UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

COMPONENTES GRÁFICOS PARA PROTOTIPAGEM E DOCUMENTAÇÃO RÁPIDA EM DELPHI

REINOLDO KRAUSE JUNIOR

BLUMENAU
2016

REINOLDO KRAUSE JUNIOR

COMPONENTES GRÁFICOS PARA PROTOTIPAGEM E DOCUMENTAÇÃO RÁPIDA EM DELPHI

Trabalho de Conclusão de Curso apresentado ao curso de graduação em Ciência da Computação do Centro de Ciências Exatas e Naturais da Universidade Regional de Blumenau como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação.

Prof.ª Luciana Pereira de Araújo, Mestra - Orientadora

BLUMENAU
2016

COMPONENTES GRÁFICOS PARA PROTOTIPAGEM E DOCUMENTAÇÃO RÁPIDA EM DELPHI

Por

REINOLDO KRAUSE JUNIOR

Trabalho de Conclusão de Curso aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II pela banca examinadora formada por:

Presidente:	Profª Luciana Pereira de Araújo, Mestra – Orientadora, FURB
Membro:	Prof. Franciso Adell Péricas, Mestre – FURB
Membro:	Prof. Matheus Carvalho Viana, Doutor – FURB

Blumenau, 04 de julho de 2016

Dedico este trabalho a minha amada namorada, a meus pais sempre presentes e que me permitiram chegar onde estou hoje, a minha querida irmã, aos meus amigos e a minha orientadora por todo incentivo.

AGRADECIMENTOS

A Deus por amar a todos de forma incondicional, mesmo sendo nós pecadores.

A minha família por todo apoio, cuidado e amor que tem comigo.

A minha namorada por toda compreensão, incentivo, desabafos e amor.

Aos meus amigos que passaram noites sem dormir, que se reuníamos finais de semana para terminar trabalhos. Amigos que viraram irmãos.

A minha orientadora Luciana Pereira de Araújo que se dedicou muito ao meu TCC e me incentivou acreditando na ideia do trabalho desenvolvido.

Ao Jacques Robert Heckmann que iniciou minha orientação ao TCC, dando as direções de pesquisa, mas que depois teve que se ausentar por motivos de saúde.

Eu não sou quem eu gostaria de ser; eu não sou quem eu poderia ser, ainda, eu não sou quem eu deveria ser. Mas graças a Deus eu não sou mais quem eu era!

Martin Luther King

RESUMO

Este trabalho apresenta a especificação, implementação e operacionalidade dos componentes gráficos desenvolvidos para auxiliar na prototipagem de telas e transformar a prototipagem realizada em modelos de casos de uso e de classes da *Unified Modeling Language* (UML) compatíveis às ferramentas de modelagem de software. A ferramenta foi desenvolvida utilizando a linguagem de programação Object Pascal, no ambiente de desenvolvimento Delphi 10 Seattle. Para geração dos modelos da UML foi utilizado o padrão *XML Metadadata Interchange* (XMI) garantindo a visualização e alteração dos artefatos da UML em diversas ferramentas de modelagem de software. Para a realização dos testes de importação e validação do XMI foram utilizados os softwares de modelagem de dados ArgoUML e Enterprise Architect. A partir de uma pesquisa de campo foi possível identificar que os diagramas de casos de uso e de classes gerados pelos componentes gráficos criados apresentaram-se compatíveis com as prototipagens de telas realizadas. Assim como, os participantes da pesquisa destacaram a agilidade que esta ferramenta traz para geração da documentação. No entanto, a ferramenta ainda apresenta algumas limitações para os diagramas de classes e caso de uso, como o uso de todos os relacionamentos entre classes, relacionamentos entre casos de uso e nos tipos de dados dos atributos gerados por um componente gráfico do tipo grade.

Palavras-chave: Componentes gráficos. Delphi. XMI. UML. Prototipagem. Documentação de sistema. Diagrama de classes. Diagrama de casos de uso.

ABSTRACT

This work presents the specification, implementation and operation of graphical components designed to assist in prototyping screens. And it has intent to turn prototyping performed in use cases and classes models of Unified Modeling Language (UML) compatible with software modeling tools. The tool was developed using the Object Pascal programming language, in development environment 10 Delphi Seattle. For generation of UML models we used the standard XML Metadadata Interchange (XMI) ensuring viewing and changing the UML artifacts in several software modeling tools. For the realization of import testing and validation of XMI data modeling we used ArgoUML software and Enterprise Architect. From a field research it was possible to identify that the use case diagram and class diagram generated by graphics components were compatible with prototyping made of screens. As the survey participants highlighted the agility that this tool brings generation of documentation. However, the tool still has some limitations for class and use case diagrams, such as the use of all relationships among classes, relationships between use cases and attributes data types generated by the graphic component grid.

Key-words: Graphics components. Delphi. XMI. UML. Prototyping. System documentation. Class diagram. Use case diagram.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

FMX – Firemonkey

IDE – *Integrated Development Environment*

RF – Requisitos Funcionais

RNF – Requisitos Não Funcionais

UML – *Unified Modeling Language*

XMI – *XML Metadadata Interchange*

XML – *Extensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

Não é de hoje que muitos projetos de software terminam fora do prazo, ou então, acabam não atendendo realmente a necessidade do cliente. De acordo com a pesquisa do Standish Group, os resultados de 2012 indicam que 39% dos projetos foram entregues dentro do prazo, 43% dos projetos foram atrasados e/ou com menos recursos e funções necessárias e que 18% falharam, ou seja, foram cancelados antes da conclusão ou entregues e nunca usados (CHAOS, 2013).

Juntamente com esta pesquisa foi elaborada uma lista dos fatores que indicam o sucesso destes 39%. Os cinco principais fatores são: suporte dos responsáveis pelo projeto, envolvimento dos usuários, otimização de processos, habilidade da equipe e a experiência em gestão de projetos (CHAOS, 2013). Nota-se que a execução de um bom projeto não depende apenas da qualificação dos desenvolvedores, mas também em saber usar a maior fonte de dados que são os usuários e responsáveis pelo produto.

Essa fonte de dados deve ser extraída na elicitação dos requisitos do sistema. É indicado utilizar a prototipação como uma parte da modelagem do software pois, segundo Galeote (2012), “O protótipo torna o produto real o suficiente para os envolvidos trazerem requisitos, que sem o protótipo não seriam capturados”. Esse processo facilita o entendimento de quem está solicitando, e de quem está elaborando o projeto.

Contudo, a modelagem do software não pode parar apenas na prototipação. Muitas empresas não mantêm modelos de software por acharem que gastarão muito tempo na sua elaboração e manutenção. Elas enganam-se, pois os modelos ajudam tanto a economizar o tempo quanto na qualidade final do produto, pelo motivo de ser a melhor forma de resolver e entender um problema em questão (LOBO, 2009).

A documentação não é somente útil no início do desenvolvimento, mas é importante também para seu ciclo de vida inteiro. Ela informa o porquê do software ter sido construído da forma que foi e reduz as chances de afetar a integridade do sistema quando realizada a manutenção do software (BROOKSHEAR, 2013).

Com base nestes fatos, o trabalho desenvolvido visa melhorar o envolvimento do usuário durante a elicitação de requisitos, assim como permitir que o processo de documentação seja mais rápido que a forma tradicional. Para isso, pretende-se desenvolver componentes gráficos específicos para prototipagem, os quais permitirão a geração de diagramas de classes e casos de uso.

De acordo com Lobo (2009, p. 18), “modelar software com UML é uma tarefa indispensável no desenvolvimento de sistemas”. Este trabalho se torna relevante por oferecer uma nova possibilidade aos analistas de sistemas modelarem um software usando a prototipagem.

Assim como Freitas (2008) reforça, “um protótipo é de extrema importância durante as primeiras fases de análise de um sistema”. A ferramenta tem o intuito de disponibilizar um ambiente em que poderão ser realizados protótipos de software. Além disso, com base no protótipo, será possível a geração de dois tipos de artefatos da UML, sendo o diagrama de casos de uso e o diagrama de classes. Nota-se também na relevância tecnológica a utilização do padrão XMI para garantir a visualização e alteração dos artefatos da UML em diversas ferramentas de modelagem de software.

1.1 OBJETIVOS

O objetivo deste trabalho é desenvolver componentes gráficos para auxiliarem na prototipagem de telas e implementar a transformação das informações da prototipagem realizada em diagramas de casos de uso e de classes da UML, compatíveis com as ferramentas de modelagem de software.

Os objetivos específicos do trabalho são:

a) desenvolver componentes gráficos para a prototipagem;
b) implementar a geração dos diagramas de casos de uso da UML;
c) implementar a geração dos diagramas de classes da UML;
d) integrar os componentes gráficos desenvolvidos na ferramenta de Delphi 10 Seatle;
e) validar se os artefatos gerados pelos componentes gráficos desenvolvidos condizem com a prototipagem realizada.

1.2 ESTRUTURA

Este trabalho está dividido em quatro capítulos. O primeiro capítulo é composto pela justificativa do trabalho, os objetivos definidos para o mesmo e a apresentação de sua estrutura.

O segundo capítulo apresenta a fundamentação teórica, abordando conceitos sobre a modelagem e documentação de software, esclarecendo pontos que devem ser considerados no desenvolvimento do trabalho. No terceiro capítulo é apresentado o desenvolvimento da ferramenta, onde são listados os requisitos principais e a especificação por meio de diagramas.

Na sequência, é detalhada a implementação da ferramenta desenvolvida, descrevendo as técnicas e ferramentas utilizadas, e são apresentados e discutidos os resultados obtidos.

Por fim, o quarto capítulo apresenta as conclusões e as sugestões para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo está organizado em seis seções. A seção 2.1 expõe a necessidade de ser feito uma documentação do sistema. A seção 2.2 detalha a etapa da prototipagem. A seção 2.3 embasa os artefatos a serem gerados pela proposta. A seção 2.4 descreve a estrutura do padrão XMI. A seção 2.5 explana como desenvolver componentes gráficos em Delphi. Por fim, a seção 2.6 traz um breve estudo de quatro trabalhos correlatos.

2.1 DOCUMENTAÇÃO DE SISTEMAS

Faz-se realmente necessário ter uma documentação do sistema? De acordo com Guedes (2011, p. 21), “um sistema de informação precisa ter uma documentação extremamente detalhada, precisa e atualizada para que possa ser mantido [sic] com facilidade, rapidez e correção, sem produzir novos erros ao corrigir os antigos”. A demanda de geração da documentação depende de cada fase do desenvolvimento do sistema computacional. Elas estão divididas em definição de requisitos, análise, projeto, desenvolvimento, teste e implantação (MACÈDO; SPÍNOLA, 2011).

Cada fase pode produzir diferentes tipos de artefatos. Por exemplo, na fase de definição de requisitos podem ser criados artefatos como modelos de casos de uso, assim como a prototipação de telas. Na fase de análise os artefatos podem ser documentos de orçamento e cronograma, documentação de regras de negócios e refinação dos casos de uso. A fase de projeto inclui modelos de componentes e diagramas de classes para visualização da arquitetura do projeto. Na fase de desenvolvimento os artefatos são propriamente os códigos fontes realizados com base na documentação construída nas fases anteriores. Na fase de testes podem-se aplicar testes unitários e de aceitação. Por fim, a fase de implantação usa planos de publicação, planos de treinamento, testes alfa e beta (SAUVÉ, 2001). Sendo assim, percebe-se que a documentação está atrelada a várias fases do desenvolvimento do sistema computacional, sendo fundamental para um software bem projetado.

A pergunta que alguns fazem é o quanto de documentação deve-se ter? Ou, o quanto se deve investir em tempo e pessoas para fazer esta tarefa? As respostas para essas perguntas impactam diretamente no método de desenvolvimento escolhido, podendo ser métodos tradicionais ou ágeis. Os métodos tradicionais basicamente focam em artefatos comuns para todos os tipos de projetos com uma documentação abrangente. Já a metodologia ágil, como o nome diz, foca na agilidade de concluir o software com uma documentação concisa (MORAIS, 2011).

Decidir qual metodologia utilizar não é apenas uma opção da empresa que desenvolverá, mas também da exigência de documentação que o cliente pode solicitar. Conforme Morais (2011), “o que se quer com a modelagem do sistema é satisfazer as necessidades dos usuários, garantindo a qualidade interna do software e respeitando as restrições de custo e prazo do projeto”.

O trabalho desenvolvido está diretamente relacionado à fase de definição de requisitos e análise do software, pois permite a realização de parte da documentação do sistema. Essa documentação está relacionada à modelagem da prototipagem, construção do diagrama de casos de uso e diagrama de classes.

2.2 PROTOTIPAGEM

A definição de prototipagem pode ser dada, conforme Bezerra (2007, p. 41), como “No contexto do desenvolvimento de software, um protótipo é um esboço de alguma parte do sistema. A construção de protótipos é comumente chamada de prototipagem.”.

Pressman (1995) ressalta a eficiência e agilidade da etapa da prototipagem no projeto do software:

> A prototipação é um paradigma eficiente da engenharia de software. A chave é definir-se as regras do jogo logo no começo; ou seja, o cliente e o desenvolvedor devem ambos concordar que o protótipo seja construído para servir como um mecanismo a fim de definir os requisitos (Pressman, 1995, p. 38).

Rogers, Sharp e Preece (2013, p. 391) complementam Pressman (1995) mostrando diferentes propósitos para os protótipos, sendo eles: “para testar a viabilidade técnica de uma ideia, para esclarecer alguns requisitos vagos, para fazer algum teste e avaliação com usuários ou para verificar se uma direção do *design* é compatível com o resto do desenvolvimento do produto”. Portanto, é de grande valia realizar a etapa da prototipagem por se manter uma documentação e por não atrasar tanto o desenvolvimento do software. Além do que, a mesma é criada e validada juntamente com o cliente, podendo até mesmo ser reutilizada para a primeira versão do sistema.

Independentemente do propósito, existem três vertentes de prototipação, sendo a prototipação de baixa, média e alta fidelidade (ROGERS; SHARP; PREECE, 2013). O protótipo de baixa fidelidade não se assemelha ao produto final, serve de base para críticas. Normalmente são rascunhos em papeis e não são usados na documentação. O protótipo de média fidelidade possui algumas definições importantes para o usuário e para o desenvolvedor. Esse tipo de protótipo é feito por ferramentas específicas pois, não são somente esboços. Por sua vez, o protótipo de alta fidelidade se assemelha bastante ao produto

final, muitas vezes realizado na mesma tecnologia do sistema final (ARAÚJO; VALE; SOUZA, 2008).

Complementando a prototipagem de alta fidelidade, ela pode se subdividir em dois tipos de protótipos, o descartável e o evolutivo. O protótipo descartável, como o próprio nome diz, será descartado depois de cumprida a sua finalidade, que é refinar a documentação para gerar qualidade para o produto que se deseja criar. O protótipo evolutivo visa ser entregue como produto final, o qual é evoluído a cada etapa de desenvolvimento do sistema sendo avaliado a cada mudança pelo usuário (ARAÚJO; VALE; SOUZA, 2008).

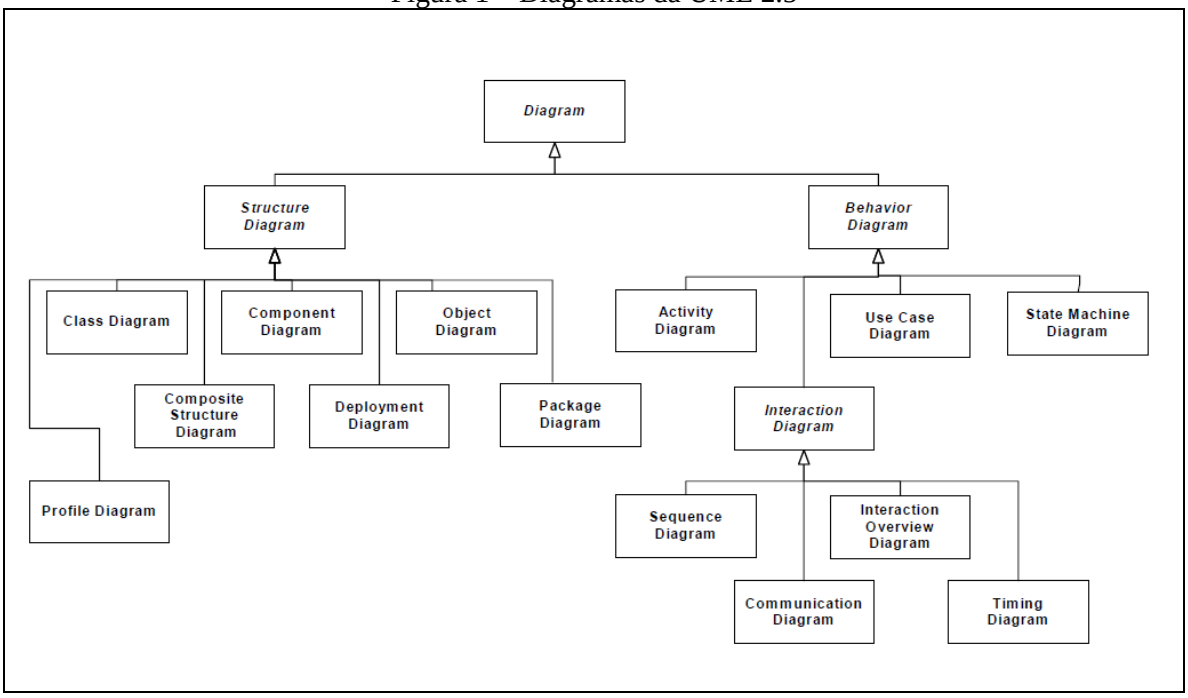
A ferramenta desenvolvida permite gerar protótipos que sejam para versão final do produto, assim como apenas para refinar a documentação. Não será impedida a aplicação de qualquer tipo de prototipagem pois, ela ficará a critério da escolha de quem usar, assim como também não implicará obrigatoriamente na geração dos diagramas.

2.3 UML

A modelagem de sistemas pode ser realizada com a linguagem UML. Ela define elementos gráficos que podem representar cinco tipos de visões do sistema, como visão de casos de uso, de projeto, de implementação, implantação e de processo. Um processo de desenvolvimento que utilize a UML como linguagem de modelagem envolve a criação de diversos documentos. Na terminologia da UML, esses documentos são denominados artefatos de software, ou simplesmente artefatos (BEZERRA, 2007).

Atualmente na versão 2.5, a UML possui quatorze diagramas, os quais podem ser visualizados na Figura 1, agrupados em diagramas de estrutura e comportamento, cada um possuindo sete diagramas. Os diagramas de estrutura são os mais à esquerda, começando de cima para baixo, sendo o diagrama de classe, diagrama de objeto, diagrama de pacote, diagrama de estrutura composta, diagrama de componentes, diagrama de perfil e diagrama de implantação. Já os diagramas de comportamento são os mais à direita, sendo o diagrama de casos de uso, diagrama de atividade, diagrama de máquina de estado, diagrama de sequência, diagrama de comunicação, diagrama de temporização e diagrama geral de interação (OBJECT MANAGEMENT GROUP, 2015, p 729).

Figura 1 – Diagramas da UML 2.5



Fonte: Object Management Group (2015).

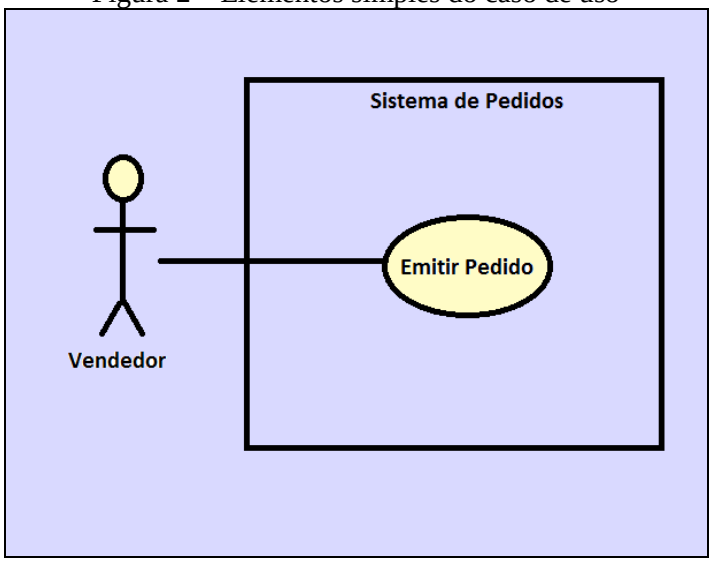
Conforme Wazlawick (2011, p. 4), “Nem todos os diagramas precisam ser usados durante o desenvolvimento de um sistema. Usam-se apenas aqueles que possam apresentar alguma informação útil para o processo”. Portanto, nas seções 2.3.1 e 2.3.2 são apresentados dois diagramas da UML de forma mais detalhada pelo fato destes serem gerados pela ferramenta que este trabalho apresenta e por servirem de base a muitos outros diagramas.

2.3.1 Diagrama de casos de uso

O diagrama de casos de uso procura identificar os atores (usuários, outros sistemas ou até mesmo algum hardware especial) que utilizarão de alguma forma o software (GUEDES, 2011). Este diagrama possui uma linguagem de fácil compreensão, ele é utilizado principalmente na elicitação dos requisitos do sistema, embora venha a ser consultado durante todo o processo de modelagem e sirva de base para todos os outros diagramas (ESPOSITO, 2013).

Alguns dos elementos utilizados no diagrama de casos de uso podem ser observados na Figura 2, onde o vendedor é o ator responsável pelo caso de uso `emitir pedido`, o retângulo maior representa uma fronteira na qual o ator pode atuar no sistema de pedidos.

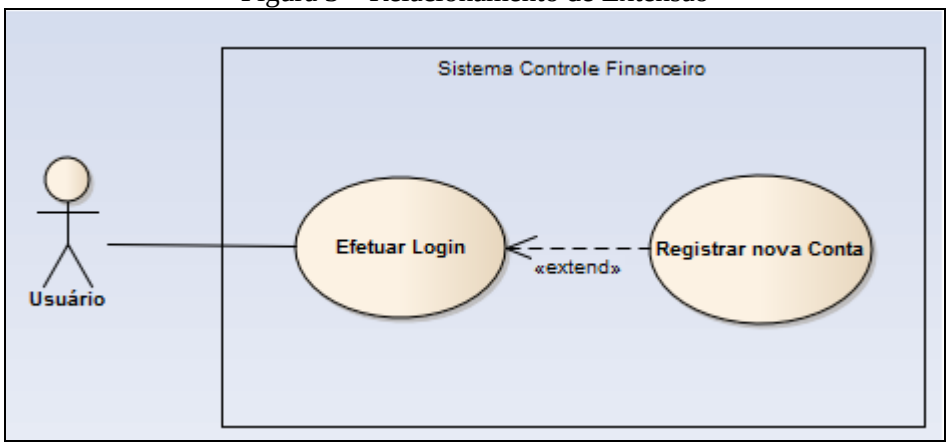
Figura 2 – Elementos simples do caso de uso



Representando de uma forma textual o caso de uso da Figura 2, seria: o sistema deve permitir ao vendedor emitir pedidos. O diagrama de casos de uso ainda possui dois elementos fundamentais que são os relacionamentos de *extends* (estender) e *include* (incluir) (WAZLAWICK, 2011). O *extends* tem como função estender a funcionalidade de um outro caso de uso e o *include* adicionar a funcionalidade de outro caso de uso (ESPOSITO, 2013).

A Figura 3 mostra o relacionamento de estender, onde o caso de uso `efetuar login` é estendido pelo caso de uso `registrar nova conta`. Assim, caso o ator usuário não possua uma conta ao efetuar o login, poderá criá-la neste instante.

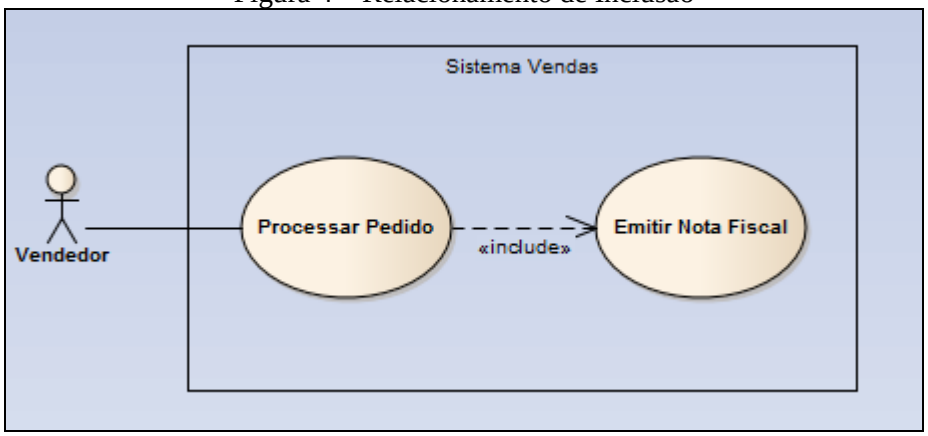
Figura 3 – Relacionamento de Extensão



Fonte: Esposito (2013).

Já a Figura 4 mostra a situação de inclusão, ou seja, o ator vendedor ao utilizar o caso de uso `processar pedido` automaticamente usará o caso de uso `emitir nota fiscal`.

Figura 4 – Relacionamento de Inclusão

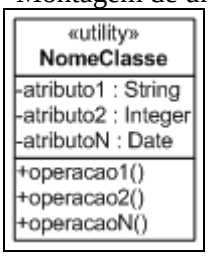


Fonte: Esposito (2013).

2.3.2 Diagrama de classes

O diagrama de classes, como o próprio nome diz, define a estrutura das classes utilizadas pelo sistema, determinando os atributos e métodos que cada classe tem, além de estabelecer como as classes se relacionam e trocam informações entre si (GUEDES, 2011). Este processo é realizado na análise do projeto com base nos casos de uso elaborados. Na Figura 5 pode ser visualizada a montagem de uma classe, a qual é dívida em três partes, a primeira sendo o nome da classe, a segunda com uma lista de seus atributos e a terceira com uma lista de operações (MELO, 2009). Cada parte tem as suas próprias especificações.

