UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
COORDENAÇÃO DE ANÁLISE E DESENVOLVIMENTO DE SISTEMAS
CURSO DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

KAMILA TAKAYAMA LYRA

EXTENSÃO DE UM EDITOR GENÉRICO DE DIAGRAMAS PARA SUPORTE A DIAGRAMAS DA UML

TRABALHO DE CONCLUSÃO DE CURSO

CORNÉLIO PROCÓPIO
2013

KAMILA TAKAYAMA LYRA

EXTENSÃO DE UM EDITOR GENÉRICO DE DIAGRAMAS PARA SUPORTE A DIAGRAMAS DA UML

Trabalho de conclusão de curso apresentado à disciplina de Trabalho de Diplomação do curso superior de Tecnologia em Análise e Desenvolvimento de Sistemas da Universidade Tecnológica Federal do Paraná – UTFPR, como requisito parcial para a obtenção do título de Tecnólogo.

Orientador: Prof. Dr. Luciano Tadeu Esteves Pansanato.

CORNÉLIO PROCÓPIO

2013

À Miyoko Takayama Lyra e José Augusto Lyra, pais dedicados e amorosos que deram todo o apoio para minha formação.

À Marina Takayama Lyra, minha irmã sempre compreensiva com meus momentos de estresse.

A toda minha família que torce sempre por minhas conquistas.

AGRADECIMENTOS

Devo minha gratidão primeiramente a Deus, onde busquei tranquilidade e força para trabalhar.

Agradeço ao professor Luciano Tadeus Esteves Pansanato pela oportunidade e pela dedicação na orientação desse trabalho e para todos os servidores da Universidade Tecnológica Federal do Paraná, campus de Cornélio Procópio, pelo ótimo trabalho contribuindo com a formação de tantos jovens.

Agradeço ao CNPq pela bolsa de iniciação científica, e à instituição da UTFPR que disponibilizou os recursos para a pesquisa.

Gostaria de agradecer também aos colegas de turma pelos momentos de estudo e aprendizado dentro da universidade e às colegas de casa pelos momentos de aprendizado fora da universidade.

Agradeço á Filipe Gustavo Salles Lasecki, pela fé.

Agradeço especialmente à minha família, pelo apoio incondicional.

“As pessoas deficientes, qualquer que seja a origem, natureza e gravidade de suas deficiências, tem os mesmo direitos fundamentais que seus concidadãos da mesma idade, o que implica, antes de tudo, o direito de desfrutar de uma vida decente, tão normal e plena quanto possível.” (Resolução ONU Nº. 2.542/1975, item 3)

RESUMO

LYRA, Kamila Takayama. **Extensão de um editor genérico de diagramas para suporte a diagramas da UML.** 2013. 64 f. Trabalho de Conclusão de Curso (Tecnologia em Análise e Desenvolvimento de Sistemas), Universidade Tecnológica Federal do Paraná. Cornélio Procópio, 2013.

Os softwares para modelagem UML existentes não trabalham corretamente com as tecnologias assistivas básicas utilizadas por alunos com deficiências visuais. Assim, surgiu a necessidade de estender os recursos de um editor de diagramas genérico existente para criar outros diagramas da UML, mantendo a acessibilidade. A extensão permite a criação e edição dos diagramas de classes, de sequência, de atividade, de componentes e de implantação. Para realizar essa extensão foi necessário modificar o código existente e aplicar os conceitos de padrão de projeto. A finalização do software permite a inclusão de pessoas com deficiência visual nas atividades de modelagem de software usando diagramas da UML.

Palavras-chave: Interação Humano-Computador. Sistemas Interativos. Acessibilidade. Diagramas. Usuários Cegos

ABSTRACT

LYRA, Kamila Takayama. **Extension of a generic diagram editor to support UML diagrams**. 2013. 64 f. Trabalho de Conclusão de Curso (Tecnologia em Análise e Desenvolvimento de Sistemas), Universidade Tecnológica Federal do Paraná. Cornélio Procópio, 2013.

The existing UML modeling software does not work properly with basic assistive technologies used by visually impaired students. So, the need arose to extend the capabilities of an existing generic diagram editor to create other UML diagrams, keeping the accessibility. The extension allows creating and editing of class, sequence, activity, component and deployment diagrams. To accomplish this extension was necessary to modify the existing code and apply the notions of design pattern. The finalization of the software allows the inclusion of people with visual impairment in activities of software modeling using UML diagrams.

Palavras-chave: Human-Computer Interaction. Interactive Systems. Accessibility. Diagrams. Blind users

LISTA DE FIGURAS

LISTA DE QUADROS

SUMÁRIO

1. INTRODUÇÃO

1.1 CONTEXTUALIZAÇÃO

O ensino de modelagem para alunos cegos da área de computação se tornou um desafio pela própria natureza gráfica dos diagramas da UML (*Unified Modeling Language*). A principal necessidade é a independência do aluno ao interagir com os colegas, é deixar de depender de um tutor ou de material tátil de auxílio para compreender e modelar diagramas.

Segundo Fowler (2003) a *Unified Modeling Language* (UML) é uma linguagem gráfica utilizada para especificar, construir e documentar os artefatos de software e para Guedes (2009) a UML é a linguagem padrão de modelagem mais adotada pela área de engenharia de software. No entanto, as ferramentas comuns de modelagem UML têm pouca acessibilidade para tecnologias assistivas como os leitores de tela, uma ferramenta básica para um usuário de computador com deficiência visual.

O editor de diagramas denominado *GEneric Diagram Editor* (GEDE) foi desenvolvido por Silva e Pansanato (2002) e estendido por Santos (2012) para atender requisitos de acessibilidade. No entanto, a versão atual do GEDE oferece suporte apenas para o Diagrama de Casos de Uso da UML. O código da versão atual não está adequado para possibilitar a extensão para os demais diagramas da UML, por exemplo, o Diagrama de Classes e o Diagrama de Sequência.

1.2 OBJETIVO

O objetivo deste trabalho é apresentar o desenvolvimento realizado e as dificuldades encontradas para estender o editor de diagramas GEDE (SILVA e PANSANATO, 2002), para tornar possível o trabalho em outros diagramas da UML.

1.3 MOTIVAÇÃO

A UML é uma linguagem visual empregada na modelagem de sistemas computacionais que utilizem o paradigma de orientação a objetos (GUEDES, 2009). Os diagramas da UML possuem uma linguagem universal bem definida, essa linguagem se torna o meio mais rápido de comunicação entre os membros da equipe, fazendo com que diferentes *stakeholders* possam compreender e discutir o mesmo ponto de um sistema. Além disso, a documentação composta por diagramas bem consolidados torna mais rápida a compreensão para novos integrantes no projeto.

O editor de diagramas usado como foco deste trabalho é o *GEneric Diagram Editor* (GEDE), desenvolvido em 2002 por Silva e Pansanato (SILVA e PANSANATO, 2002) e estendido posteriormente com recursos de acessibilidade (SANTOS, 2012). Atualmente o GEDE não possibilita a edição de outros tipos de diagramas UML além do Diagrama de Casos de Uso.

Atualmente, a busca de acessibilidade não está mais restrita apenas à superação de obstáculos tecnológicos. Segundo Dias (2006) a inclusão de usuários cegos no campo da tecnologia está facilitada pela existência de softwares que trazem a acessibilidade para diferentes tipos de deficiência. Como exemplo mais utilizado temos os leitores de tela*, s*istemas que acessam as interfaces gráficas de outros sistemas e páginas e convertem informações textuais em áudio por meio de um sintetizador de voz. Porém, o verdadeiro obstáculo encontrado nos softwares disponíveis para aprendizagem de diagramas é que suas interfaces não são acessíveis aos leitores de tela. Esse obstáculo intensifica a necessidade de auxílio para a utilização desses recursos pelo usuário cego, tornando-o dependente de outro profissional, além de restringir a participação do aluno cego em projetos relacionados à modelagem de software.

1.4 ESTRUTURA DO TRABALHO

Este documento está dividido em seis capítulos, sendo este o primeiro, voltado á contextualizar o leitor, destacar o objetivo da escrita e descrever as principais motivações que despertaram o interesse nesse estudo.

O capítulo dois, formado por parte das pesquisas realizadas durante o desenvolvimento, apresenta conceitos necessários tanto para o desenvolvimento quanto para melhor compreensão do trabalho.

O capítulo três apresenta os recursos de hardware e software utilizados durante o desenvolvimento do trabalho além da utilização do conhecimento sobre padrões de projeto adquiridos durante o curso.

O capítulo quatro descreve como foi realizada a pesquisa e apresenta os resultados do desenvolvimento realizado.

O capítulo cinco descreve a realização do cronograma.

O capítulo seis apresenta a conclusão sobre o trabalho, mostrando as dificuldades encontradas e propostas para trabalhos futuros.

2. FUNDAMENTAÇÃO TEÓRICA

2.1 ACESSIBILIDADE E TECNOLOGIAS ASSISTIVAS

O conceito de acessibilidade define que um software deve se permitir ser acessado por pessoas com deficiências, disponibilizando maneiras de uso diferentes para cada tipo de usuário. Todos os usuários devem poder acessar as mesmas funcionalidades e atingir os mesmos resultados (DIAS, 2006).

Tornar um software acessível é remover as barreiras que impedem que um usuário com deficiência o utilize. Para remover essas barreiras, em muitos casos é necessário que as pessoas com deficiência utilizem de outros dispositivos alternativos aos mouses e teclados comuns de um computador. Esses dispositivos alternativos são chamados de tecnologias assistivas.

Conforme Hogetop e Santarosa (2001) as tecnologias assistivas são o conjunto de recursos que, de alguma maneira, contribuem para proporcionar às pessoas com deficiências um maior nível de independência, de qualidade de vida ou de inclusão social, potencializando suas capacidades.

Existe uma grande diversidade de tecnologias assistivas usadas como recursos computacionais. Provavelmente, para usuários cegos, a mais conhecida é o leitor de telas, um software que, utilizando um sintetizador de voz, realiza a “leitura” de todo o conteúdo textual da tela. O NVDA *(Non Visual Desktop Access)* (NV ACCESS, 2013*)*, o Leitor de Telas do CPqD (MINISTÉRIO DAS COMUNICAÇÕES, 2013) e o Orca (GNOME, 2005) são exemplos de leitores de tela desenvolvidos e distribuídos na forma de software livre.

Para pessoa com limitações motoras severas que impedem o uso do teclado e do mouse convencionais uma das mais simples tecnologias assistivas existentes para este público são os teclados virtuais (TANAKA, 2009). Trata-se de um teclado apresentado na tela do computador no qual há um sistema de varredura pelas teclas e o usuário deve pressionar algum tipo de acionador para selecionar a

tecla desejada. O tipo de acionador difere dependendo das necessidades e habilidades do usuário, podendo ser um pedal, um botão circular ou ainda um acionador a base de sopro.

Pessoas que possuem sua amplitude de movimentos do braço ou das mãos diminuídas necessitam de teclados reduzidos para facilitar o alcance dos botões. Assim, os teclados expandidos são direcionados para pessoas que não conseguem realizar movimentos curtos. Ainda, para usuários sem nenhuma possibilidade de movimentação existe o rastreador ocular que simplesmente move o cursor do mouse para o ponto na tela onde o usuário está olhando (MAIA, 2008).

É possível observar que cada tipo de tecnologia assistiva é direcionada para um público de usuários. Portanto, uma interface que é acessível para um usuário surdo, pode não ser acessível para um usuário cego. Adotar recomendações de acessibilidade em um projeto faz com que o produto aceite um maior número de tecnologias assitivas e assim atingir maior quantidade de usuários, pois, além de acessível, o produto se torna fácil de usar (DIAS, 2006).

2.2 AVALIAÇÃO

Avaliação é o processo ordenado de coleta de dados responsável por obter informações sobre o modo como um determinado usuário ou grupo de usuários deve utilizar um sistema para uma determinada tarefa em certo tipo de ambiente (PREECE et al., 2002). Segundo Dias (2003) os métodos de avaliação podem ser classificados em três grandes grupos: métodos de inspeção, métodos de teste com usuários e métodos baseados em modelos.

Os métodos de inspeção caracterizam-se pela não participação direta dos usuários do sistema na avaliação. Os avaliadores, especialistas em usabilidade ou desenvolvedores de sistemas, se baseiam em regras, recomendações, princípios e/ou conceitos previamente estabelecidos para identificar problemas de usabilidade relacionados à interação dos usuários com o sistema.

Os métodos baseados em modelos têm como objetivo prever a usabilidade de um sistema a partir de representações de sua interface e/ou de seus usuários. Esses métodos pretendem representar a interação dos usuários com um sistema, isto é, modelam aspectos do entendimento, conhecimento, intenções ou reações dos usuários.

Os métodos de teste com usuários caracterizam-se pela participação direta dos usuários na avaliação. Esses métodos podem ser prospectivos, como questionários e entrevistas, ou empíricos, ao adotar técnicas de observação ou monitoramento do uso do sistema em situações reais.

Atualmente existem diversos guias para desenvolvimento de software com interfaces acessíveis. Porém, são poucas as recomendações e métodos de avaliar a acessibilidade que estão voltadas para sistemas desktop; a maioria dos guias está voltada para páginas web. A seguir serão citados métodos de avaliar acessibilidade ou guias de desenvolvimento para softwares acessíveis que podem ser aplicados em sistemas desktop (não web).

- Em Método baseado em heurísticas para avaliação de acessibilidade em sistemas de informação (TANAKA, 2009), o autor propõe, com base em um estudo, cinco heurísticas de acessibilidade. Assim como é feito com as heurísticas de usabilidade de Nielsen (NIELSEN, 1994), essa tese propõe que a avaliação de acessibilidade de um software deve ser feita por um grupo de avaliadores que tenham como objetivo interagir com o software e encontrar erros de acessibilidade. Esses erros devem estar associados a uma ou mais heurísticas de acessibilidade propostas. Pode ser usado em sistemas web ou desktop.
- A IBM disponibiliza diversos *checklists*, listas de verificação, direcionadas para o desenvolvimento de software. A *Software Checklist* (IBM, 2013) fornece recomendações para criar sistemas com maior acessibilidade, além de informações sobre ferramentas

para testes de certos pontos da acessibilidade. A *Documentation Checklist* (IBM, 2012) fornece uma lista de pontos que devem ser notados ao criar a documentação de um sistema ou de um site.

- Um trabalho mais antigo do Instituto de Tecnologia da Geórgia (FAIN, 2001) oferece um guia para conduzir uma validação de acessibilidade. O texto aconselha desde definir os objetivos da validação, até montar sua própria lista de verificação com os pontos principais que devem ser validados pelo usuário.
- A organização TIRESIAS (2009) dispõe de mais de quarenta listas de verificação direcionadas para sistemas de tecnologia da informação e comunicação. Alguns exemplos são: Lista de verificação para Aplicações de Software, Lista de verificação para Acessibilidade Web, Lista de verificação para Telas Sensíveis ao Toque, Lista de verificação para Fontes. Cada ponto abordado na lista é classificado de acordo com o tipo de usuário que afeta e com o nível de importância para esse usuário. Esse foi o método escolhido para ser aplicado e avaliar a acessibilidade da extensão desenvolvida neste trabalho. O resultado está descrito na seção 4.3.
- Está disponível para desenvolvedores de aplicativos para a *Windows Store* em *JavaScript* e HTML, um texto de recomendações que descreve o procedimento a ser seguido para testar a acessibilidade do produto. A base do procedimento é a execução de ferramentas de análise de código, para então corrigir os erros apontados como prioridade 1 pela ferramenta (MICROSOFT, 2013).
- Alves (2011) propõe além de uma lista para verificação da acessibilidade em software livre, uma pesquisa de opinião com usuários portadores de diferentes tipos de necessidades especiais.

2.3 UML

Segundo GUEDES (2009), a UML (*Unified Modeling Language*) é uma linguagem visual utilizada para modelar softwares baseados no paradigma de orientação a objetos. A UML se tornou, nos últimos anos, a linguagem padrão de modelagem adotada internacionalmente pela indústria de software.

O objetivo da linguagem é auxiliar os engenheiros de software na definição das características do sistema. Cada diagrama detalha um nível de características, por exemplo, requisitos, comportamento, estrutura lógica de execução e necessidades físicas de planejamento.

Os diagramas UML são artefatos de extrema importância gerados na fase de projeto do ciclo de desenvolvimento de um software. Projetar um sistema, mesmo um muito simples, deixa claro o quanto implementar antes de modelar pode ser arriscado para o sistema. Projetos mal modelados costumam crescer em complexidade ao longo do desenvolvimento. O crescimento está relacionado também com as mudanças que surgem com a evolução do mercado nas necessidades dos clientes. As correções necessárias podem produzir novos erros se a documentação não estiver detalhada e sempre atualizada.

