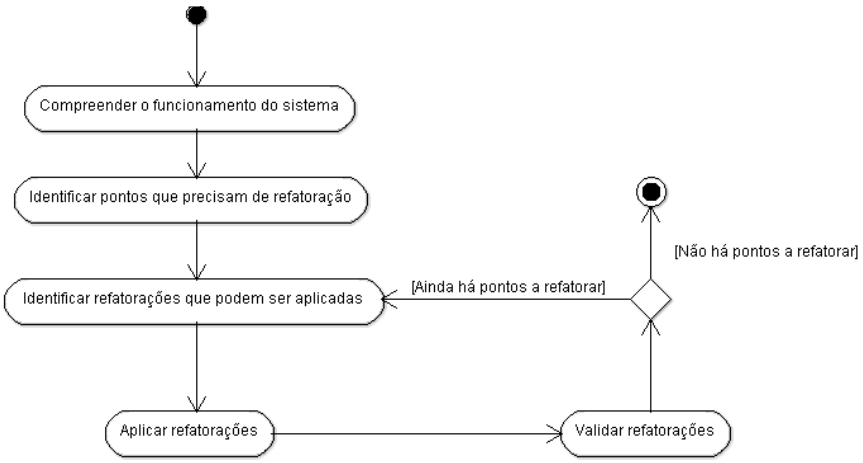

Figura 15: Diagrama de atividades usado para nortear o processo de refatoração.

O Proface 3.0 não dispõe de nenhum tipo de documentação formal, sendo o código fonte a única referência para a compreensão do funcionamento do sistema. Sendo

assim, para facilitar o entendimento fez-se o uso da engenharia reversa do código da ferramenta.

O processo de engenharia reversa possibilita a geração de múltiplas visões do *software* fazendo com que suas características mais notáveis sejam reveladas (WILLS; NEWCOMB, 1996). Para realizar esse procedimento foi utilizada a ferramenta *Enterprise Architect* 11[15] em sua versão *Trial*, disponível gratuitamente para avaliação de 30 dias. Assim, foram obtidos os diagramas de classes do sistema que possibilitaram entender melhor o funcionamento do software, como as classes estão relacionadas e a função exercida por cada uma.

Após a compreensão do funcionamento do sistema, foram identificadas possibilidades de refatoração pela análise do código fonte. Com os pontos identificados, o *loop* descrito no diagrama da Figura 15 se inicia fazendo com que refatorações sejam aplicadas até não existirem mais pontos a refatorar.

A fim de verificar se o comportamento do software não foi alterado após a aplicação das refatorações, alguns testes foram feitos antes e depois de realizá-las. Cada teste tinha como objetivo avaliar algum recurso oferecido pelo Proface 3.0 que poderia ser afetado pelas refatorações. Os testes executados são apresentados no Apêndice A.

Finalizado o processo de refatoração, foi realizada a implementação da nova paleta de componentes do Proface no estilo *Windows* 7. Após a implementação, além dos mesmos testes do Apêndice A, os testes do Apêndice B também foram realizados para validação dos novos recursos.

Todo o processo de refatoração, bem como a adição dos novos recursos no Proface foram realizados utilizando o paradigma de programação orientado a objetos e a linguagem *Java SE*[16]. O ambiente de desenvolvimento integrado (IDE) usado foi o *Eclipse Luna* 4.4[17]. O Eclipse foi escolhido porque o Proface 3.0 foi inicialmente

[15] http://www.sparxsystems.com/products/ea/
[16] https://docs.oracle.com/javase/specs/
[17] https://www.eclipse.org

desenvolvido nesta IDE e, também, por possuir recursos que facilitaram a refatoração. Esses recursos estão disponíveis através do menu *Refactor* localizado na barra de menus principal.

4 RESULTADOS OBTIDOS

4.1 REFATORAÇÃO DO CÓDIGO DO PROFACE

Na etapa de identificação dos pontos a serem refatorados, foi possível observar alguns "*bad smells*" no código, sendo estes classificados por Fowler como sinais ou sintomas que indicam que o código pode ser melhorado (FOWLER, 1999).

O primeiro *bad smell* encontrado foi um código repetido nas classes BorderColorEditor, FillColorEditor e FontColorEditor que herdam da superclasse ColorEditor. Estas classes são responsáveis por gerenciar mudanças nas cores de borda de um componente (BorderColorEditor), preenchimento de um componente (FillColorEditor) e fonte de um componente (FontColorEditor). O código encontrado estava duplicado em todas as subclasses dentro do método herdado selectColor() como pode ser visto na Figura abaixo.

```
420     if (selectedColor != null) {
421         Color oldColor = color;
422         Color newColor = selectedColor;
423         label.setValue(newColor);
424         color = newColor;
425         firePropertyChange(oldColor, newColor);
426     }
```

Figura 16: Trecho de código duplicado dentro do método selectColor()

Para a solução desse *bad smell* foram utilizadas as refatorações *Extract Method*, *Pull up Field* e *In line temp*. A refatoração *Extract Method* pode ser aplicada quando se tem classes herdeiras que possuem expressões repetidas. O método selectColor() da superclasse ColorEditor foi extraído, criando-se nesta mesma classe um novo método denominado changeColor(Color pSelectedColor). Este novo método recebe como parâmetro o campo que se repetia nas classes filhas, configurando assim uma nova refatoração denominada *Pull up Field.*

Por fim, a refatoração *In line temp* foi aplicada nas classes filhas. Ela permitiu que a variável “selectedColor” que é atribuída uma única vez, como pode ser visto na linha 533 da Figura17, fosse passada diretamente como parâmetro para o novo método criado na superclasse.

```
531            if (selectedColor != null) {
532                Color oldColor = color;
533                Color newColor = selectedColor;
534                label.setValue(newColor);
535                color = newColor;
536                firePropertyChange(oldColor, newColor);
537            }
```