Figura 5 – Montagem de uma Classe



Fonte: Melo (2009).

A expressão `<<utility>>` representa o estereótipo da classe, ou seja, segundo Melo (2009) “é um mecanismo de extensibilidade da UML que representa uma subclasse de um elemento já existente, com o mesmo formato, porém com objetivos diferentes e bem definidos”. Portanto, se informado, significa que será uma classe do tipo utilitária, uma classe que oferece recursos úteis a outras classes.

O nome da classe deve respeitar um padrão, ser em negrito, centralizado sem acentuação e as iniciais devem ser maiúsculas. Caso seja um nome composto deve ser escrito sem espaço entre o primeiro e o segundo nome. A primeira letra do segundo nome deve ser maiúscula. Na lista de atributos e de operações, os nomes devem iniciar com letra minúscula,

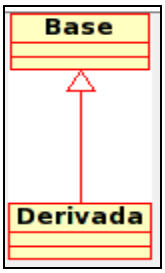
não devem possuir espaçamento e, se forem compostos, a primeira letra do nome composto deve ser em maiúscula.

Na sintaxe do atributo e da operação o primeiro caractere indica a visibilidade, podendo ser público (+), privado (-) ou protegido (#). Essas definições respeitam os significados de visibilidade da orientação de objetos. O atributo pode ter a definição de um tipo de dado (`string`, `integer`, `float`...) (MELO, 2009; WAZLAWICK, 2011).

O diagrama de classes ainda possui seus relacionamentos com outras classes que podem ter cinco variações, sendo elas a generalização, associação, agregação, composição e dependência (WAZLAWICK, 2011). Além disso é possível informar a multiplicidade em cada relacionamento de cada classe. A multiplicidade indica a quantidade de vezes que um objeto de uma classe pode estar relacionado com outra classe.

A generalização entre classes é informada por uma linha contínua com uma seta no lado da classe base. Portanto, a classe que estiver apontando para a outra, estará herdando todas as suas características (HENSGEN, 2013). Na Figura 6 pode ser observado que a classe derivada herda a classe base.

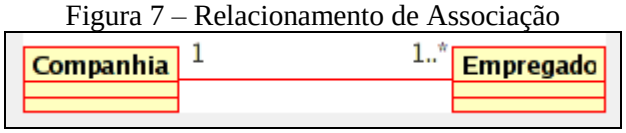
Figura 6 – Relacionamento de Generalização



Fonte: Hensgen (2013).

A associação entre classes é informada por uma linha contínua sem setas. Ela indica apenas uma relação simples podendo esta ter qualquer tipo de multiplicidade (HENSGEN, 2013). A Figura 7 ilustra a associação das classes na qual a classe `Empregado` possui uma companhia e a classe `Companhia` pode possuir um ou mais empregados.

Figura 7 – Relacionamento de Associação

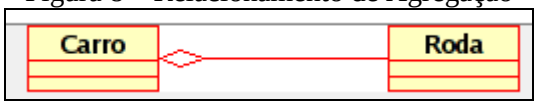


Fonte: Hensgen (2013).

A agregação entre classes é informada por uma linha contínua com um losango no lado da classe que possui toda a outra classe. A agregação é utilizada para especificar que uma classe é parte de outra classe, ou seja, o objeto gerado pela classe é parte de um todo (WAZLAWICK, 2011). Observa-se na Figura 8 que toda classe carro tem a classe roda.

Porém, neste tipo de relacionamento, se o objeto carro deixar de existir, a roda ainda existirá (HENSGEN, 2013).

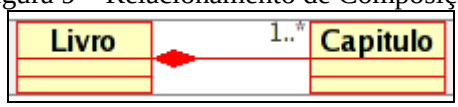
Figura 8 – Relacionamento de Agregação



Fonte: Hensgen (2013).

A composição entre classes é informada por uma linha contínua com um losango pintado no lado da classe que possuir toda a outra classe. A composição também é utilizada para representar parte de um todo, porém se a classe que recebeu o losango deixar de existir, as outras partes também deixam de existir (HENSGEN, 2013). A Figura 9 mostra um exemplo de composição onde a classe capítulo não existe sem a classe livro e vice-versa.

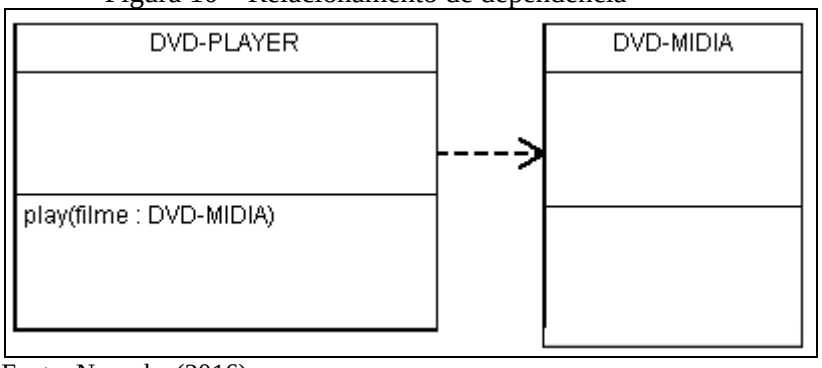
Figura 9 – Relacionamento de Composição



Fonte: Hensgen (2013).

Por fim, a dependência entre classes é informada por uma linha tracejada com um terminal em seta para indicar quem depende de quem. Conforme Noronha (2016) “nesse contexto, diz-se que uma classe utiliza a outra como argumento em sua assinatura”. A Figura 10 mostra um exemplo de dependência que segundo Noronha (2016) “a assinatura do método play da classe DVD-PLAYER recebe como parâmetro um objeto ou instância da classe DVD-MIDIA”.

Figura 10 – Relacionamento de dependência



Fonte: Noronha (2016).

2.4 XMI

Quando a UML apareceu pela primeira vez, não houve nenhum formato padrão para a troca de modelos. A maioria das ferramentas de modelagem de software tinha seu próprio formato textual. Para compreender era necessário executar a engenharia reversa. Portanto, foi projetado o XMI pela Object Management Group (OMG) (STEVENS, 2003).

O XMI é um padrão baseado em *Extensible Markup Language* (XML) para exportar modelos definidos em UML. Esse padrão é importante, pois permite a interoperabilidade entre diversas ferramentas de modelagem de softwares (BEZERRA, 2007). A diferença entre os dois padrões é que o XML armazena as informações em modo de árvore estruturada e não é orientada a objetos. Já o XMI estende do XML para torná-lo orientado a objeto. Antes de apresentar a formatação do XMI, é necessário esclarecer alguns termos como *schema* e *namespace*.

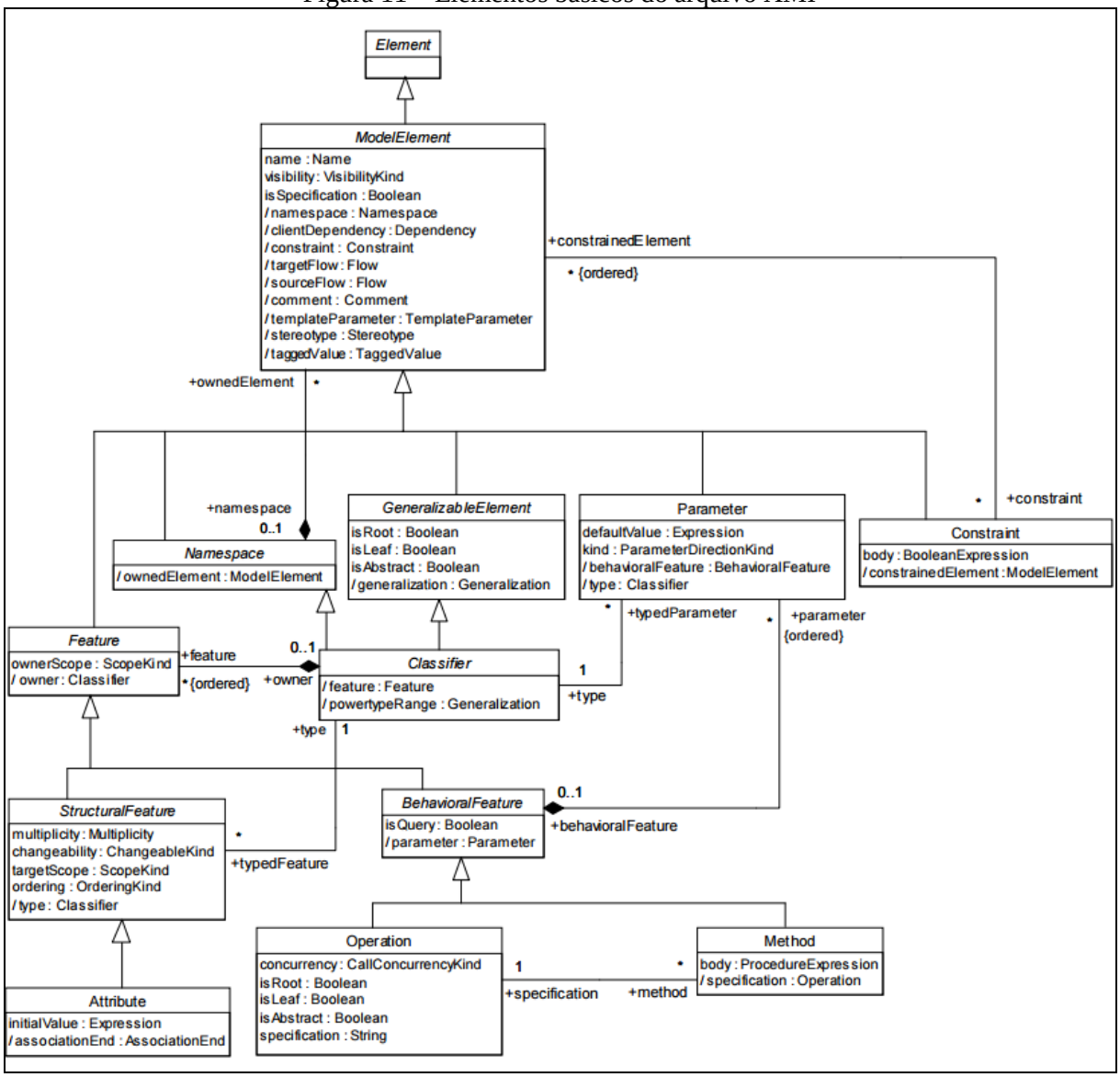
O *schema* XML é um arquivo codificado em linguagem baseada em padrão XML que contém a definição da estrutura de um documento XML, as definições de tipo, tamanho, ocorrência e regras de preenchimento dos elementos que compõem documento XML. O *namespace* é um método para evitar conflitos com nomes de elementos do XML.

Cada arquivo XMI é composto por declarações que geralmente são conhecidas como *tags*. A seguir, são listadas algumas dessas declarações, retiradas do documento provido pela Object Management Group (OMG) a qual é responsável pelas especificações da UML (OBJECT MANAGEMENT GROUP, 2015):

a) uma instrução de versão de processamento XML. Exemplo: <? Xml version = “1.0”>;
b) uma declaração de codificação opcional que especifica o conjunto de caracteres que segue a ISO-10646 (também chamado estendido Unicode) padrão (exemplo: <? Xml version = "1.0" encoding = "UCS-2">);
c) quaisquer outras instruções de processamento XML válidos;
d) um elemento XML *schema*;
e) um elemento XML de importação para o *namespace* XMI;
f) as declarações de um modelo específico.

A Figura 11 representa os elementos básicos que podem ser usados em um arquivo XMI, com suas respectivas propriedades. Dessa forma, é possível identificar quais elementos do arquivo XMI são utilizados para a comunicação com os modelos da UML.

Figura 11 – Elementos básicos do arquivo XMI



Fonte: Object Management Group (2001).

A modelagem exemplificada na Figura 11, demonstra algumas das classes a serem utilizadas na especificação de um arquivo XMI. A classe *model* representa o modelo a ser exportado. Logo após, a classe *namespace* é utilizada para definir os elementos pertencentes ao *model*. Por fim, a classe *classifier* representa a classe do diagrama de classes a ser exportada. Com esses passos é possível ter uma classe exportada, porém sem atributos ou métodos. Ou seja, visualizando o arquivo apenas com as classes criadas sendo cada uma representada em *tags* XML sem os valores das propriedades definidas ficaria conforme o Quadro 1. Portanto, para complementar cada *tag* poderia ser adicionado os atributos que cada classe dispõe na modelagem da Figura 11.

Quadro 1 – *Tags* arquivo XMI

```
<UML:Model>
    <UML:Namespace.ownedElement>
        <UML:Class>
        </UML:Class>
    </UML:Namespace.ownedElement>
</UML:Model>
```

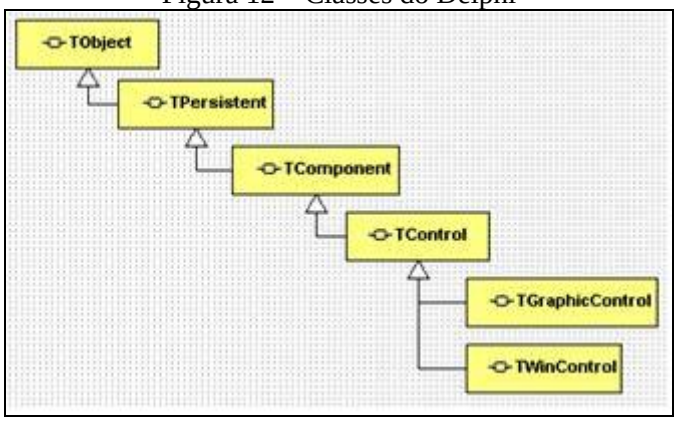
2.5 COMPONENTES GRÁFICOS EM DELPHI

Os componentes gráficos, também chamados de artefatos de software, são classes especiais que descendem de uma das classes principais do Delphi, que é a `TComponent`. Desta forma, podem ser instalados no ambiente de desenvolvimento integrado (FARIA, 2006). Eles podem ser definidos em dois tipos de componentes gráficos: os visuais e os não visuais.

Componentes gráficos visuais são os que aparecem tanto em tempo de *design* do projeto quanto em tempo de execução, como por exemplo, botões. Os não visuais não aparecem em tempo de execução. Um exemplo são componentes gráficos de acesso a dados. Eles não aparecem na aplicação, mas proveem funcionalidade para acessar bancos de dados (PAULI, 2008).

Conforme Pauli (2008), a criação de componentes gráficos no Delphi é simples. É um processo manual, feito em linha de código. O primeiro passo a fazer é identificar a classe base que será herdada. O segundo passo é codificar o comportamento adicional que o componente terá. O terceiro passo é instalá-lo como forma de *plugin* no Delphi. A identificação de qual classe herdar pode ser vista na Figura 12, a qual apresenta seis classes, todas elas herdeiras da classe `TObject`.

Figura 12 – Classes do Delphi



Fonte: Pauli (2008).

A `TObject` está no topo da hierarquia, ela serve de base direta ou indireta para todos as outras classes. A classe `TPersistent` introduz os métodos de persistência, permitindo salvar

e recuperar dados do componente. Geralmente são gravados nos arquivos com formato DFM (específico para plataforma Windows) ou com formato FMX (para diversas plataformas) dependendo do tipo de aplicação que está sendo criado. A `TComponent` foi esclarecida no início desta seção. A classe `TControl` é a base para todos componentes visuais, introduzindo o comportamento comum a todos eles. Componentes que herdam da classe `TGraphicControl` tem como propósito primário exibir textos ou gráficos. Por fim, a classe `TWinControl` permite processar entradas do teclado e mouse, receber o foco e conter outros componentes (LAPORTE, 2011).

Além da decisão de qual classe o componente herdará, é preciso se preocupar com quais propriedades serão publicadas para a interação do componente com o programador. As propriedades e os eventos publicados são visualizados na paleta da IDE, denominada *Object Inspector*. Ela é responsável por mostrar todas as informações de cada objeto selecionado no ambiente de desenvolvimento.

Pode-se observar no Quadro 2, um exemplo básico de estrutura de um componente. A classe `TEstereotipo` tem por objetivo não ser um componente visual por isso herda da classe `TComponent`, e possui apenas uma propriedade chamada Tipo que está declarada dentro do bloco `published`, ou seja, tudo que estiver neste bloco será visível no *Object Inspector*. Os demais blocos, `private`, `protected` e `public` respeitam o conceito de encapsulamento da programação orientada a objetados. Outra situação a ser notada é a procedure `Register`. Ela é responsável por registrar o componente na paleta de componentes da IDE, nesse caso o componente `TEstereotipo` será registrado no grupo `Prototipagem` da paleta (LAPORTE, 2011).

Quadro 2 – Estrutura de um componente

```
unit uEstereotipo;

interface

uses
  System.SysUtils, System.Classes;

type
  TEstereotipo = class(TComponent)
  private
    { Private declarations }
    FTipo: String;
  protected
    { Protected declarations }
  public
    { Public declarations }
  published
    { Published declarations }
    property Tipo: String read FTipo write FTipo;
  end;

procedure Register;

implementation

procedure Register;
begin
  RegisterComponents('Prototipagem', [TEstereotipo]);
end;

end.
```

2.6 TRABALHOS CORRELATOS

Foram selecionados quatro trabalhos correlatos. O primeiro é sobre a utilização da prototipagem, o segundo é sobre uma ferramenta de criação de interfaces, o terceiro é sobre uma integração de softwares com uso do XMI e o quarto é um software que através do código fonte lido gera o diagrama de classes. O item 2.6.1 descreve o artigo de prototipação de software e *design* participativo aplicado por uma empresa do Brasil do estado do Ceará chamada Atlântico (ROSEMBERG, 2008). O item 2.6.2 descreve a ferramenta Elicitar (BATTISTI, 2014), que é um protótipo de gerador de especificações com padrões de requisitos. O item 2.6.3 descreve a ferramenta QEA (SOUZA, 2011), a qual realiza a estruturação de um banco de dados com diagramas de classes. O item 2.6.4 descreve o software ESS-Model desenvolvido pela Eldean AB o qual recebe como entrada códigos fontes e transforma em diagramas de classe (CAETANO, 2009). Por fim, no item 2.6.5, é apresentada uma tabela comparativa em relação aos trabalhos correlatos.

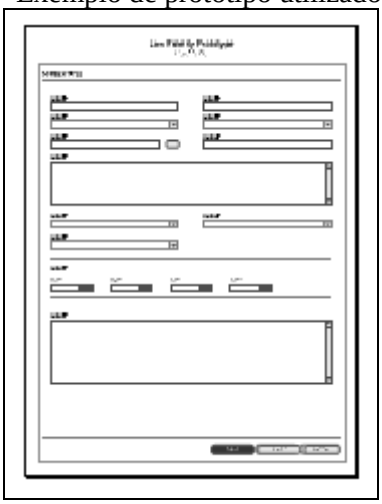
2.6.1 Atlântico

O Atlântico é uma empresa de pesquisa e desenvolvimento para soluções em TI para o mercado nacional e internacional com certificação nível 3 de maturidade no modelo

Capability Maturity Model Integration (CMMI) versão 1.1, localizada no Brasil em Fortaleza, no estado do Ceará (ROSEMBERG et al., 2008). Ela recebeu um desafio de uma empresa multinacional para fazer um produto, o qual poderia ocasionar muitas dificuldades por depender de informações da empresa matriz e filiais. Por isso elaboraram um processo de análise para ser assertivo na necessidade do cliente (ROSEMBERG et al., 2008).

Este processo envolveu uma equipe específica para a elicitação dos requisitos. No momento em que se desenhava a prototipagem com a ferramenta Microsoft Visio juntamente com um conjunto de componentes visuais web, já eram questionados os requisitos do sistema, sendo validados pelo cliente e pelo especialista pelo negócio. Segue um exemplo de protótipo conforme a Figura 13 (ROSEMBERG et al., 2008).

Figura 13 – Exemplo de protótipo utilizado no projeto



Fonte: Rosemberg et al. (2008).

Conforme Rosemberg et al. (2008, p. 315), “O estudo de caso realizado mostrou os benefícios obtidos com a prototipação em baixa fidelidade na elicitação de requisitos, gerando otimização de recursos e aumentando a satisfação do cliente”.

2.6.2 Elicitar

Battisti (2014) teve como proposta a criação de uma ferramenta (Elicitar) que tenha como entrada a informação de requisitos escritos em língua portuguesa e como saída código para o protótipo da tela correspondente. Para alcançar seu objetivo, Battisti (2014) utilizou as tecnologias e ferramentas como CoGrOO para analisador morfológico, a linguagem de programação Java, a biblioteca Primefaces para desenvolvimento da interface web do protótipo, Eclipselink para abstração do banco de dados e Velocity para geração do arquivo HTML a partir do arquivo de definições geradas pela ferramenta.

A Figura 14 ilustra a tela inicial da ferramenta. Nela estão dispostos os menus de acesso da ferramenta, os quais possuem as funções de cadastrar requisitos, gerar definição,

gerar HTML, cadastrar características, enviar feedback, abrir proposta, novidades da versão e abrir ajuda.

Figura 14 – Tela principal do Elicitar

Fonte: Battisti (2014).

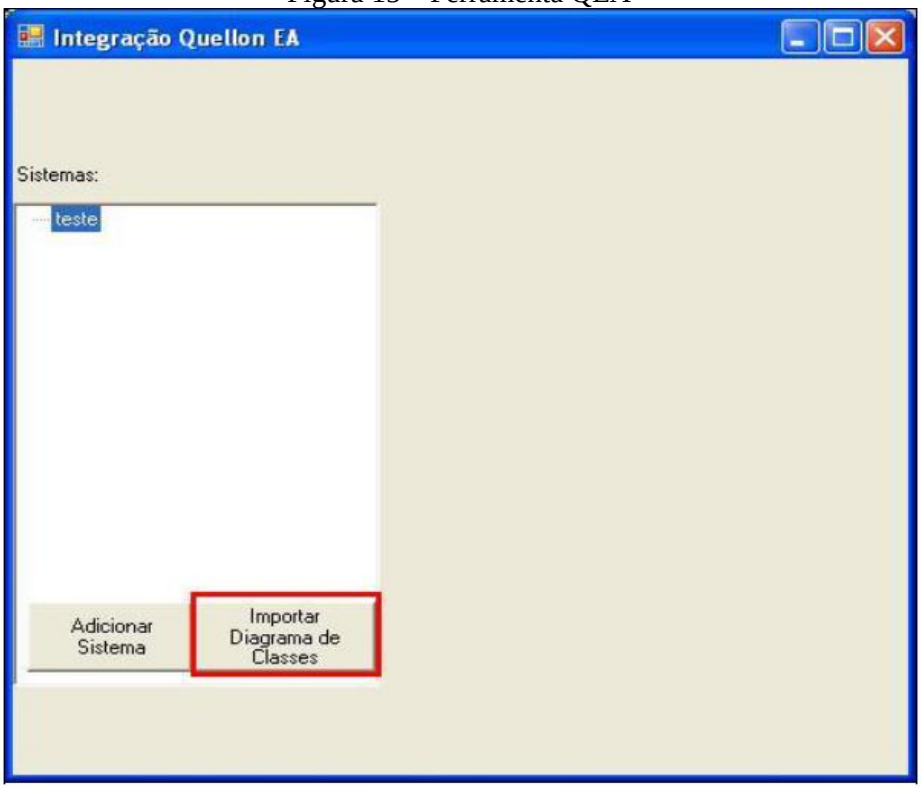
2.6.3 QEA

Souza (2011) teve como proposta criar uma ferramenta visando atender a necessidade de integrar a ferramenta Quellon com a Enterprise Architect (EA) chamada QEA. A Quellon é uma ferramenta utilizada para desenvolvimento e gestão de aplicações web na plataforma .NET fornecida pela empresa Quellon do Brasil Sistemas S.A.

A ferramenta QEA automatiza a passagem da estrutura lógica de um sistema exposto em um diagrama de classes gerado na plataforma do EA, para o banco de dados do sistema que está sendo desenvolvido na ferramenta Quellon. O processo de integração acontece através do uso de XML com o padrão XMI. As técnicas utilizadas para o desenvolvimento foram a linguagem C#, ASP e .NET, sobre a plataforma Microsoft .Net com versão 4.0 de *framework* (SOUZA, 2011).

A Figura 15 representa a execução da ferramenta QEA na etapa de importar os diagramas de classes para estruturar a base de dados do sistema selecionado.

Figura 15 – Ferramenta QEA



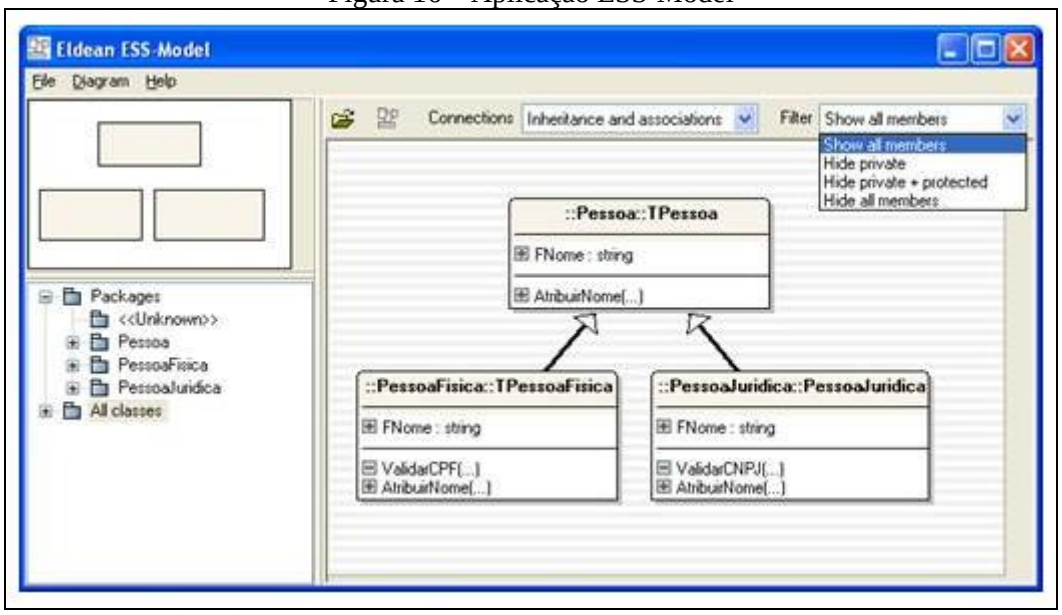
Fonte: Souza (2011).