A comunicação constitui um dos principais desafios da engenharia de software. Cada envolvido possui uma dificuldade diferente em compreender os conceitos abstratos que são as necessidades dos clientes, e transformá-los em conceitos concretos através da modelagem. A padronização gerada pela UML aproxima a curva de entendimento dos envolvidos, assim seu objetivo não é apenas diminuir a necessidade de manutenção do software, mas também abater esforços na compreensão de códigos feitos com estilos de desenvolvimento diferentes.

O objetivo da variedade de diagramas existente na UML é fornecer múltiplas visões do projeto. Cada visão está em um nível de detalhamento do projeto, umas superficiais, como o diagrama de caso de uso, outras mais

detalhadas, como o diagrama de sequência. A união dessas visões deve se completar e fornecer uma visão ampla da modelagem.

2.4 ENSINO DE UML COM O USO DE TECNOLOGIAS ASSISTIVAS

O software desenvolvido neste trabalho tem como foco os usuários com deficiência visual. Esse termo se refere à diminuição irreversível da capacidade visual, em diferentes graus. Esses usuários são os que têm o maior grau de dificuldade em acessar conteúdo digital, pela própria natureza gráfica do conteúdo, assim como dos diagramas e das ferramentas utilizadas para a edição de diagramas.

Existem outros estudos sobre sistemas computacionais que têm o objetivo de ilustrar uma informação gráfica a um usuário cego. Esses trabalhos adicionam à interface a utilização de áudio ou de dispositivo tátil e de força. Esses casos específicos exigem colaboração especializada na construção dos modelos táteis, além de, não permitirem o desenvolvimento e edição dessas informações, apenas apresentarem seu conteúdo (KENNEL, 1996; METATLA et al., 2006; ROTARD et al., 2005; KING et al., 2004; HORSTMANN et al., 2004). O mais alto nível de interação entre o usuário e o diagrama foi encontrado nos casos em que existe a possibilidade de exploração do diagrama (BROWN et al., 2004; COHEN et al., 2006; BLENKHORN; EVANS, 1998; HORSTMANN et al., 2004).

Como alternativa, têm sido utilizadas as tecnologias assistivas. Essas tecnologias auxiliam os usuários com deficiência visual no acesso ao conteúdo digital que está na tela do computador. São programas que transformam o conteúdo em Braille, ou aumentam o tamanho das fontes ou imagens, ou leem a parte do conteúdo que é textual com uma voz sintetizada. Estes últimos costumam ser os mais utilizados.

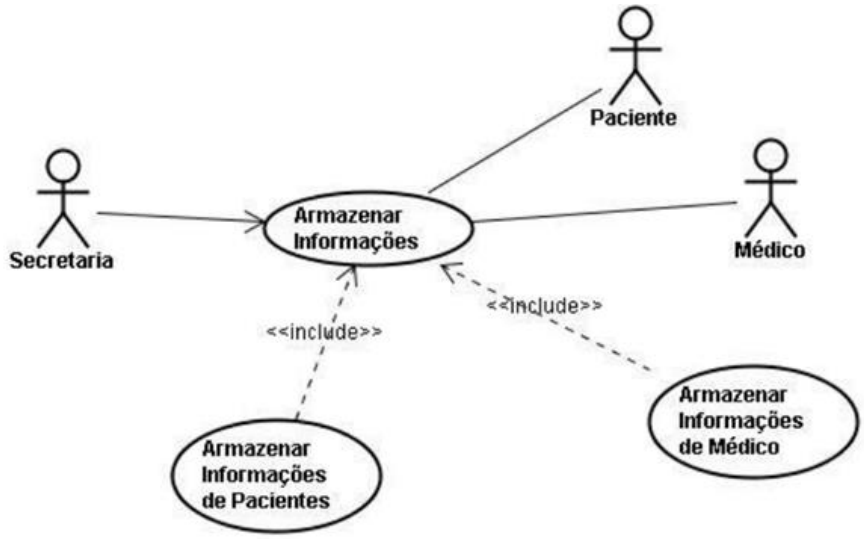
Um software leitor de tela exige o controle do teclado ou do mouse ou de outra tecnologia assistiva para poder navegar pelos botões e textos da tela. Ele

devolve para o usuário uma descrição sonora da posição do foco, ou seja, do elemento que está em foco, um botão, um texto, um menu, etc.

A natureza dos diagramas deixa claro que a UML não é uma linguagem de programação e sim uma linguagem de modelagem visual. A interação dos usuários com as ferramentas atuais mais usadas para construção de diagramas UML é basicamente visual. Para os alunos com deficiência visual na área da computação o aprendizado da UML se torna um desafio, tanto quanto é para os professores realizarem esse ensino.

O principal problema é a falta de acessibilidade encontrada nas ferramentas de modelagem UML, a leitura dos diagramas feita pelos sintetizadores de voz não é correta e muitas vezes nem executada. Em 2010, na Universidade Tecnológica Federal do Paraná, foi desenvolvido um método de representar de forma alternativa o diagrama de Caso de Uso da UML (SILVA et al, 2010). Essa representação possibilita a um estudante cego entender as associações sem o auxílio de outra pessoa ou de algum equipamento especial.

No exemplo apresentado a seguir, Figura 1, o Diagrama de Casos de Uso é convertido em uma tabela, onde é necessário incluir informações textuais equivalentes às descrições dos nós e arestas do diagrama. Os leitores de tela têm acesso total aos elementos textuais presentes na tabela. Além disso, a navegação e exploração da tabela por meio do teclado são bem conhecidas pelos usuários.



A	B	C	D	E
Ator	Caso de Uso	Include	Extend	Ator
Secretaria	Armazenar Informações			Paciente
				Médico
		Armazenar Informações de Pacientes		
		Armazenar Informações de Médicos		

Figura 1 – Conversão de diagrama para tabela
Fonte: SILVA (2010)

Apesar dessa nova representação, em forma de tabelas, facilitar o aprendizado do aluno com deficiência visual, uma vez que desperta o mesmo grau de capacidade em elaborar modelos que é despertado pela notação gráfica, essa representação ainda possui baixa agregação quanto à interação entre o estudante cego e o de visão normal, pois este último deve se tornar capacitado na compreensão das duas formas de representação, a gráfica e a em forma de tabela.

2.5 GEDE (GEneric Diagram Editor)

Este trabalho descreve a extensão desenvolvida para uma versão do editor GEDE, desenvolvido por Silva e Pansanato (2002). Em 2012 foi adicionada uma extensão de acessibilidade ao software. Essa extensão adicionou recursos de navegação facilitada pelo diagrama utilizando apenas o teclado (SANTOS, 2012). Atalhos para toda a interface podem ser acessados por meio das combinações de diferentes teclas. Os diferentes modos de explorar o diagrama permitem a navegação por grupo de elementos, minimizando a questão da complexidade dos

diagramas. A navegação padrão se assemelha ao estilo de navegação encontrado nos navegadores web, onde a tecla Tab passa o foco para o próximo elemento e a combinação Shift + Tab foca o elemento anterior. A Figura 2 mostra um exemplo da interface exibindo um Diagrama de Casos de Uso.

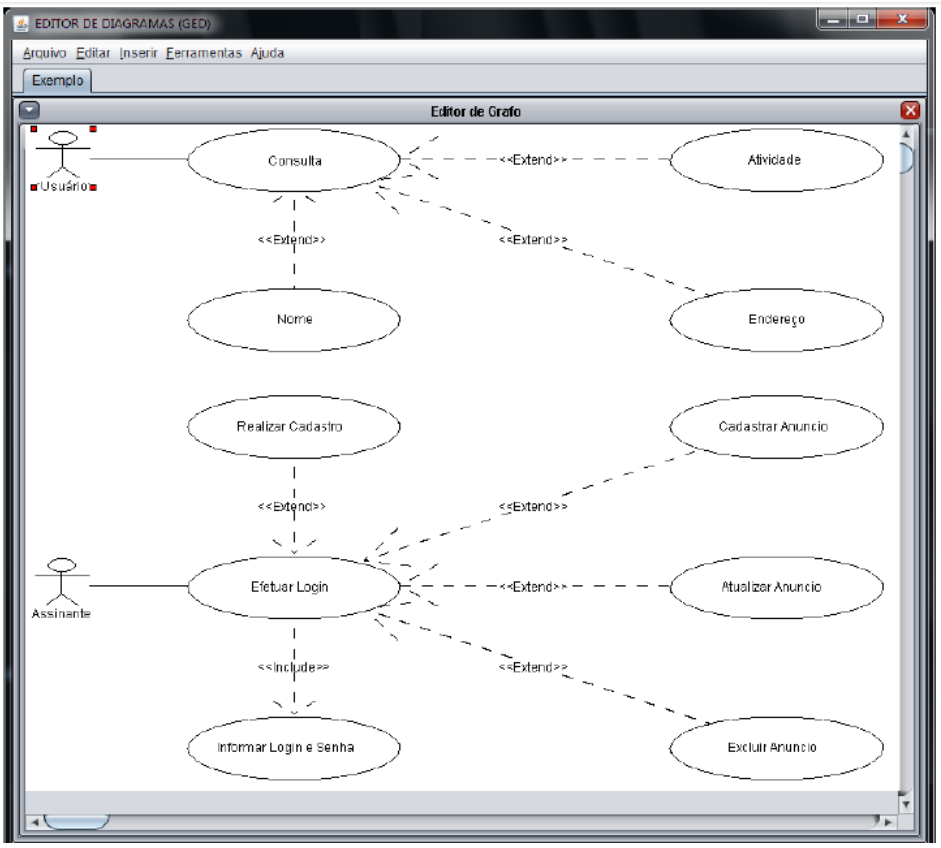


Figura 2 – Interface do editor GEDE
Fonte: SANTOS (2012)

O GEDE está desenvolvido em linguagem Java. Apesar do bom funcionamento do editor na criação do Diagrama de Casos de Uso, o modo como o código se encontra organizado dificulta a extensão para os outros diagramas UML.

A Figura 3 e a Figura 4 mostram uma abstração do diagrama de classes contendo apenas as classes mais importantes para compreender a estrutura do código. A classe *Menu* utiliza o método *AdicionarAbas()* da classe GUI para criar novas abas de diagramas de Casos de Uso. A aba instanciada é composta por um objeto da classe *Grafo* com seus objeto da classe *EditorAplicativoGrafo*, responsável

pela área onde será desenhado o diagrama. A classe *Grafo* também mantém uma lista para cada tipo de vértice existente na aba. Na versão atual somente é possível criar diagramas de Casos de Uso, logo, só existem duas listas de vértices na classe GUI, a *ListaAtor* e *a ListaCasoUso*.

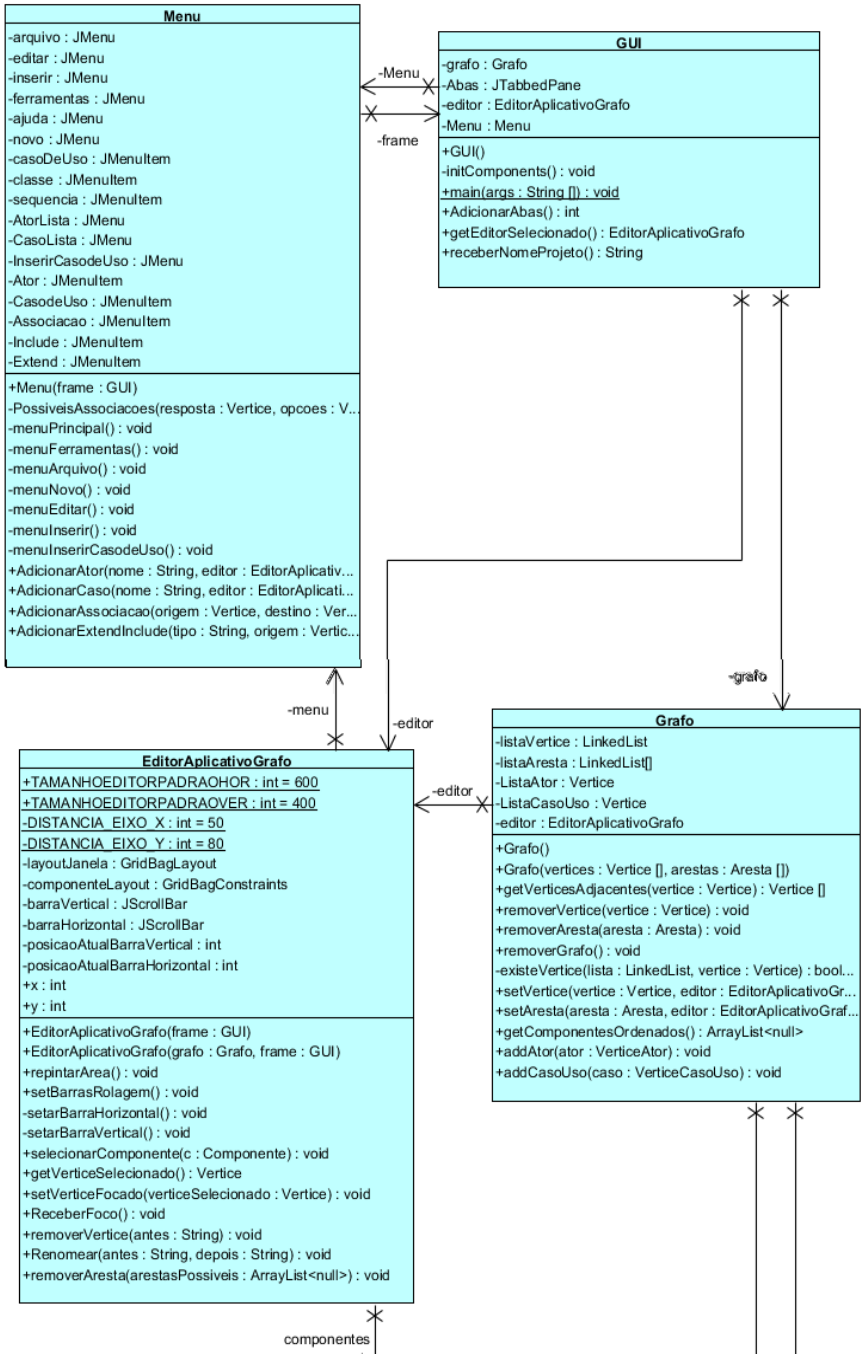


Figura 3 - Relação com a classe *Menu*
Fonte: Autoria própria

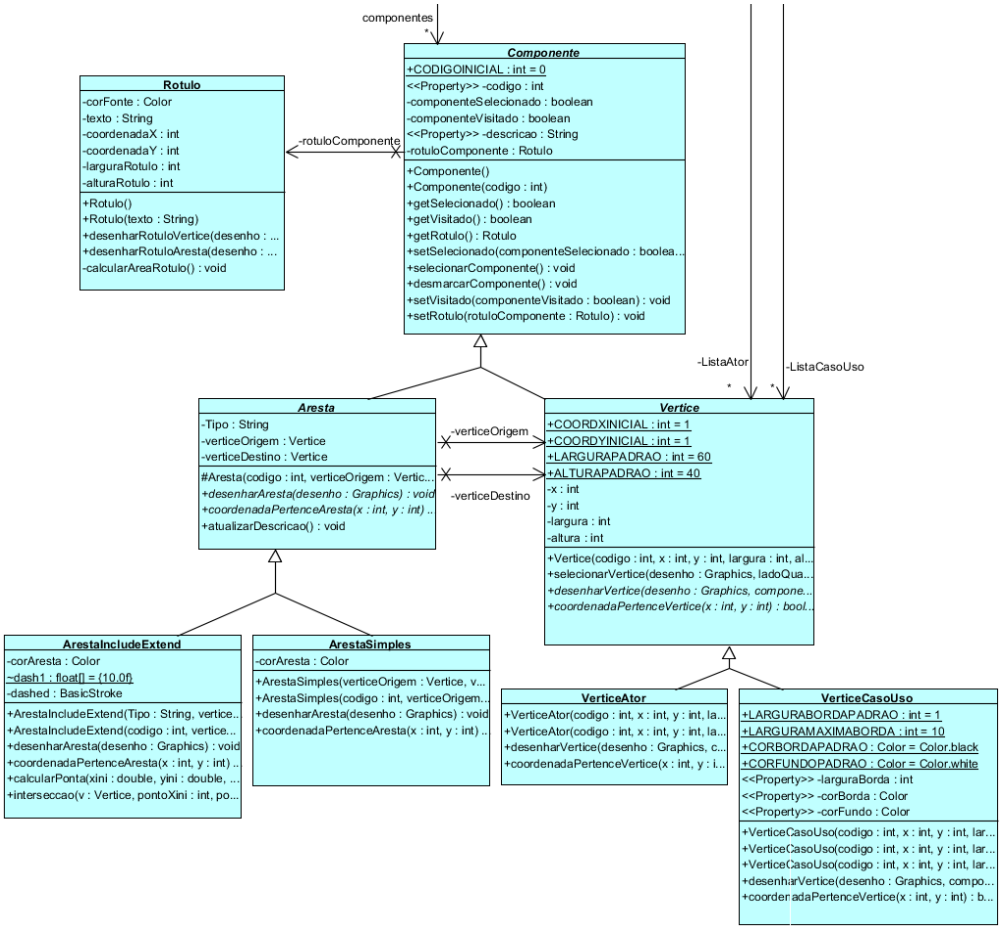


Figura 4 - Hierarquia de componentes
Fonte: Autoria própria

A Figura 4 mostra as classe *Aresta* e *Vertice* que são herdeiras da classe *Componente*, todo componente de um diagrama deve ter um rótulo para ser lido pelo leitor de tela. Os tipos de vértice de um diagrama de Casos de Uso herdam da classe *Vertice*, do mesmo modo, os tipos de aresta herdam da classe *Aresta*.