Figura 17: Trecho do método selectColor() na classe BorderColorEditor

O código atualizado na superclasse após a criação do método changeColor(Color pSelectedColor) pode ser visto na Figura 18.

```
423        protected void changeColor(Color pSelectedColor) {
424            if (pSelectedColor != null) {
425                Color oldColor = color;
426                Color newColor = pSelectedColor;
427                label.setValue(newColor);
428                color = newColor;
429                firePropertyChange(oldColor, newColor);
430            }
431        }
432
433        protected void selectColor() {
434            changeColor(Color.black);
435        }
```

Figura 18: Aplicação do *Extract Method* na classe ColorEditor

A Figura 19 mostra como as classes filhas agora utilizam o novo método changeColor(Color pSelectedColor) após a aplicação das refatorações.

```
515        protected void selectColor() {
516            areaTrabalho.alteraCorBordaComponenteSelecionado();
517            super.changeColor(componente.getCorBorda());
518        }
```

Figura 19: Método selectColor() da classe BorderColorEditor após refatoração

Outra refatoração aplicada ao código foi a *Substitute Algorithm.* Essa refatoração, segundo Fowler(1999), é usada para fazer com que um determinado trecho de

código fique mais claro e mais compreensível, uma vez que esse trecho é substituído por um novo algoritmo.

A aplicação dessa refatoração ocorreu na classe “Utils”. Esta classe tem por objetivo, como o próprio nome sugere, ser uma classe utilitária. Ela possui dois métodos principais: getExtension(File f) e createImageIcon(String path). O primeiro é responsável por retornar a extensão de um arquivo passado como parâmetro, e o segundo é responsável por criar um ícone com base no endereço (path) da imagem que também é passada por parâmetro.

Além dos dois métodos principais, a classe possui alguns campos estáticos que servem como constantes que são utilizadas em várias partes do sistema. O trecho de código com as constantes existentes na classe pode ser visto na Figura 20.

```
11      public final static String jpeg = "jpeg";
12      public final static String jpg = "jpg";
13      public final static String gif = "gif";
14      public final static String tiff = "tiff";
15      public final static String tif = "tif";
16      public final static String png = "png";
```

Figura 20: Constantes da classe utilitária Utils

Diversas classes fazem uso da classe “Utils”. A caráter demonstrativo, é apresentado na Figura 21 como a classe ImageFilter faz o uso dessa classe no método accept(File f).

```
12      public boolean accept(File f) {
13          if (f.isDirectory()) {
14              return true;
15          }
16
17          String extension = Utils.getExtension(f);
18          if (extension != null) {
19              if (extension.equals(Utils.tiff) ||
20                  extension.equals(Utils.tif) ||
21                  extension.equals(Utils.gif) ||
22                  extension.equals(Utils.jpeg) ||
23                  extension.equals(Utils.jpg) ||
24                  extension.equals(Utils.png)) {
25                      return true;
26              } else {
27                  return false;
28              }
29          }
30
31          return false;
32      }
```

Figura 21: Método accept(File f) da classe ImageFilter

A aplicação da refatoração *Substitute algorithm* se deu inicialmente pela criação de um *array* que armazena todos os tipos de arquivos de imagens manipulados pelo sistema atualmente e outro para armazenar os arquivos com extensão .prj ou .PRJ, que são arquivos de projeto do Proface. Também foi criado um campo estático que armazena a extensão .pro, utilizada para identificar arquivos de protótipos do Proface. A criação do campo estático nesta classe se fez necessária pois a verificação estava sendo feita de maneira não padronizada no sistema. Com a criação do campo, mudanças futuras ficarão mais fáceis de serem realizadas. Os *arrays* criados e o campo podem ser vistos na Figura 22.

```
14      public final static List<String> extensoesImagens = Arrays.asList(new String[] {"jpeg", "jpg", "gif", "tiff", "tif", "png"});
15      public final static List<String> extensoesProjetoProface = Arrays.asList(new String[] {"prj", "PRJ"});
16      public final static String extensaoArquivoProface = "pro";
```

Figura 22: *Arrays* e campos criados na classe Utils

Com a refatoração, todas as classes que fazem uso da classe Utils foram adaptadas. O método da classe ImageFilter, apresentado na Figura 21, agora usa a classe Utils da seguinte forma:

```
12      public boolean accept(File f) {
13          if (f.isDirectory()) {
14              return true;
15          }
16
17          String extension = Utils.getExtension(f);
18          if (extension != null) {
19              return (Utils.extensoesImagens.contains(extension));
20          }
21
22          return false;
23      }
```

Figura 23: Método accept(File f) da classe ImageFilter após a classe Utils ser refatorada

É possível perceber, comparando as Figuras 21 e 23, que o uso da classe Utils se tornou muito mais simples, pois não é necessário comparar extensão por extensão, bastando apenas verificar se uma determinada extensão existe no array pelo uso do método contains(Object) do Java.