2.6.4 ESS-Model

O objetivo principal da ferramenta ESS-Model é obter diagramas de classes da UML por meio de engenharia reversa de arquivos fonte aceitando dois tipos de linguagens de programação, sendo Object Pascal e Java. Ele foi desenvolvido na linguagem Object Pascal por uma empresa da Suécia chamada Eldean AB. Pode ser integrado na IDE Delphi e o seu projeto é *Open Source*. O ESS-Model não possui muita documentação e sua última *release* foi à versão 2.2, realizada em janeiro de 2003 (CAETANO, 2009).

O aplicativo não precisa ser instalado. Basta executá-lo e informar onde estão os arquivos fontes. O ESS-Model faz a leitura do código fonte e gera a documentação mostrando o diagrama em seu aplicativo sem ter recursos para manipulação, conforme é visualizado na Figura 16. O diagrama pode ser exportado em formato XMI e também pode ser visualizado em relatórios exportados em formato HTML (CAETANO, 2009).

Figura 16 – Aplicação ESS-Model



Fonte: Caetano (2009).

2.6.5 Comparação entre os trabalhos correlatos

O Quadro 3 apresenta de forma comparativa algumas características em relação aos trabalhos correlatos apresentados neste trabalho.

Quadro 3 – Características dos trabalhos

Características	Atlântico	Elicitar	QEA	ESS-Model
utiliza a geração de prototipagem de telas	Sim	Sim	Não	Não
utiliza o padrão XMI para exportação/leitura de diagramas da UML	Não	Não	Sim	Sim
ferramenta é dependente de aplicações comerciais	Sim	Não	Sim	Não
busca otimizar o processo de desenvolvimento de softwares	Sim	Sim	Sim	Sim

Cada trabalho tem sua finalidade de atuação, por isso apresentam várias diferenças. Mas, como se pode perceber através do Quadro 3, os trabalhos correlatos buscam otimizar alguma parte do processo de desenvolvimento de software. O Elicitar e o ESS-Model não dependem de aplicações comerciais, pois o Elicitar utiliza bibliotecas de uso gratuito e o ESS-Model já é um aplicativo pronto de uso gratuito. Já o Atlântico depende do Microsoft Visio para a realização das prototipagens e o QEA usa o Enterprise Architect para geração dos arquivos XMI. O QEA e o ESS-Model são os únicos que utilizam o padrão XMI para exportação/leitura de diagramas da UML. Nesse caso o ESS-Model utiliza como forma de exportação dos diagramas de classes e o QEA faz a leitura dos diagramas de classe para importar sua estrutura em sua aplicação.

3 DESENVOLVIMENTO

Neste capítulo são apresentadas as etapas do desenvolvimento da ferramenta. Na seção 3.1 são enumerados os principais requisitos do projeto desenvolvido. A seção 3.2 apresenta a especificação da ferramenta com diagramas de casos de uso, classes e de sequência. A seção 3.3 detalha a implementação da ferramenta, destacando os principais pontos. Por fim, a seção 3.4 apresenta os experimentos realizados e resultados obtidos.

3.1 REQUISITOS

Os requisitos do sistema desenvolvido são:

a) o sistema deverá permitir o uso de componentes gráficos visuais pré-definidos para auxiliar na criação da prototipagem e extração das informações para geração da documentação (Requisito Funcional – RF). Os componentes gráficos são:
   - área de texto,
   - botão,
   - botão de seleção,
   - caixa de combinação,
   - caixa de seleção,
   - caixa de texto,
   - grade,
   - imagem,
   - rótulo;

b) o sistema deverá permitir o uso de componentes gráficos não visuais pré-definidos para auxiliar na criação da prototipagem e extração das informações para geração da documentação. Sendo os componentes gráficos para definição do estereótipo da classe e do ator para o caso de uso (Requisito Funcional – RF);

c) o sistema deverá permitir a exportação das informações da prototipagem das telas em padrão XMI com configurações que resultem na diagramação de classes para importar em ferramentas de modelagem de software (RF);

d) o sistema deverá permitir a exportação das informações da prototipagem das telas em padrão XMI com configurações que resultem na diagramação de casos de uso para importar em ferramentas de modelagem de software (RF);

e) o sistema deverá ser integrado com a ferramenta RAD Studio Delphi 10 Seattle em forma de *plugin* (Requisito Não Funcional – RNF);

f) o sistema deverá utilizar a linguagem de programação Object Pascal (RNF).

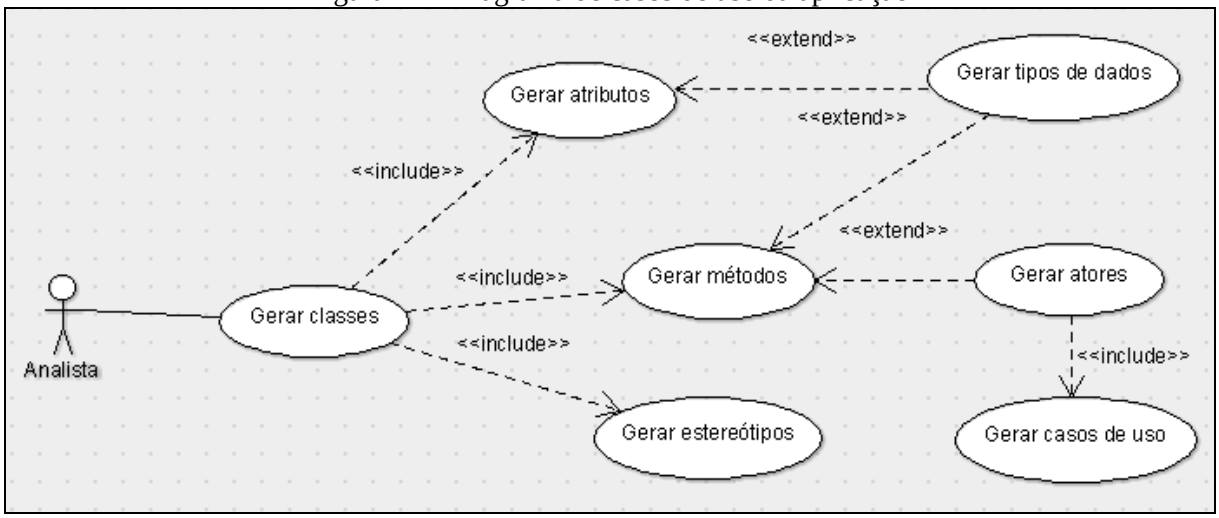
3.2 ESPECIFICAÇÃO

A especificação deste trabalho foi desenvolvida através do diagrama de casos de uso, diagrama de classes e diagrama de sequência, todos da UML, realizados nas ferramentas de modelagem ArgoUML e Astah.

3.2.1 Diagramas de casos de uso

Na Figura 17 é mostrado o diagrama de casos de uso da aplicação. Nela estão os elementos da UML que o usuário pode gerar através do uso dos componentes gráficos utilizados para a prototipagem das telas. A especificação do diagrama modela os requisitos funcionais da alínea “c” e “d”.

Figura 17 – Diagrama de casos de uso da aplicação

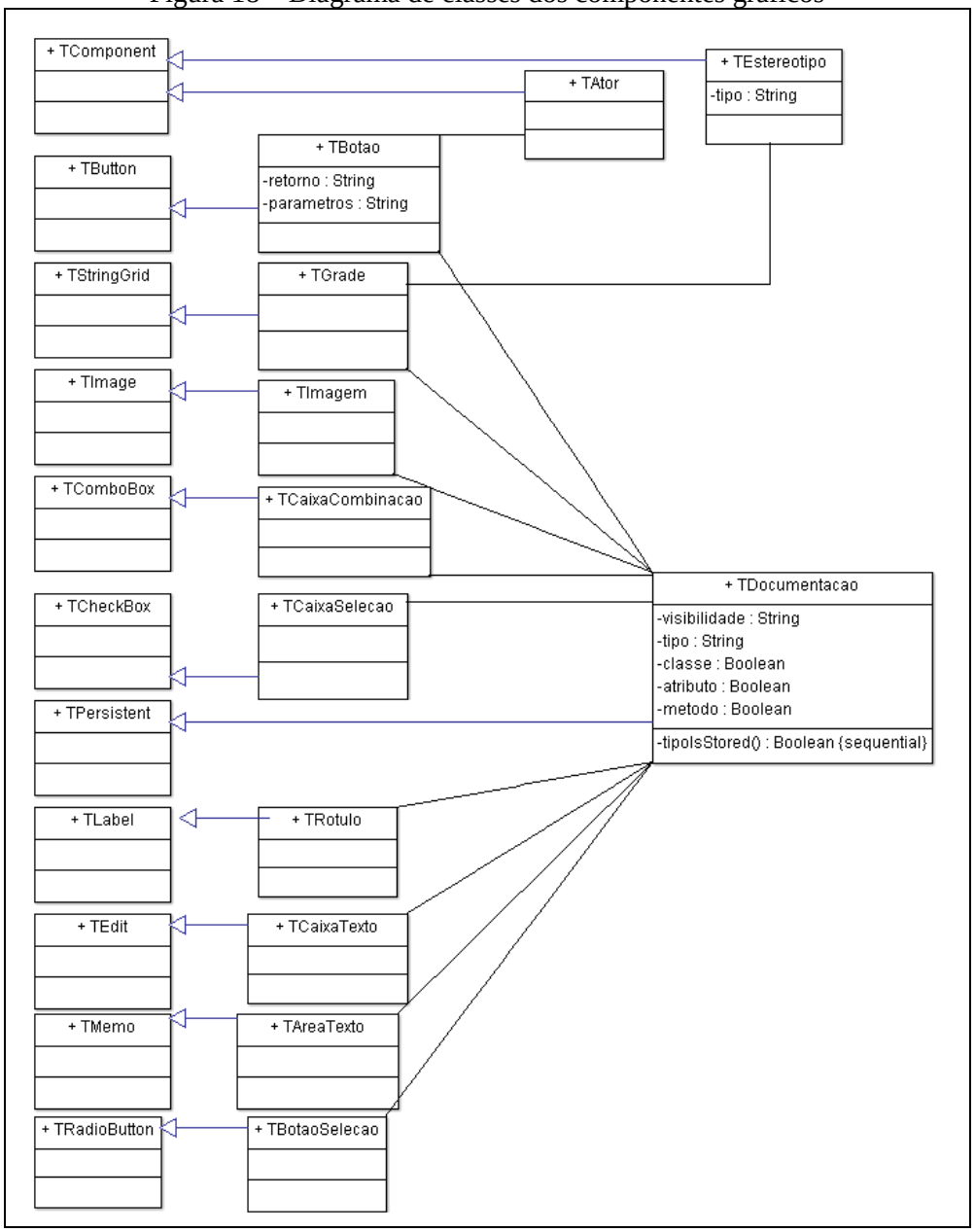


O caso de uso principal é o gerar classes, pois através dele é possível executar os demais casos de uso. O analista pode criar novos tipos de dados através dos casos de uso gerar atributos e gerar métodos. Além disso, é possível criar os elementos do diagrama de casos de uso através dos casos de uso `gerar atores` e `gerar casos de uso`.

3.2.2 Diagrama de classes

A Figura 18 apresenta a modelagem do diagrama de classes dos componentes gráficos disponibilizados para a realização da prototipagem.

Figura 18 – Diagrama de classes dos componentes gráficos



Foram criados onze componentes gráficos para a prototipação das telas conforme os requisitos funcionais definidos. Dois desses componentes gráficos são não visuais e herdam diretamente da classe `TComponent`, sendo esta uma classe definida pelo Delphi. Os demais são visuais e foram herdados diretamente dos componentes gráficos padrões disponibilizados pela IDE Delphi, para que sejam mantidas as propriedades bases dos componentes. Dessa forma, esses componentes herdam de uma classe que estabelece toda a aparência do componente gráfico já conhecida por um usuário da IDE.

No Quadro 4 podem-se observar os componentes gráficos visuais criados neste trabalho, com seu respectivo nome, significado e sua respectiva classe herdada.

Quadro 4 – Descrição dos componentes gráficos

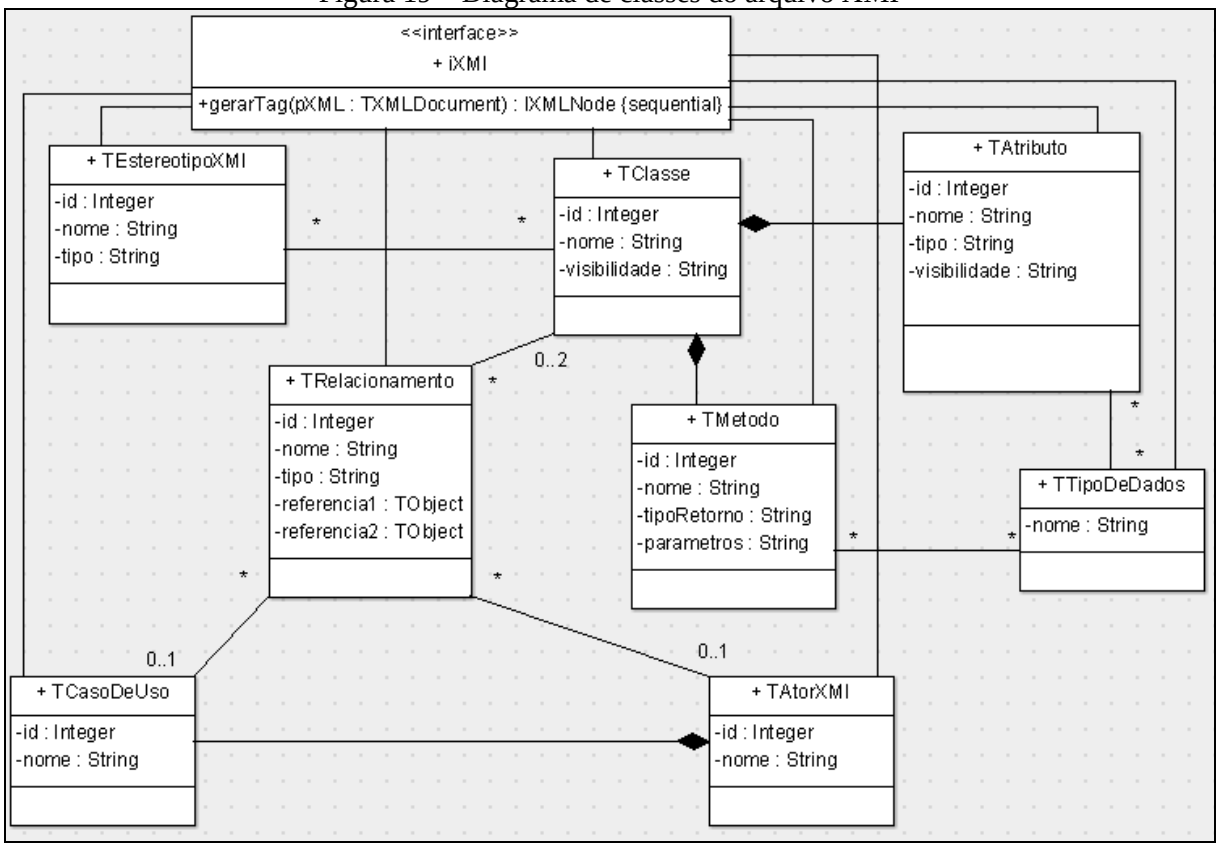
Nome	Significado	Herança
TAreaTexto	Permite a entrada de textos com várias linhas.	TMemo
TBotao	Permite a execução de uma ação.	TButton
TBotaoSelecao	Representa uma opção a ser marcada. Quando utilizado em conjunto com outros botões de seleção só pode ser marcado um deles.	TRadioButton
TCaixaCombinacao	Representa uma opção que pode ser marcada ou não.	TCheckBox
TCaixaSelecao	Permite a seleção de uma única opção dentre uma lista de opções.	TComboBox
TCaixaTexto	Permite a entrada de um texto em uma linha	TEdit
TGrade	É uma tabela formada por linhas e colunas, permitindo a entrada de vários dados.	TStringGrid
TImagem	Representa uma imagem.	TImage
TRotulo	Descrição de alguma informação. Componente visual para o usuário.	TLabel

Já na parte não visual, foram desenvolvidos os componentes gráficos `TAtor` e `TEstereotipo`. O `TAtor` representa o ator responsável pela execução de cada função disponibilizada no formulário. Portanto, ao utilizar o componente para a realização do protótipo ele sempre será atrelado a um ou mais botões do formulário. Por sua vez, os botões podem ter nenhum ou vários atores referenciados. O `TEstereotipo` representa os estereótipos que a classe poderá ter. Ao utilizá-lo na prototipação, ele estará atrelado a um formulário ou a uma grade, sendo que os estereótipos que não estiverem referenciados a uma grade pertencerão a um formulário.

A maioria dos componentes gráficos criados possui uma propriedade chamada `Documentacao`. Esta propriedade é herdada da classe `TDocumentacao`. Ela é responsável por registrar os dados necessários para serem visualizados em um diagrama de classes e/ou casos de uso quando utilizado a opção de exportar a prototipagem das telas em um arquivo XMI.

Na implementação da geração do XMI conforme o diagrama de classes da Figura 19 foi criada a interface `iXMI` e as classes `TAtor`, `TCasoDeUso`, `TMetodo`, `TTipoDeDados`, `TRelacionamento`, `TAtributo`, `TClasse` e `TEstereotipoXMI`. Todas implementam a interface `iXML` que possui a função `gerarTAG` que é responsável por gerar as *tags* XML para uso no arquivo XMI. Cada uma dessas classes representa um elemento para o diagrama de classes ou diagrama de casos de uso.

Figura 19 – Diagrama de classes do arquivo XMI

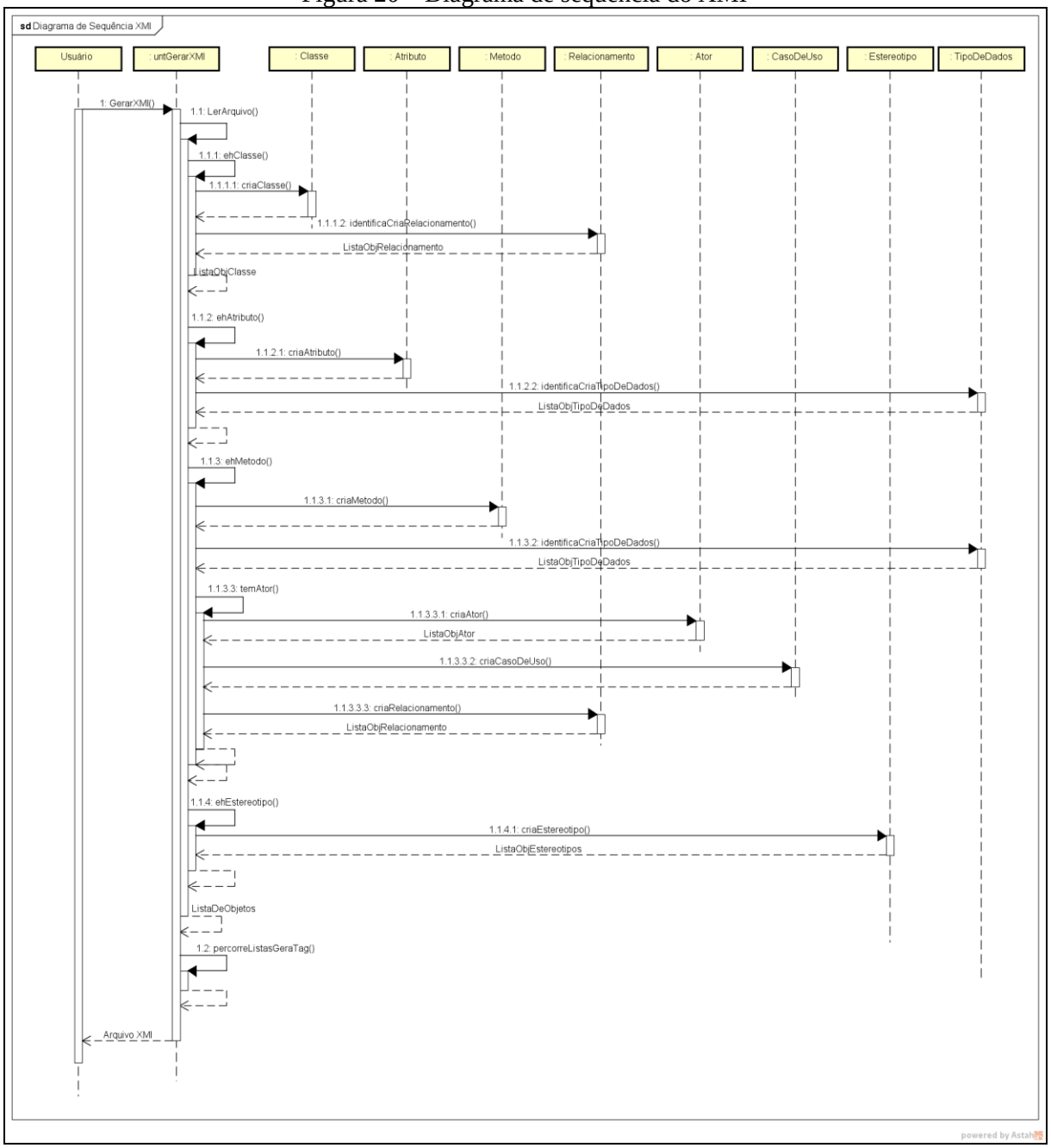


Sendo assim, a classe `TClasse` é o elemento classe, a classe `TAtributo` é o elemento atributo e a classe `TMetodo` é o elemento método do diagrama de classes da UML. A classe `TEstereotipoXMI` serve para representar qual estereótipo a classe possui no diagrama de classes. A classe `TRelacionamento` cria o relacionamento de agregação entre as classes criadas no formulário, sendo representadas pelos componentes gráficos `TForm` e `TGrade`. A classe `TRelacionamento` também é responsável por relacionar o caso de uso com o ator, quando for gerado o diagrama de casos de uso. A classe `TTipoDeDados` é utilizada para identificar todos os tipos de dados utilizados nos atributos, tipos de retornos e parâmetros dos métodos. Por fim, as classes `TAtorXMI` e `TCasoDeUso` representam os elementos ator e caso de uso no diagrama de casos de uso.

3.2.3 Diagrama de sequência

Na Figura 20 pode ser observado o diagrama de sequência correspondente à geração do arquivo XMI. O usuário inicia a geração do arquivo XMI, no qual o sistema irá ler os componentes posicionados no protótipo e identificar cada objeto que deverá ser criado, podendo ser objetos do tipo `Classe`, `Atributo`, `Metodo`, `Ator`, `CasoDeUso`, `Relacionamento`, `TipoDeDados` e `Estereotipo`.

Figura 20 – Diagrama de sequência do XMI



O diagrama de sequências apresentado na Figura 20 apresenta todos os possíveis componentes a serem gerados, com suas respectivas *lifelines* e os métodos de ativação de cada uma delas. É importante destacar que os métodos de criação relacionados aos componentes dos diagramas da UML só serão invocados caso haja os componentes gráficos correspondentes no protótipo realizado pelo usuário. Devido a uma baixa visualização da Figura 20, a mesma pode ser consultada no Apêndice C dividida em três partes em modo paisagem.

3.3 IMPLEMENTAÇÃO

A seguir são apresentadas as técnicas e ferramentas utilizadas para o desenvolvimento deste trabalho, bem como a operacionalidade da implementação.

3.3.1 Técnicas e ferramentas utilizadas

A ferramenta foi desenvolvida utilizando a linguagem de programação Object Pascal, no ambiente de desenvolvimento Delphi 10 Seattle. O Delphi 10 Seattle é uma IDE para o Object Pascal que possui uma ampla biblioteca nativa de componentes gráficos multiplataforma disponibilizada pela empresa Embarcadero.

Optou-se por utilizar a ferramenta Delphi, pois ela permite que a prototipação de telas seja realizada de forma simples, tendo um tempo de aprendizado reduzido. Por sua vez, optou-se por utilizar esta versão da IDE devido a licença acadêmica disponibilizada pela FURB e por poder realizar a prototipagem das telas para qualquer dispositivo suportado pela IDE, incluindo dispositivos móveis. Para a implementação da funcionalidade de exportação dos dados no padrão XMI foi utilizada a versão 1.2. Para a construção dos diagramas da UML foi utilizada a versão 1.4. Para a realização dos testes de importação e validação do XMI foram utilizados os softwares de modelagem de dados ArgoUML versão 0.34 e Enterprise Architect versão 12.

3.3.2 Desenvolvimento da ferramenta

O desenvolvimento da ferramenta foi dividido em duas etapas: na primeira foram implementados os componentes gráficos para a prototipação das telas e na segunda, a exportação para o arquivo XMI. A seguir são apresentados os trechos de código mais significativos dessas etapas.

3.3.2.1 Criação dos componentes

Conforme o Quadro 5, a montagem da classe `TDocumentacao` constitui de três propriedades e uma função. A propriedade `fVisibilidade` é uma enumeração e permite os valores `Público`, `Privado`, `Protegido` e `Pacote`. A propriedade `fTipo` espera o tipo que poderá ser digitado pelo usuário, ou então, pode ser escolhido um dos valores pré-definidos sendo `inteiro`, `real`, `lógico`, `texto`, `data` e `hora` que aparecem no momento de configurar o componente gráfico na prototipagem. A propriedade `fClasse` indica se o componente gráfico em questão é uma classe e a propriedade `fAtributo` indica se o componente gráfico é um atributo no diagrama de classe. A função `TipoIsStored` é responsável por informar o valor inicial da propriedade `fTipo`.

