Franciscarlos Nascimento Ávila Pereira

Modelagem de um Editor Gráfico 2D Orientado a Objeto

Monografia de Graduação apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras como parte das exigências do Curso de Ciência da Computação para obtenção do título de Bacharel em Ciência da Computação.

Orientador
Prof. Bruno de Oliveira Schneider

Lavras
Minas Gerais - Brasil
2003

Franciscarlos Nascimento Ávila Pereira

Modelagem de um Editor Gráfico 2D Orientado a Objeto

Monografia de Graduação apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras como parte das exigências do Curso de Ciência da Computação para obtenção do título de Bacharel em Ciência da Computação.

Aprovada em *16 de junho de 2003*

Prof. Renato de Souza Gomes

Prof[a]. Olinda Nogueira Paes Cardoso

Prof. Heitor Augustus Xavier Costa

Prof. Bruno de Oliveira Schneider
(Orientador)
Lavras
Minas Gerais - Brasil

A minha mãe e ao meu pai que sempre me apoiaram.

Agradecimentos

Aos professores que me incentivaram sempre, especialmente ao prof. Bruno de Oliveira Schneider.

Modelagem de um Editor Gráfico 2D Orientado a Objeto

Este trabalho propõe a modelagem de um editor gráfico 2D multiplataforma e extensível. Para isso, os elementos fundamentais de um *software*, a saber, atributos e métodos são pensados de forma a atender os conceitos principais da orientação a objetos, ou seja, reusabilidade, extensibilidade, herança, polimorfismo, etc. São propostas classes a serem implementadas além de métodos a serem usados pela interface do programa para criar figuras usando dados advindos do *mouse*. As classes propostas estão hierarquizadas para possibilitar uma boa extensibilidade e reusabilidade. Foram analisados alguns editores gráficos para que modelagem consiga abranger o máximo possível dos recursos normalmente encontrados em programas desse tipo. As propostas contidas nesse trabalho se restringem à descrição de métodos e atributos, sendo independente da funcionalidade de qualquer biblioteca, linguagem de programação ou sistema operacional, facilitando o desenvolvimento de um programa multiplataforma.

Modeling of a Object Oriented Graphic 2D Editor

This work proposes modeling of a extensible, multiplataform 2D graphics editor. In the proposed model, the basic elements of a sofware (attributes and methods) are represented in a way to comply with the main concepts of object orientation, that is, reusability, extensibility, inheritance, polymorphism, etc. Object classes are proposed in a hierarchic way to achieve good extensibility and reusability. Some popular 2D graphic editors were analised so that the modelling could enclose as such common features as possible. The ideas proposed in this work are restricted to methods and attributes, therefore being independent of any programming library, language or operating system, in order to ease the implementation of a multiplataform software.

Sumário

Lista de Figuras

Capítulo 1

Introdução

Atualmente quase não existe documentação para implementação sobre um editor 2D (estruturas de dados usadas, objetos e classes, etc). Dessa forma seria bom ter algo que pudesse orientar o programador na construção de um editor gráfico (um editor para desenhar gráficos).

O trabalho tem como proposta desenvolver uma modelagem de um editor gráfico Orientado a Objeto e Extensível. A orientação a objeto dá suporte para modelar um editor gráfico que permita acrescentar novos recursos ao editor gráfico sem contudo ter que redefinir praticamente toda a estrutura de dados já criada. Do contrário, uma implementação de um editor gráfico, sem um estudo prévio profundo das estruturas de dados, dados e métodos à luz dos conceitos de orientação a objetos, gera segundo estudos e técnicas de engenharia de software e também através da experiência prática uma implementação ruim em diversos aspectos. Alguns dos aspectos mais importantes é o alto índice de manutenção exigido no programa e difícil extensibilidade.

Para ressaltar a importância de se fazer um projeto orientado à objeto é importante verficar o que diz Deitel:

"Construir software de forma rápida, correta e econômica é uma tarefa difícil de atingir, ainda mais em tempos que exigem software cada vez mais complexo e poderoso" [2].