Na sequência do processo de refatoração foi aplicada a *Consolidate conditional expression*. Essa refatoração é aplicada quando se tem uma sequência de testes condicionais que produzem o mesmo efeito. Para solucionar o problema, todos os testes são substituídos por um único teste equivalente usando operadores do tipo *Or* e *And*. Para isso, cria-se uma função do tipo *boolean* que faz o teste das condicionais. Essa refatoração foi aplicada na classe PainelPrototipacao, responsável por gerenciar os recursos onde os protótipos são construídos. A Figura 24 apresenta parte do método mouseReleased(MouseEvent event) antes da refatoração ser aplicada.

```
956         public void mouseReleased(MouseEvent event) {
957             if (componente != null)
958                 return;
959
960             if (componenteSelecionado == null)
961                 return;
962
963             if (arrastou == 0)
964                 return;
965
966             if (event.isPopupTrigger() && componenteSelecionado != null) {
967                 exibePopup(event);
968             }
969
970             arrastou = 0;
971             Componente temp = (Componente) event.getSource();
```

Figura 24: Parte do método mouseReleased(MouseEvent event) antes da aplicação da refatoração

Consolidade Conditional Expression

A solução para o *bad smell* encontrado foi a criação de um método chamado isArrastado() onde os testes feitos nas linhas 957, 960 e 963 da Figura 24 foram consolidados. O resultado é mostrado na Figura 25.

```
1039        private boolean isArrastado() {
1040            return ((componente != null) || (componenteSelecionado == null) || (arrastou == 0));
1041        }
```

Figura 25: Método criado na classe PainelPrototipacao para a solução do *bad smell* encontrado

Agora o método mouseReleased(MouseEvent event) apenas chama o novo método criado no início de sua execução.

Outra refatoração executada foi a *Decompose conditional.* Esta refatoração, como o próprio nome sugere, tem por objetivo simplificar condicionais complexas. Para fazer a simplificação, as condicionais são extraídas e são criados métodos que descrevem seus testes. Por fim faz-se as chamadas dos métodos onde as condicionais eram verificadas. Essa refatoração foi aplicada na classe ImageFileView. A Figura 26 ilustra como o método getIcon(File f) da classe ImageFileView testa os tipos de extensões de imagens para retornar um ícone equivalente.

```
50      public Icon getIcon(File f) {
51          String extension = Utils.getExtension(f);
52          Icon icon = null;
53
54          if (extension != null) {
55              if (extension.equals(Utils.jpeg) ||
56                  extension.equals(Utils.jpg)) {
57                  icon = jpgIcon;
58              } else if (extension.equals(Utils.gif)) {
59                  icon = gifIcon;
60              } else if (extension.equals(Utils.tiff) ||
61                         extension.equals(Utils.tif)) {
62                  icon = tiffIcon;
63              } else if (extension.equals(Utils.png)) {
64                  icon = pngIcon;
65              }
66          }
67          return icon;
68      }
```

Figura 26: Método getIcon(File f) da classe ImageFileView.

Para decompor as condicionais, os métodos apresentados na Figura 27 foram criados.

```
66      private boolean eJPEG(String extension) {
67          return (extension.equals(Utils.extensoesImagens.get(Utils.extensoesImagens.indexOf("jpeg")))
68                  || extension.equals(Utils.extensoesImagens.get(Utils.extensoesImagens.indexOf("jpg"))));
69      }
70
71      private boolean eGIF(String extension) {
72          return extension.equals(Utils.extensoesImagens.get(Utils.extensoesImagens.indexOf("gif")));
73      }
74
75      private boolean eTIFF(String extension) {
76          return (extension.equals(Utils.extensoesImagens.get(Utils.extensoesImagens.indexOf("tiff")))
77                  || extension.equals(Utils.extensoesImagens.get(Utils.extensoesImagens.indexOf("tif"))));
78      }
79
80      private boolean ePNG(String extension) {
81          return extension.equals(Utils.extensoesImagens.get(Utils.extensoesImagens.indexOf("png")));
82      }
```

Figura 27: Métodos criados na classe ImageFileView para a aplicação da refatoração *Decompose Conditional*

Agora o método getIcon(File f) testa as condicionais da seguinte forma:

```
48      public Icon getIcon(File f) {
49          String extension = Utils.getExtension(f);
50          Icon icon = null;
51
52          if (extension != null) {
53              if (eJPEG(extension)) {
54                  icon = jpgIcon;
55              } else if (eGIF(extension)) {
56                  icon = gifIcon;
57              } else if (eTIFF(extension)) {
58                  icon = tiffIcon;
59              } else if (ePNG(extension)) {
60                  icon = pngIcon;
61              }
62          }
63          return icon;
64      }
```

Figura 28: Método getIcon(File f) da classe ImageFileView após refatoração *Decompose Conditional*

Os métodos criados agora também são usados pelo método getTypeDescription(File f) da classe ImageFileView.

No decorrer de todo o processo de refatoração diversas variáveis foram renomeadas para que descrevessem o seu real propósito. Foi também substituída a interface MouseListener por MouseAdapter nas classes Componente e PainelPrototipação, fazendo com que somente os métodos necessários da interface MouseListener fossem implementados. Algumas classes que possuíam o método Main foram

retiradas do sistema, pois estas não estavam sendo usadas. Essas classes realizavam testes no desenvolvimento de versões anteriores do Proface.

Nenhum problema foi encontrado na execução dos testes após as refatorações. O único procedimento em que ocorreu erro foi na criação de um componente de “Imagem” com a alteração de sua imagem padrão. Havia um defeito no software que estava impedindo que a imagem fosse alterada. O defeito foi corrigido e o procedimento foi executado normalmente.

A execução normal dos procedimentos após a aplicação das refatorações mostra que o comportamento externo do Proface 3.0 não foi alterado, mas sua estrutura interna foi melhorada.

As refatorações aplicadas no código fonte do Proface 3.0 foram:

- Renomeação de variáveis para nomes mnemônicos;
- Extract method;
- Pull up field;
- Substitute algorithm;
- In line temp;
- Decompose conditional;
- Consolidate conditional expression.

Com a aplicação dessas refatorações, o código tornou-se mais compacto e a modularização foi facilitada. Além disso, o código ficou mais limpo, permitindo assim melhor legibilidade para novas evoluções da ferramenta.