Quadro 5 – Classe `TDocumentacao`

```
TDocumentacao = class(TPersistent)
private
  { Protected declarations }
  fVisibilidade: TVisibilidade;
  fTipo: String;
  fClasse: Boolean;
  fAtributo: Boolean;
  function TipoIsStored:Boolean;
protected
  { Protected declarations }
public
  { Public declarations }
  constructor create(AOwner: TPersistent);
  property Tipo: String read fTipo write fTipo stored TipoIsStored;
  property Classe: Boolean read fClasse write fClasse;
  property Atributo: Boolean read fAtributo write fAtributo;
published
  { Published declarations }
  property Visibilidade: TVisibilidade read fVisibilidade write fVisibilidade;

end;
```

A classe `TDocumentacao`, como mencionado anteriormente, é fundamental para vários componentes gráfico, pois possui as propriedades importantes para eles. Um exemplo da sua aplicação pode ser visualizado no Quadro 6, na qual a classe `TDocOwner` herda a classe `TDocumentacao` e a classe `TCaixaTexto` tem uma propriedade do tipo `TDocOwner`. A classe `TDocOwner` sobrescreve a `TDocumentacao` para identificar quais atributos são visíveis para o usuário configurar o componente. No Quadro 6 nota-se o uso da classe `TDocOwner`, deixando visível as propriedades `Tipo` e `Atributo`.

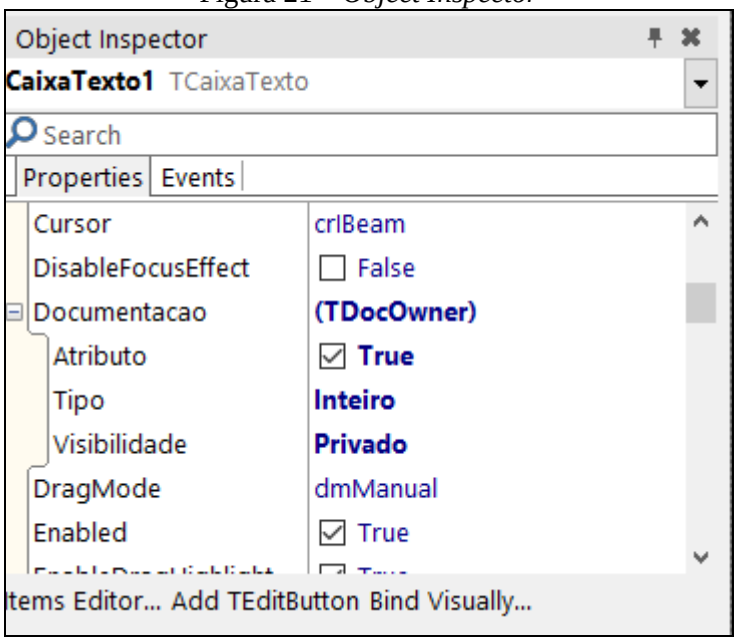
Quadro 6 – Uso da classe `TDocOwner`

```
TDocOwner = class(TDocumentacao)
published
  property Tipo;
  property Atributo;
end;

TCaixaTexto = class(TEdit)
private
  fDoc : TDocOwner;
protected
  { Protected declarations }
public
  { Public declarations }
  constructor Create (AOwner: TComponent); override;
  destructor Destroy; override;

published
  { Published declarations }
  property Documentacao : TDocOwner read fDoc write fDoc ;
end;
```

Dessa forma, a propriedade `Documentacao` presente na classe `TCaixaTexto` agrupa todas as informações necessárias para a aplicação utilizar na geração dos diagramas,

facilitando também a localização das propriedades para o usuário manipulá-las. A visualização destas informações fica agrupada na propriedade `Documentacao`, disponível no *Object Inspector*, conforme apresentado na Figura 21. Essa propriedade `Documentacao` é exibida para todos os componentes gráficos gerados pelo trabalho.

Figura 21 – *Object Inspector*



As propriedades relacionadas à documentação podem ser alteradas pelo usuário de forma que sejam definidas as características necessárias para a geração dos diagramas. Quando o usuário informa os dados nas propriedades da `Documentacao`, essa informação é gravada no próprio arquivo que o Delphi mantém, correspondente a parte visual da aplicação. Essa gravação é possível devido a classe `TDocumentacao` herdar a classe `TPersistent`, a qual controla a persistência das informações no arquivo que o Delphi mantém para gravar os componentes utilizados. Conforme o Quadro 7, pode-se observar que o objeto `CaixaTexto1` ficou gravado no arquivo com as informações de `tipo`, `visibilidade` e `atributo`, de acordo com o que foi selecionado em suas propriedades.

Quadro 7 – Objeto gravado no FMX

```
object CaixaTexto1: TCaixaTexto
  Touch.InteractiveGestures = [LongTap, DoubleTap]
  TabOrder = 13
  Position.X = 24.000000000000000000
  Position.Y = 40.000000000000000000
  Documentacao.Visibilidade = Privado
  Documentacao.Tipo = 'Inteiro'
  Documentacao.Atributo = True
end
```

A maioria dos componentes gráficos visuais desenvolvidos pelo trabalho segue este mesmo padrão. Exceto os componentes `TGrade`, `TBotao` que possuem algumas propriedades

diferentes e os componentes gráficos não visuais `TAtor` e `TEstereotipo` que não possuem a herança para a classe `TDocumentacao`.

O componente `TGrade` também cria a classe `TDocOwner` herdada da classe `TDocumentacao`, porém publica apenas a propriedade Classe. Além disso, ele cria uma nova propriedade chamada `DocumentacaoEstereotipo`, a qual não fica dentro da propriedade `Documentacao` por motivo de limitação da IDE do Delphi não habilitar uma lista de objetos dentro de uma propriedade. A `DocumentacaoEstereotipo` é uma lista de itens de estereótipos que podem ser usados pelo usuário ao definir os estereótipos das classes.

O componente `TBotao` também cria a classe `TDocOwner` herdada da classe `TDocumentacao`, porém não publica nenhuma propriedade a mais, apenas cria novas como a `Parametros`, `Retorno` e `DocumentacaoAtor`, a qual não fica dentro da propriedade pelo mesmo motivo da propriedade `DocumentacaoEstereotipo` do componente `TGrade`. A `DocumentacaoAtor` é uma lista de itens de atores que podem ser usados pelo usuário ao definir os atores dos casos de uso.

3.3.2.2 Geração do XMI

Para a geração do arquivo XMI foi criado uma unidade de código principal, sendo a `uGerarXMI`. Essa unidade de código é responsável por ler os arquivos FMX e controlar todos os tipos de objetos que serão criados. Nela foi criada a função `gerarXMI`, a qual recebe por parâmetro o local de onde deverão ser buscados os arquivos FMX e onde deverá ser gravado o arquivo XMI.

Todas as classes que utilizam a propriedade nome recebem o valor que estiver na propriedade `name` dos componentes gráficos utilizados. A identificação de qual objeto será criado funciona da seguinte forma. A primeira linha de todo arquivo FMX é um objeto de classe, portanto todo formulário criado representa uma classe. A visibilidade por padrão é privada. Caso haja um objeto do tipo `TGrade`, poderá ser criado mais de uma classe no mesmo arquivo FMX, desde que a propriedade classe da documentação tiver o valor verdadeiro. Quando uma `TGrade` é definida como classe, as suas colunas são consideradas atributos da classe com visibilidade privada e com tipo de dados `String`.

Para identificar se o componente gráfico é um atributo, é verificado se o arquivo FMX possui algum objeto do tipo `TAreaTexto`, `TBotaoSelecao`, `TCaixaCombinacao`, `TCaixaSelecao`, `TCaixaTexto`, `TImagem`, `TRotulo` e se os mesmos contém a propriedade `Atributo` da `Documentacao` com o valor verdadeiro. Para identificar se existirá algum

método, é verificado se o arquivo FMX possui um objeto do tipo `TBotao`. Caso exista, o botão será considerado como um método. Ainda relacionado à classe `TBotao`, se ele contiver a informação de `Ator` é criado o objeto `TAtorXMI` e uma classe `TCasoDeUso` com a funcionalidade informada no nome do botão. Para cada atributo e método identificado é verificado o tipo de dados utilizado neles e é criado a um objeto do tipo `TTipoDeDados` para armazenar estes valores.

Sempre que é identificada a criação de um ou mais objetos do tipo `TClasse` no mesmo formulário é criado um relacionamento de agregação utilizando a classe `TRelacionamento`. Ao realizar o relacionamento é passado no atributo `referencia1` do objeto `Relacionamento` a classe do formulário e na `referencia2` a classe da grade criada. Assim como, ao identificar a criação de um objeto `TAtorXMI` e um objeto `TCasoDeUso` é criado o objeto `Relacionamento` com os atributos `referencia1` e `referencia2` recebendo os objetos criados. Para identificar os estereótipos das classes é verificado no arquivo se possui o objeto `TEstereotipoXMI` e se eles estão preenchidos na propriedade `DocumentacaoEstereotipo` dos objetos `TGrade`, caso sim, o estereótipo pertence a essa classe, caso não, ele pertence ao formulário criando assim o objeto do tipo `TEstereotipoXMI` que armazena estes valores.

Após a leitura do arquivo FMX é realizada a identificação dos objetos utilizados para a construção dos diagramas. Através da identificação desses objetos, são criados os novos objetos que representarão cada elemento nos diagramas. Todos eles são adicionados em listas genéricas que recebem estes tipos de objetos. Elas são percorridas e, então, é gerada a respectiva *tag* do objeto, respeitando uma ordem de composição no XMI. Essas listas são separas em tipos, sendo `TClasse`, `TTipoDeDados`, `TAtorXMI`, `TCasoDeUso` e `TRelacionamentoXMI`. A interação com essas listas para a geração das *tags* pode ser visualizada no Quadro 8.

Quadro 8 – Interação de listas

```
for Obj in ListaObjDados.Values do
begin
  nameespace.ChildNodes.Add(TTiposDeDados(Obj).gerarTag(XMLDocument));
end;

for I := 0 to ListaObj.Count - 1 do
begin
  Obj := ListaObj.Items[i];
  nameespace.ChildNodes.Add(TClasse(Obj).gerarTag(XMLDocument));
end;

for Obj in ListaObjAtor.Values do
begin
  nameespace.ChildNodes.Add(TAtorXMI(Obj).gerarTag(XMLDocument));
end;

for Obj in ListaObjCaso.Values do
begin
  nameespace.ChildNodes.Add(TCasoDeUso(Obj).gerarTag(XMLDocument));
end;

for Obj in ListaObjRelacionamento.Values do
begin
  nameespace.ChildNodes.Add(TRelacionamento(Obj).gerarTag(XMLDocument));
end;
```

Através do quadro anterior (Quadro 8) pode-se observar que cada classe está invocando o método `gerarTag`, este método é implementado na classe base das listas. A lista `ListaObjAtor` possui todos os atores, a lista `ListaObjCaso` possui todos os casos de uso, a lista `ListaObjRelacionamento` possui todos os relacionamentos. A lista `ListaObj` que contém todas as classes criadas. Em específico, o método `gerarTag` invocado pela classe base da `ListaObj`, percorre todos os atributos, métodos e estereótipos definidos para os componentes gráficos. Portanto, além de gerar a própria *tag* relacionada ao componente gráfico, gera também para seus descendentes, conforme pode ser visualizado no Quadro 9.

Quadro 9 – Gerar *tags* do XMI

```
nodeAtributo := pXML.CreateNode('name', ntAttribute);
nodeAtributo.Text := Self.Nome;
nodeClasse.AttributeNodes.Add(nodeAtributo);

nodeAtributo := pXML.CreateNode('xmi.id', ntAttribute);
nodeAtributo.Text := IntToStr(Self.Id);
nodeClasse.AttributeNodes.Add(nodeAtributo);

for ObjEstereotipo in Self.ListaEstereotipos do
begin
  nodeClasse.ChildNodes.Add(ObjEstereotipo.gerarTag(pXML));
end;

nodeClassificador := pXML.CreateNode('UML:Classifier.feature', ntElement);
nodeClasse.ChildNodes.Add(nodeClassificador);

for ObjAtributo in Self.ListaAtributos do
begin
  nodeClassificador.ChildNodes.Add(ObjAtributo.gerarTag(pXML));
end;

for ObjMetodo in Self.ListaMetodos do
begin
  nodeClassificador.ChildNodes.Add(ObjMetodo.gerarTag(pXML));
end;

Result := nodeClasse;
```

A estrutura do XMI gerada pelo método gerarTag pode ser visualizada nos Quadros 10 e 11. Basicamente é declarado um bloco de cabeçalho (header) e um bloco de conteúdo (content). No cabeçalho são definidas as informações de versão, meta modelo e o exportador dos dados, conforme visto no Quadro 10. No conteúdo são definidos os modelos que serão exportados, contendo todos os elementos dos diagramas, conforme visto no

Quadro 11.

Quadro 10 – Cabeçalho do arquivo XMI gerado

```
<XMI.header>
    <XMI.documentation>
        <XMI.exporter>TCC Reinoldo</XMI.exporter>
    </XMI.documentation>
    <XMI.metamodel xmi.version="1.4" xmi.name="UML"/>
</XMI.header>
```

Quadro 11 – Conteúdo do arquivo XMI gerado

```
<XMI.content>
    <UML:Model isAbstract="false" isLeaf="false" isRoot="false" isSpecification="false" name="Modelo" xmi.id="
    1">
        <UML:Namespace.ownedElement>
            <UML:DataType isAbstract="false" isLeaf="false" isRoot="false" isSpecification="false" name="
            Integer" xmi.id="Integer"/>
            <UML:Class isActive="false" isAbstract="false" isLeaf="false" isRoot="false" isSpecification="
            false" visibility="private" name="Pedido" xmi.id="4">
```

Para auxiliar na geração do XMI foi utilizada a biblioteca `TXMLDocument`, que permite a criação de um nó para cada *tag* e, por sua vez, permite a definição dos atributos e seus respectivos valores para cada um desses nós. Desta forma, não é necessário criar um arquivo texto e inserir as *tag* manualmente. A utilização da biblioteca `TXMLDocument` pode ser observada no Quadro 12, por meio da criação do cabeçalho do XMI.

Quadro 12 – Gerar cabeçalho do arquivo XMI

```
// Criando outro nodulo na RAIZ (um node filho de RAIZ)
cabecalho := XMLDocument.CreateNode('XMI.header', ntElement);
Raiz.ChildNodes.Add(cabecalho);

documentacao := XMLDocument.CreateNode('XMI.documentation', ntElement);
cabecalho.ChildNodes.Add(documentacao);

nodoElemento := XMLDocument.CreateNode('XMI.exporter', ntElement);
nodoElemento.Text := 'TCC Reinoldo';
documentacao.ChildNodes.Add(nodoElemento);

nodoElemento := XMLDocument.CreateNode('XMI.metamodel', ntElement);
cabecalho.ChildNodes.Add(nodoElemento);

nodoAtributo := XMLDocument.CreateNode('xmi.version', ntAttribute);
nodoAtributo.Text := '1.4';
nodoElemento.AttributeNodes.Add(nodoAtributo);

nodoAtributo := XMLDocument.CreateNode('xmi.name', ntAttribute);
nodoAtributo.Text := 'UML';
nodoElemento.AttributeNodes.Add(nodoAtributo);
```

3.3.3 Operacionalidade da implementação

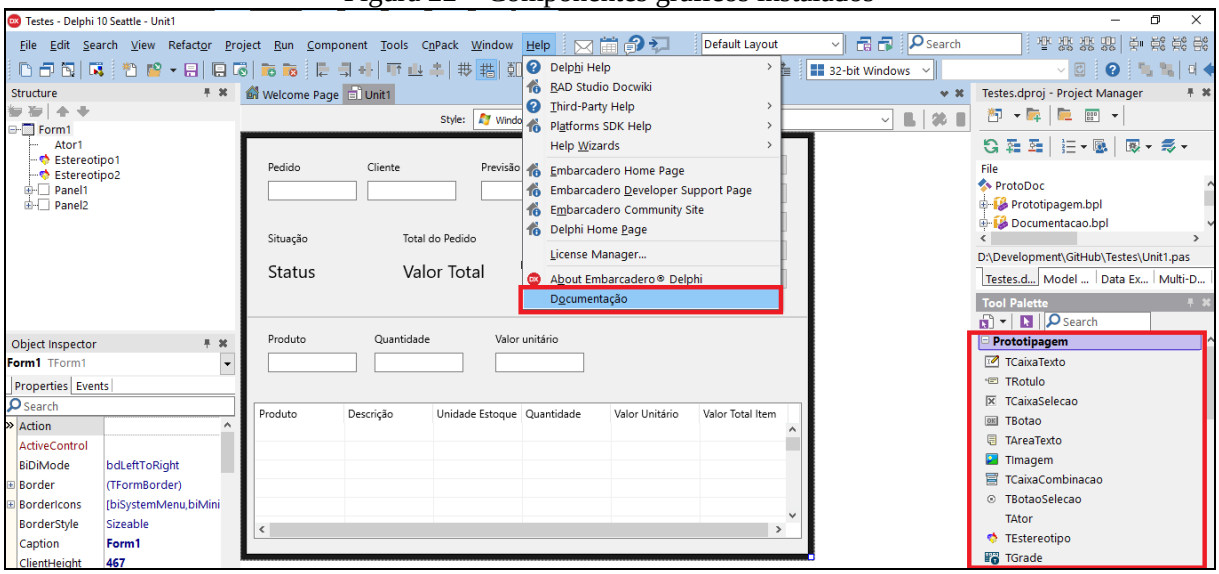
Para usar os componentes gráficos criados é necessário seguir três passos: primeiro deve-se realizar a instalação dos componentes desenvolvidos, depois deve-se realizar a prototipagem com um olhar de diagramação e por fim, gerar o arquivo XMI para manipular em uma ferramenta de modelagem de software. Como pré-requisito para a utilização dos componentes criados, é necessário ter o Delphi 10 Seatle instalado no computador. Já para a manipulação do arquivo XMI gerado deve-se ter instalado uma ferramenta de modelagem de software, como o ArgoUML, utilizado para validar este trabalho. Cada uma dessas etapas é detalhada nas subseções seguintes.

3.3.3.1 Instalação

Os componentes gráficos desenvolvidos neste trabalho estão disponibilizados no link https://drive.google.com/open?id=0B7ZvS1wjEhJSdENVRTRsQWRJNlE. Para instalar os componentes, basta baixar o pacote e salvá-los em um diretório do computador. Após isso, deverá ser aberto a IDE Delphi 10 Seatle, acessar o meu *File*, ir na opção *Open*. Feito isso,

deve-se localizar o arquivo `ProtDoc.dproj` que está na pasta baixada e então abri-lo. Dentro deste arquivo estarão agrupados dois projetos: o de prototipagem e o de documentação. O usuário deverá selecionar a opção *Build* e *Install* para os dois projetos. A instalação da prototipagem resultará nos componentes gráficos disponibilizados pelo trabalho na paleta de ferramentas (*Tool Palette*) da IDE Delphi conforme Figura 22. A instalação da documentação resultará no acesso da tela de geração do arquivo XMI que ficará situada no menu *Help* da IDE Delphi conforme a Figura 22.

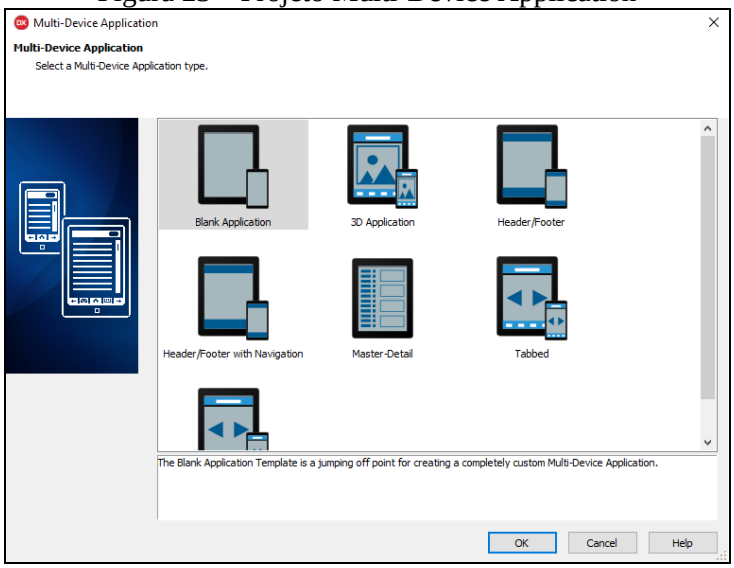
Figura 22 – Componentes gráficos instalados



3.3.3.2 Prototipagem

Para iniciar a prototipagem, deve-se criar um novo projeto *Multi-Device Application* conforme a Figura 23.

Figura 23 – Projeto Multi-Device Application



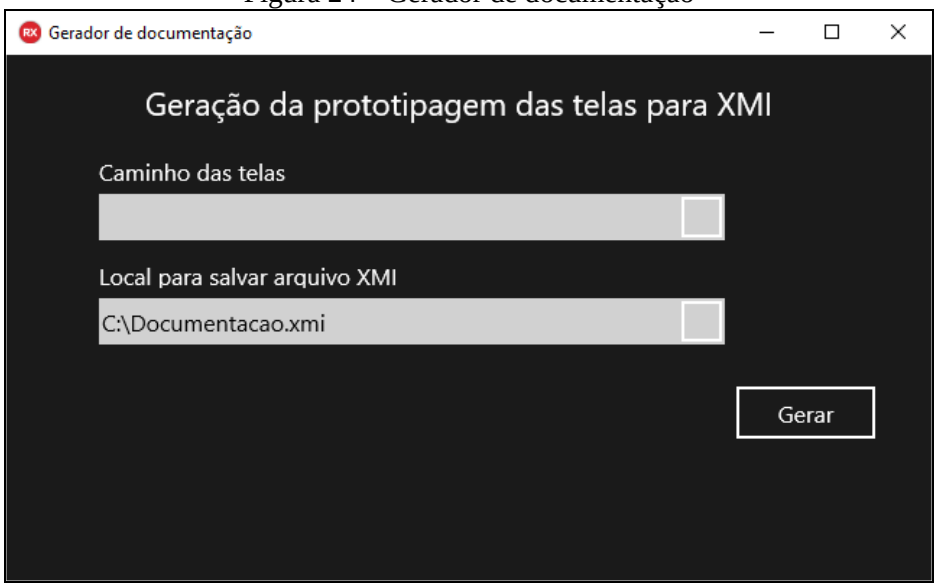
Na janela apresentada na Figura 23, deve-se escolher um tipo de *template* para a aplicação. Após isso, o usuário/analista poderá prototipar a tela com os componentes gráficos situados na paleta de ferramentas no grupo de Prototipagem. É necessário preencher no mínimo a propriedade name de cada componente com um nome referente ao elemento que será exibido nos diagramas de classe e casos de uso. Para complementar a diagramação podem ser utilizadas as propriedades disponíveis nos componentes pertecentes ao grupo Documentacao.

Caso o usuário queira ir além, é possível realizar a programação do protótipo criado. Como exemplo tem-se a exportação do XMI, na qual a tela foi feita com os componentes gráficos desenvolvidos neste trabalho, sendo adicionadas funcionalidades para realizar a exportação do arquivo.

3.3.3.3 Exportando XMI

Para realizar a exportação dos dados atribuídos nas telas prototipadas é necessário acessar o menu Help da IDE e localizar a opção Documentação, a qual abrirá a tela que está na Figura 24. Nessa tela é necessário informar o caminho de onde estão as telas prototipadas (`.fmx`) e o local de onde será gravado o arquivo gerado, junto com o seu nome e sua extensão (`.xmi`). Após definir os locais é necessário clicar no botão `gerar`. Ao terminar a geração o sistema informará que houve a geração com sucesso ou com falha dependendo da situação encontrada.

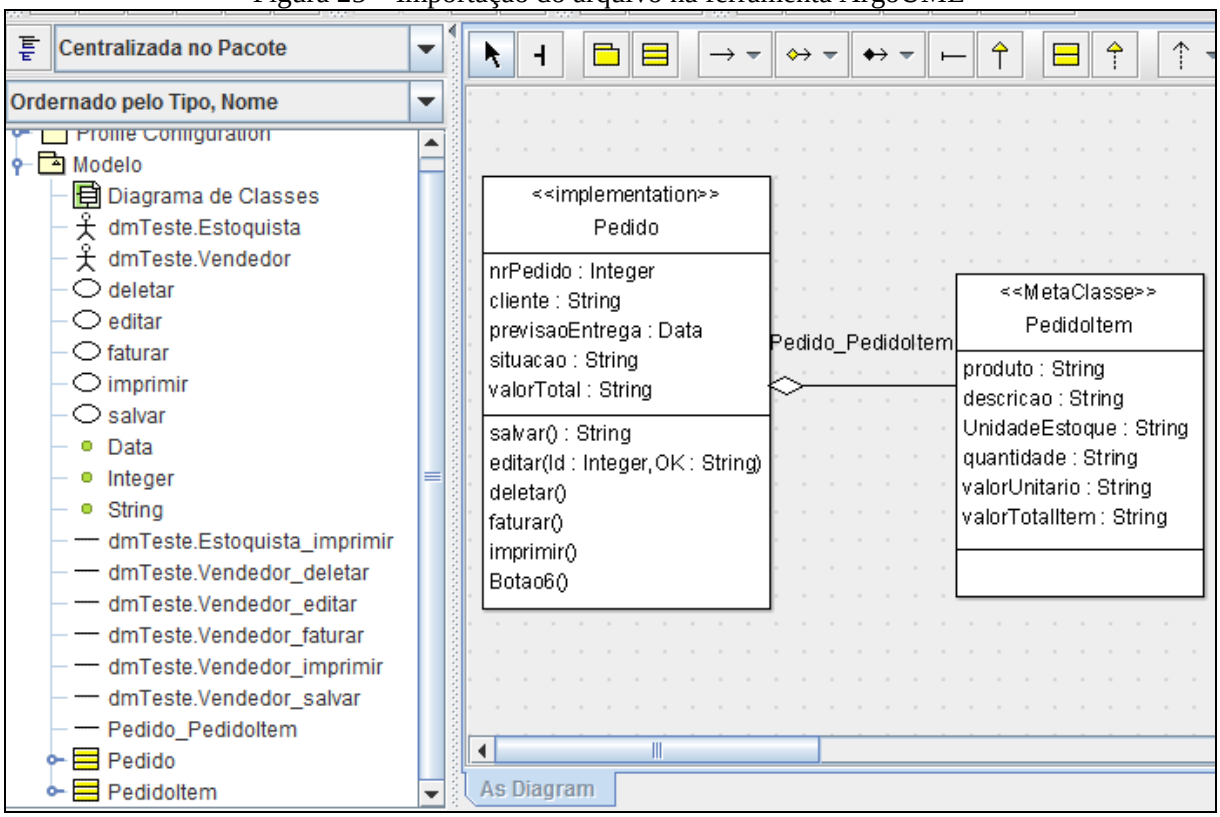
Figura 24 – Gerador de documentação



Com o arquivo gerado é possível importar o arquivo XMI em ferramentas de modelagem de software, como por exemplo, a Figura 25 ilustra a importação do arquivo na

ferramenta ArgoUML. Depois de importado o arquivo, basta arrastar os elementos para o diagrama. Os elementos que possuem relacionamentos aparecerão no momento em que os elementos do relacionamento estejam presentes no diagrama. Para mais detalhes da operacionalidade dos componentes, prototipação e exportação, há um tutorial que pode ser seguido conforme o Apêndice A. Este foi aplicado durante a pesquisa de campo realizada, para que os usuários participantes tivessem um aprendizado de como instalar e utilizar a ferramenta desenvolvida.