A classe *Menu* é responsável por instanciar os itens de menu e definir as ações para cada item. O principal item é o *JMenuItem casoDeUso*. O clique nesse item utiliza o método *AdicionarAbas()* da classe principal GUI, como no código do Quadro 1, para adicionar uma aba com a área para o desenho do diagrama na interface do GEDE.

```
private GUI frame;
[...]
casoDeUso.addActionListener(new ActionListener() {
      @Override
      public void actionPerformed(ActionEvent e) {
            int AdicionarAbas = frame.AdicionarAbas();
            if(AdicionarAbas==1) {
                  InserirCasodeUso.setVisible(true);
                  Navegacao.setVisible(true);
            }
      }
});
```

Quadro 1 - Chamada do método *AdicionarAbas()*
Fonte: SILVA (2012)

Outra ação importante é a de adicionar componentes, como um ator ou uma associação, ao diagrama de Casos de Uso. Essas ações também estão definidas dentro da classe *Menu*.

Os itens *Ator* e *CasodeUso* são responsáveis por adicionar vértices ao diagrama. O código executado quando o usuário escolhe o item *Ator* está no Quadro 2. No caso de existir mais de uma aba de diagrama de Casos de Uso é necessário saber em qual delas será colocado o novo ator, isso é feito utilizando o método *getEditorSelecionado()* da classe GUI. Após receber um nome válido para o vértice é chamado o método *AdicionarAtor()* escrito também na classe *Menu*. A execução do código para o item *CasodeUso* é semelhante é semelhante à do item *Ator*. No entanto, ao invés do método *AdicionarAtor()* é utilizado o método *AdicionarCaso().*

```
Ator.addActionListener(new ActionListener() {
@Override
public void actionPerformed(ActionEvent e) {

      EditorAplicativoGrafo editor = frame.getEditorSelecionado();
      String nome="";

      while ("".equals(nome)) {

            nome = JOptionPane.showInputDialog(editor,"Digite o
            rótulo do ator?","Digite o rótulo do ator?",1);

            if ("".equals(nome))  {
                  JOptionPane.showMessageDialog(editor, "Rótulo
                  Inválido");
            }
      }
      AdicionarAtor(nome, editor, 1);
}

});
```

Quadro 2 - Chamada do método *AdicionarAtor()*
Fonte: SILVA (2012)

O método *AdicionarAtor(),*Quadro 3, é responsável pela instanciação do vértice *VerticeAtor*, definindo pela passagem de parâmetros a posição *x* e *y* em que o desenho será colocado no Grafo, e a largura e altura do desenho do vértice ator. A instanciação do vértice caso de uso segue a mesma estrutura.

```
public void AdicionarAtor(String nome, EditorAplicativoGrafo
editor, int converter){
      Grafo grafo = editor.getGrafo();
      [...]
//Parametros:codigo, posicaox, posicaoy, largura, altura, nome
      VerticeAtor v1 = new VerticeAtor(1, editor.x, editor.y, 50,
50, nome);
       [...]
       //Adiciona o novo ator em uma lista de atores
       grafo.addAtor(v1);
       [...]
}
```

Quadro 3 - Instanciando um vértice do tipo *VerticeAtor*
Fonte: SILVA (2012)

Para a adição de arestas no diagrama de Casos de Uso é necessário saber qual será o vértice origem e o vértice destino da aresta. O programa exibe

uma lista de vértices para associar e recebe a reposta do usuário. Os vértices escolhidos são passados por parâmetro para o método *AdicionarAssociacao()* ou para o método *AdicionarExtendInclud()*, dependendo do tipo de aresta que foi escolhida pelo usuário. A escolha dos vértices e a chamada do método *AdicionarAssociacao()* é mostrada no Quadro 4.

```
Associacao.addActionListener(new ActionListener() {
@Override
public void actionPerformed(ActionEvent e) {
      EditorAplicativoGrafo editor = frame.getEditorSelecionado();
      Grafo grafo = editor.getGrafo();

      Vertice resposta;
     Vertice resposta2;
     Vertice[] opcoes = grafo.getTodosVertices();

      resposta = (Vertice) JOptionPane.showInputDialog(editor,
      "Escolha o vertice origem?", "Escolha o vertice origem?",
      JOptionPane.PLAIN_MESSAGE, null, opcoes, "");

     [...]
     Vertice[] opcoes2 = PossiveisAssociacoes(resposta,opcoes);

      resposta2 = (Vertice) JOptionPane.showInputDialog(editor,
"Escolha o vertice destino?", "Escolha o vertice destino?",
JOptionPane.PLAIN_MESSAGE, null, opcoes2, "");

      [...]
     AdicionarAssociacao(resposta, resposta2, editor, 1);
}
});
```

Quadro 4 - Chamada do método *AdicionarAssociacao()*
Fonte: SILVA (2012)

No método *AdicionarAssociacao()* é instanciado o tipo de aresta desejada, no caso uma *ArestaSimples*, passando os vértices origem e destino, como mostrado no Quadro 5. É obrigatório para toda aresta ter um vértice origem e um vértice destino. O mesmo ocorre para o método *AdicionarExtendInclud()*, que também está na classe Menu.

```
public void AdicionarAssociacao(Vertice origem, Vertice destino,
EditorAplicativoGrafo editor, int converter) {
        Grafo grafo = editor.getGrafo();

        ArestaSimples a1 = new ArestaSimples (origem,destino);

         grafo.setAresta(a1,editor);

        [...]
}
```

Quadro 5 - Instanciando uma aresta do tipo *ArestaSimples*
Fonte: SILVA (2012)

3. TECNOLOGIAS UTILIZADAS

Neste capítulo são descritos os recursos de hardware e software que foram pesquisados, aprendidos, utilizados e explorados para a melhor realização do trabalho.

3.1 RECURSOS DE HARDWARE

- Alto-falantes Realtek para o uso do leitor de tela.

3.2 RECURSOS DE SOFTWARE

- Editor de diagramas Visual Paradigm for UML Community Edition. Version 10.1
- Ambiente de Desenvolvimento NetBeans 7.2.1

3.2.1 *NVDA*

NVDA *(NonVisual Desktop Access)* é uma tecnologia assistiva criada por Michael Curran e James Teh (NV Access, 2013). Desenvolvida em 2006, essa tecnologia é o primeiro leitor de tela gratuito para computadores que utilizam o sistema operacional Windows. O NVDA foi traduzido para mais de 40 línguas e é usado em mais de 120 países. A expansão e melhoramento do NVDA têm contínua contribuição de desenvolvedores de todo o mundo, pois é uma tecnologia de código aberto.

Esse software pode ser instalado diretamente no computador do usuário ou executar totalmente a partir de um dispositivo USB ou qualquer outro dispositivo portátil. O NVDA acessa a maioria dos programas mais populares, como navegadores, *e-mails* e programas do pacote Microsoft Office. O sintetizador de voz anuncia automaticamente o texto que se encontra sob o ponteiro do mouse ou o que

recebe o foco através da navegação pelo teclado. O NVDA também pode ser configurado para ler a formatação do texto, como o estilo, tamanho e tipo de fonte, e também anunciar por meio de um sinal sonoro a posição do mouse na tela. Na Figura 5 é mostrada a interface inicial do NVDA; assim que é iniciado, o sintetizador de voz lê as instruções iniciais para o usuário.

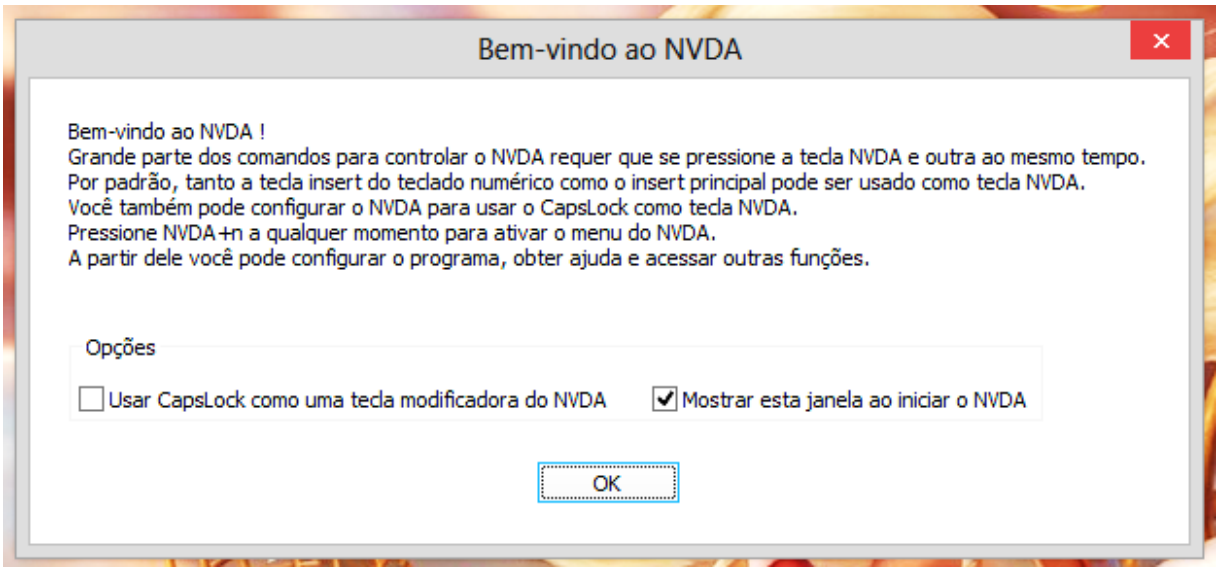


Figura 5 - Interface inicial do NVDA
Fonte: NV Access (2013)

O NVDA foi escolhido para testar se a acessibilidade que existia no editor de diagramas continuou presente na extensão para os outros diagramas adicionados. O principal motivo é que o software suporta o acesso às *APIs* de acessibilidade, como *Java Access Bridge*, utilizado no desenvolvimento do editor. O link para realizar o download pode ser encontrado no site oficial[1].

3.2.2 *API de Acessibilidade Java*

Para tornar o editor acessível aos leitores de tela, foram usados os recursos da *Java Accessibility API* que faz parte da *Java Accessibility Utilities*. Essa *API*, nativa da linguagem, provê os recursos necessários para que as tecnologias assistivas possam ter acesso às aplicações Java.

[1] www.nvaccess.org

Todos os componentes de um software que desejam suportar a *Java Accessibility API* devem implementar sua interface *Accessible*. Seu único método, *getAccessibleContext()*, devolve uma instância da classe *AccessibleContext* que fornece as informações sobre os componentes da interface Java (descrição e texto) para a tecnologia assistiva.

Neste trabalho a API de acessibilidade foi utilizada para garantir o acesso dos leitores de tela, principalmente aos elementos dos diagramas da UML e aos menus do editor. A seguir, nas Figura 6 e Figura 7, são apresentados dois exemplos que ilustram apenas as classes, métodos e atributos que são necessários para a compreensão da utilização da API. No pacote *javax.accessibility* se encontra a interface *Accessible* que é implementada pela classe *JMenuBar*, nativa da linguagem Java. A classe *Menu,* criada para instanciar os itens de menu do editor, define seu nome acessível em seu construtor através do método *setAccessibleName().* Esse método se encontra na classe *AccessibleContext*, uma instância dessa classe é retornada quando o método *getAccessibleContext()* é chamado. Desse modo quanto um item de menu receber o foco do mouse o leitor de tela consegue descrever esse item acessando o atributo *accessibleName*.

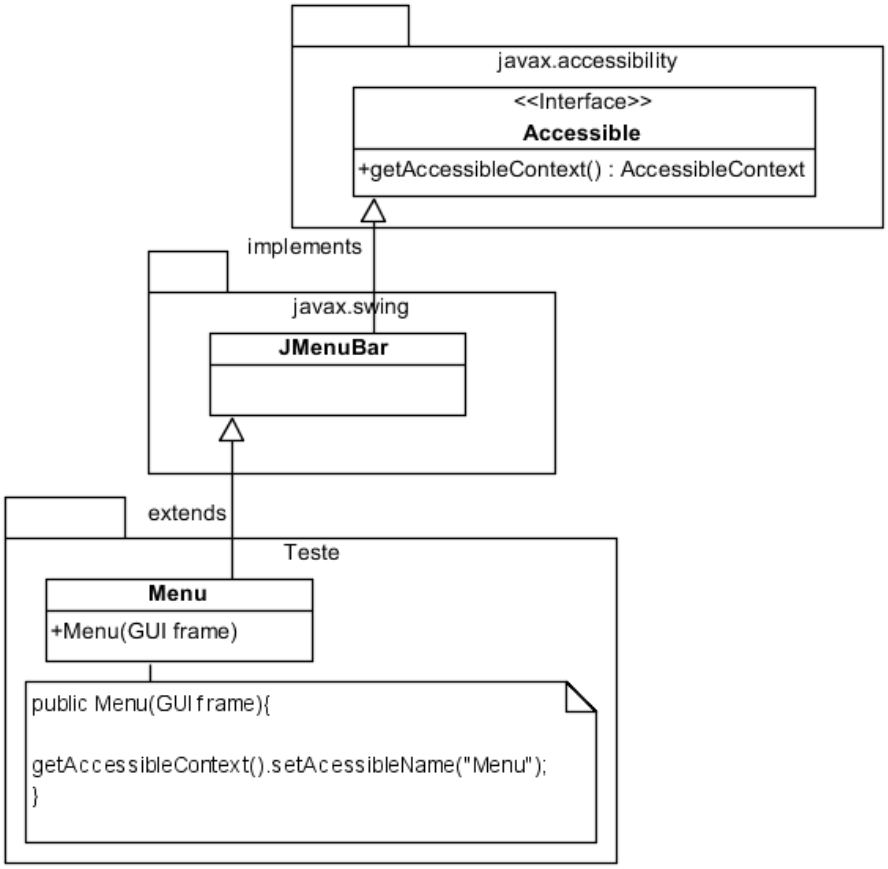


Figura 6 - Diagrama de classe para acessibilidade no menu
Fonte: Autoria própria

A acessibilidade para os elementos do diagrama UML foi implementada segundo o diagrama simplificado na Figura 7.

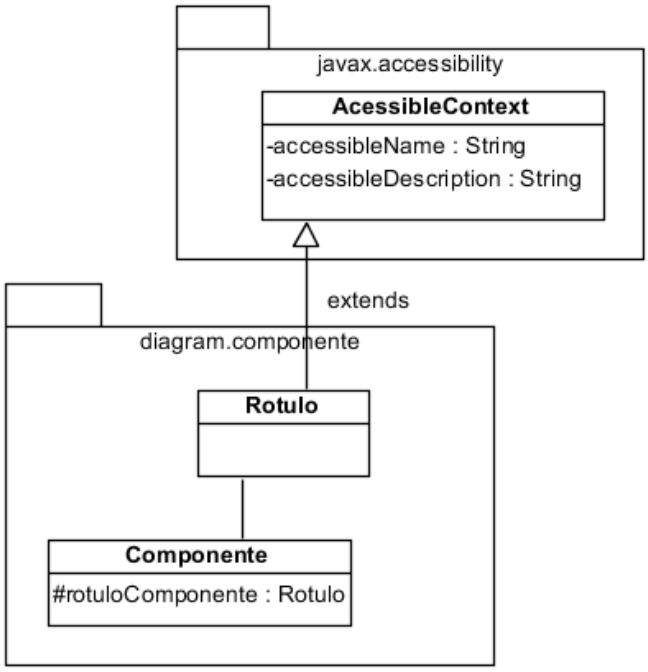


Figura 7 - Diagrama de classe para acessibilidade nos componentes
Fonte: Autoria própria

Cada componente do diagrama, por exemplo, um ator ou uma classe possui um rótulo; esse rótulo é o próprio contexto acessível do componente. Ao invés de implementar a interface *Accessible*, a classe *Rotulo* estende da classe abstrata *AccessibleContext.*

3.2.3 *Java Access Bridge*

Java Access Bridge é a tecnologia que faz a comunicação entre a API de Acessibilidade Java e a tecnologia assistiva utilizada. Uma tecnologia assistiva em um sistema operacional Windows, como o leitor de tela NVDA, se comunica com as DLLs da Java Access Bridge que, por sua vez, troca informações com a Máquina Virtual Java. A Figura 8 ilustra a ordem de comunicação entre os recursos utilizados.