4.2 IMPLEMENTAÇÃO DA NOVA PALETA DE COMPONENTES

Inicialmente, foi necessário criar as imagens para compor o *background* dos botões na nova paleta de componentes. Após a geração das imagens, foi criada uma classe para cada componente. As classes criadas foram:

- PAbaWin7;
- PBarraProgressoWin7;
- PBarraRolHorizontalWin7;
- PBarraRolVerticalWin7;
- PBotaoWin7;
- PCaixaDialogoWin7;
- PCheckBoxWin7;
- PComboBoxWin7;
- PJanelaComAbaWin7;
- PJanelaWin7;
- PRadioButtonWin7;
- PSliderWin7;
- PSpinnerWin7;
- PTextAreaWin7;
- PTextFieldWin7.

Todas estas classes são filhas de um dos tipos das classes do diagrama da Figura 29.

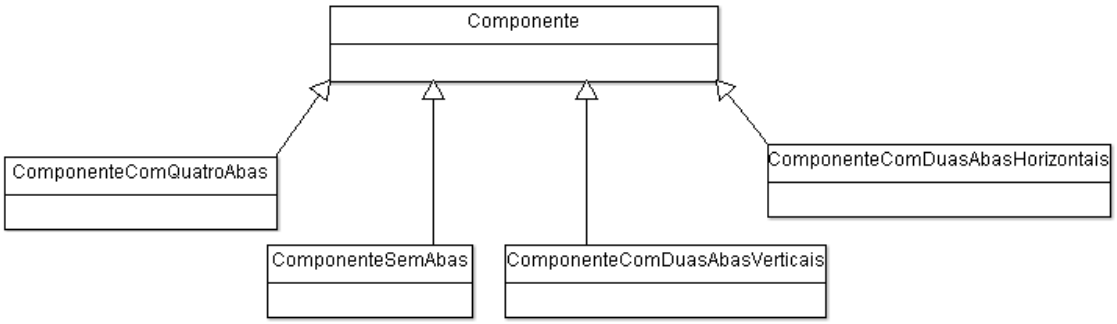

Figura 29: Diagrama de classes dos components

A classe Componente contém todos os atributos e métodos necessários para a criação de um componente no Proface. A classe Componente com todos os seus atributos e métodos pode ser vista no Apêndice C. O número de abas especificado nas classes filhas de Componente diz respeito ao número de pontos de redimensionamento que cada componente possui.

Para inicializar cada barra de componentes, o Proface utiliza um arquivo no formato .xml. Este arquivo é carregado dinamicamente quando o sistema é iniciado. Um trecho do arquivo criado para a nova paleta de componentes pode ser visto na Figura 30.

```xml
<?xml version="1.0" encoding="iso-8859-1"?>
<proface>
    <componente>
        <imagem>resource/imagensPaletaWin7/BotaoJanelaWin7.gif</imagem>
        <classe>br.ufv.dpi.proface.componentes.win7.PJanelaWin7</classe>
        <rotulo>Janela</rotulo>
        <texto1>Pressione o botão e clique na área de trabalho no</texto1>
        <texto2>local desejado para adicionar uma janela.</texto2>
    </componente>
    <componente>
        <imagem>resource/imagensPaletaWin7/BotaoDialogoWin7.gif</imagem>
        <classe>br.ufv.dpi.proface.componentes.win7.PCaixaDialogoWin7</classe>
        <rotulo>Caixa de diálogo</rotulo>
        <texto1>Pressione o botão e clique na área de trabalho no</texto1>
        <texto2>local desejado para adicionar uma caixa de diálogo.</texto2>
    </componente>
```

Figura 30: Parte do arquivo no formato .xml usado para a geração da nova paleta no estilo *Windows* 7

Os atributos do arquivo são:

- Componente: Tag responsável por caracterizar um componente completo.
- Imagem: Caminho completo da imagem criada para o respectivo componente. Esta imagem é mostrada como ícone na paleta.
- Classe: Caminho para classe que representa o componente a ser prototipado.
- Rótulo: Título do texto que aparece como dica quando o mouse é posicionado sobre a área do botão.
- Texto 1 e texto 2: Corpo do texto que aparece como dica quando o mouse é posicionado sobre a área do botão.

Com esse arquivo criado, foram implementadas as rotinas que fazem a sua leitura e montam a barra de componentes na tela principal do Proface. Essas rotinas foram criadas na classe chamada Proface. Esta classe é responsável por gerenciar os recursos da tela principal da ferramenta. A parte principal da rotina responsável pela leitura do arquivo ".xml" é apresentada na Figura 31.

```
894     // Recupera o elemento raiz
895     Element proface = document.getRootElement();
896     // Recupera os elementos filhos
897     Iterator<?> i = proface.getChildren().iterator();
898     JToggleButton temp;
899     Element element;
900
901     while (i.hasNext()) {
902         element = (Element) i.next();
903         temp = new JToggleButton();
904         temp.setIcon(new ImageIcon(getClass().getClassLoader().getResource(element.getChildText("imagem"
            ))));
905         temp.setToolTipText("<html> <b>  "+element.getChildText("rotulo")+"</b> <br>"+
906                 "  "+element.getChildText("texto1")+"  <br>"+
907                 "  "+element.getChildText("texto2")+"  "+
908                 "</html>");
909         temp.addActionListener(new TrataEventosBotaoComponentes(element.getChildText("classe")));
910         barraComponentesWin7.add(temp);
911     }
```

Figura 31: Parte principal da rotina implementada para efetuar a leitura do arquivo .xml