Figura 25 – Importação do arquivo na ferramenta ArgoUML



3.4 RESULTADOS E DISCUSSÕES

A fim de validar se os artefatos gerados pelos componentes gráficos desenvolvidos condizem com a prototipagem realizada, assim como a usabilidade do conjunto proposto, foi realizado uma pesquisa de campo com os alunos dos cursos de Ciência da Computação e Sistemas de Informação da Universidade Regional de Blumenau (FURB). A pesquisa foi respondida por dezoito alunos, sendo que treze estão cursando o terceiro semestre, três o quinto semestre e dois o sétimo semestre. Todos os alunos envolvidos já cursaram disciplinas de modelagem orientada a objetos e programação orientada a objetos, sendo que possuem conhecimento suficiente para identificar os artefatos gerados pelos diagramas de casos de uso e diagramas de classes.

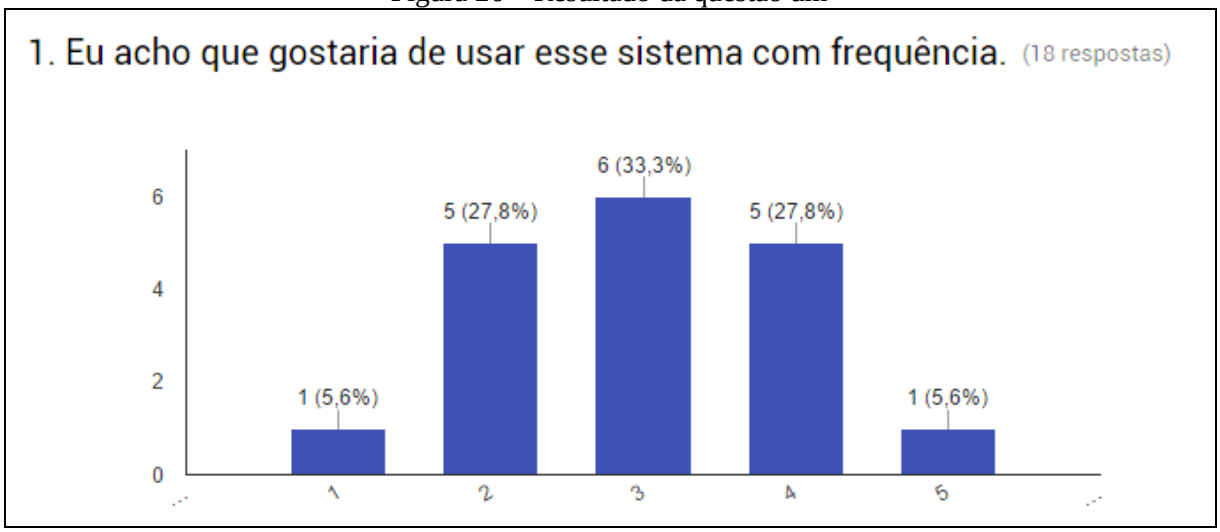
Antes de iniciar a pesquisa foi solicitado aos participantes a leitura e confirmação afirmativa ou negativa de um termo de consentimento livre e esclarecido para que estivessem cientes que a contribuição era voluntária, sem riscos e que poderiam desistir a qualquer momento. Após concordarem com o termo, iniciou-se a pesquisa de campo na qual os participantes seguiram um tutorial para aprenderem a utilizar os componentes do trabalho. O tutorial foi disponibilizado via Google Forms e além de disponibilizar os passos para instalação dos componentes, apresenta um passo-a-passo do uso dos componentes desenvolvidos, uma atividade para ser resolvida e um questionário final. O tutorial pode ser visualizado na íntegra no Apêndice A. Os participantes foram convidados a seguir todo o tutorial, bem como realizar a atividade proposta e, por fim, responder um questionário de usabilidade a respeito do trabalho desenvolvido.

Para a pesquisa de usabilidade foi aplicado o questionário *System Usability Scale* (SUS) (TEIXEIRA, 2015), por ser um método consolidado por vários pesquisadores e por ser simples de aplicar. O SUS é um questionário quantitativo e contém dez perguntas a serem respondidas na escala de *likert* de um (1) a cinco (5), onde um (1) significa discordo completamente e cinco (5) concordo completamente (TEIXEIRA, 2015). Foram adicionadas duas perguntas qualitativas baseada no questionário *Software Usability Measurement Inventory* (SUMI) (TEIXEIRA, 2015), sendo elas descritivas. Essas perguntas questionam qual é a melhor característica da ferramenta e qual característica deveria ser melhorada. Para ambas as perguntas é solicitado o motivo. Além destas, foram adicionadas mais quatro perguntas para identificar o curso e semestre do participante, assim como para identificar se os diagramas gerados foram de acordo com o que o participante tinha prototipado. No tutorial eles utilizaram a ferramenta ArgoUML para importação dos diagramas gerados, porém, em testes além do ArgoUML foi realizada a validação da importação no Enterprise Architect. O questionário pode ser visualizado na íntegra no mesmo Apêndice A, após o tutorial aplicado.

Nesta seção são ressaltadas as questões mais relevantes para a pesquisa. A pesquisa completa pode ser visualizada no Apêndice B. A primeira pergunta a ser destacada é “Eu acho que gostaria de usar esse sistema com frequência”. Essa pergunta é a primeira do questionário SUS e indica se os participantes acharam a ferramenta eficiente, fácil de utilizar e se sentiram confortáveis com ela. Conforme o gráfico apresentado na Figura 26, seis (6) dentre os dezoito participantes possivelmente usariam com mais frequência a ferramenta desenvolvida. Outros seis (6) participantes mantiveram sua resposta neutra, selecionando a escala intermediária e seis (6) participantes provavelmente não usariam.

Um terço dos participantes provavelmente não usaria a ferramenta por não se encaixarem em um perfil de um analista, ou então, por motivos de não gostar de prototipagem apenas de codificação, ou, por não trabalharem com a IDE Delphi, ou realmente por não terem gostado da ferramenta. Por outro lado, a ferramenta teve um público que possivelmente usaria a ferramenta, sendo estes um terço também dos participantes. Estes resultados incentivam a continuação do desenvolvimento destes componentes gráficos.

Figura 26 – Resultado da questão um



Outras duas perguntas que merecem destaque são “Eu achei que o diagrama de classes ficou compatível com minha prototipagem” e “Eu achei que o diagrama de casos de uso ficou compatível com minha prototipagem”, sendo estas perguntas externas adicionadas no questionário. Essas perguntas estão diretamente relacionadas à geração da modelagem através dos componentes gráficos desenvolvidos. Com base nas respostas, pode-se afirmar que o resultado da geração da modelagem de dados foi positivo, conforme podem ser observados os gráficos nas Figura 27 e 28, a prototipagem correspondeu com os diagramas gerados. Pode ser observado que um participante apontou como discordar um pouco mais do diagrama de casos de uso, mas foi o mesmo que apontou que não usaria o sistema com frequência e também respondeu que precisaria aprender várias coisas antes de utilizar os componentes e precisaria de ajuda de uma pessoa com conhecimento técnico para usar o sistema.

Figura 27 – resultado da questão onze

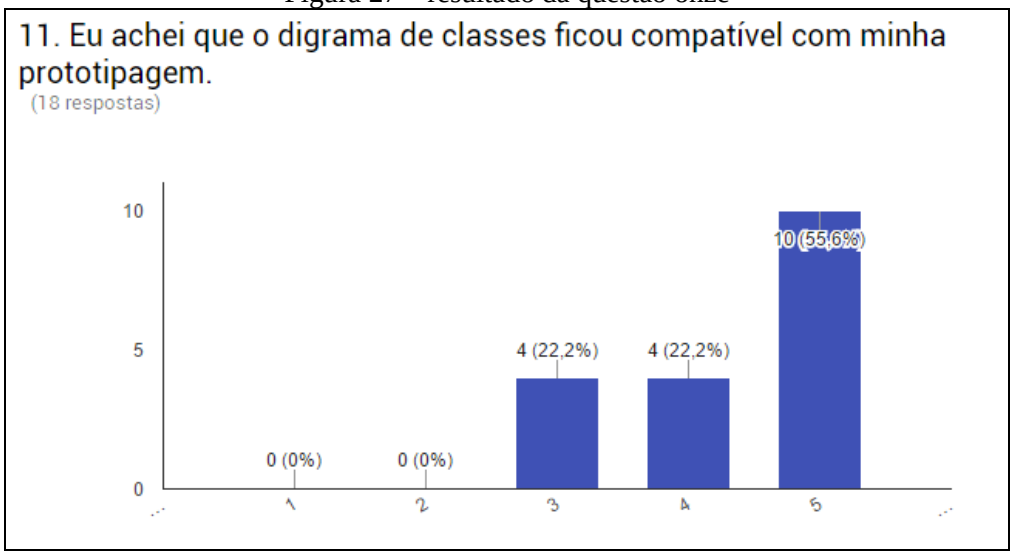
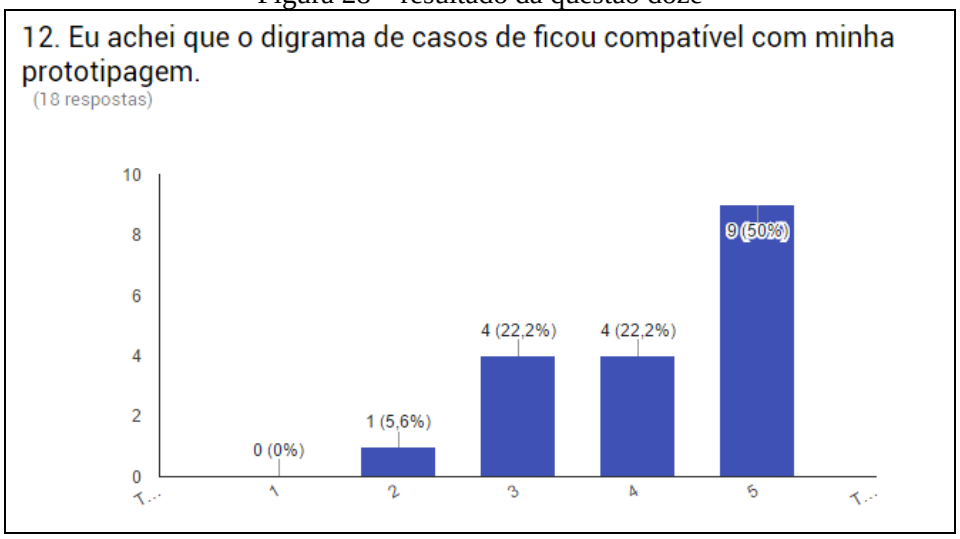


Figura 28 – resultado da questão doze



Como ponto chave da pesquisa, destaca-se a pergunta “o que você acha que é a melhor característica desta ferramenta, e por quê?”, pois no momento da apresentação da pesquisa não foi comentado sobre o objetivo dos componentes gráficos criados auxiliarem numa documentação rápida. As respostas para essa pergunta são apresentadas no Quadro 13.

Quadro 13 – Respostas das melhores características da ferramenta

A facilidade de criação, porque nem todo mundo tem muito tempo para gastar nisso então uma ferramenta que gere isso assim rapidamente é muito util
Agilidade.
Melhor visibilidade na criação de projetos.
Gerar diagramas automaticamente.
Eu só gerei a tela e ele me gerou todo o diagrama de caso de uso e classe com facilidade. Isso me economizaria muito tempo se o sistema fosse complexo, pois normalmente, você primeiro programa e depois modela.
Gera diagramas automáticamente, porque é chato fazer na mão
Facilita a criação da documentação por base da prototipagem de telas, ficando mais fácil fazer estas atividades poupando tempo.
Acelera as entregas de projetos poupando tempo em criar manualmente os diagramas e casos de usos
Gerar diagramas a partir de uma interface, porque quando for fazer um protótipo muitas vezes já tem-se a interface.
Modelar os diagramas de forma rápida e automatizada.
A facilidade de conseguir os diagramas
Praticidade
Aceitar arquivos xmi
Automatizar a etapa de casos e diagrama
Conseguir gerar os diagramas após a prototipagem automaticamente.
a facilidade de criar um diagrama

Destacam-se do Quadro 13 as palavras como: automatizar, praticidade, agilidade, facilidade, útil e acelera a entrega de projetos, definem como foi a experiência de ter uma documentação do sistema gerada pela a elaboração de uma prototipagem de telas. Estas respostas afirmam mais uma vez o objetivo do trabalho desenvolvido ter sido alcançado.

Na pergunta do que deveria ser melhorado na ferramenta e o porquê foi apontado a seguinte situação de que os participantes tiveram um pouco de dificuldade na execução do tutorial, pois o mesmo deveria ser mais detalhado e claro. Assim como um pouco de dificuldade no uso da IDE Delphi por ter sido para alguns a primeira vez a ser utilizada. Conforme o Quadro 14 pode-se notar a resposta do participante.

Quadro 14 – Resposta do participante

Quanto a ferramenta, aparentemente é algo simples de usar, porém senti alguma dificuldade na utilização devido a falta de algumas informações no tutorial, como por exemplo salvar todo o projeto. Seguindo passo a passo o tutorial para alguem que desconhece o Delphi como eu, chegamos em alguns pontos que gera dúvidas de como proceder.

A partir da pesquisa realizada, pode-se afirmar que os componentes gráficos desenvolvidos neste trabalho permitem a geração de diagramas de classe e casos de uso compatíveis com o protótipo realizado, o que é algo relevante para a pesquisa. Sabe-se que há

ainda vários itens a serem melhorados nos componentes gráficos, mas através do primeiro teste de campo, obteve-se um *feedback* positivo de um terço dos participantes, ressaltando que outro um terço ficou indefinido por escolher a escala de *likert* intermediária, sendo este também um resultado positivo. Sendo assim, entende-se que a documentação é importante e que ter uma ferramenta eficaz que agilize este processo é imprescindível.

O Quadro 15 apresenta de forma comparativa algumas características em relação aos trabalhos correlatos apresentados e este trabalho.

Quadro 15 – Características de todos trabalhos

Características	Atlântico	Elicitar	QEA	ESS-Model	Trabalho desenvolvido
utiliza a geração de prototipagem de telas	Sim	Sim	Não	Não	Sim
utiliza o padrão XMI para exportação/leitura de diagramas da UML	Não	Não	Sim	Sim	Sim
ferramenta é dependente de aplicações comerciais	Sim	Não	Sim	Não	Sim
busca otimizar o processo de desenvolvimento de softwares	Sim	Sim	Sim	Sim	Sim

Em comparação aos trabalhos correlatos, o trabalho realizado contempla um pouco de cada um. Neste trabalho é utilizada a prototipagem como entrada de dados fundamental, conforme o trabalho do Atlântico que destacou como uma parte bem importante na busca de informações com os usuários envolvidos. Em comparação ao Elicitar, este trabalho faz o inverso, ou seja, ao invés de ter a prototipagem como saída tem como entrada de dados. O QEA e o ESS-Model utilizam o padrão XMI para exportação/leitura de diagramas da UML, porém único modelo que utilizam é o diagrama de classes. Já o trabalho desenvolvido utiliza a o padrão XMI para exportação a diagrama de classes e diagrama de casos de uso. Cada um teve a sua otimização de acordo com sua necessidade, mas todos buscaram otimizar alguma parte do processo de desenvolvimento de software tendo seus resultados positivos.

4 CONCLUSÕES

Este trabalho apresentou o desenvolvimento de componentes gráficos para prototipagem e documentação rápida em Delphi. Os objetivos dos componentes gráficos eram auxiliar na prototipagem de telas e transformar a prototipagem realizada em modelos de casos de uso e de classes da UML sendo compatíveis às ferramentas de modelagem de software, o qual foi alcançado.

A ferramenta foi desenvolvida utilizando a linguagem de programação Object Pascal, no ambiente de desenvolvimento Delphi 10 Seattle. Para geração dos modelos da UML foi utilizado o padrão XMI com o auxílio da biblioteca `TXMLDocument` para criação das *tags* garantindo assim, a visualização e alteração dos artefatos da UML em diversas ferramentas de modelagem de software. Para a realização dos testes de importação e validação do XMI foram utilizados os softwares de modelagem de dados ArgoUML e Enterprise Architect.

Com os resultados obtidos a partir da pesquisa de campo foram identificados que aproximadamente 78% dos diagramas de casos de uso e aproximadamente 77% dos diagramas de classes gerados pelos componentes gráficos criados apresentaram-se compatíveis com as prototipagens de telas realizadas. Não significando que os componentes gráficos criados não funcionam, mas que pela percepção dos participantes da pesquisa os diagramas gerados seriam modelados/interpretados diferente por eles. Assim como, um terço dos participantes disseram que usariam com frequência a ferramenta, além de destacarem a agilidade e facilidade que esta ferramenta traz para geração da documentação.

A ferramenta ainda apresenta algumas limitações para os diagramas de classes e caso de uso, como apenas a possibilidade do uso do relacionamento de agregação entre classes, a indisponibilidade dos relacionamentos `include` e `extends` de casos de uso e o componente gráfico `TGrade` gerar os atributos com a visibilidade e o tipo de dados padrões sem ter a opção na prototipagem para o usuário escolher. No entanto, ao importar os modelos nas ferramentas de modelagem de softwares essas opções podem ser alteradas sem nenhum problema.

Os componentes gráficos para prototipagem e documentação rápida em Delphi podem servir para o desenvolvimento de novas ferramentas para a geração de diversos modelos da UML. É possível concluir que o desenvolvimento deste trabalho possibilitou uma alternativa para documentar sistemas de forma rápida e que as limitações da ferramenta poderão ser alteradas na modelagem dos diagramas.

4.1 EXTENSÕES

Existem pontos a serem melhorados e incrementados nos componentes gráficos desenvolvidos, sendo eles:

a) implementar os demais tipos de relacionamentos para o diagrama de classes: generalização, associação, composição e dependência;
b) implementar os relacionamentos *extends* e *include* para o diagrama de casos de uso;
c) implementar para que possam ser utilizados diferentes tipos de dados dos atributos gerados por um componente gráfico do tipo `TGrade`;
d) implementar a exportação para demais diagramas da UML, como por exemplo o de sequência, podendo ser utilizada a comunicação entre os formulários e a chamada dos botões;
e) criar os demais componentes gráficos para complementar a prototipagem, como por exemplo `TListBox`, `TListView`, `TDateEdit` e demais componentes conforme for a necessidade;
f) implementar para que seja adicionado na lista de tipos de dados os novos tipos digitados pelo usuário, para que o mesmo não precise sempre redigitar;
g) complementar na classe `TDocumentacao` para ter a propriedade que indique se o componente será um método e implementá-la no componente `TBotao`, com o mesmo intuito da propriedade atributo e classe.

REFERÊNCIAS

ARAÚJO, Marco; VALE, Ricardo; SOUZA Vinícius. Prototipação no desenvolvimento de software. **Engenharia de Software**, [S.l.], v. 17, n. 2, p. 44-50. 2008. Disponível em: <http://www.devmedia.com.br/artigo-engenharia-de-software-17-prototipacao-no-desenvolvimento-de-software/14502>. Acesso em 01 nov. 2015.

BATTISTI, Leandro V. **Elicitar**: Protótipo de gerador de código a partir de especificações com padrão de requisitos. 2014. 54 f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

BEZERRA, Eduardo. **Princípios de análise e projeto de sistemas com UML**. 2ª edição. Rio de Janeiro: Elsevier, 2007.

BROOKSHEAR, Glenn J. **Ciência da Computação**: Uma visão abrangente. 11. ed. Porto Alegre: Bookman, 2013. 573 p. Tradução Eduardo Kessler Piveta.

CAETANO, Cristiano. ESS-Model. **Clube Delphi**, Rio de Janeiro, v. 67, n. 5, p. 48-49. 2005. Disponível em: <http://www.devmedia.com.br/artigo-clube-delphi-67-ess-model/13832>. Acesso em 22 maio 2016.

CHAOS, The Standish Group. **The CHAOS Manifesto 2013**. 2013. Disponível em <https://larlet.fr/static/david/stream/ChaosManifesto2013.pdf>. Acesso em: 18 set. 2015.

ESPOSITO, Adriano. Diagrama de casos de uso: principais desafios na elaboração. **Engenharia de Software**, [S.l.], v. 65, n. 5, p. 52-56. 2013. Disponível em: <http://www.devmedia.com.br/diagrama-de-casos-de-uso-principais-desafios-na-elaboracao/29790>. Acesso em 01 nov. 2015

FARIA, Rodolfo. Introdução à criação de componentes. **Clube Delphi**, Rio de Janeiro, v. 71, n. 5, p. 12-16. 2006. Disponível em: <http://www.devmedia.com.br/artigo-clube-delphi-71-criacao-de-componentes/11548>. Acesso em 01 nov. 2015.

FREITAS, Marcio L. **UML**. [S.l.], 2008. Disponível em <http://www.devmedia.com.br/uml/8579>. Acesso em 01 nov. 2015.

HENSGEN, Paul. **Manual do Umbrello UML Modeller**. [S.l.], 2013. Tradução Marcus Gama. Disponível em <https://docs.kde.org/trunk4/pt_BR/kdesdk/umbrello/index.html>. Acesso em 01 nov. 2015.

GALEOTE, Sidney. **Conceitos e importância da prototipação de requisitos**. [S.l.], 2012. Disponível em <http://www.galeote.com.br/blog/2012/11/conceitos-e-importncia-da-prototipao-de-requisitos/>. Acesso em 18 out. 2015.

GUEDES, Gilleanes T. A. **UML 2**: uma abordagem prática. 2ª ed. São Paulo: Novatec Editora, 2011.

LAPORTE, Daniel S. Conheça a base da VCL. **Clube Delphi**, Rio de Janeiro, v. 127, n. 9, p. 6-10. 2011. Disponível em: <http://www.devmedia.com.br/conheca-a-base-da-vcl-artigo-revista-clube-delphi-127/20109>. Acesso em 22 maio 2016.

LOBO, Edson J. **Guia prático de engenharia de software**. São Paulo: Digerati Books, 2009.

MACÈDO, Ana; SPÍNOLA, Rodrigo. Ciclos de vida do software. **Engenharia de Software**, [S.l.], v. 36, n. 3, p. 21-28. 2011. Disponível em: <http://www.devmedia.com.br/ciclos-de-vida-do-software-artigo-revista-engenharia-de-software-magazine-36/21099>. Acesso em 01 nov. 2015.

MELO, Ana. UML: Diagrama de classes. **Engenharia de Software**, [S.l.], v. 12, n. 1, p. 18-23. 2009. Disponível em: <http://www.devmedia.com.br/artigo-engenharia-de-software-12-uml-diagrama-de-classes/12563>. Acesso em 01 nov. 2015.

MORAIS, Lenildo. Modelagem em uma visão ágil. **Engenharia de Software**, [S.l.], v. 32, n. 3, p. 9-12. 2011. Disponível em: <http://www.devmedia.com.br/modelagem-em-uma-visao-agil-engenharia-de-software-32/19006>. Acesso em 01 nov. 2015.

NORONHA, Robinson Vida. **Relacionamento entre classes**. Disponível em: <http://www.professorvida.com.br/if62c/material/relacionamentos.pdf>. Acesso em: 25 fev. 2016.

OBJECT MANAGEMENT GROUP. **OMG Unified Modeling Language TM (OMG UML) version 1.4**. Needham: OMG, 2001. Disponível em: <http://www.omg.org/spec/UML/1.4/PDF>. Acesso em 22 jun. 2016.

_____. **OMG Unified Modeling Language TM (OMG UML) version 2.5**. Needham: OMG, 2015. Disponível em: <http://www.omg.org/spec/UML/2.5/PDF>. Acesso em 01 nov. 2015.

PAULI, Guinther. Introdução à criação de componentes. **Clube Delphi**, Rio de Janeiro, v. 98, n. 8, p. 54-58. 2008. Disponível em: <http://www.devmedia.com.br/artigo-clube-delphi-magazine-98-introducao-a-criacao-de-componentes/10450>. Acesso em 01 nov. 2015.

PRESSMAN, Roger S. **Engenharia de software**. Tradução José Carlos Barbosa dos Santos. São Paulo: Makron Books, 1995.

ROGERS, Yvonne; SHARP, Helen; PREECE, Jennifer. **Design de Interação**: Além da interação humano-computador. 3. ed. Porto Alegre: Bookman, 2013. 585 p. Tradução Isabela Gasparini.