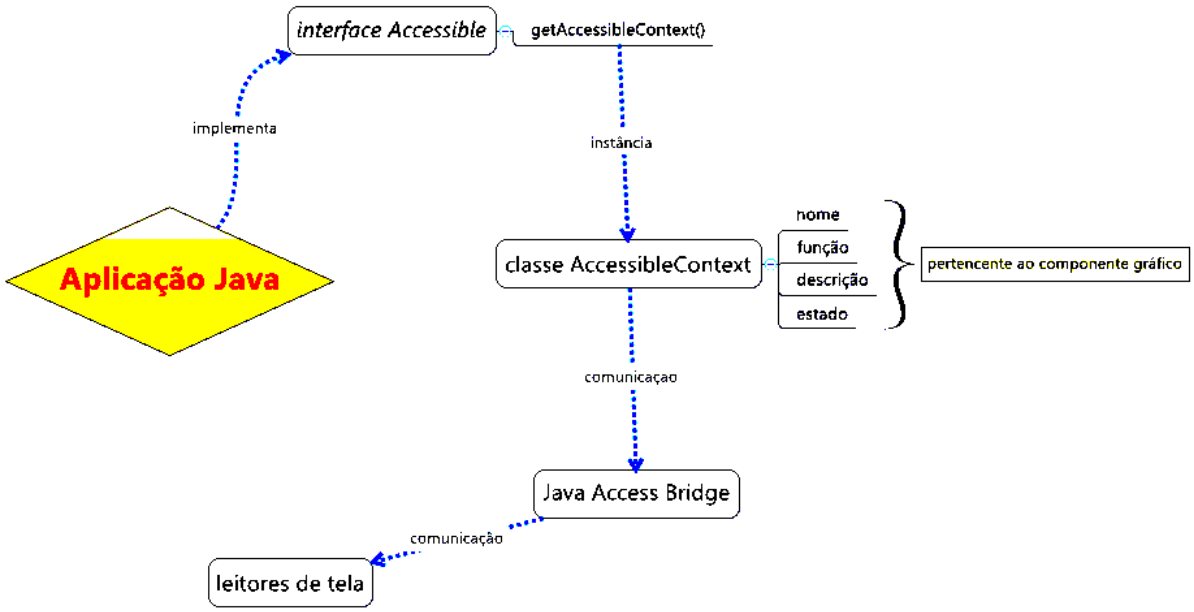


Figura 8 – Mapa mental da comunicação implementada
Fonte: Autoria própria

A *Java Access Bridge* fornece um subconjunto da API de Acessibilidade Java como uma DLL do Windows, a *WindowsAccessBridge.dll*. As tecnologias assistivas no sistema operacional Windows carregam e se ligam a esta DLL. A *Java Access Bridge* também oferece um par de DLLs, *JavaAccessBridge.dll* e *JAWTAccessBridge.dll*, que são carregados em tempo de execução. Este par de

DLLs se comunica com o aplicativo através da *Java Accessibility API* e da *Java Accessibility Utilities*, uma coleção de classes que fornecem acesso para tecnologias assistivas. O componente *access-bridge.jar* é um conjunto de classes que gerencia a comunicação entre as DLLs e o código Java carregados em tempo de execução. Esse arquivo JAR é carregado através do arquivo *accessibility.properties* (ORACLE, 2011).

O resultado dessa comunicação é o áudio descritivo dos elementos que compõe do diagrama UML em que se está navegando, provocando a construção da imagem mental dos conceitos de maneira semelhante ao que ocorre quando a leitura é tátil.

3.3 PADRÕES DE PROJETO

Considerando que este projeto foi realizado com base na programação em linguagem Java, uma linguagem orientada a objetos, e que ao realizar o estudo do código do editor, a fim de implementar as extensões, surgiu a necessidade de modificação e organização do código. Nessas modificações foram utilizados os conceitos sobre Padrões de projeto.

Padrões de projeto são descrições de conjuntos de classes e objetos que precisam ser personalizados para resolver um problema geral de projeto num contexto particular (GAMMA, 2000). Assim, um padrão de projeto abstrai os aspectos-chave da estrutura de um projeto para torná-la reutilizável em outros, identificando classes e instâncias com seus papéis e responsabilidades, ajudando a projetar sistemas, independente da linguagem que será utilizada. Esses padrões resolvem problemas específicos de projetos orientados a objetos, tornando-os mais flexíveis e reutilizáveis.

No livro *Padrões de Projeto: Soluções reutilizáveis de software orientado a objetos* (GAMMA, 2000) são descritos 23 padrões de projeto, cada padrão nomeia,

explica e avalia um aspecto de projeto importante e recorrente, tornando mais fácil reutilizar técnicas e arquiteturas bem sucedidas, testadas e aprovadas.

Os 23 padrões de projetos podem ser agrupados em categorias segundo sua finalidade ou escopo, como mostrado no Quadro 6. A finalidade demonstra o que um padrão faz, podendo ser criacional, estrutural ou comportamental. O escopo define se o padrão é aplicado primariamente a classes ou a objetos. Se aplicado a classes, o padrão modela os relacionamentos estáticos entre as classes e suas subclasses. Quando aplicado a objetos, o padrão refere-se aos relacionamentos dinâmicos entre os objetos.

		Finalidade		
		De criação	*Estrutural*	*Comportamental*
Escopo	*Classe*	Factory Method	Adapter	Interpreter Template Method
	Objeto	Abstract Factory Builder Prototype Singleton	Adapter Bridge Composite Decorator Façade Flyweight Proxy	Chain of Responsability Command Iterator Mediator Memento Observer State Strategy Visitor

Quadro 6 - Classificação dos padrões de projeto
Fonte: GAMMA (2000)

As pricipais modificações realizadas no código se basearam nos padrões *Factory Method* e *Façade*. O padrão *Factory Method* tem como intenção definir uma interface para a criação de objetos, essa interface fica com a responsabilidade de decidir que tipo de objeto instanciar. A Figura 9 mostra como deve ser a estrutura de um código que utiliza o padrão *Factory Method*. O método *FactoryMethod()* fica responsável por instanciar um objeto concreto da hierarquia da classe *Product*.

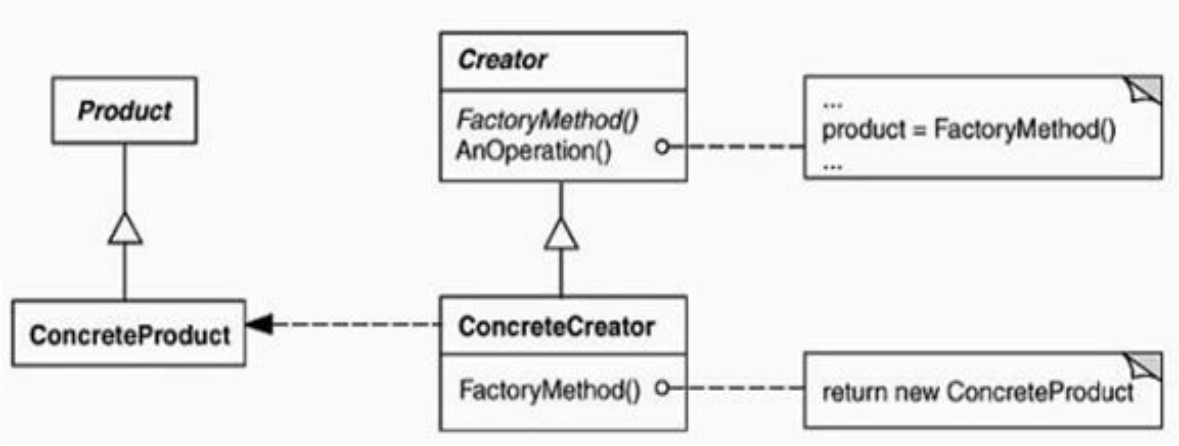


Figura 9 - Estrutura do padrão *Factory Method*
Fonte: GAMMA (2000)

A estrutura da utilização do padrão *Façade* é composta pela classe *Façade*, responsável pelo objeto fachada que fornece aos clientes uma interface única e simples para poder utilizar um subsistema ou subclasse. A classe *Façade* delega a solicitações do cliente para os objetos responsáveis do subsistema. As classes do subsistema implementam as funcionalidades do sem manter referência para a classe *Façade*. A Figura 10 mostra a estrutura do padrão *Façade*.

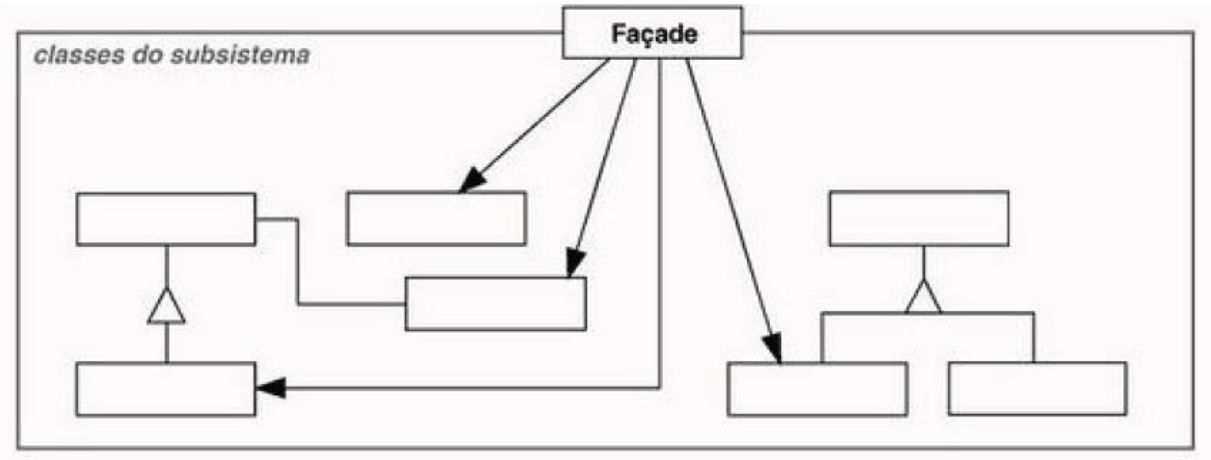


Figura 10 - Estrutura do padrão *Façade*
Fonte: Autoria própria

4. PESQUISA E DESENVOLVIMENTO

A execução deste trabalho foi realizada em duas etapas, a pesquisa e o desenvolvimento.

4.1 A PESQUISA

A etapa de pesquisa realizada inicialmente pode ser classificada como pesquisa bibliográfica. A pesquisa bibliográfica é feita a partir do levantamento de referências teóricas publicadas por meios escritos e eletrônicos, como livros, artigos científicos e páginas de web sites. Todo trabalho científico deve iniciar-se com uma pesquisa bibliográfica, para permitir ao pesquisador conhecer o que se estudou sobre o assunto (FONSECA, 2002, p. 32). Nessa etapa foram adquiridos os conhecimentos sobre o editor GEDE, sobre o leitor de tela NVDA e levantados os métodos de validação de acessibilidade. A pesquisa bibliográfica realizada também buscou associação com o ensino de diagramas UML para alunos com deficiência visual.

Ao ter definido, como problema inicial e foco de estudo, a adição dos recursos de criar outros tipos de diagramas UML ao editor de diagramas GEDE, a pesquisa passa a ser exploratória. A pesquisa exploratória objetiva proporcionar maior familiaridade com o problema e assim torná-lo mais explícito (GIL, 2007). Os conhecimentos sobre a Java Access Bridge, e a exploração do código em Java do editor tiveram inicio nessa etapa da pesquisa.

A pesquisa aplicada, por fim, visou gerar conhecimentos para a aplicação prática da solução do problema, ou seja, conhecimentos para a codificação. O estudo do uso da API de acessibilidade Java verificou como foi a implementação da acessibilidade e como manter essa acessibilidade nas extensões. O estudo mais aprofundado da codificação existente levantou a necessidade de adaptação e a

busca de meios menos acoplados para a implementação. Esses meios foram encontrados no uso dos padrões de projeto.

4.2 O DESENVOLVIMENTO

As extensões realizadas na interface do GEDE permitem a criação e edição dos outros diagramas utilizados na UML. Além disso, as adaptações que foram necessárias tornaram facilitada a adição de novas extensões pelo programador.

As adaptações tiveram início na classe *Menu*. Essa classe possui todos os objetos do tipo *JMenu* e *JMenuITem* utilizados dentro do programa. Foram adicionados os itens de menu necessários para os outros tipos de diagramas, como os outros itens do menu *Novo*. A instanciação desses objetos fica distribuída em métodos, um para cada item do menu principal, como mostrado no Quadro 7. Esses métodos são chamados diretamente no construtor da classe *Menu*. Porém, a instanciação dos itens do menu *Inserir*, responsáveis pela inserção de novos elementos nos diagramas, foi transferida para os itens do menu *Novo*. O quadro a seguir mostra que os métodos *menuSelecionar(), menuInserirCasodeUso()* e *menuInserirClasse()*, assim como os outros métodos referentes aos diagramas de atividade, sequência, componentes e implantação, não são mais chamados no construtor da classe *Menu*.

```
public Menu(GUI frame) {
        [...]
      menuPrincipal();
      menuArquivo();
            menuNovo();
      menuEditar();

            //menuSelecionar(); OS SUBMENUS DO Selecionar SÓ SÃO
      INSTANCIADOS AO SE INSERIR UM VERTICE

      //menuInserirCasodeUso(); SÓ É CHAMADO AO CRIAR UMA NOVA ABA
DIAGRAMA DE CASO DE USO
      //menuInserirClasse();  SÓ É CHAMADO AO CRIAR UMA NOVA ABA
DIAGRAMA DE CLASSES
     [...]
      menuFerramentas();
            menuNavegacao();
     menuAjuda();

      eventos();
      [...]

         getAccessibleContext().setAccessibleName("Menu");
}
```

Quadro 7 - Modificação da classe Menu
Fonte: Autoria própria.

Desse modo, a opção de inserir uma classe ou uma associação ao diagrama de classes, por exemplo, é habilitada somente após se criar uma aba com um diagrama de classes. Essa é uma das alterações realizadas para nível de organização e modularização da classe *Menu*. Os métodos eventos, *eventosDClasse(), eventosDCasoDeUso(), eventosDAtividade()...,* foram criados para aglutinar as responsabilidades relacionadas aos referentes diagramas. Esses métodos primeiramente instanciam os itens do menu *Inserir* e assim podem definir as ações para esses itens. O Quadro 8 a seguir, mostra o evento de clique no item *Classe* do menu *Novo*. Após adicionar uma aba para o tipo de diagrama de classes, o método *eventosDClasse()* é chamado para instanciar os itens do menu *Inserir* referentes à diagramas de classes. Nesse mesmo método são definidas as ações para os itens instanciados.

```
private void eventos() {
[...]

// EVENTO DE CLIQUE NO ITEM Diagrama de Classe DO MENU Novo
      classe.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e){
                  if(frame.AdicionarAbas("classe")){

                        eventosDClasse();

                        Navegacao.setVisible(true);
                  }
            }
      });
}

[...]
private void eventosDClasse(){

      menuInserirClasse();
      [...]
      Classe.addActionListener(new ActionListener() {
            [...]
      });

      Associacaoc.addActionListener(new ActionListener() {
            [...]
      });

}
```

Quadro 8 - Chamada de métodos na classe Menu
Fonte: Autoria Própria.

A principal modificação realizada no código se baseou no padrão *Factory Method*. Antes da adaptação, o método *AdicionarAbas()* da classe *GUI* instanciava um objeto do tipo *Grafo*, esse objeto continha alguns métodos responsáveis pela manipulação do Diagrama de Casos de Uso, outros se encontravam na classe *Menu*.

Com a adição dos outros tipos de diagramas UML, a classe *Grafo* se tornou uma classe abstrata e suas classes herdeiras passaram a ser instanciadas no novo método *AdicionarAbas()*. Esse método é a interface que verifica através de um parâmetro recebido qual o objeto a ser instanciado. O diagrama da Figura 11 abstrai a estrutura das classes para que seja possível notar a utilização do padrão.