Com a paleta carregada na tela principal, as classes criadas foram implementadas. Na implementação de cada classe fez-se o uso da interface de programação de aplicações (API) Java 2D[18]. Este recurso permite a construção de primitivas gráficas e manipulação de cores de forma dinâmica. Na Figura 32 pode-se ver parte do método paintComponent(Graphics gp) da classe PBotaoWin7 fazendo o uso deste recurso.

```
63      public void paintComponent(Graphics gp)
64      {
65          // Cor da borda
66          gp.setColor(new Color (112, 112, 112));
67          gp.fillRoundRect(03, 03, largura - 6, altura - 6, 5, 5);
68
69          gp.drawRoundRect(04, 04, largura - 7, altura - 7, 5, 5);
70
71          // Sombreamento interno bordas superiores arredondadas
72          gp.setColor(getBackground());
73          gp.fillRoundRect(04, 04, largura-8, altura-8, 5, 5);
74
75          Color corSombreamento = getBackground();
76          int r = getBackground().getRed(),
77                  g = getBackground().getGreen(),
78                  b = getBackground().getBlue();
79
80          gp.setColor(new Color(
81                  Math.min(243, (int)(r * 1.109)),
82                  Math.min(243, (int)(g * 1.109)),
83                  Math.min(243, (int)(b * 1.109))));
84
85          gp.fillRoundRect(04, 04, largura-8, (altura / 2) - 4, 5, 5);
86
87          // Sombreamento interno bordas inferiores arredondadas
88          gp.drawRoundRect(04, 04, largura - 8, altura - 8, 5, 5);
89
```

Figura 32: Parte do método paintComponent(Graphics gp) da classe PBotaoWin7

[18] https://docs.oracle.com/javase/tutorial/2d/

Para criar os componentes com as cores fiéis às do *Windows* 7 foi utilizado um aplicativo gratuito denominado Color Cop[19]. Esse aplicativo captura o RGB de qualquer cor representada na tela do computador. Cada classe criada para representar um componente implementa os métodos abstratos não implementados na sua classe mãe. A Figura 33 mostra como a classe PJanelaWin7 implementa 3 dos métodos abstratos da classe Componente não implementados na classe ComponenteComQuatroAbas.

```
142     public Componente copiar(){
143         return new PJanelaWin7(this);
144     }
145
146     public boolean isWindow(){
147         return true;
148     }
149
150     public boolean podeInserirSobre(){
151         return true;
152     }
153 }
```

Figura 33: Métodos da classe PJanelaWin7

Para que os componentes sejam adicionados ao painel de prototipação do Proface, é adicionado um *listener* para cada botão da nova paleta através do método addActionListener(ActionListener) da variável “temp” na linha 909 da Figura 31. Esse *listener* é efetivamente adicionado pela classe “TrataEventosBotaoComponentes” que implementa a interface *ActionListener*, onde seu *actionPerformed* é efetivamente configurado.

Após todos os *listeners* serem adicionados, o método insereComponente (Componente objeto) da classe PainelPrototipação é responsável por inserir um objeto no painel e adicionar os *listeners* necessários à manipulação desse componente inserido. Parte do método insereComponente pode ser visto na Figura 34.

[19] http://colorcop.net/

```
1   private void insereComponente(Componente objeto) {
2
3       objeto.setBounds(new Rectangle(objeto.getX(),
4               objeto.getY(),
5               objeto.getLargura(),
6               objeto.getAltura()));
7
8       int posicaoTemp = getComponentCount();
9
10      if (objeto.isWindow() && (posicaoTemp > 0))
11      {
12          Componente temp;
13          int i;
14          for(i = posicaoTemp - 1; i >= 0; i--)
15          {
16              temp = (Componente)getComponent(i);
17              if (!temp.isWindow())
18                  break;
19          }
20          posicaoTemp = i + 1;
21          add(objeto, posicaoTemp);
22      }
```

Figura 34: Parte do método insereComponente(Componente object) da classe PainelPrototipacao

Ao término da implementação da paleta, alguns defeitos foram corrigidos e recursos foram adicionados. Os defeitos corrigidos foram:

- Mensagem de erro exibida ao abrir o manual de ajuda pelo menu principal ou pela barra de ferramentas;
- Mensagem de erro que também era exibida ao abrir o manual de diretrizes de elaboração de protótipos pelo menu principal;
- Erro que ocorria ao tentar importar um protótipo para um projeto usando o botão direito do mouse pelo painel de arquivos de um projeto;
- Erro que estava impedindo que o recurso de adicionar imagem a um componente fosse utilizado.

Os recursos adicionados foram:

- Mudança na cor da interface principal da ferramenta. Ela agora possui a cor azul padrão do Microsoft Office do *Windows*;
- Inserção do ícone do Proface na sua tela principal e nas janelas padrão que a ferramenta utiliza, tais como: Importar protótipo, abrir projeto, selecionar imagem, etc;

- Inserção de ícones mnemônicos, onde estes não existiam nos submenus da barra de menus principal. Os ícones foram inseridos nos submenus: Fechar projeto, imprimir e Dados do projeto do menu Arquivo. Manual do sistema, diretrizes para projeto de interfaces e sobre o Proface do menu Ajuda;
- Inserção de botões para criar novo projeto e abrir projeto existente na barra de ferramentas principal;
- Adição de uma régua nas bordas do painel de prototipação para facilitar a medição dos componentes e ajudar no processo de prototipação;
- Adição de uma grade no fundo do painel de prototipação também usando a API *Java* 2D, para facilitar o alinhamento dos componentes;

Dois recursos que merecem destaque são a régua e a grade inseridos no painel de prototipação. A motivação para a inclusão desses recursos foram os pedidos dos usuários de versões anteriores do Proface. A paleta com componentes no estilo *Windows* 98 foi retirada da nova versão, visto que o sistema não possui mais suporte da *Microsoft* desde 2006[20].