ROSEMBERG, Carlos et al. Prototipação de Software e Design Participativo: uma Experiência do Atlântico. In: SIMPÓSIO SOBRE FATORES HUMANOS EM SISTEMAS COMPUTACIONAIS, 8., 2008, Porto Alegre. **Anais...** Porto Alegre: IHC, 2008. p. 312-315.

SAUVÉ, Jacques P. **Processos de Desenvolvimento de Software**, Campina Grande, 2001. Disponível em: <http://www.dsc.ufcg.edu.br/~jacques/cursos/proj/gerenciadesenv/processos.htm>. Acesso em 22 maio 2016.

SOUZA, Bruna E. D. O. **QEA**: integração entre a ferramenta para desenvolvimento de sistemas web Quellon e o Enterprise Architect. 2011. 57 f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) – Centro de Ciências Extas e Naturais, Universidade Regional de Blumenau, Blumenau.

TEIXEIRA, Fabricio. **O que é o SUS (System Usability Scale) e como usá-lo em seu site**, [S.l.], 2015. Disponível em: < http://arquiteturadeinformacao.com/usabilidade/o-que-e-o-sus-system-usability-scale-e-como-usa-lo-em-seu-site/>. Acesso em 18 maio 2016.

WAZLAWICK, Raul Sidnei. **Análise e projeto de sistemas de informação orientados a objetos**. 2. ed. Rio de Janeiro: Elsevier, 2011. 330 p.

APÊNDICE A – Tutorial e pesquisa aplicada em campo

Este apêndice contém o passo a passo que o participante precisava fazer ao participar da pesquisa. Este conteúdo foi retirado do Google Forms.

Figura 29 – Passo 1

Tutorial de uso dos componentes gráficos para prototipagem e documentação rápida em Delphi

Leia o termo de consentimento e siga o passo a passo deste tutorial e no fim responda a um questionário

*Obrigatório

Termo de consentimento livre e esclarecido

Convido-o (a) para participar da pesquisa de componentes gráficos para prototipagem e documentação rápida em Delphi, sob a reponsabilidade do pesquisador Reinoldo Krause Junior, o qual pretende testar a usabilidade de sua ferramenta para desenvolvimento do TCC.

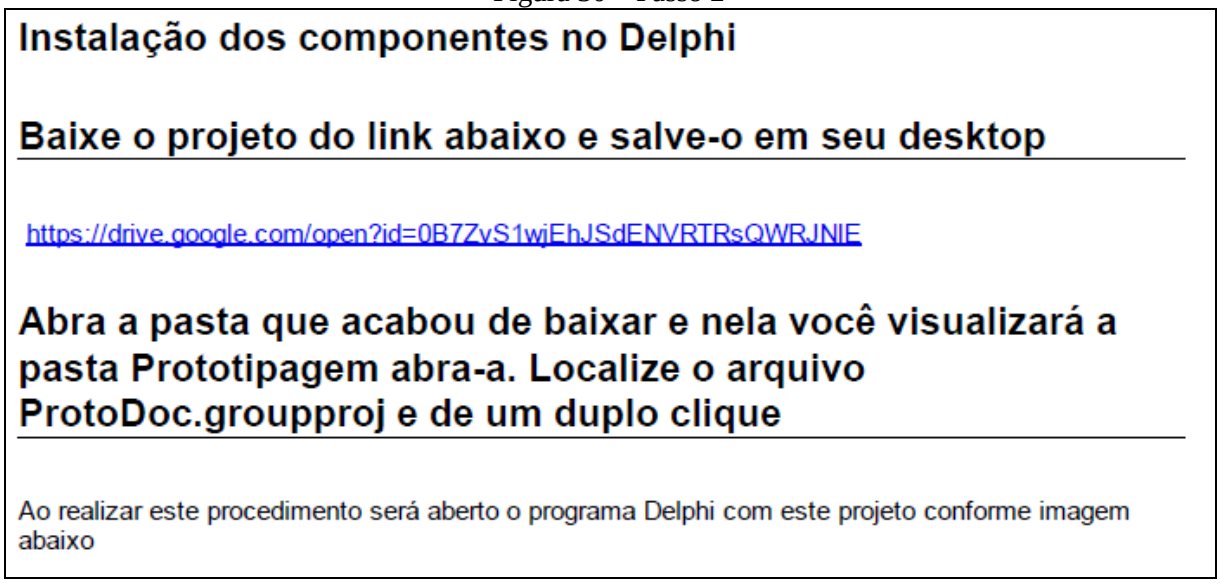
Sua participação é voluntária.

Se depois de consentir em sua participação o Sr (a) desistir de continuar participando, tem o direito e a liberdade de retirar seu consentimento em qualquer fase da pesquisa, seja antes ou depois da coleta dos dados, independente do motivo e sem nenhum prejuízo a sua pessoa. O (a) Sr (a) não terá nenhuma despesa e também não receberá nenhuma remuneração. Os resultados da pesquisa serão analisados e publicados, mas sua identidade não será divulgada, sendo guardada em sigilo.

Para qualquer outra informação, o (a) Sr (a) poderá entrar em contato com o pesquisador no endereço de email reiinoldo@gmail.com, ou pelo telefone (47) 9907-4810.

Prosseguindo para o próximo passo o (a) Sr (a) está de acordo com este termo.

Figura 30 – Passo 2



Instalação dos componentes no Delphi

Baixe o projeto do link abaixo e salve-o em seu desktop

https://drive.google.com/open?id=0B7ZvS1wjEhJSdENVRTRsQWRJNlE

Abra a pasta que acabou de baixar e nela você visualizará a pasta Prototipagem abra-a. Localize o arquivo ProtoDoc.groupproj e de um duplo clique

Ao realizar este procedimento será aberto o programa Delphi com este projeto conforme imagem abaixo

Figura 31 – Passo 3

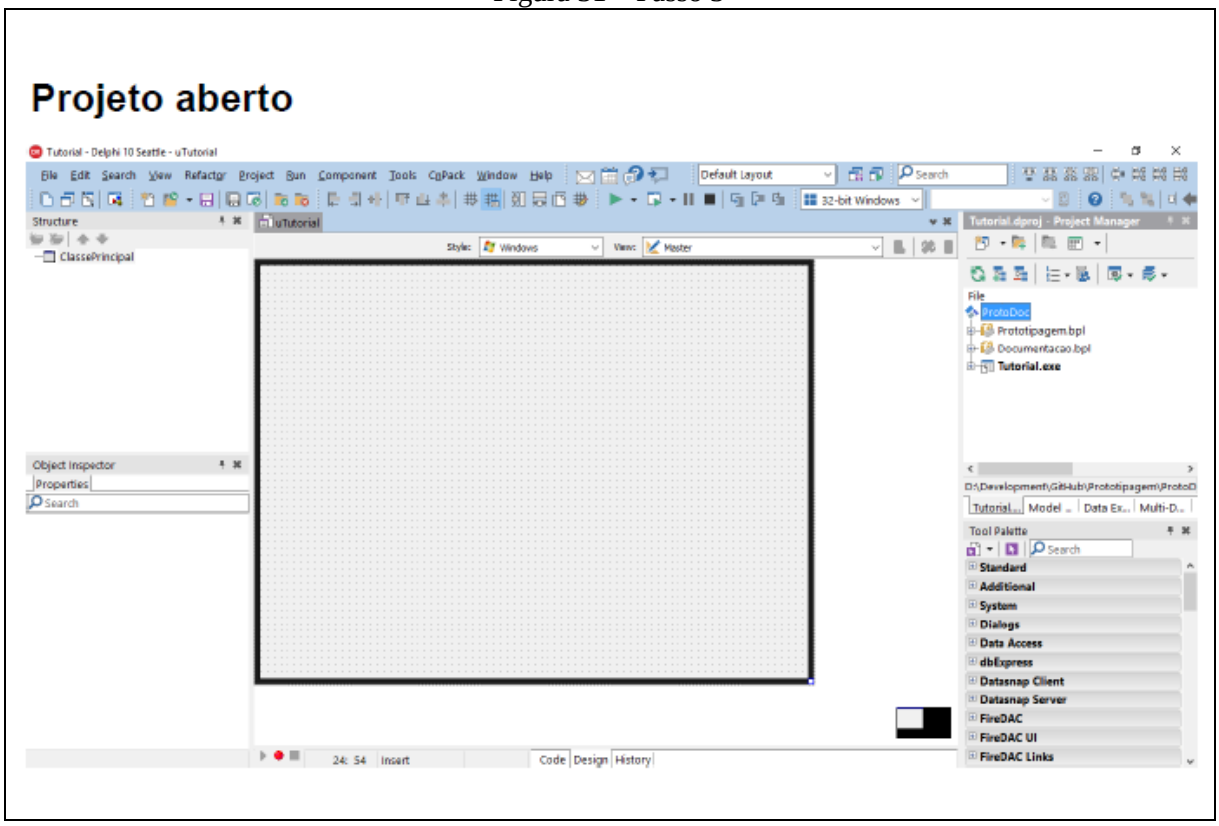


Figura 32 – Passo 4

No canto direito superior pode ser visualizado o arquivo ProtDoc, clique com o botão direito e vá na opção Build All. Após dar o Build, selecione os dois arquivos Prototipagem.bpl e Documentacao.bpl clique com o botão direito sobre eles e vá na opção Install

Realizado este procedimento você verá que os dois arquivos .bpl ficaram com uma cor roxa significando que foram instalados, conforme imagem abaixo

Arquivos instalados

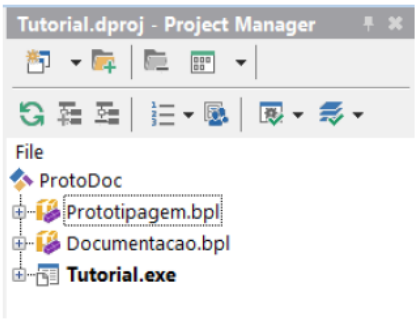


Figura 33 – Passo 5

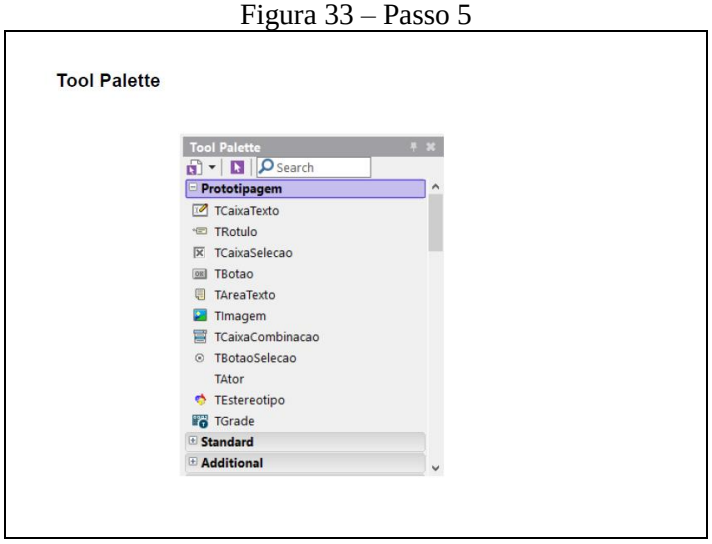


Figura 34 – Passo 6

Modele o formulário utilizando os seguintes componentes

TCaixaTexto para informar o código, nome e email do cliente
TCaixaSelecao para marcar se o cliente deseja receber email marketing, email com noticiais
TAreaTexto para informar uma observação do cliente
TImagem para adionar uma imagem do cliente
TCaixaCombinacao para informar se o cliente é pessoa física, jurídica ou exterior
TBotaoSelecao para informar se o cliente é consumidor ou revendedor
TGrade para informar os contatos do cliente, onde deverá ter 3 colunas, Tipo do Contato, Telefone, Responsável
TAtor para informar dois tipos de atores, Ator Secretária, Ator Supervisor
TBotao para Salvar, Excluir e Exportar onde o Salvar poderá ser feito pelo ator Secretária e Supervisor , e o Excluir e Exportar apenas pelo ator Supervisor
TRotulo para identificar o que é cada campo

Use a criatividade para ficar um cadastro simples e fácil para o usuário utilizar

Figura 35 – Passo 7

Importante para cada componente adicionado coloque um nome na propriedade name situada no objetc inspector

Dica selecione o objeto e aperta a tecla F11. Exemplo de inserção do nome na propriedade name do componente

Object Inspector

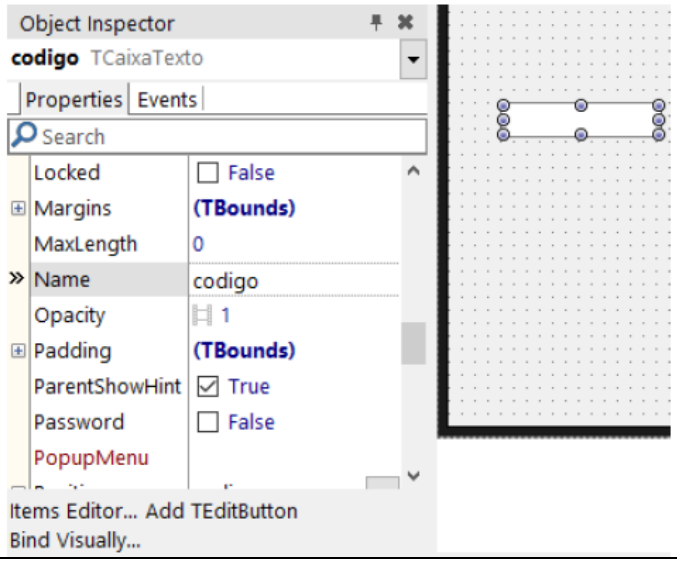


Figura 36 – Passo 8

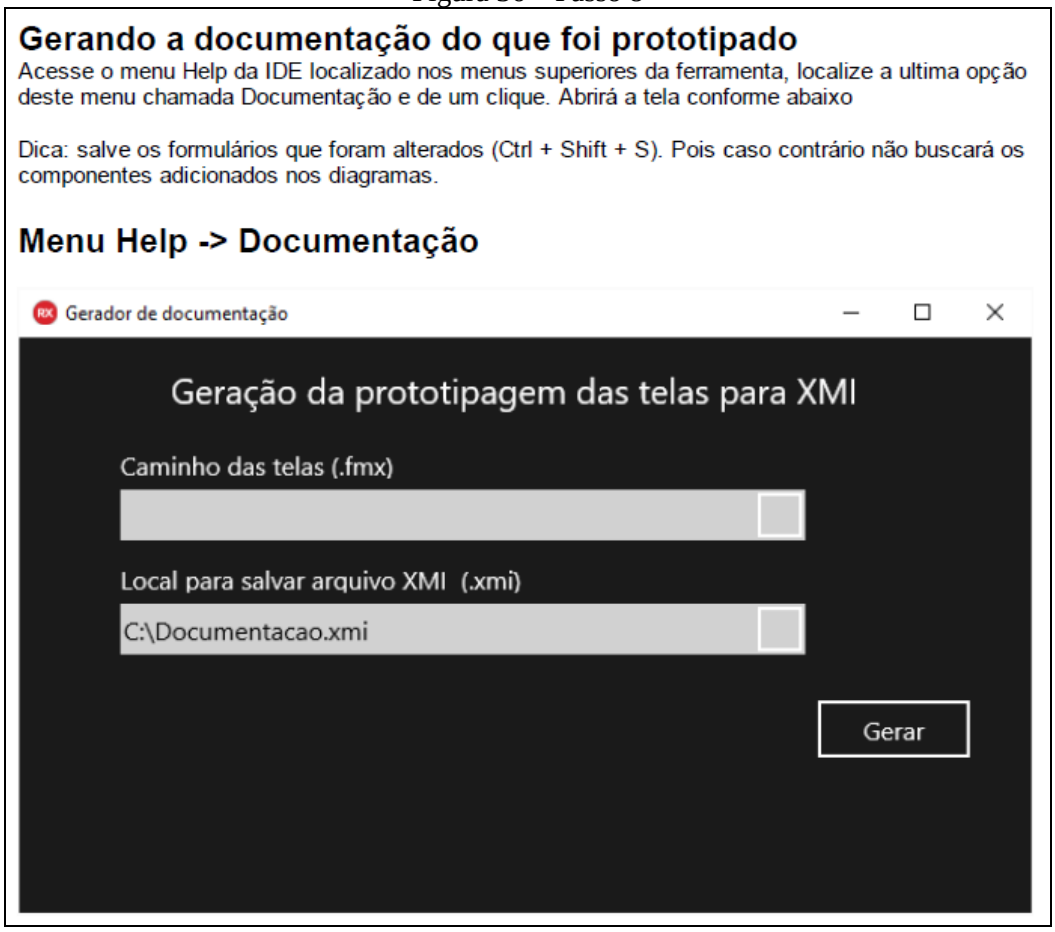


Figura 37 – Passo 9

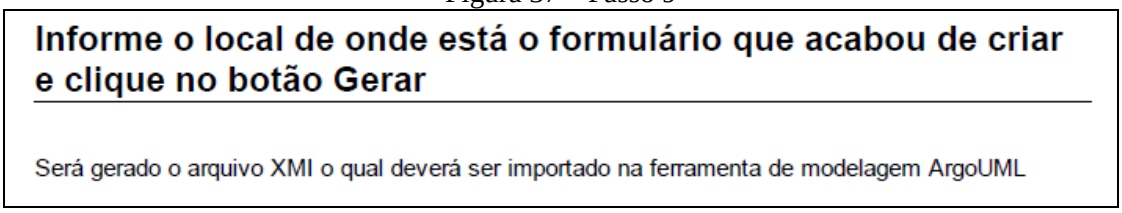


Figura 38 – Passo 10

Acesso o programa de modelagem de software ArgoUML e importe o arquivo gerado

Acesse o menu Arquivo situado no canto superior da ferramenta e vá na opção Importar XMI, selecione o arquivo e importe, conforme a figura abaixo

Importação do arquivo XMI

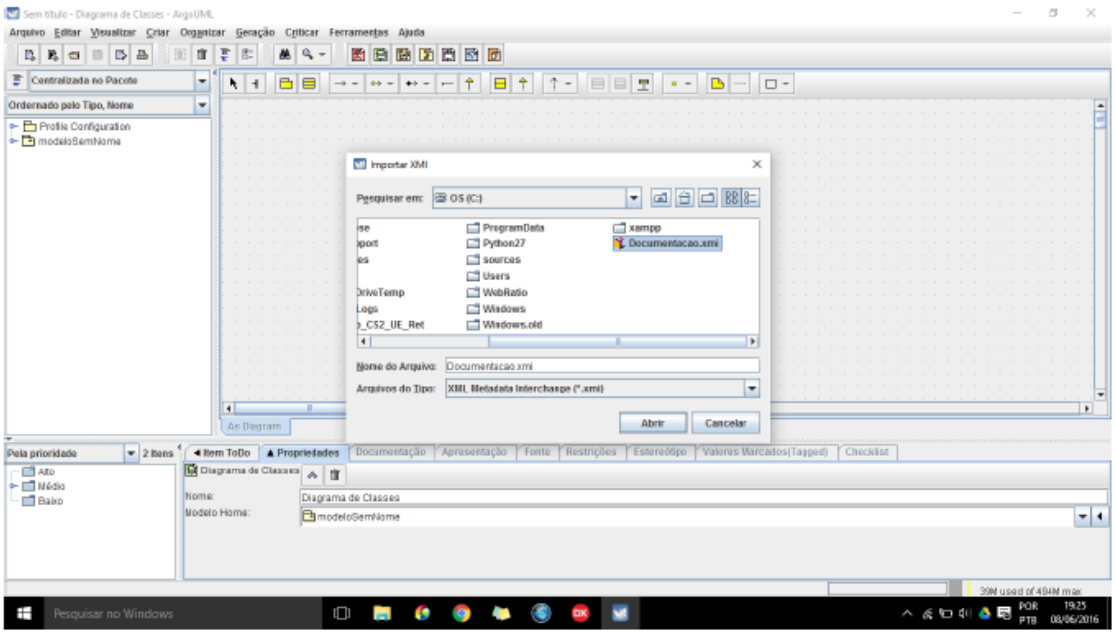


Figura 39 – Passo 11

Verificando a documentação gerada

Ao importar no canto esquerdo terá um pacote com nome de Modelo abra-o e identificará a que possui a ClassePrincipal araste-a para o Diagrama e ficará conforme a imagem abaixo

ClassePrincipal no diagrama de classes

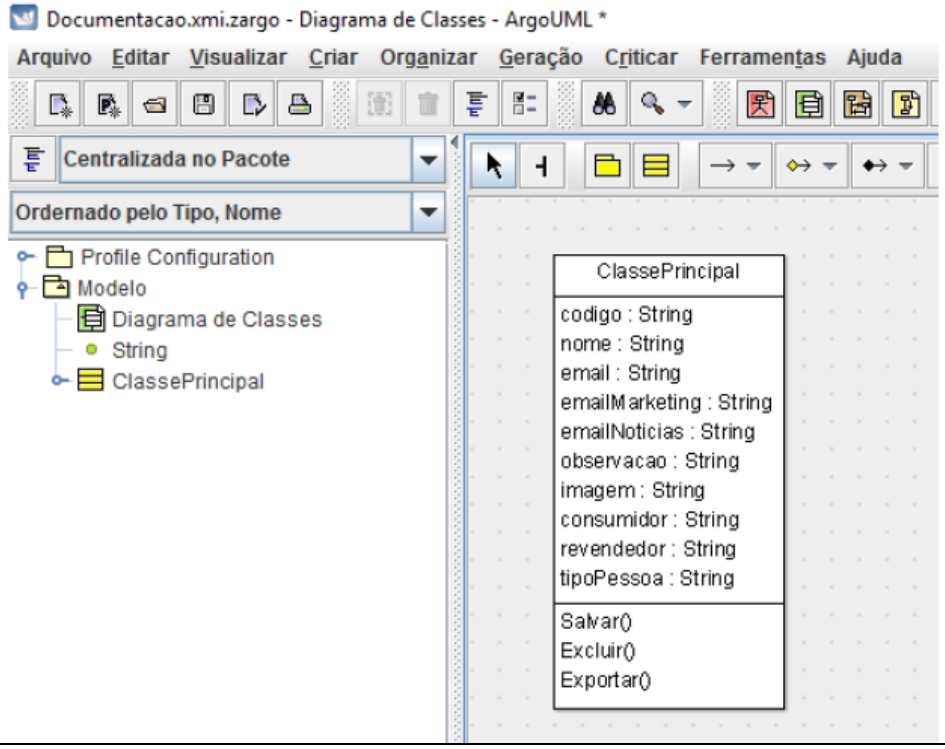


Figura 40 – Passo 12

Através de uma simples prototipagem você criou um diagrama de classes. Que tal agora através desta prototipagem você gerar um caso de uso?

Vá no Delphi novamente, e selecione o botão salvar, procure pela propriedade DocumentacaoAtor. Dica pode ser digitado para procurar esta palavra conforme a imagem abaixo

Object Inspector do botão Salvar

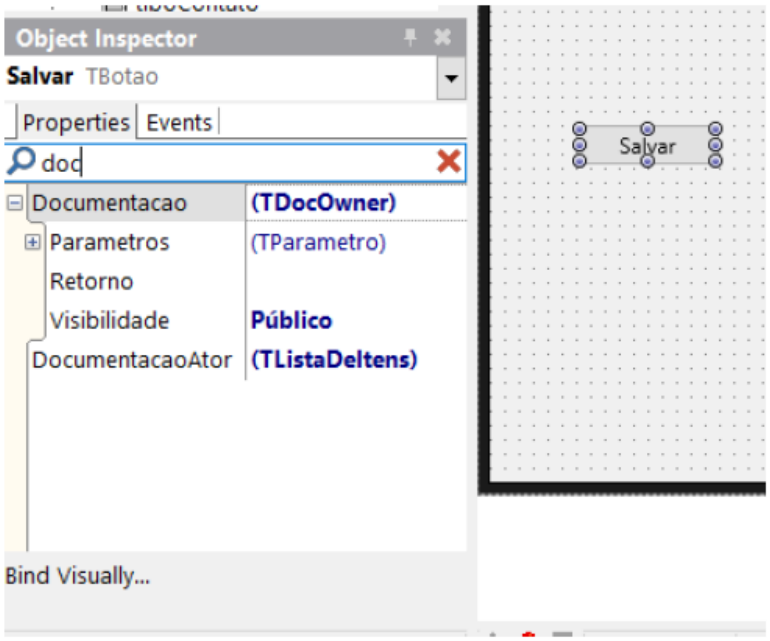


Figura 41 – Passo 13

De um duplo clique em cima da propriedade (TListaDeItens). Abrirá uma janela conforme a imagem abaixo. Clique em inserir e selecione o ator responsável por esta ação. Faça a mesma situação para os demais botões.

Selecione um ator

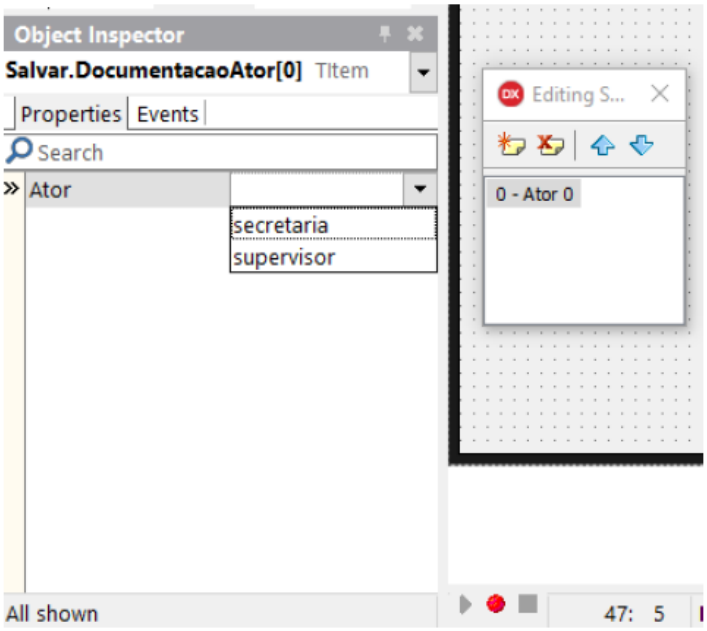


Figura 42 – Passo 14

Vá no menu Help, acesse a opção documentação e gere novamente o arquivo. Importe no ArgoUML e arraste os novos elementos ator e casos de uso para o diagrama e verificará que ficará conforme a imagem abaixo.