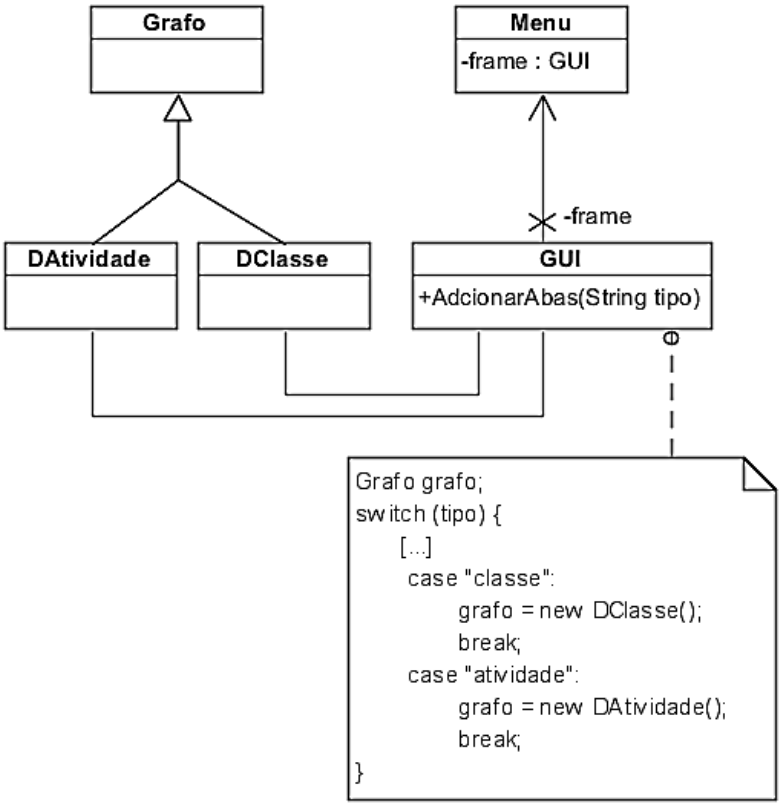


Figura 11 - Utilização do padrão Factory Method
Fonte: Autoria própria

Como consequência, o *Factory Method* elimina a necessidade de anexar as classes específicas do pacote *controle*[2] no código da classe *Menu*. A classe *Menu* lida somente com a classe abstrata *Grafo*; portanto, a classe *Menu* pode trabalhar com qualquer classe que herde de *Grafo*.

Ao modificar a classe *Grafo* para uma classe abstrata e criar classes que implementem seus métodos abstratos, foi utilizado o padrão *Façade*. Esse padrão define uma interface para a utilização das classes herdeiras de *Grafo*. A classe *Menu* só precisa conhecer os métodos da classe fachada *Grafo*, como o método *AdicionarVertice()*. De acordo com o tipo de classe concreta que foi instanciado e com a sobrescrita que foi realizada desse método ele transfere a criação de um vértice para outro método correspondente. Por exemplo, como mostrado na Figura 12, a sobrescrita do método *AdicionarVertice()* na classe *DClasse* chama o método *AdicionarClasse()*.

[2] Pacote *diagram.controle*

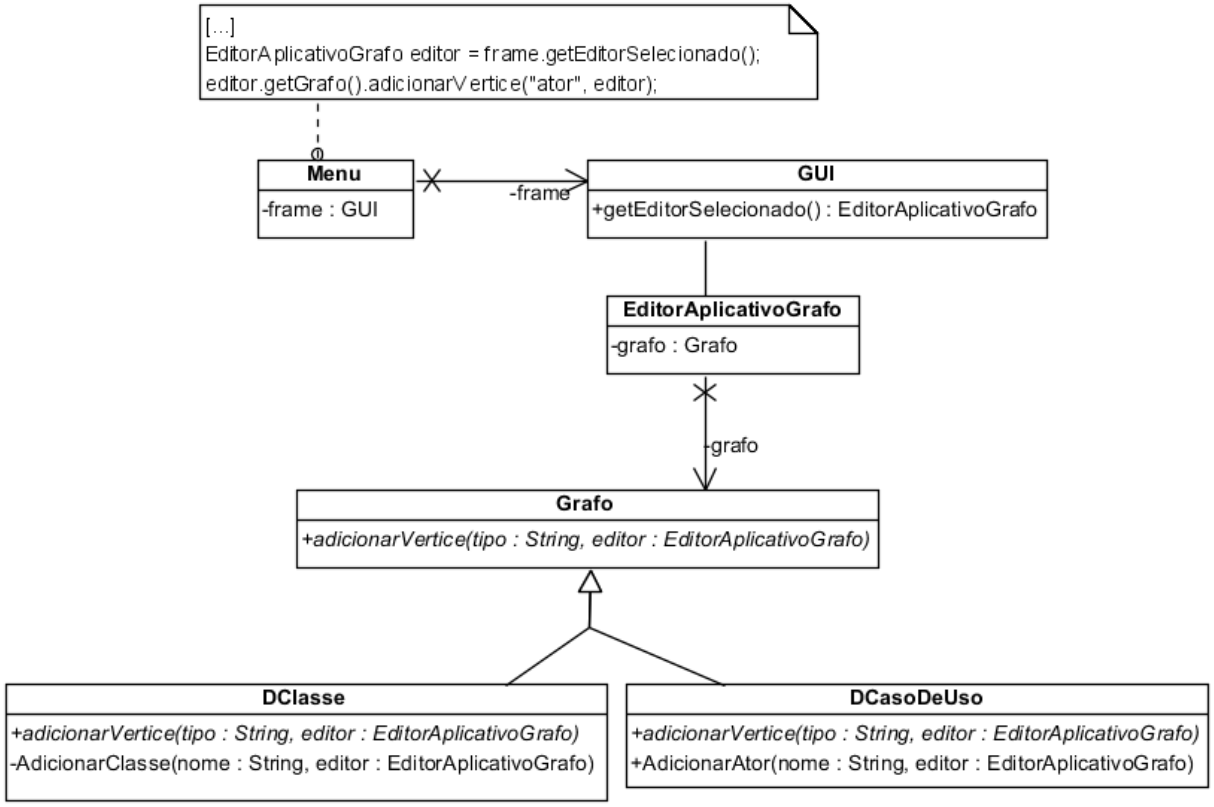


Figura 12 - Utilização do padrão Façade
Fonte:Autoria própria

A classe *Teste*.*GUI*, mostrada na Figura 13, estende a classe *JFrame* e contém o método *main()*. O principal método desta classe é o método *AdicionarAbas()* que passou por adaptações para permitir a adição dos outros diagramas. No código anterior, esse método apenas era chamado no evento de clique do item Caso de uso do menu *Novo*. Atualmente o método *AdicionarAbas()* é chamado quando o usuário escolhe o novo tipo de diagrama que deseja criar.

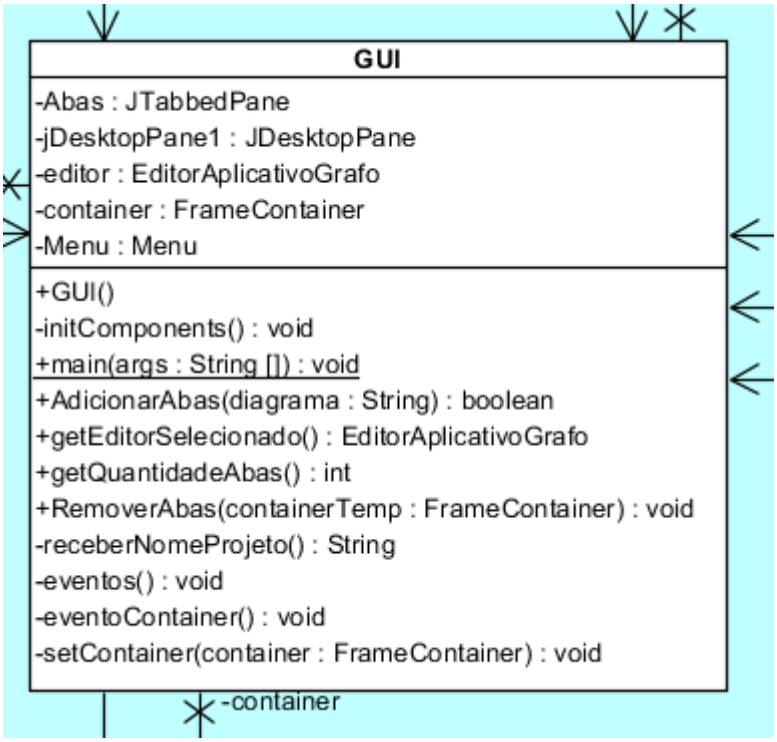


Figura 13 - Classe GUI
Fonte: Autoria própria.

O Quadro 9 apresenta uma abstração da principal alteração realizada nesse método para assim ser possível instanciar o diagrama de acordo com o tipo escolhido pelo usuário.

```
public boolean AdicionarAbas(String diagrama) {

 Grafo grafo = null;
        String rotulo = this.receberNomeProjeto();

if (rotulo != null) {
            try {
                switch (diagrama) {
                    case "caso de uso":
                        grafo = new DCasoDeUso();
                        break;
                    case "classe":
                        grafo = new DClasse();
                        break;
                    case "sequencia":
                        grafo = new DSequencia();
                        break;
                    case "atividade":
                        grafo = new DAtividade();
                        break;
                    case "componentes":
                        grafo = new DComponentes();
                        break;
                    case "implantacao":
                        grafo = new DImplantacao();
                        break;
                    case "exemplo":
                        grafo = new DCasoDeUso();
                        break;
                    default:
                        grafo = new DCasoDeUso();
                }
} catch (ExceptionInInitializerError e) {
                System.out.println(e.getCause());
     }
            editor = new EditorAplicativoGrafo(grafo, this);
            editor.setMenu(Menu);

            [...]
            return true;
        }
    return false;
}
```

Quadro 9 - Principal modificação da classe Grafo
Fonte: Autoria própria

A classe *Grafo* era a responsável pelos métodos de manipulação do diagrama de caso de uso. Para que fosse possível adicionar outros diagramas a classe *Grafo* se tornou uma classe abstrata, não podendo ser instanciada. Ela permaneceu com os métodos e atributos comuns a todo tipo de diagrama e recebeu da classe *Menu* dois métodos que possuem essa característica, *receberRotulo()* e

existeVertice(). Os métodos referentes a apenas diagramas de caso de uso foram unidos em uma nova classe, *diagram.controle.DCasoDeUso*. Assim, para cada extensão de um novo diagrama UML existe uma classe que estende Grafo no novo pacote *diagram.controle* e que implementa os métodos abstratos de maneira única para cada diagrama. A abstração do diagrama de classes na Figura 14 ilustra essa nova herança.

As classes, *DClasse, DAtividade, DComponentes, DImplantacao* e *DSequencia*, foram criadas para a adição da extensão dos diagramas. Cada uma possui algoritmos próprios da sobrescrita dos métodos de *Grafo* para retornar as possíveis associações de um vértice, por exemplo.

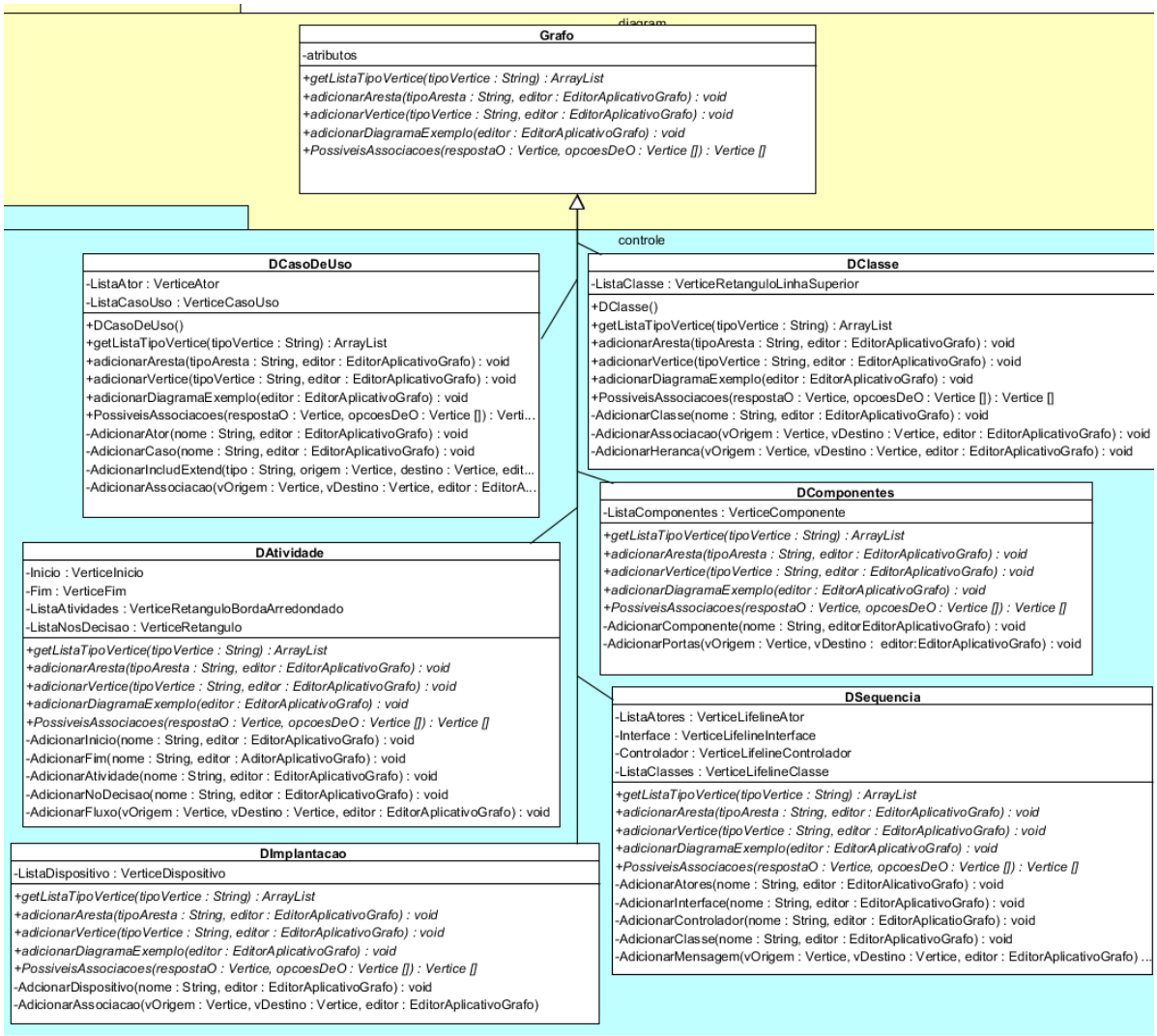


Figura 14 - Herança da classe Grafo
Fonte: Autoria própria

As classes do pacote *controle* implementam os métodos abstratos da classe *Grafo*. Esses métodos abstratos possuem responsabilidades diferentes dependendo da classe concreta. Para o diagrama de caso de uso, por exemplo, a implementação do método *adicionarVertice*(*String*, *EditorAplicativoGrafo*) verifica através da *String* recebida qual o tipo de vértice que se deseja adicionar, para então, utilizar um dos métodos internos correspondentes: *AdicionarAtor()* ou *AdicionarCaso()*. A mesma verificação ocorre na utilização do método *AdicionarAresta(),* que pode chamar os métodos *AdicionarAssociacao(),* colocando uma associação simples no diagrama, ou o método *AdicionarIncludExtend()* que adiciona uma associação do tipo *include* ou *extend* entre os vértices recebidos por parâmetro.

A criação do pacote *controle,* e de suas classes, permitiu restringir a chamada de métodos e a instanciação de objetos. Apenas uma aba que possui um grafo do tipo *DClasse* pode utilizar o método *AdicionarClasse()* e acessar o atributo *ListaClasses*.

Outro exemplo de alteração realizada no código é a transferência de métodos para outras classes, com a intenção de suprimir o uso de certas variáveis. Por exemplo, o método *SelecionarComponente(),* da classe *Grafo* foi transferido para a classe EditorAplicativoGrafo. No código antigo era o único método que utilizava uma variável global do tipo *EditorAplicativoGrafo.* A Figura 15 e a Figura 16 ilustram a alteração realizada. Consequentemente, todos os usos do método *SelecionarComponente()* não utilizam mais um objeto do tipo *Grafo* e sim do tipo *EditorAplicativoGrafo*.

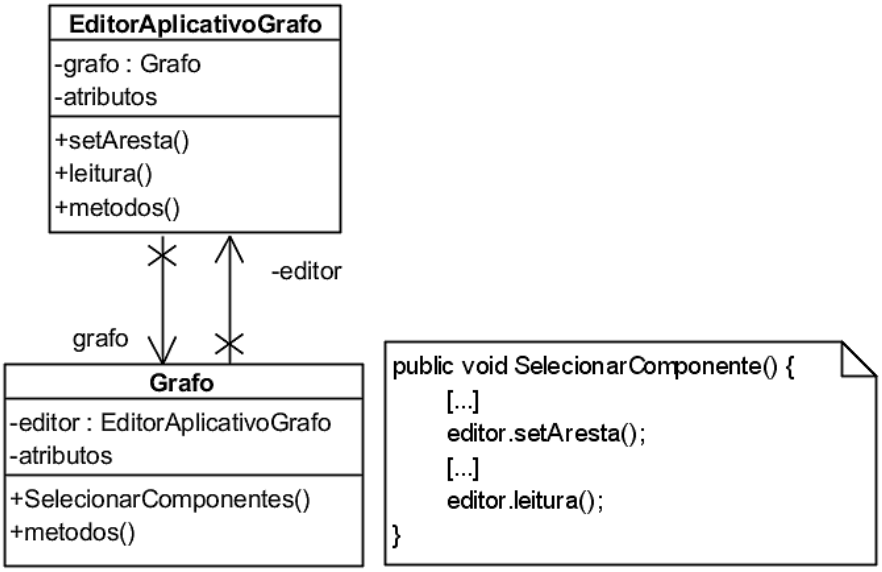


Figura 15 - Abstração da comunicação antiga entre Grafo e EditorAplicativoGrafo
Fonte:Autoria própria.