Capítulo 2

Referencial Teórico

A seguir, serão definidos termos freqüentes e conceitos importantes nesse trabalho.

Polígono é "uma figura plana formada por uma linha poligonal fechada" [3].

Widgets é "um símbolo gráfico de uma interface que possibilita a interação entre o usuário e o computador" [6].

Toolkit é "conjunto de ferramentas, conjunto de ferramentas para o desenvolvimento, grupo de programas e rotinas usadas como base para programação em um novo sistema ou de um hardware especial" [6].

Como já foi mencionado, o trabalho do editor 2D busca desenvolver uma modelagem à luz da orientação à objetos, uma vez que a orientaçãoà objetos dispõem de uma série de vantagens que vão de encontro com os objetivos propostos na modelagem do Editor Gráfico 2D. As vantagens principais de se usar orientação a objetos são:

1. Fazer um programa de fácil correção e aproveitamento de código.

2. Programas construídos utilizando a orientação objeto são mais fáceis de entender, modificar e corrigir.

Segundo [2], "como o mundo é constituído de objetos que interagem entre si, programar orientado a objetos é um processo mais natural que programar não orientado à objeto, pois é similar à maneira com que o programador tem percepção dos objetos do mundo real. Já na programação não orientada a objetos unidades do programa não correspondem, mas "imitam" a unidades do mundo real".

Nos programas utilizando programação não orientado a objetos são comuns o reinicio do projeto a fim de poder colocar um novo recurso.

Conforme [2], “empresas de software estima-se que 80% dos custos de software não estão associados com os esforços originais de desenvolvimento do software, mas sim com a evolução continuada e manutenção deste software ao longo de sua vida útil”.

Atualmente há uma demanda cada vez maior na utilização de software gratuitos, o que implica uma sensível economia para os usuários. Trabalhando com editores atuais, são constatados alguns erros e tais editores poderiam ter alguns novos recursos a fim de facilitar a criação de figuras específicas. Nesse sentido, o trabalho de modelagem do editor gráfico vem contribuir também para a solução dos dois itens anteriormente mencionados. No que diz respeito a demanda cada vez maior da utilização do software gratuito o trabalho tem como um de seus objetivos oferecer uma diretriz para o desenvolvimento futuro de um editor gráfico de código aberto seguindo a licença GPL *(Gnu Public License)*. Quanto às facilidades para a criação de figuras específicas pode ser implementada sem prejuízo do sistema pois a modelagem foi feita à luz da orientação a objeto, evitando assim que seja reinventada novamente a “roda”. O motivo de fazer uma modelagem de um editor 2D vetorial é devido a fato que em um editor vetorial as partes da figura podem ser facilmente alteradas, uma vez que a figura é composta de elementos geométricos tais como: círculo, quadrado, triângulo, ponto, reta, etc. Além disso, foi proposto o desenvolvimento do Editor vetorial, pois é mais complicado pensar nas estruturas de dados e objetos presentes em um Editor vetorial do que em um editor matricial. No matricial tem-se basicamente uma matriz de pixels enquanto no vetorial temos vários elementos geométricos que se relacionam para formar uma nova figura.

Existem dois tipos de representação de imagens:

Vetorial

1. Permite uma representação do desenho em poucos bytes;
2. Permite alterar as características da imagem de forma rápida e eficiente;

Matricial

1. A maioria dos dispositivos existentes no mercado são matriciais;
2. Úteis em figuras ricas em detalhes;
3. Exige um quantidade maior de memória do que a representação vetorial.
4. Dispositivos matriciais são mais baratos.

Dessa forma, é importante construir um editor Gráfico 2D a partir de estruturas de dados do elementos geométricos.

2.1 Histórico

"A orientação a objeto começou em meados da década de 60. As facilidades no desenvolvimento de software começa com a programação estrutura mas somente com a orientação objeto que foi amplamente utilizada na década de 80 e consolidada na década de 90 a sua importância no desenvolvimento de software" [2].