ArgoUML Casos de Uso

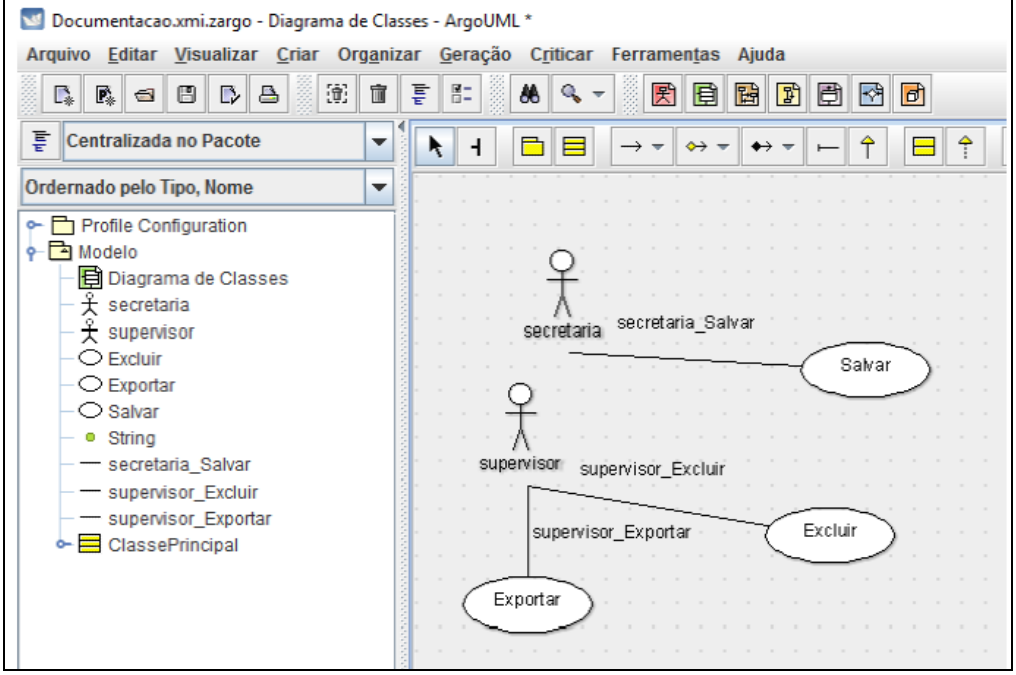


Figura 43 – Passo 15

Agora que já pegou um pouco mais de experiencia explore os demais componentes que você adicionou no formulário, verificará que eles tem informações que afetarão na visualização dos diagramas. O próximo passo será resolver um exercício.

Exercício

Explicando um pouco mais do que representa cada componente no formulário em relação aos diagrama de classes e caso de uso.

Todo formulário criado representa o elemento classe no diagrama de classes.
Toda TGrade adicionada e marcada a propriedade "classe" da "Documentacao" com o valor de verdadeira, se torna o elemento classe no diagrama de classes

Para identificar se classe possui um estereótipo é utilizado o objeto TEstereotipo, atrelado a uma TGrade, ou simplesmente no formulário.

Para identificar se é elemento atributo do diagrama de classes verifica se no formulário possui os objetos TAreaTexto, TBotaoSelecao, TCaixaCombinacao, TCaixaSelecao, TCaixaTexto, TImagem, TRotulo e se as mesmas contiverem a propriedade "atributo" da "Documentacao" com o valor de verdadeira.

Para identificar se é um elemento Método do diagrama de classes verifica se no formulário possui o objeto TBotao. Se no mesmo contiver a informação de Ator (DocumentacaoAtor) é criado o elemento Ator e e um elemento CasoDeUso com a funcionalidade informada no nome do botão para o diagrama de casos de uso.

Figura 44 – Passo 16

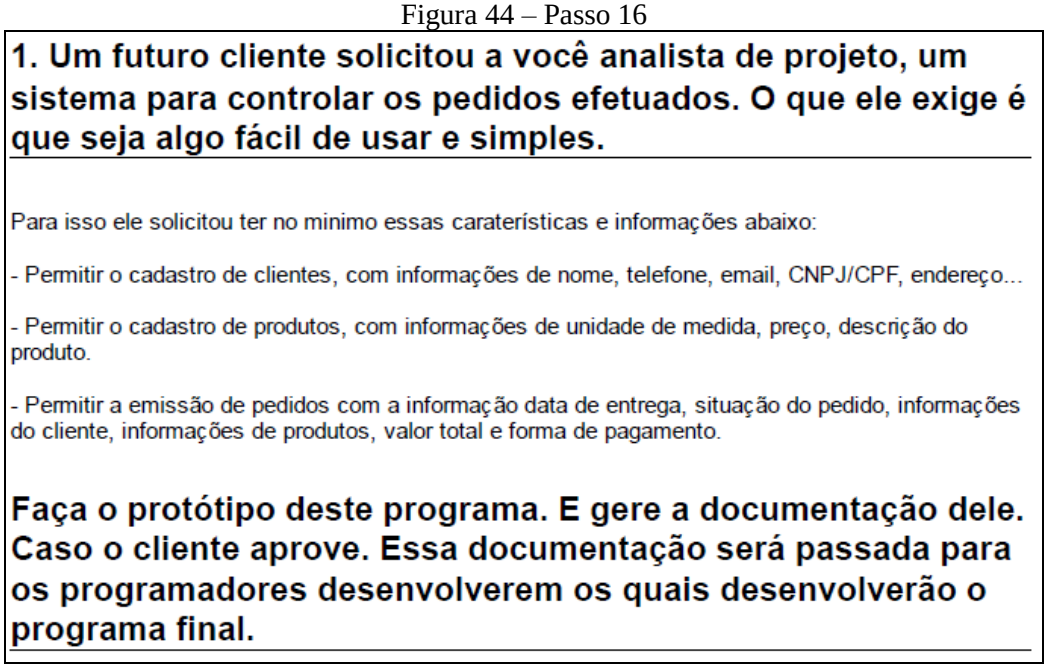


Figura 45 – Grupo de questões 1

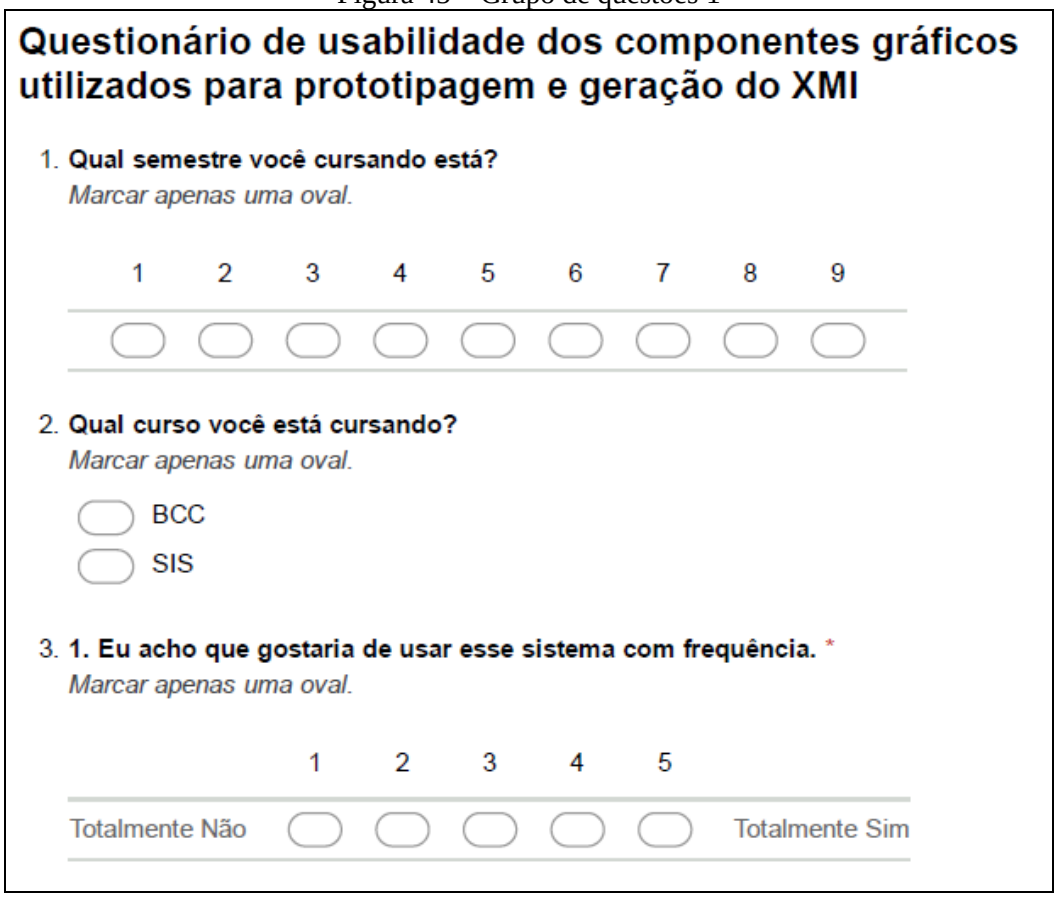


Figura 46 – Grupo de questões 2

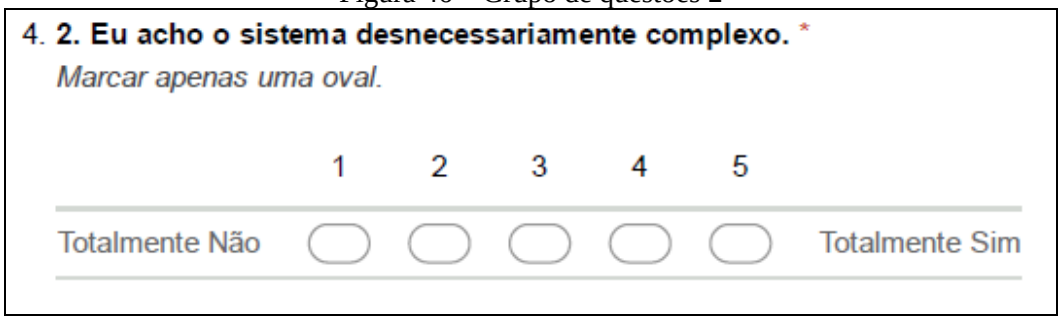


Figura 47 – Grupo de questões 3

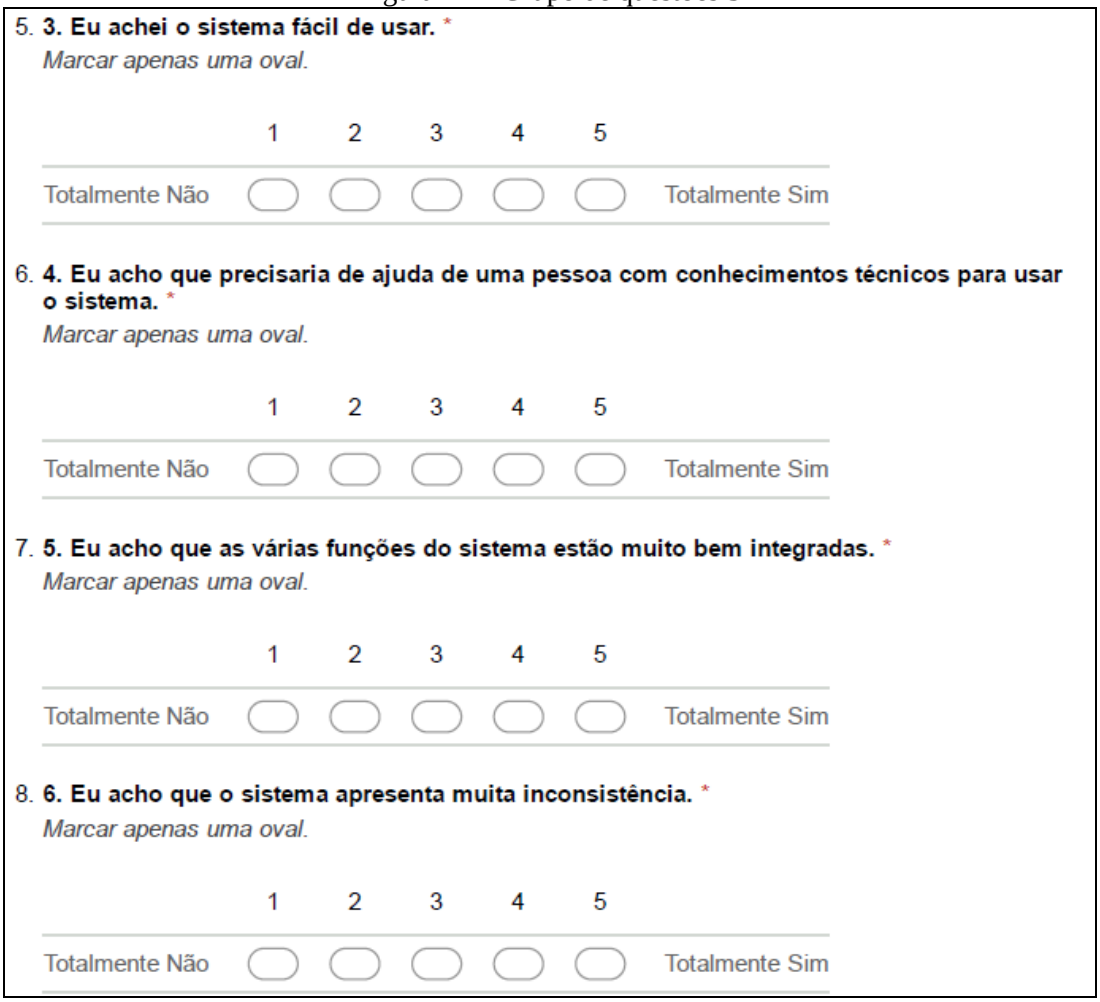


Figura 48 – Grupo de questões 4

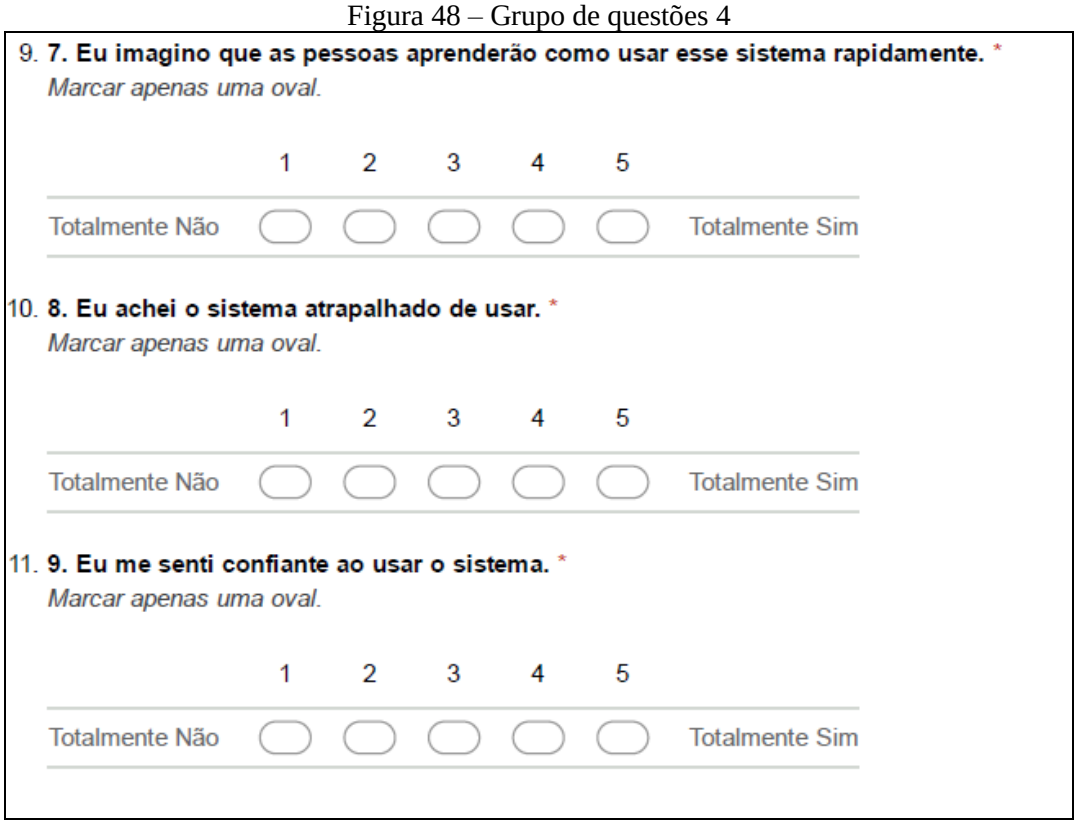


Figura 49 – Grupo de questões 5

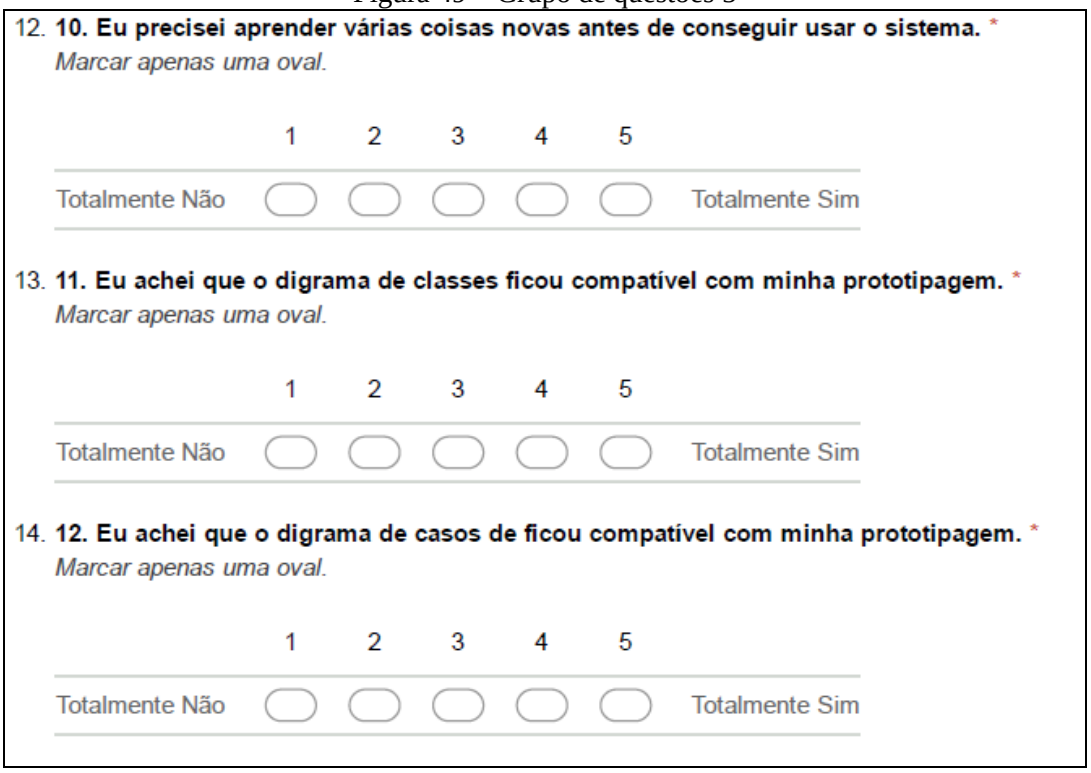


Figura 50 – Grupo de questões 6

15. **13. O que você acha que é a melhor característica desta ferramenta, e por quê?**

---

16. **14. Qual característica desta ferramenta você acha que deve ser melhorado, e por que?**

Apêndice B – Resultado do questionário

Este apêndice contém todos os resultados da pesquisa de campo aplicada, sendo os gráficos gerados pelo Google Forms.