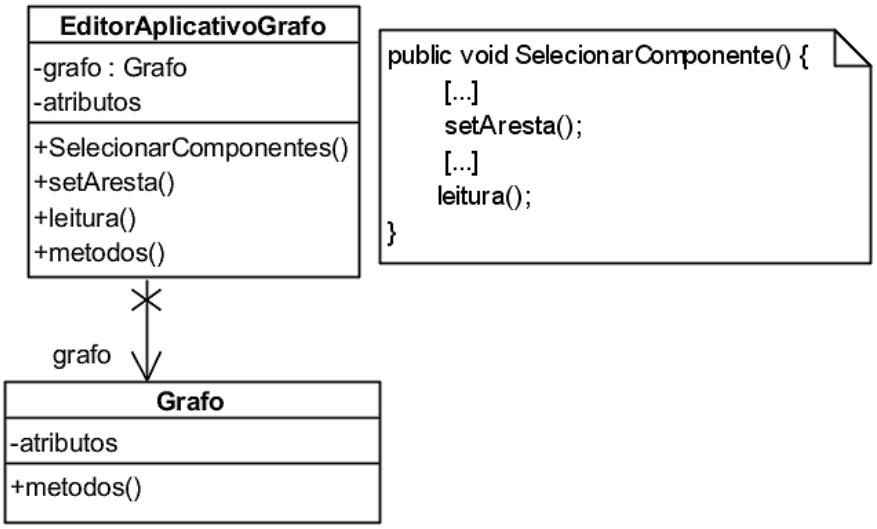


Figura 16 - Abstração da nova comunicação entre Grafo e EditorAplicativoGrafo
Fonte Autoria própria

Para realizar a extensão dos recursos e tornar possível a criação de outros diagramas UML foi necessário criar novas classes para o desenho de vértices e arestas.

Como herdeiras da classe *Aresta* foram criadas as classes: *ArestaHeranca, ArestaPortas, ArestaFluxo* e *ArestaMensagem*, as outras foram apenas mantidas. A Figura 17 mostra a os tipos de aresta que podem ser instanciados. Para cada nova aresta é necessário sobrescrever o método

desenharAresta(), trabalhando com as coordenadas *x* e *y* e com os métodos da classe nativa *Graphics*, por exemplo: *desenho.drawLine(Xinicial, Yinicial, Xfinal, Yfinal).* Para alguns vértices também é necessário sobrescrever o método *calcularPonta(),* que desenha a ponta da aresta.

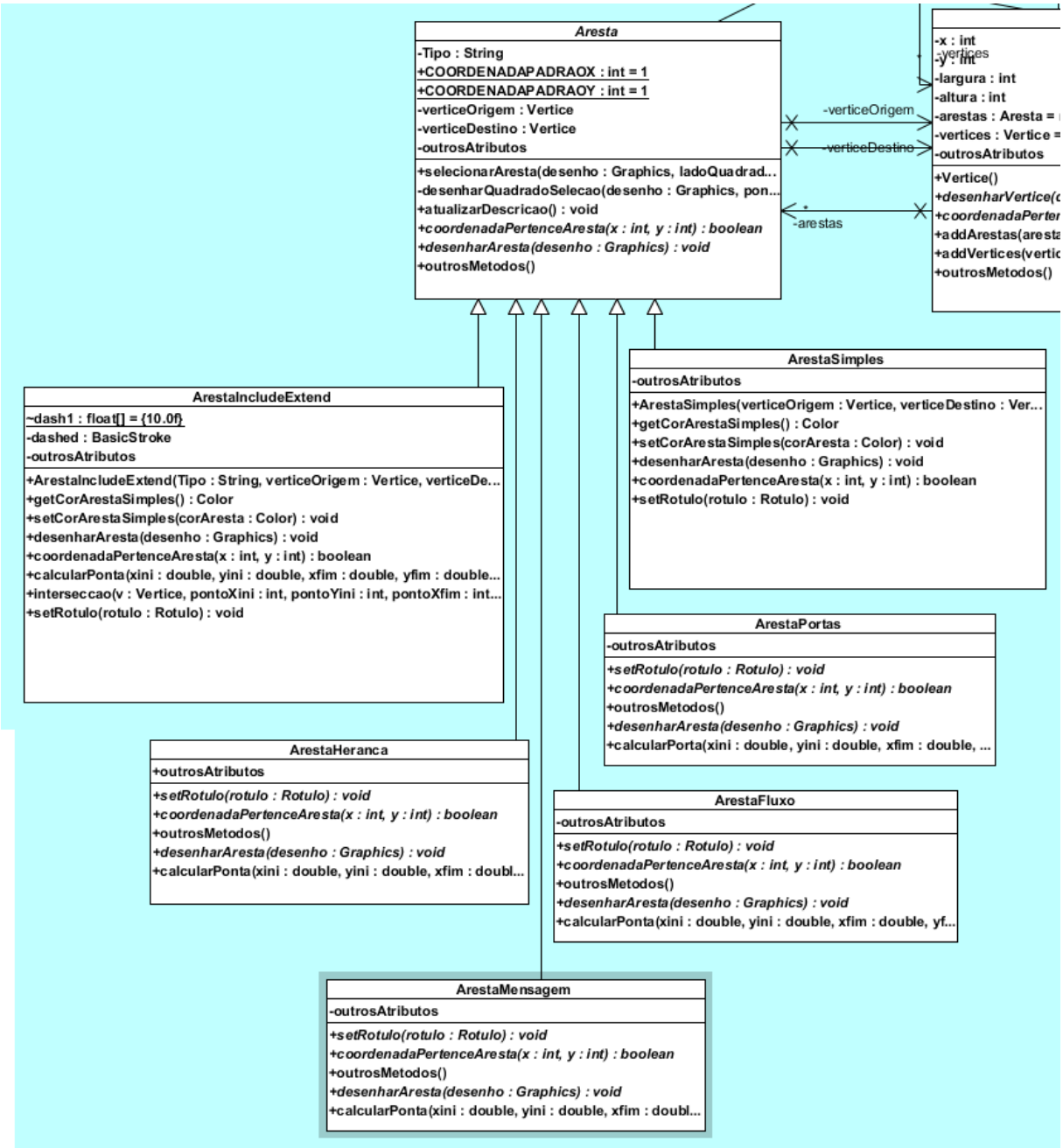


Figura 17 - Herança da classe *Aresta*
Fonte: Autoria própria

Para criar os novos tipos de vértices para os diagramas adicionados, foram implementadas novas classes que estendem da classe *Vertice,* mostrada na Figura 18. As classes *VerticeComponente, VerticeDispositivo, VerticeInicio, VerticeFim, VerticeLifelineAtor*, *VerticeLifelineControlador* e *VerticeLifelineInterface* foram criadas segundo o modelo dos tipos de vértices já existentes, porém a implementação do método abstrato *desenharVertice()* precisou ser estudada e modificada de acordo com o desenho desejado para cada vértice.

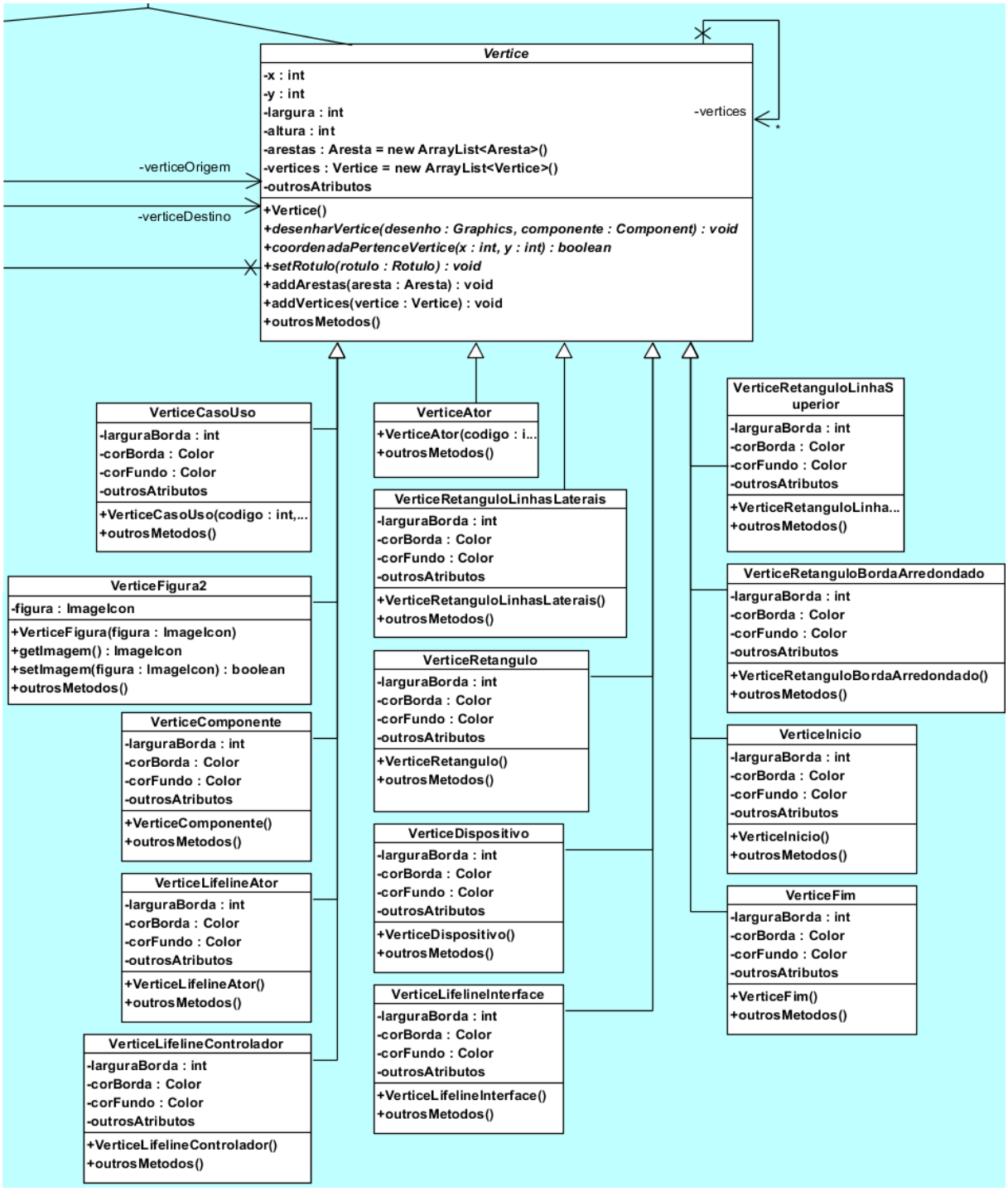


Figura 18 - Herança da classe *Vertice*
Fonte: Autoria própria

A Figura 19 mostra um exemplo de um diagrama de classes, ainda sem atributos ou métodos, que pode ser desenhado pelo GEDE após a realização da extensão. A Figura 20 mostra o diagrama de sequencia realizado pela interface.

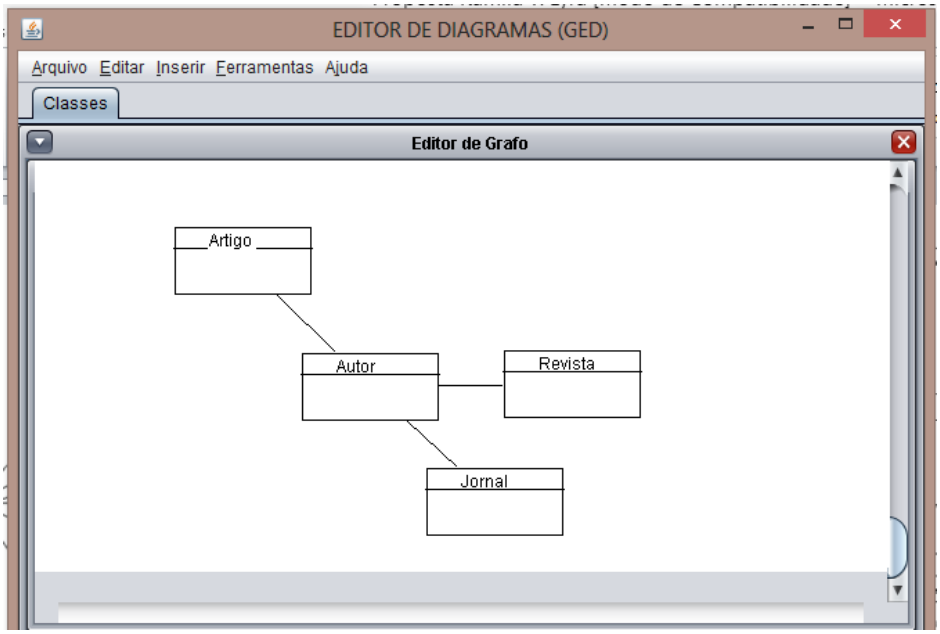


Figura 19 - Interface exibindo um diagrama de classes
Fonte: Autoria própria

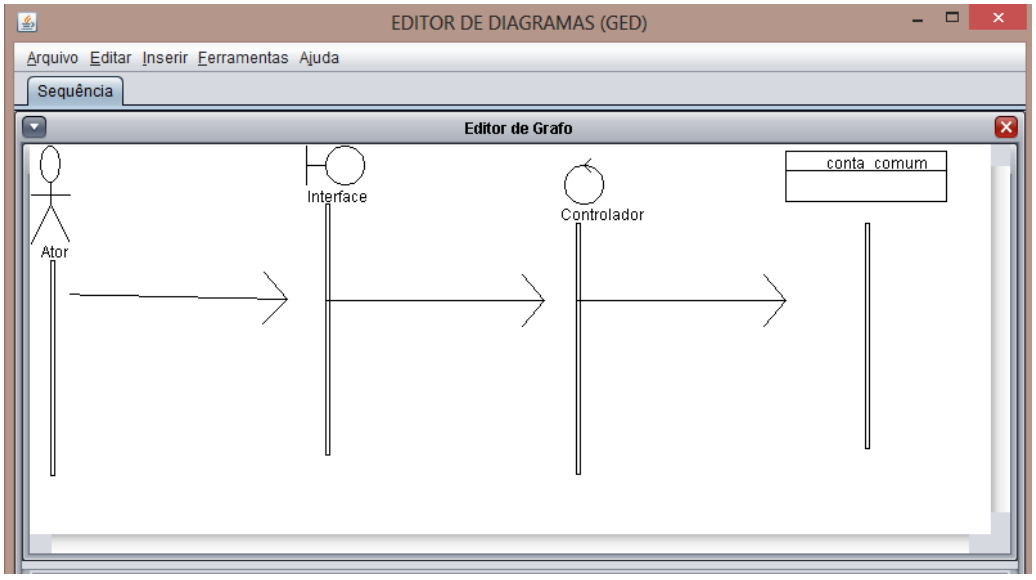


Figura 20 - Interface exibindo um diagrama de sequência
Fonte: Autoria própria

A Figura 21 exibe o diagrama de implantação e a Figura 22 mostra um diagrama de atividade desenhados utilizando a ferramenta GEDE.