Com a implementação da nova paleta concluída, os recursos extras adicionados e os defeitos corrigidos, a nova versão do *software* passou a ser denominada Proface 4.0. O arquivo .jar gerado foi embutido em um arquivo no formato .exe usando o aplicativo JSmooth[21] para que possa ser atualizado no sítio do *sourceforge*[22] onde poderá ser baixado. A nova tela principal do Proface pode ser vista na Figura 35.

[20] http://support2.microsoft.com/lifecycle/search/?sort=ES&alpha=Windows+98
[21] http://jsmooth.sourceforge.net/
[22] http://sourceforge.net/projects/proface/

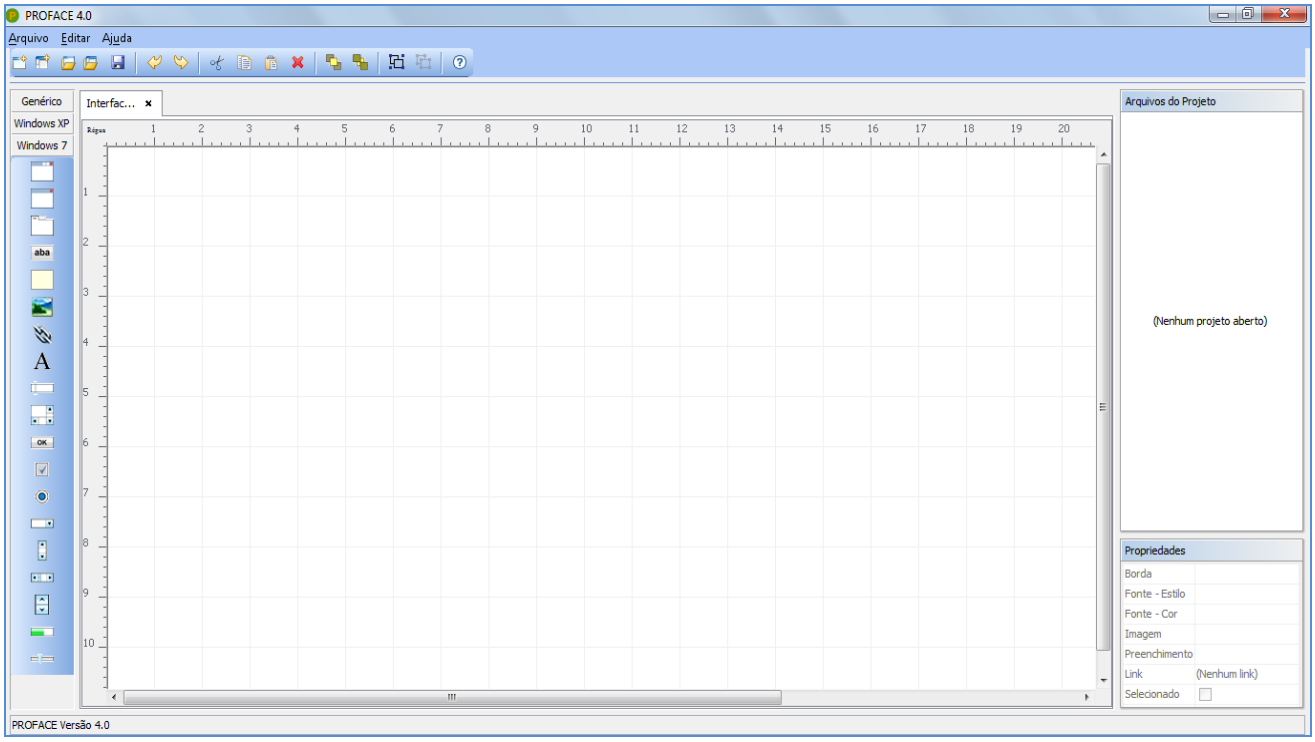

Figura 35: Tela principal Proface 4.0

Como resultado final da implementação da paleta no estilo *Windows* 7, foram elaborados dois protótipos usando os novos componentes. Estes podem ser vistos nas próximas Figuras.