Figura 51 – Gráfico 1

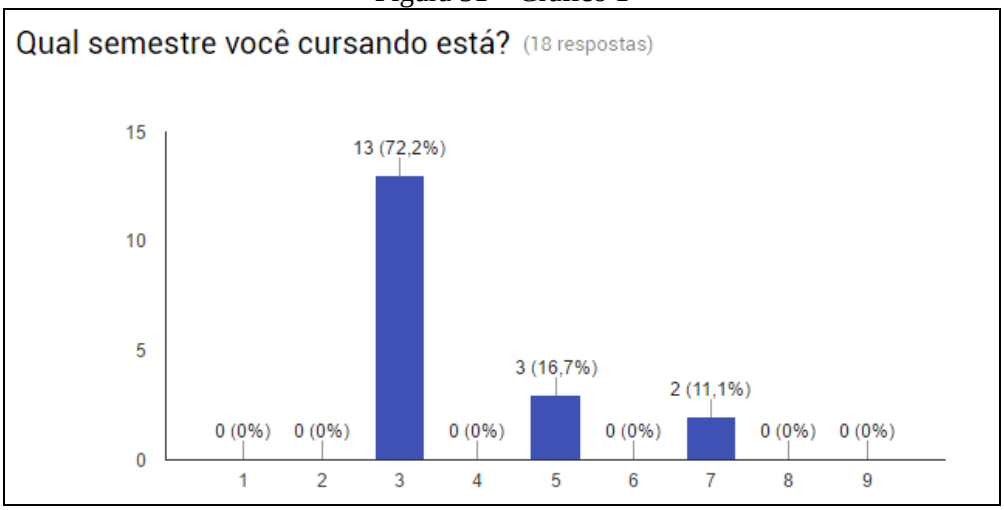


Figura 52 – Gráfico 2

Figura 53 – Gráfico 3

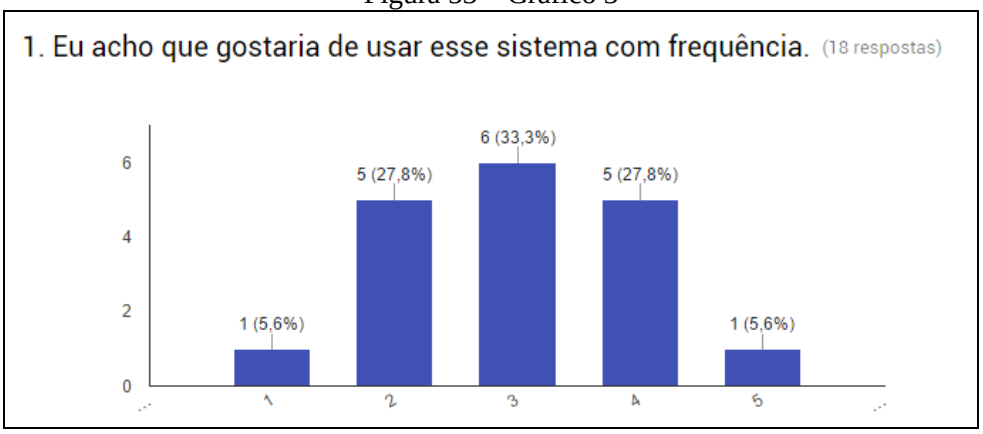


Figura 54 – Gráfico 4

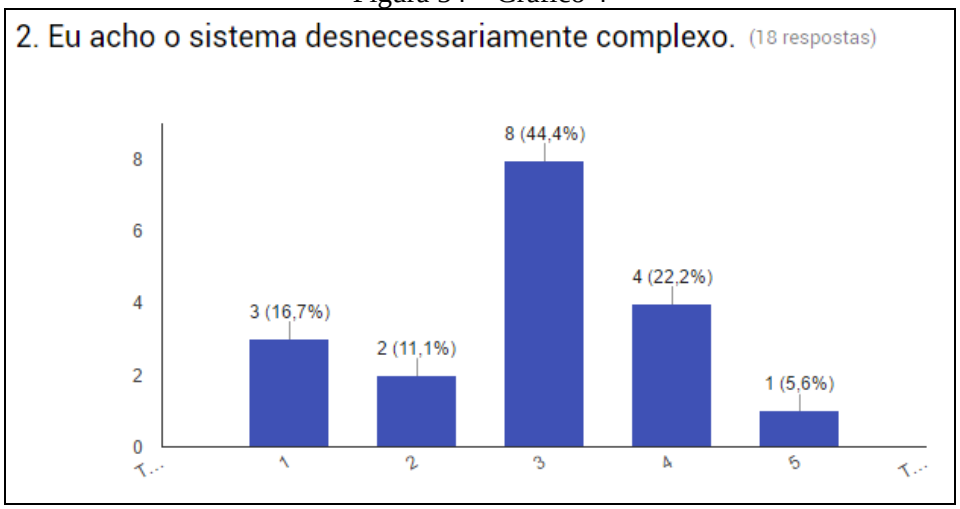


Figura 55 – Gráfico 5

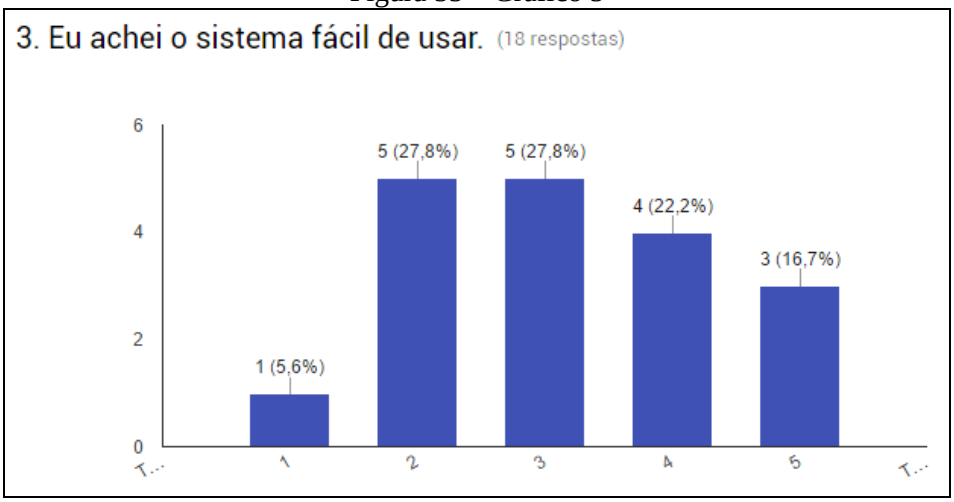


Figura 56 – Gráfico 6

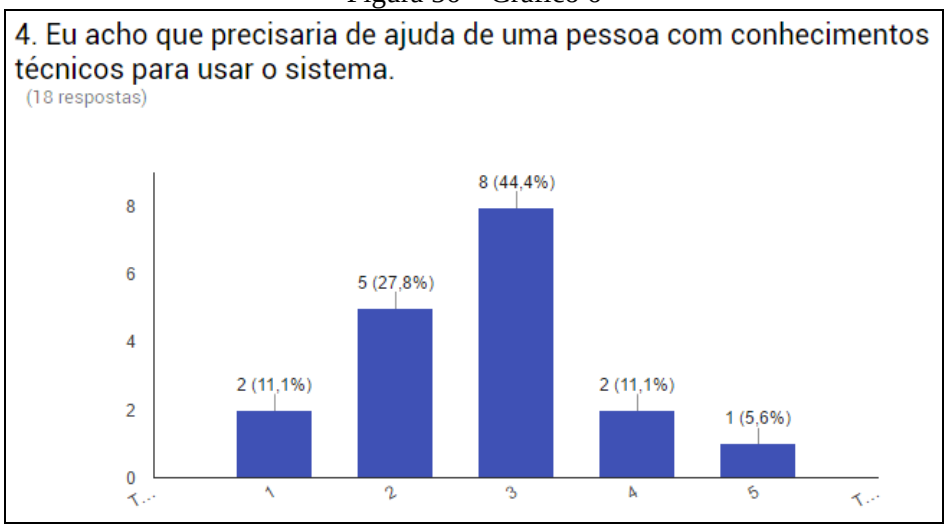


Figura 57 – Gráfico 7

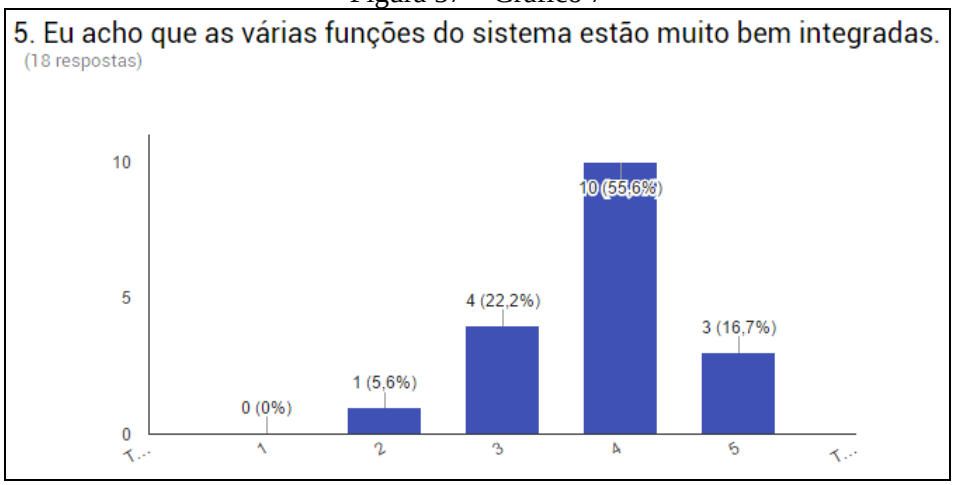


Figura 58 – Gráfico 8

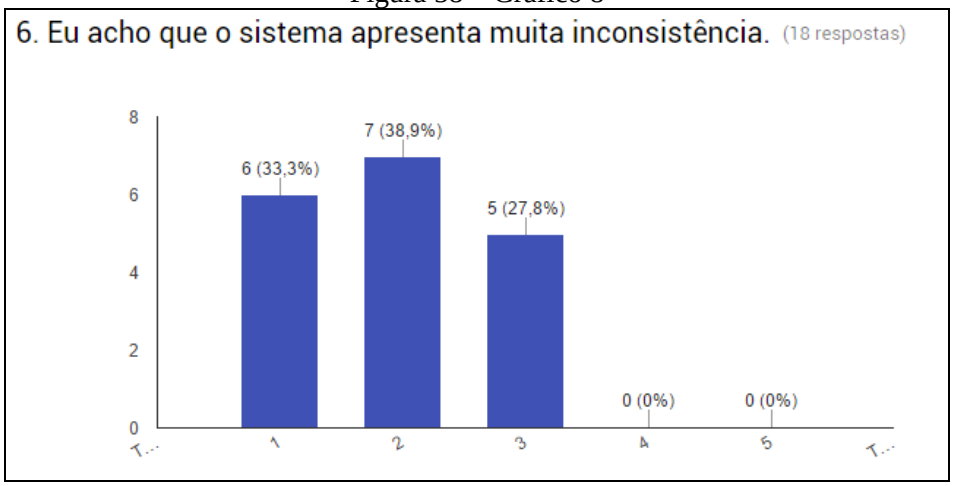


Figura 59 – Gráfico 9

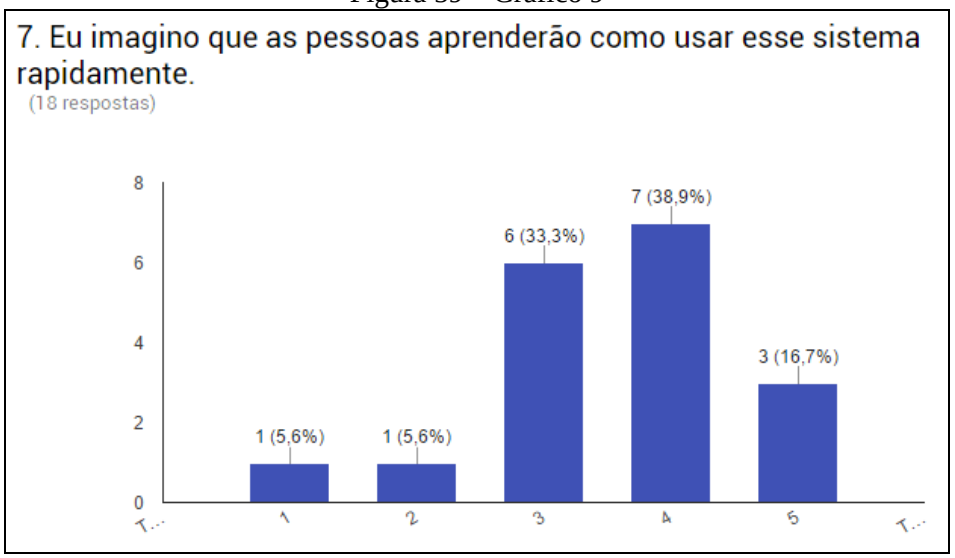


Figura 60 – Gráfico 10

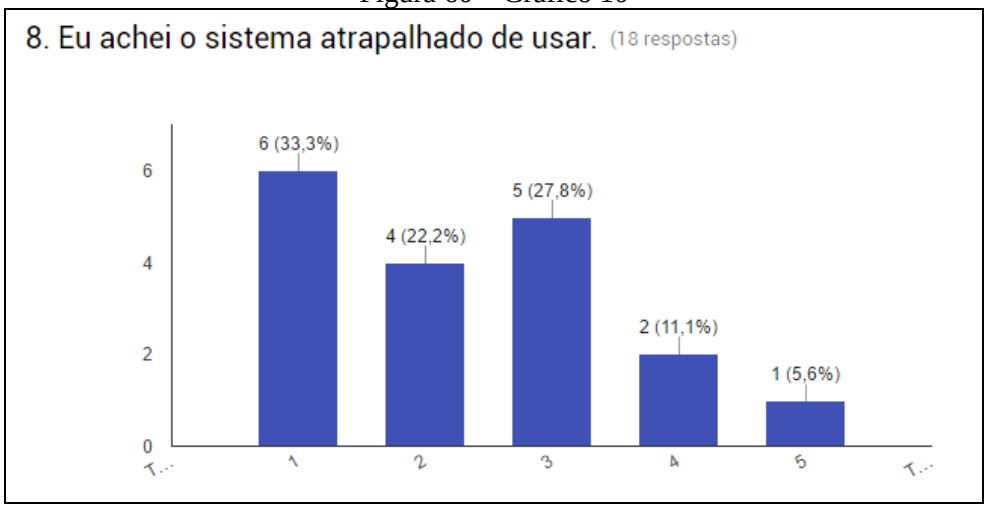


Figura 61 – Gráfico 11

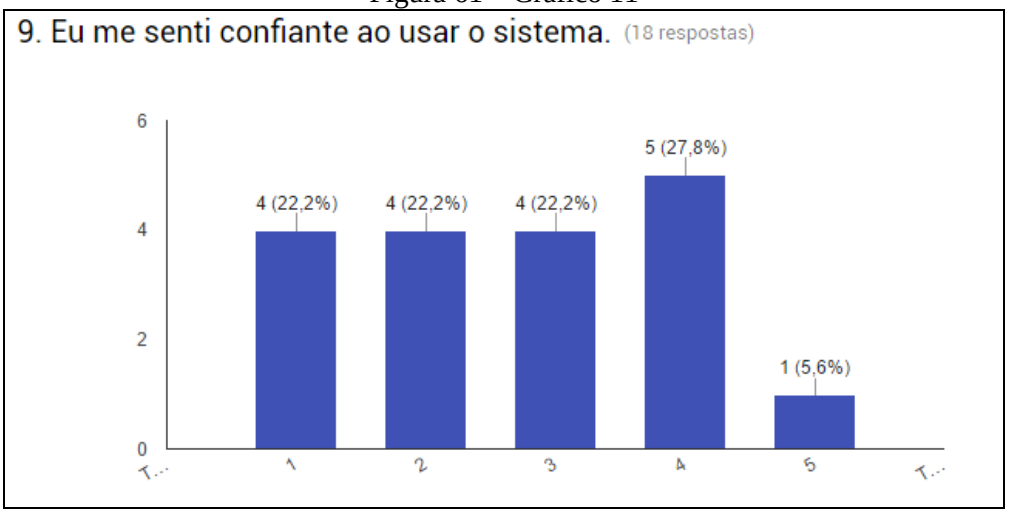


Figura 62 – Gráfico 12

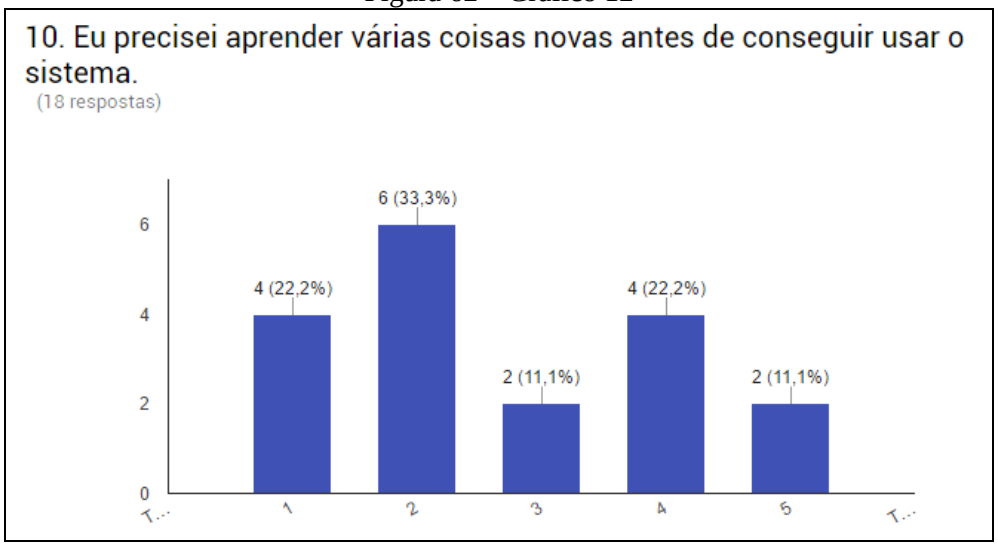


Figura 63 – Gráfico 13

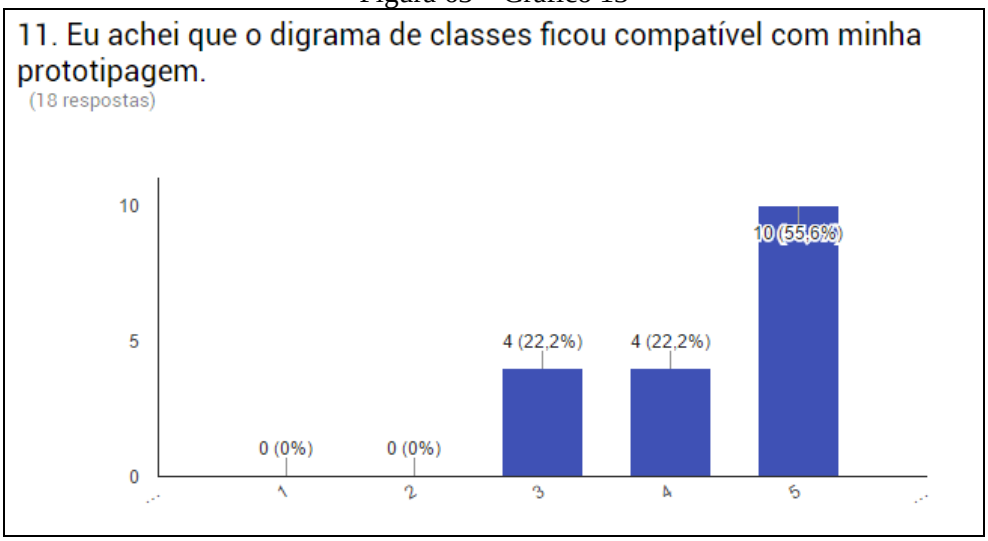
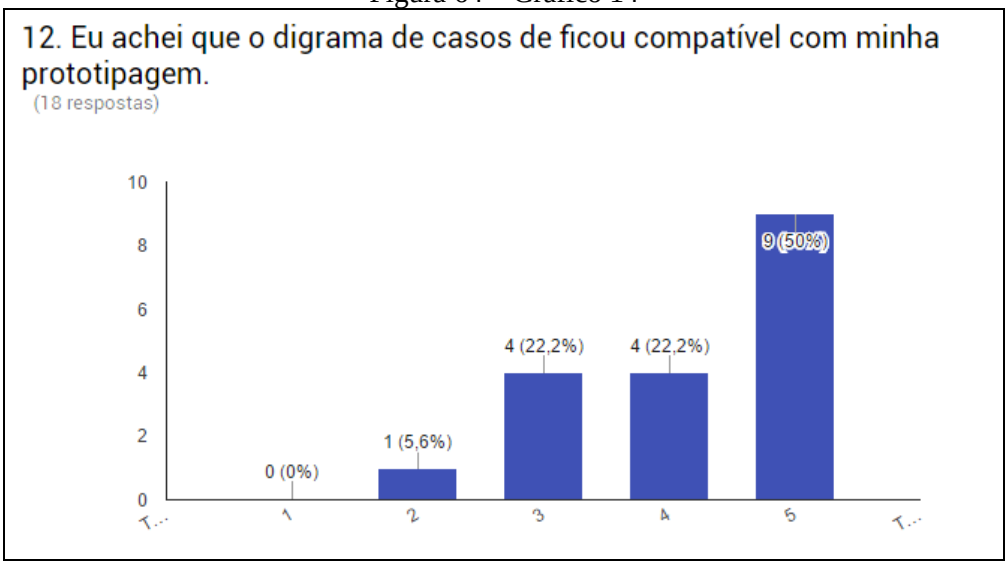


Figura 64 – Gráfico 14



Quadro 16 – Respostas da pergunta 13

13. O que você acha que é a melhor característica desta ferramenta, e por quê? (16 respostas)
A facilidade de criação, porque nem todo mundo tem muito tempo para gastar nisso então uma ferramenta que gere isso assim rapidamente é muito util
Agilidade.
Melhor visibilidade na criação de projetos.
Gerar diagramas automaticamente.
Eu só gerei a tela e ele me gerou todo o diagrama de caso de uso e classe com facilidade. Isso me economizaria muito tempo se o sistema fosse complexo, pois normalmente, você primeiro programa e depois modela.
Gera diagramas automáticamente, porque é chato fazer na mão
Facilita a criação da documentação por base da prototipagem de telas, ficando mais fácil fazer estas atividades poupando tempo.
Acelera as entregas de projetos poupando tempo em criar manualmente os diagramas e casos de usos
Gerar diagramas a partir de uma interface, porque quando for fazer um protótipo muitas vezes já tem-se a interface.
Modelar os diagramas de forma rápida e automatizada.
A facilidade de conseguir os diagramas
Praticidade
Aceitar arquivos xmi
Automatizar a etapa de casos e diagrama
Conseguir gerar os diagramas após a prototipagem automaticamente.
a facilidade de criar um diagrama

Quadro 17 – Respostas da pergunta 14

14. Qual característica desta ferramenta você acha que deve ser melhorado, e por que?
(14 respostas)

o dagrama de casos, nao ficou completamente nítido com oque eu desenhei
Nada.
Quanto a ferramenta, aparentemente é algo simples de usar, porém senti alguma dificuldade na utilização devido a falta de algumas informações no tutorial, como por exemplo salvar todo o projeto. Seguindo passo a passo o tutorial para alguem que desconhece o Delphi como eu, chegamos em alguns pontos que gera dúvidas de como proceder.
A parte de 'Gerar' poderia verificar se existe permissão na pasta, (tratar a exceção) caso não tenha informar ao usuário e já pedir um novo endereço. Outra sugestão seria gerar os diagramas utilizando somente o código-fonte, com isso seria mais fácil de levar essa mesma ideia para várias outras linguagens.
Gerar métodos a partir do código fonte também, e não somente através dos botões. Não precisar arrastar todos os componentes (atores, casos de uso e etc.) no ArgoUML seria interessante. Ambas as funções deixariam o ambiente melhor e mais fácil de se usar.
Ao invés de desenhar a tela o programa poderia identificar automaticamente no código os atributos, métodos e etc...porque desenhar tela é chato
Talvez criar a associação entre as classes no diagrama de classe, mas o maior problema na minha opinião é ter que ter um conhecimento anterior com o Delphi para poder usar bem a ferramente, e para contornar isto sugiro um manual/tutorial bem completo
Nenhuma. Ceifador Noturno aprovou tibia rules !!!!
A integração dos softwares
Arrumar pequenos probleminhas, como por exemplo no tipo do doc
Tutorial precisa ser explicado de forma mais clara e com mais detalhes.
Facilidade de uso, sem precisar explorar ferramentas do programa usado
Na primeira vez que gerei os componentes do diagrama no ArgoUML os atores apareceram corretamente, mas quando voltei ao Delphi para alterar as propriedades de classes e atributos dos objetos, gerei o .xmi novamente e no ArgoUML o ator secretaria e o caso do botao salvar nao existiam mais. Por tempo não deu para concluir os testes.
sem pontos significantes para serem melhorados

APÊNDICE C – Diagrama de Sequência do XMI

Este apêndice contém três figuras que representam um único diagrama de sequência do processo de geração de XMI da aplicação desenvolvida. Elas foram cortadas para uma melhora visualização.

Figura 65 – Sequência 1

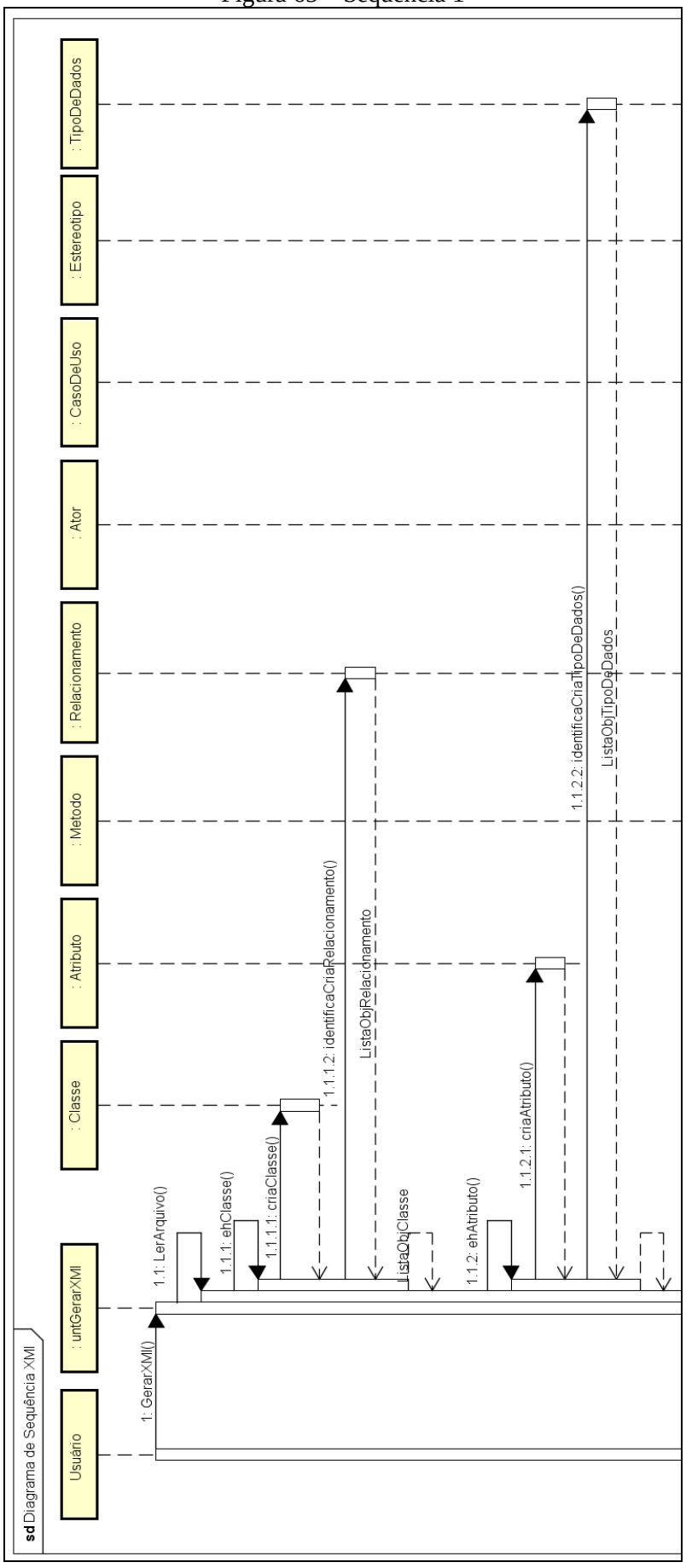


Figura 66 – Sequência 2

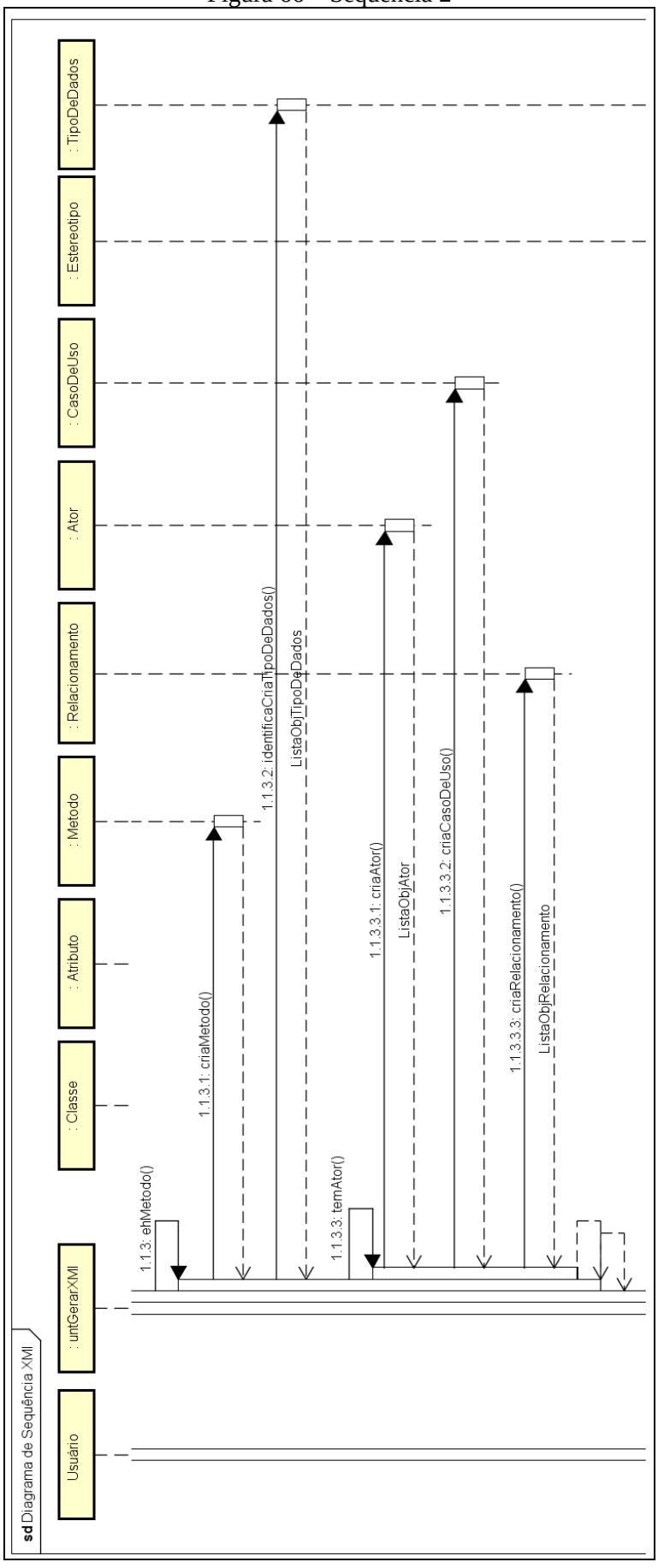


Figura 67 – Sequência 3