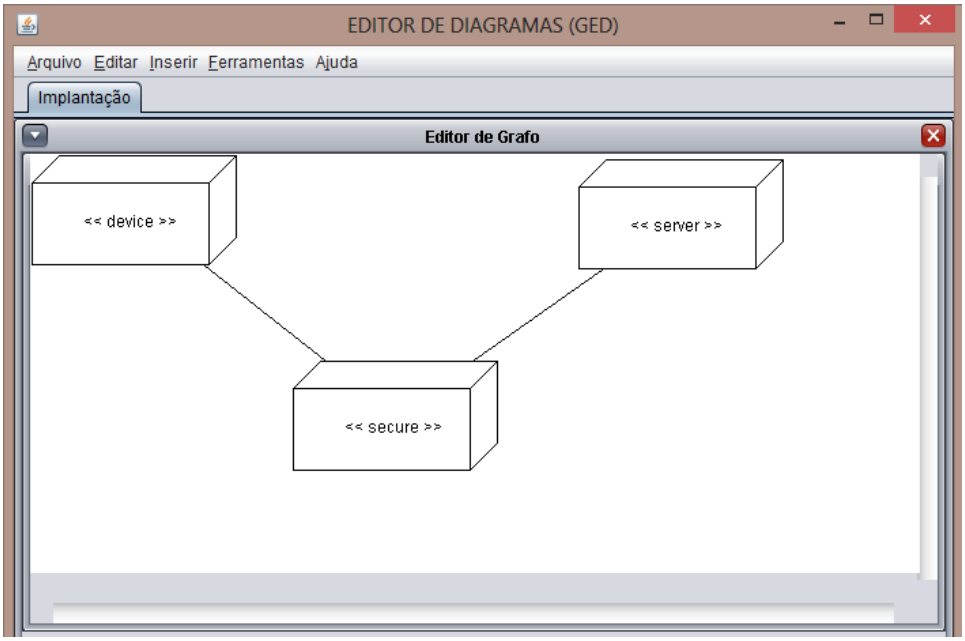


Figura 21 - Interface exibindo um diagrama de implantação
Fonte: Autoria própria

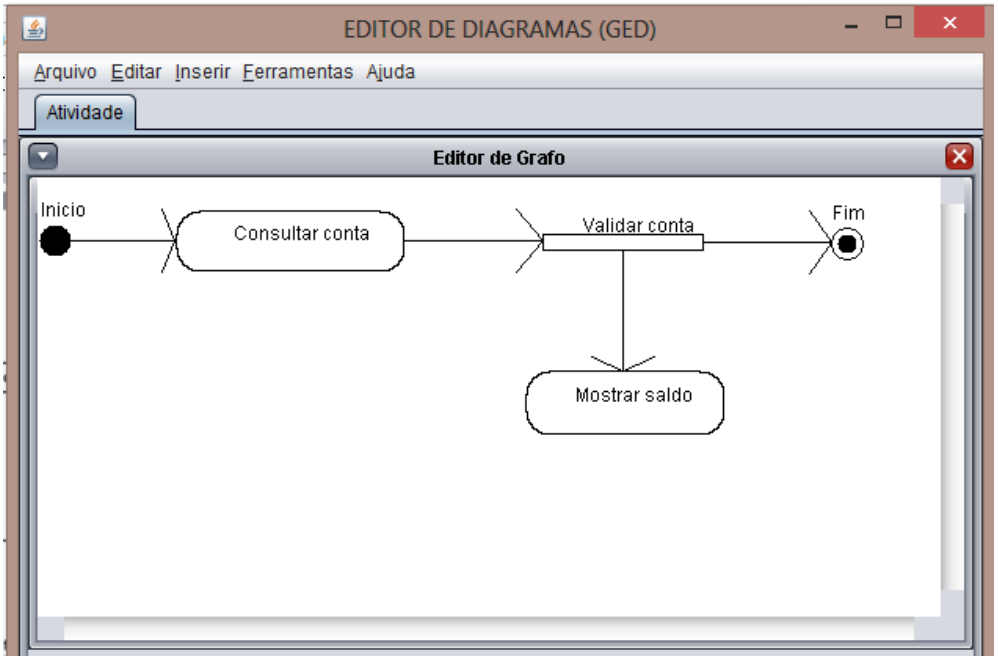


Figura 22 - Interface exibindo um diagrama de atividades
Fonte: Autoria própria

A Figura 23 mostra a interface do GEDE exibindo o diagrama de componentes da extensão.

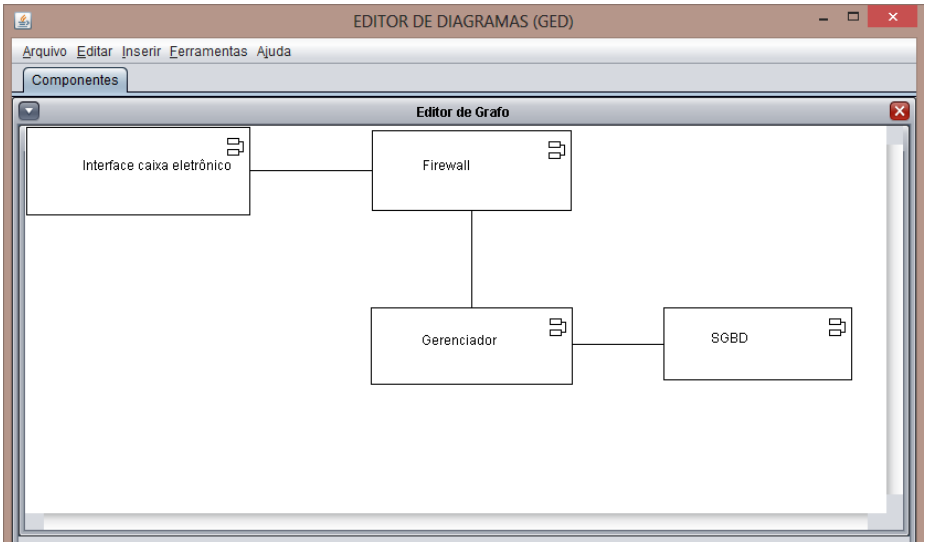


Figura 23 - Interface exibindo um diagrama de componentes
Fonte: Autoria própria

4.3 AVALIAÇÃO

A organização TIRESIAS (2009) dispõe de mais de quarenta listas de verificação direcionadas para sistemas de tecnologia da informação e comunicação. A Lista de Verificação para Aplicação de Software foi utilizada para avaliar os pontos de acessibilidade do GEDE. Essa lista funciona como uma lista de recomendações para aumentar o nível de acessibilidade do software através da adição dos recursos elucidados. O Quadro 10 mostra a lista de verificação; as recomendações marcadas com um “sim” foram atendidas pelo GEDE. A organização ainda classifica cada ponto abordado na lista de acordo com o nível de importância para cada tipo de deficiência de um usuário.

		Tipo de deficiência[3]							
		F	**FG**	**V**	**C**	**A**	**S**	**C**	**O**
Tecnologias assistivas									
A aplicação evita interromper ou desativar recursos ativados que são identificados como recursos de acessibilidade?	sim	+	++	+	++	-	+	+	-
A aplicação faz uso do sistema de entrada / saída padrão?	sim	-	+	-	+	--	-	--	--
O sistema operacional provê os serviços para que a aplicação seja acessível?	sim	-	+	-	+	--	-	-	-
O usuário pode selecionar os dispositivos de entrada / saída?	não	+	++	-	++	-	-	--	--
O usuário pode mudar as alternativas de entrada / saída?	não	-	-	-	-	-	-	--	--
O usuário pode realizar a tarefa de forma eficaz com qualquer dispositivo de entrada?	sim	+	++	-	++	--	-	--	--
O sistema operacional fornece perfis de preferência do usuário?	não	-	-	-	-	-	-	-	-
O aplicativo fornece navegação para grupos de tarefas de controles?	não	+	++	-	++	--	-	-	--
O sistema mantém o foco de entrada?	não	-	-	-	+	--	--	+	--
Acesso ao teclado									
Todas as funções do aplicativo (incluindo navegação) são acessíveis pelo teclado?	sim	+	++	-	++	--	--	-	-
Existe uma alternativa sequencial para a ação simultânea de teclado?	não	+	++	--	--	--	--	-	--
A aplicação respeita as convenções de acesso teclado do sistema operacional?	sim	-	+	-	+	--	--	-	--
A sequência para se mover de um ponto a outro em uma caixa de diálogo usando o teclado está acordo com o layout da tela?	sim	-	+	-	+	--	--	+	--

3

F: Deficiências físicas
FG: Deficiências físicas graves
V: Deficiência visual (inc. daltonismo)
C: Cegos
A: Deficiências auditivas leves a moderadas
S: Surdos (pessoas com deficiências auditivas graves)
C: Cognitivas
O: Outros (inc. canhoto, língua diferente,...).

-- Problemas significativos
- Problemas menores
+ Problemas moderados
++ Problemas graves

A aplicação fornece teclas de documentado?	sim	-	+	-	+	--	--	--	--
A aplicação evita interferir no sistema operacional com recursos de acessibilidade do teclado?	não	+	++	-	+	--	--	--	--
As funções do teclado podem ser remanejadas?	não	-	+	--	--	--	--	-	--
O teclado de ativação é separado do teclado de navegação?	sim	+	++	+	++	--	--	+	-
Existe um método para distinguir entrada macro de entrada de teclado?	não	-	-	--	--	--	--	--	--
Dispositivos apontadores									
As funções do botão do dispositivo apontador são ajustáveis?	não	+	++	--	--	--	--	--	-
Existe a possibilidade de permitir vários cliques com o pressionar ou soltar de uma tecla?	não	+	++	--	--	--	--	+	--
Existe a possibilidade de habilitar botão de espera com um clique?	não	+	++	--	--	--	--	+	--
Pode ser definida a aceitação de um atraso no pressionamento de um botão?	não	+	++	--	--	--	--	+	--
Pode ser definida a aceitação de um atraso após o movimento do ponteiro do mouse?	não	+	++	--	--	--	--	+	--
A área alvo de múltiplos cliques é modificável?	não	+	++	--	--	--	--	+	--
A velocidade de movimento do ponteiro é personalizável?	não	+	++	--	--	--	--	+	--
Existe a possibilidade de utilizar um método alternativo para a entrada normalmente feita com um dispositivo apontador?	não	-	+	-	+	--	--	--	--
Informações sobre o objeto									
Existe uma indicação na tela bem definida do foco atual?	sim	--	--	+	--	--	--	+	--
São fornecidas informações semânticas sobre objetos de interface do usuário?	sim	--	--	+	++	--	--	-	--
Os rótulos são associados com controles, objetos, ícones e imagens?	sim	--	--	+	++	--	--	-	--
Se forem utilizados formulários electrónicos, as pessoas que utilizam tecnologias assistivas conseguem utilizar os formulários?	__[4]	-	+	-	++	--	--	--	--

[4] Sem formulários eletrônicos

O sistema operacional fornece texto ou linguagem de sinais como sistema base de ajuda?	não	--	--	--	--	+	++	--	--
O menu do aplicativo e janela de navegação são circulares?	não	-	+	-	+	--	--	-	--
A notificação de eventos alcançam as tecnologias assistivas?	sim	-	+	+	++	--	--	--	--
Os atributos do objeto são disponíveis para tecnologias assistivas?	sim	-	+	+	++	--	--	--	--
São disponibilizados designadores implícitos (letras sublinhadas em menus, botões,...)?	sim	-	-	-	-	--	--	-	--
Sons e multimídia									
Há funções que oferecem a possibilidade de enviar as informações de texto para a saída de voz?	sim	--	--	-	++	--	--	--	--
A saída de voz ocorre imediatamente após o evento que o gerou?	sim	--	--	-	++	--	--	-	--
Avisos auditivos de alertas são apresentados de forma visual?	sim	--	--	--	--	+	++	--	--
Existe a possibilidade de ajustar a frequência do som de avisos sonoros?	não	--	--	--	--	+	--	--	--
Existem alternativas acessíveis para áudio e vídeo?	não	--	--	+	++	+	++	--	--
Existe uma opção para ajustar o volume do som?	não	--	--	+	--	+	--	--	--
Exibição									
Informações textuais são fornecidas pelo sistema para textos exibidos?	não	--	--	-	++	--	--	--	--
A aplicação evita substituir as preferências de contraste e seleções de cores e outros atributos de exibição individuais do usuário?	não	--	--	+	--	--	--	--	--
Existe uma opção para exibir animação em um modo de apresentação sem animação?	não	--	--	-	++	--	--	-	--
Existe um esquema de configuração de cor projetado para pessoas que têm deficiência visual?	não	--	--	+	--	--	--	--	--
Todo o texto exibido tem um cursor para navegar pelo texto?	não	--	--	-	+	--	--	--	--
É possível ajustar o tamanho e a posição dos objetos exibidos pelo sistema?	sim	-	+	+	--	--	--	-	--
Todos os ícones possuem um rótulo de texto, e o usuário pode optar por exibir somente o rótulo?	sim	--	--	+	+	--	--	-	--

Em imagens, as legendas estão colocadas no lado esquerdo e horizontalmente alinhadas com a primeira linha do componente correspondente?	não	--	--	-	++	--	--	--	--
O sistema permite que janelas fiquem "sempre no topo"?	não	-	-	-	-	--	--	-	--
É possível ajustar o tamanho e a posição das janelas na tela?	sim	-	-	+	--	--	--	--	--
Há opções para minimizar, maximizar e fechar janelas?	sim	-	-	+	--	--	--	--	--
É possível alternar entre janelas usando o teclado?	sim	+	++	-	++	--	--	--	--
A aplicação disponibiliza informação visual de pelo menos um modo para os usuários com acuidade visual de 20/70 e 20/200?	não	--	--	+	--	--	--	--	--
O aplicativo suporta definições de alto contrate por toda a interface do usuário?	não	--	--	+	--	--	--	--	--
As janelas dos aplicativos herdam as configurações do sistema operacional para a fonte, tamanho e cor?	não	--	--	+	--	--	--	--	--
As fontes podem ser personalizadas para obter uma melhor legibilidade?	não	--	--	+	--	--	--	--	--
A escala e a disposição dos objetos são ajustadas quando o tamanho da fonte muda?	não	--	--	+	--	--	--	--	--
O aplicativo usa uma apresentação de tabelas adequada?	não	--	--	+	++	--	--	--	--
A aplicação usa as configurações de cor do usuário?	não	--	--	+	--	--	--	--	--
O sistema permite o controle do usuário para diversas telas que estejam “sempre no topo”?	não	-	-	-	-	--	--	-	--
Notificação do usuário									
Será que o sistema usa mensagens curtas e simples?	sim	--	--	--	--	-	+	+	-
O formato das mensagens e o tipo de exibição são sempre os mesmos?	sim	--	--	--	--	-	+	+	-
A notificação do usuário é apresentada de forma relevante?	sim	-	-	-	-	-	-	-	-
Cronometragem									
O sistema evita texto, objetos ou outros elementos piscando com uma frequência de flash superior a dois Hz e menor que 55 Hz?	sim	--	--	--	--	--	--	--	++

O usuário pode personalizar ou eliminar respostas cronometradas?	—[5]	+	++	-	++	--	+	+	+
A informação de aviso ou erro de tarefa persiste até a aceitação do usuário?	sim	+	++	-	++	-	+	+	+
A aplicação permite ao usuário controlar a apresentação de informação cronometrada?	—[6]	+	++	-	++	-	+	+	+
Outros									
O aplicativo está desenvolvido para minimizar o número de passos necessários para ativar qualquer opção?	sim	+	++	-	++	--	--	+	--
A aplicação fornece a função finalizar?	sim	-	-	-	-	-	-	-	-
A interface do sistema fornece uma funcionalidade desfazer?	sim	+	++	-	+	--	-	+	-
As preferências do usuário podem ser personalizadas?	não	-	-	-	-	-	-	-	-
As preferências do usuário podem ser usadas em todos os locais?	não	-	-	-	-	-	-	-	-

Quadro 10 - Lista de verificação para aplicações de software
Fonte: TIRESIAS (2009)

Foram atendidas 31 recomendações, de um total de 73 recomendações. Sendo que as categorias com cem por cento de atendimento foram as de Cronometragem e de Notificação do usuário. O sistema não possui limite de tempo para a execução das tarefas deixando que o usuário construa o diagrama à seu tempo, além disso as mensagens de notificação do usuário são simples e possuem sempre os mesmo formatos, facilitando a compreensão do usuário. O Quadro 11 mostra a porcentagem de conformidade para cada categoria da lista.

[5] Sem limite de tempo para tarefas
[6] Sem limite de tempo para tarefas

Cronometragem:	100%
Notificação do usuário:	100%
Informações sobre o objeto:	75%
Acesso ao teclado:	55,5%
Sons e Multimídia:	50%
Tecnologias assistivas:	44,4%
Exibição:	25%
Dispositivos apontadores:	0%

Quadro 11 - Porcentagem de conformidade por categoria
Fonte: Autoria própria

A categoria de Informações sobre o objeto, com 75 % de conformidade, reúne recomendações sobre os objetos da interface, como botões ou figuras que devem ser sempre acessíveis aos leitores de tela e possuírem teclas de atalho para o acesso. Para atender 100% a essa categoria o sistema deve fornecer um sistema de ajuda no formato de texto ou linguagem de sinais e ainda modificar seu menu para suportar um formato circular.

A categoria de Exibição, que trata da visualização da interface de acordo com as preferências e necessidades do usuário, teve apenas vinte e cinco por cento das recomendações atendidas.

Esse software não oferece nenhum suporte a configurações dos dispositivos apontadores, assim o software atendeu 0% das recomendações dessa categoria. Nota-se que não cumprir algumas das recomendações da Lista de Verificação para Aplicação de Software não traz um mau funcionamento para o sistema, porém, torna-o difícil de ser usado por pessoas com quaisquer tipos de necessidades especiais. Cada recomendação que é atendida pelo software significa uma funcionalidade a mais que pode ser executada de modo acessível.

5. CONCLUSÃO

É um desafio ensinar a lógica da modelagem para alunos com deficiência visual, considerando que o método de ensino geralmente é totalmente baseado em ilustração gráfica. A UML é a principal linguagem para a modelagem de software e seu ensino nas universidades possui obstáculos para esses alunos. O principal obstáculo é a falta de acessibilidade das usuais ferramentas de modelagem, onde as tecnologias assistivas normalmente utilizadas não funcionam corretamente ao descrever os componentes gráficos da interface e dos diagramas.