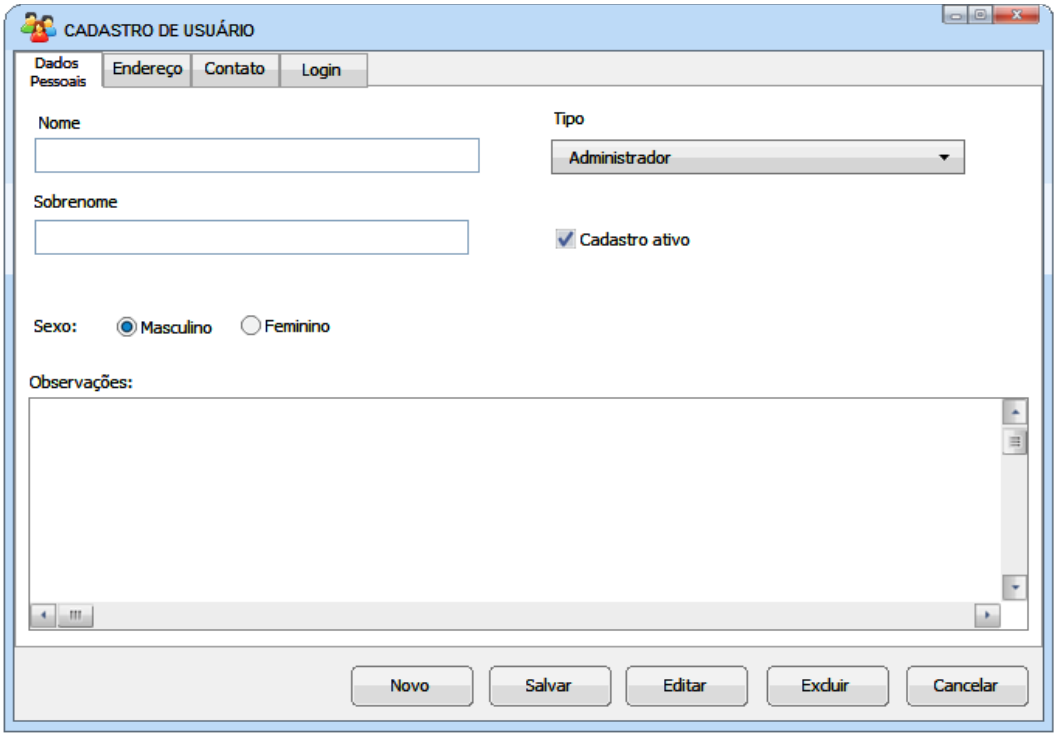

Figura 36: Protótipo de tela criado no Proface 4.0 com componentes estilo *Windows* 7

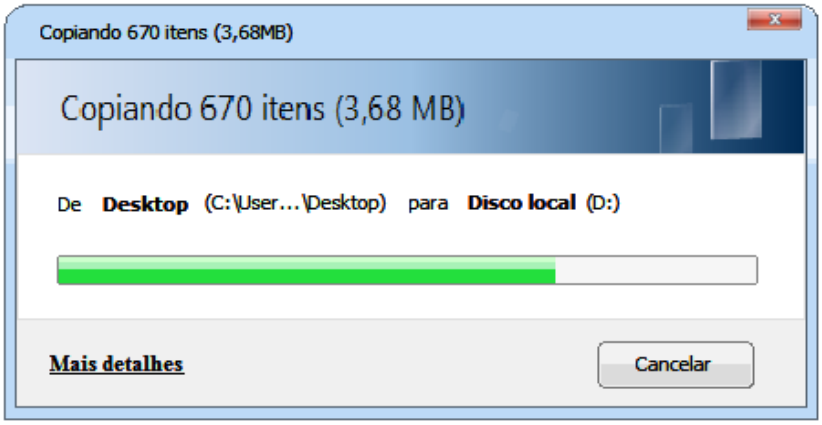

Figura 37: Protótipo janela de transferência de arquivos estilo *Windows* 7 criado no Proface 4.0

É possível observar nas Figuras 36 e 37 que os componentes da nova paleta permitem criar protótipos com um ótimo nível de detalhes e com cores fiéis às do *Windows* 7.

5 CONCLUSÃO

A prototipação de software compreende uma etapa crucial no desenvolvimento de sistemas, pois antecipa a validação de requisitos e possibilita que o cliente tenha um contato com uma abstração bem próxima do produto final.

Este trabalho teve como objetivo a evolução da ferramenta de prototipação de software Proface. Essa evolução se deu com algumas refatorações no código, inclusão de uma nova paleta de componentes no estilo *Windows* 7, uma régua e uma grade inseridos no painel de prototipação, melhorias visuais e alguns defeitos corrigidos.

Com as refatorações aplicadas, a compreensão e a legibilidade do código foram facilitadas permitindo assim que futuras mudanças sejam feitas de forma menos custosa. Com as melhorias, protótipos mais modernos podem ser construídos. Essa construção agora se faz de uma forma mais padronizada pela inclusão da régua e da grade que auxiliam no tamanho e alinhamento dos componentes.

Como trabalhos futuros para ferramenta pode-se destacar a inclusão das seguintes funcionalidades:

- Mudar o tamanho de componentes pelo painel de propriedades;
- Visualização de mudança no tamanho de um componente ao redimensioná-lo arrastando uma de suas extremidades;
- Exportar os protótipos para .html de forma dinâmica, de modo que os *links* criados sejam exportados juntamente com os protótipos facilitando a simulação entre as telas ao visualizá-las no *browser*;
- Paleta que permita construção de protótipos no estilo *Windows* 8;
- Componente tabela (*grid)* nas paletas de componentes.

Uma refatoração identificada e não aplicada neste trabalho por restrições no cronograma, também pode ser feita como trabalho futuro. A refatoração "*Replace Inheritance with Delegation*", que tem por objetivo substituir uma herança por delegação, pode ser aplicada na classe PComponentesSelecionados.

6 REFERÊNCIAS

Aguiar, Y. P. C. SMILE – uma Ferramenta para Geração Automática, Edição e Simulação de Protótipos de Interface do Usuário. 2007. 201 p. Dissertação (Mestrado em Informática) – COPIN/UFCG, Campina Grande, Brasil.

BECK, K; FOWLER, M. (2000).Planning Extreme Programming. Addison-Wesley Professional, 2000. 160 p.

FOWLER, M. Refactoring: Improve the design of existing code. United States: Addison Wesley Longman, Inc, 1999. 431p.

GREENBERG, Saul; CARPENDALE, Sheelagh; MARQUARDT, Nicolai; BUXTON, Bill.**The Narrative Storyboard Telling a story about use and context over time.** Interactions, GroupLab University of Calgary, Fev. 2012. Disponível em: <http://grouplab.cpsc.ucalgary.ca/grouplab/uploads/Publications /Publications/2012-NarrativeStoryboard.Interactions.pdf>. Acesso em: 24 jan. 2014.

MAGUIRE, Martin. Methods to support human-centred design. Husat Research Institute, Leicestershire, Uk, p.587-634, 26 jun. 2001. Disponível em: <http://www.cse.chalmers.se/research/group/idc/ituniv/courses/06/ucd/papers/maguire 2001a.pdf>. Acesso em: 23 jan. 2014.

MENS, Tom. TOURWÉ, Tom. A Survey of Software Refactoring. **IEEE Transactions on Software Engineering**, v.30, n.2, 2004.