O desenvolvimento de um editor de diagramas UML que seja totalmente acessível aos leitores de tela torna possível o aprendizado e desenvolvimento de diagramas por alunos cegos. A utilização dessa ferramenta faz com que o usuário deixe de depender de um tutor para confeccionar as notas de aula e modelos táteis de diagramas proporcionando independência para o aluno.

Por fim, além de tornar o aluno com deficiência independente, o editor permite sua inclusão, pois viabiliza sua participação em projetos disciplinares. Permite a criação e edição de diagramas pelo aluno com deficiência de maneira acessível e com funcionalidades para navegação, ao mesmo tempo, cria a ilustração nos padrões da linguagem UML que será entendida pelo aluno sem deficiência, sem necessidade de ser traduzida.

5.1 DIFICULDADES ENCONTRADAS

Durante a execução deste trabalho de pesquisa algumas dificuldades foram encontradas para que os objetivos que foram propostos fossem concretizados. A primeira dessas dificuldades foi a compreensão do código já existente. Em um código alheio, com documentação restrita, existe a dificuldade de se sintetizar as principais ideias da implementação sem o auxílio do autor.

Um dos objetivos apresentados era realizar a extensão dos recursos do editor de modo a não interferir no código fonte existente. Porém, durante o estudo do código, foi notado que a distribuição dos métodos e atributos seria um obstáculo para realizar a extensão. Assim, a alteração para elevar o grau de orientação a objeto na implementação foi indispensável, e por não ser prevista, causou atraso na execução do cronograma.

Devido à estrutura da implementação existente e das modificações necessárias, a opção de adicionar um diagrama que fosse configurável pelo usuário se tornou inviável. Essa opção exigiria do programa manter existente todas as formas gráficas possíveis para um vértice e todas as formas gráficas possíveis para diferentes tipos de arestas, além de não ser possível dar a opção ao usuário de inserir as regras de negócio do diagrama, como restrições nas associações entre vértices.

5.2 TRABALHOS FUTUROS

O trabalho de pesquisa científica se caracteriza pela aquisição de conhecimento, descoberta de novos fatos, dados, relações ou leis (ANDER-EGG, 1978). A aplicação de outros métodos de avaliação de acessibilidade traria novos dados para o conhecimento sobre a real acessibilidade oferecida durante o uso do aplicativo.

A extensão dos recursos para os diagramas mais complexos e menos utilizados da UML se torna um passo mais simples após as modificações realizadas, além de outros diagramas utilizados durante o projeto de um software como o Diagrama Entidade-Relacionamento.

Para aumentar o grau de interação do usuário com deficiência visual com os colegas seria desejável implementar meios de importar arquivos no formato XMI de outros editores de diagramas. Desse modo o editor GEDE poderá disponibilizar a acessibilidade para diagramas que não foram confeccionados por sua interface.

REFERÊNCIAS

ALVES, D.D. **Acessibilidade no desenvolvimento de software livre.** 2011. 134 f. Dissertação (Mestrado em Ciência da Computação) – Faculdade de Computação, Universidade Federal de Mato Grosso do Sul, Campo Grande, 2011.

ANDER-EGG, E. **Introducción a las técnicas de investigación social:** para trabajadores sociales. 7. ed. Buenos Aires: Humanitas, 1978.

BLENKHORN, P., EVANS, D. G**. Using speech and touch to enable blind people to access schematic diagrams.** J. Netw. Comput. Appl. 21, 1 (Jan.), 17-29, 1998.

BROWN, A., PETTIFER, S., STEVENS, R. **Evaluation of a non-visual molecule browser.** In: Proceedings of the 6th international ACM SIGACCESS Conference on Computers and Accessibility, pp. 40-47. ACM, New York, NY, 2004.

COHEN, R. F., MEACHAM, A., SKAFF, J. **Teaching graphs to visually impaired students using an active auditory interface.** SIGCSE Bull. 38, 1 (Mar. 2006), 279-282, 2006.

DIAS, C. **Usabilidade na Web: Criando portais mais acessíveis**. 2 ed. Alta Books, Rio de Janeiro, 2006.

FAIN, W. B.; FOLDS, D. **Accessibility evaluation methodology.** 2001. Georgia institute of technology. Disponível em: < http://accessibility.gtri.gatech.edu/aem/ > Acesso em: 20 mai. 2013.

FONSECA, J. J. S. **Metodologia da pesquisa científica**. FORTALEZA: UEC, 2002.

FOWLER, M. **UML Distilled: A Brief Guide to the Standard Object Modeling Language**. 3 ed. Addison-Wesley, New York, 2003.

GAMMA, E., HELM, R., JOHNSON, R., VLISSIDES, J. **Padrões de Projeto:** Soluções reutilizáveis de software orientado a objetos. Porto Alegre: Ed. Bookman, 2000.

GIL, A. C**. Como elaborar projetos de pesquisa**. 4. ED. SÃO PAULO: ATLAS, 2007.

GNOME. **Orca, a free, open source, flexible and extensible screan reader**. 2005. Disponível em: < https://projects.gnome.org/orca/ > Acesso em: 23 jan. 2013.

GUEDES, G. T. A. **UML 2:** uma abordagem prática. São Paulo: Ed. Novatec, 2009.

HOGETOP,L., SANTAROSA, L.M.C. **Tecnologias Assistivas / Adaptativas: Viabilizando a acessibilidade ao potencial individual.** Disponível em: <http://www.nied.unicamp.br/~proinesp >. Acesso em: 16 jul. 2013.

HORSTMANN, M., LORENZ, M., WATKOWSKI, A., IOANNIDIS, G., HERZOG, O., KING, A., EVANS, D. G., HAGEN, C., SCHLIEDER, C., BURN, A., KING, N., PETRIE, H., DIJKSTRA, S., CROMBIE, D. **Automated interpretation and accessible presentation of technical diagrams for blind people.** New Review of Hypermedia Multimedia 10, 2 (Dec.), 141-163, 2004.

IBM CORPORATION. **Software Checklist.** 2013. Disponível em: < http://www-03.ibm.com/able/guidelines/software/swsystemset.html#reqtechjava > Acesso em: 23 mai. 2013.

_____. **Documentation Checklist.** 2012. Disponível em: < http://www-03.ibm.com/able/guidelines/documentation/accessdoc.htm l> Acesso em: 23 mai. 2013.

KENNEL, A. R. **Audiograf**: a diagram-reader for the blind. In: Proceedings of the Second Annual ACM Conference on Assistive Technologies, pp. 51-56. ACM, New York, NY, 1996.

KING, A., BLENKHORN, P., CROMBIE, D., DIJKSTRA, S., EVANS, D. G. and WOOD, J. **Presenting UML Software Engineering Diagrams to Blind People**. In: Proc. 9th Int. Conf. on Computers Helping People with Special Needs, p 522-529, 2004.

MAIA M. **Processos bottom-up e top-down no rastreamento ocular de imagens.** Veredas on-line, Juiz de fora, p. 01-07, fev. 2008.

METATLA, O., BRYAN-KINNS, N., STOCKMAN, T. **Diagrams As Sonified Trees:** The Design and Implementation of Auditory UML. In: Poster presentation at Haptic and Audio Interaction Design, First International Workshop, HAID 2006, Glasgow, UK, 2006.

MICROSOFT CORPORATION. **Testando a acessibilidade de seu aplicativo**. 2013. Disponível em: < http://msdn.microsoft.com/pt-br/library/windows/apps/hh452726.aspx > Acesso em: 21 mai. 2013.

MINISTÉRIO DAS COMUNICAÇÕES. **Leitor de telas.** Disponivel em: < http://www.mc.gov.br./component/content/article/170-sem-categoria/22727-leitor-de-telas > Acesso em: 18 jan. 2013.

NIELSEN, J. **Heuristic evaluation.** In: Nielsen, J.; Mack, R. L. (Ed.). Usability Inspection Methods. New York: John Wiley & Sons, 1994.

NV ACCESS. **NVDA scream reader.** 2013. Disponível em: < http://www.nvaccess.org > Acesso em: 15 jan. 2013.

ORACLE. **Java Access Bridge Installation and Application Developer's Guide Release 2.0.2 for Microsoft Windows (32-Bit and 64-Bit).** 2011. Dísponivel em:< http://docs.oracle.com/javase/accessbridge/2.0.2/introduction.htm > Acesso em: 05 jun. 2013.

PREECE, J.; ROGERS, Y.; SHARP, H. **Interaction Design: Beyond Human-Computer Interaction.** New York: John Wiley & Sons, 2002.

RATIONAL SOFTWARE CORPORATION. **Rational Unified Process:** Artefatos. Disponível em: <http://www.wthreex.com/rup/process/artifact/ovu_arts.htm> Acesso em: 11 jan. 2013.

ROTARD, M., KNÖDLER, S., ERTL, T. **A tactile web browser for the visually disabled**. In: Proceedings of the Sixteenth ACM Conference on Hypertext and Hypermedia, pp. 15-22. ACM, New York, NY, 2005.

SANTOS, L. G., BANDEIRA, A. L. M., PANSANATO, L. T. E., PAIVA, D. M. B. **Recursos de Acessibilidade para Auxiliar a Navegação de Estudantes Cegos em um Editor de Diagramas**. In: Simpósio Brasileiro de Informática na Educação. Rio de Janeiro, 2012

SILVA, L. H. C.; PANSANATO, L. T. E. **Uma Ferramenta para Auxílio à Apresentação e Edição de Diagramas**. In: Anais do VI Encontro de Atividades Científicas. Londrina: UNOPAR, 2002.

SILVA, C. E. ; PANSANATO, L. T. E. ; FABRI, J. A. **Ensinando Diagramas UML para Estudantes Cegos**. In: XVIII Congreso Iberoamericano de Educación Superior en Computación, 2010, Asunción. Anales CLEI 2010 del XXXVI Conferencia Latinoamericana de Informática. Asunción: Faculdad Politécnica - Universidad Nacional de Asunción, Universidad Autónoma de Asunción, 2010.

SILVA, L. H. C. **Editor de Diagramas.** Código fonte – Universidade Tecnológica Federal do Paraná, Cornélio Procópio, 2012.

TANAKA, E. H. **Método baseado em heurísticas para avaliação de acessibilidade em sistemas de informação.** 2009. 190 f. Tese (Doutorado em Ciência da Computação) – Instituto de Computação, Universidade Estadual de Campinas, Campinas, 2009.

TIRESIAS.ORG. **Checklists.** 2009. Disponível em: < http://www.tiresias.org/research/guidelines/checklists/index.htm > Acesso em: 22 mai. 2013.

Apêndice A – Cronograma

O cronograma apresentado na Figura 24 refere-se às atividades realizadas durante o desenvolvimento deste trabalho. A modelagem gerou as abstrações de diagramas de classes para as modificações necessárias. A fase de testes foi executada juntamente com as modificações, assim cada modificação ou extensão realizada era logo testada de corrigida.

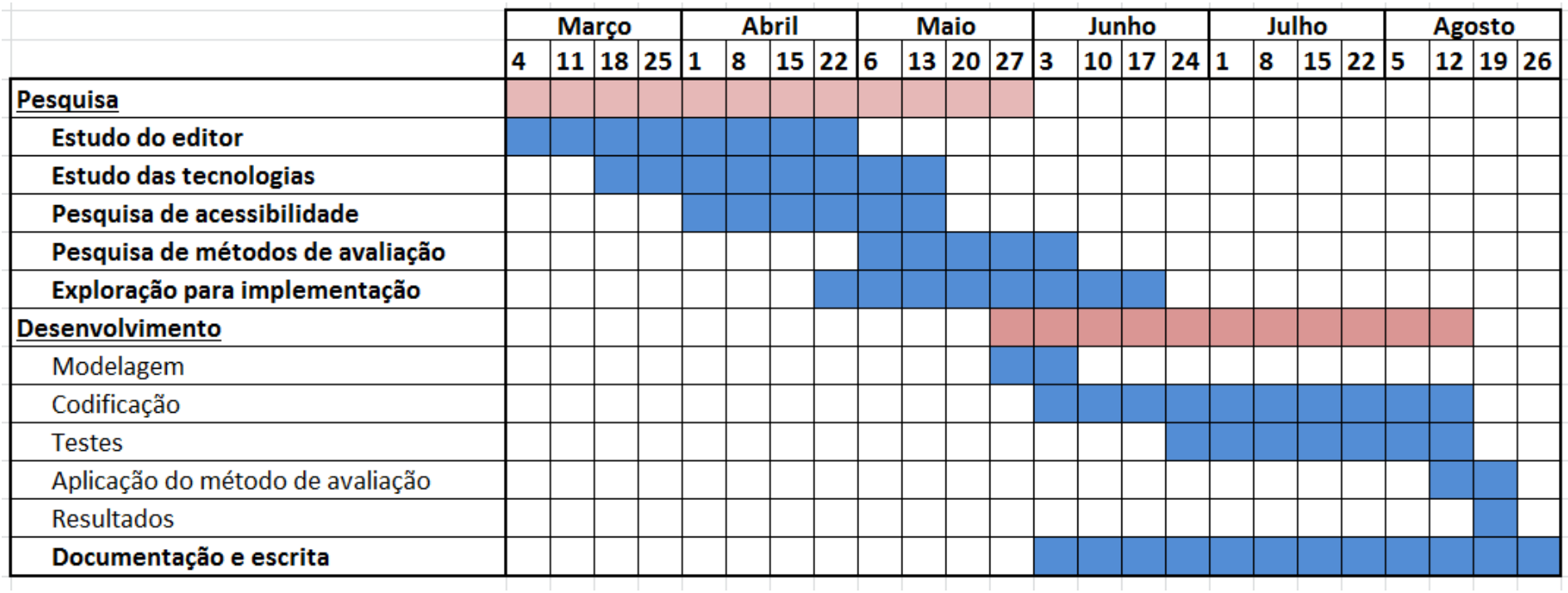


Figura 24 - Cronograma executado

O cronograma apresentado na Figura 25 é o cronograma que foi previsto para a execução do trabalho.

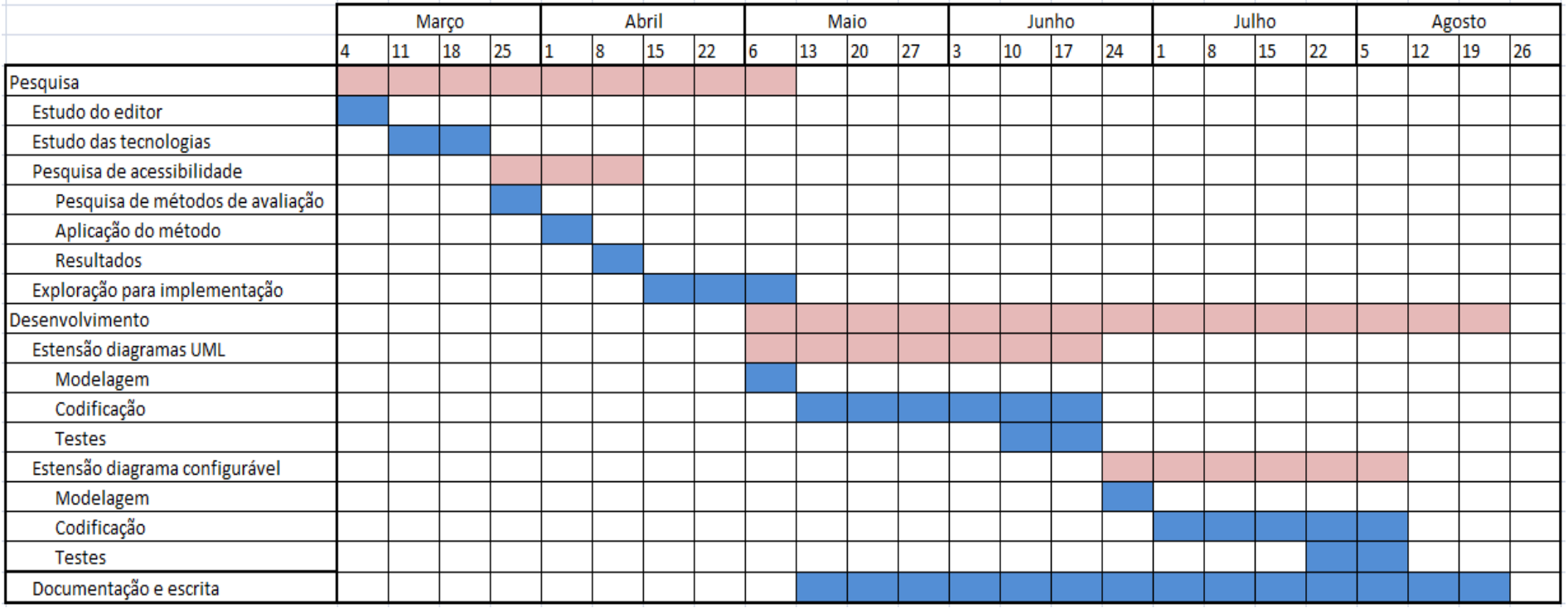


Figura 25 - Cronograma previsto