MINUZZI, T. S. Ustory-refactory: ferramenta de refatoração de requisitos aplicada em cartões user stories (crc cards). 2001. 118 f. Dissertação (Mestrado em Computação Aplicada). Centro de Ciências Exatas e Tecnológicas, Universidade do Vale do Rio dos Sinos – UNISINOS, São Leopoldo. 2007.

OPDYKE, W. F.1992. 206 f. Refactoring Objetc-Oriented Frameworks. Tese (Doutorado em Ciência da Computação), University of IIlinois, Urbana-Champaign. 1992.

PREECE, Jennifer; ROGERS, Yvonne; SHARP, Helen. **Design de Interação – Além da interação homem-computador.** Porto Alegre: Bookman, 2005.

PRESSMAN, Roger S. **Engenharia de *Software***. 6º edição. Rio de Janeiro: McGrawHill, 2002.

ROBERTS, D.; BRANT, J.; JOHNSON, R. E. A Refactoring Tool for Smalltalk: **Theory and Practice of Object Systems**, [S.l], v.3, n.4, p.253-263, 1997.

ROBERTS, D.; BRANT, J.; JOHNSON, R. E. An Automated Refactoring Tool. In: INTERNATIONAL CONFERENCE ON ADVANCED SCIENCE AND TECHNOLOGY, 1996, Chicago, EUA. **Anais**...[S.l.:s.n], 1996.

ROBERTS, D. B. 1999.137 f. Practical Analysis for Refactoring.Tese (Doutorado em Ciência da Computação), University of IIlinois, Urbana-Champaign. 1999.

SILVA, Silvio S.; SAVOINE, Márcia M. Análise comparativa de ferramentas computacionais para prototipação de interfaces. **Revista Científica do ITPAC**, Tocantins, v. 3, n. 3, p. 45-54, 2010.

SOMMERVILLE, Ian. **Engenharia de *Software***. São Paulo: Addison Wesley, 2003.

WILLS, Linda; NEWCOMB, Philip. **Reverse Engeneering**. 3 ed. reimpressa. Netherlands: Kluwer Academic Publishers, 1996.

APÊNDICE A - Conjunto de testes realizados antes e após o processo de refatoração

- Criação de um novo componente no painel de prototipação;
- Mudança na cor de preenchimento dos componentes “Background” e “Botão”;
- Mudança na cor de borda do componente “Background”;
- Criação do componente “Imagem” e alteração da imagem deste mesmo componente;
- Cópia, deleção e recorte dos componentes e em seguida execução do comando desfazer;
- Arrastar componentes no painel de prototipação;
- Executar salvamento de um protótipo;
- Executar abertura de um protótipo salvo;
- Criação de um projeto e adição de protótipos;
- Executar salvamento de um projeto;
- Executar abertura de um projeto salvo.

APÊNDICE B - Conjunto de testes realizados após a evolução do Proface

- Redimensionamento e alinhamento de cada componente a régua e grade adicionadas;
- Exportação de um projeto para “.html”;
- Criação de um novo projeto pelo novo botão adicionado a barra de ferramentas;
- Abertura de um projeto pelo novo botão adicionado a barra de ferramentas;
- Exportação de um protótipo para “.PNG” e “.JPG”;
- Impressão de um protótipo;

APÊNDICE C – Classe Componente com todos os seus atributos e métodos

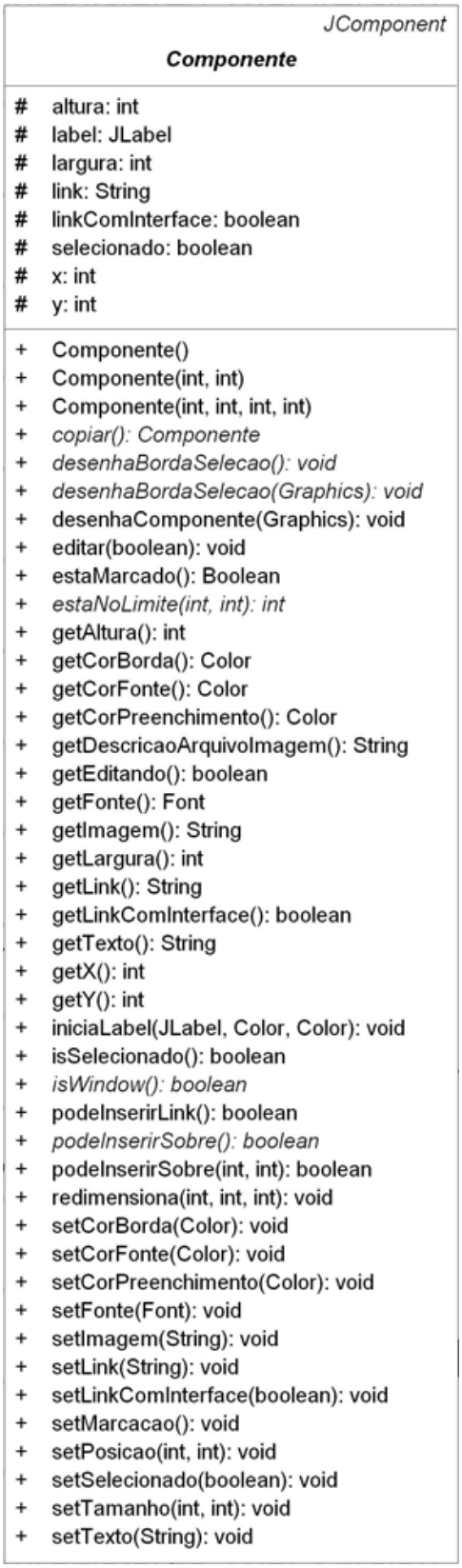

Figura 38: Classe componente com todos os atributos e métodos