2.2 Definição de Objeto

"Objetos são essencialmente componentes de software reutilizáveis que modelam coisas do mundo real, em outras palavras, é a concretização do gabarito (casas feitas a partir da mesma planta)". Objetos estão em todo lugar, basta dar uma olhada em nossa volta para ver animais, prédios, carros, computadores, cadeiras, etc. Nesse sentido, as pessoas já estão acostumadas a pensar em objetos, pois desde cedo aprendem sobre os objetos do mundo real, observando seus atributos e comportamentos. Dessa forma, utilizar orientação a objetos facilita escrever programas de computador. Os desenvolvedores de software estão descobrindo que usar uma abordagem de implementação e projeto modulares, orientada a objetos, pode tornar os grupos de desenvolvimento de software muito mais produtivos do que é possível usando-se técnicas de programação anteriormente populares, como a programação estruturada. Essa facilidade de criam unidades de software com significado próprio é porque objetos são um esquema de empacotamento. Como o projeto propõem o desenvolvimento da modelagem de um editor gráfico orientado a objetos, é necessário conhecer a respeito desse paradigma ou estilo, quais as vantagens, etc. "Seres humanos pensam em termos de objetos e possuem uma habilidade de abstração e percepção muito grande quando se trata de objetos (figuras), por exemplo: consegue-se rapidamente compreender o significado de uma imagem o que não acontece com textos". Dessa forma modelar um problema do mundo real é pensar nos atributos e comportamentos mais importantes que uma objeto possui, visto que um objeto pode ter vários comportamentos e atributos. Na modelagem de um problema do mundo real, tem-se que verificar a relação desse objeto analisado como os objetos que interagem com ele. A OOP (programação orientada a objetos encapsula (esconde) dados (atributos) e métodos (comportamentos). A definição teórica das características e comportamentos do objetos são

feitos na classe. Assim, o objeto é uma instância da classe, ou seja, o objeto concretiza as definições da classe. Por exemplo, classes estão para objetos, assim como, plantas arquitetônicas estão para casas. “Pode-se assim criar várias casas a partir de uma única planta da mesma forma vários objetos a partir de uma únicas classe” [2].

2.3 Reutilização de Herança

Reutilização de herança refere-se ao uso de herança em sua aplicação para aproveitar o comportamento implementado em classes existentes.

Herança é um dos conceitos fundamentais de orientação a objetos, permitindo que você modele relacionamentos "é um", "é como"e "é do tipo". Por exemplo, os objetos quadrado, retângulo, círculo reutilizarão todo o comportamento implementado na classe Entidade. “A vantagem da reutilização de herança é que você tira proveito de comportamento previamente desenvolvido, o que diminui tanto o tempo de desenvolvimento como o custo da aplicação” [7].

2.4 Reutilização de Componente

“Reutilização de componentes refere-se ao uso de componentes pré-construídos e totalmente encapsulados no desenvolvimento de suas aplicações. Componentes são tipicamente auto-suficientes e encapsulam um único conceito. A reutilização de componentes difere da reutilização de código pelo fato de que você não tem acesso ao código fonte. Difere-se da reutilização de herança, porque não faz criação de subclasses. Exemplos comuns de componentes são os *Java Beans* e componentes ActiveX. Há muitas vantagens na reutilização de componentes. Primeiro, ela oferece um escopo maior de reusabilidade do que o da reutilização de código ou de herança, porque os componentes são auto-suficientes - você literalmente os “pluga” e eles fazem o trabalho. Segundo, a utilização ampla de plataformas comuns, como o sistema operacional Win32 e o *Java Virtual Machine*, provêem um mercado amplo o bastante para que terceiros criem e vendam componentes para você a baixo custo. A principal desvantagem da reutilização de componente é que os componentes são pequenos e encapsulam um único conceito e você acaba precisando de uma grande biblioteca deles. O caminho mais fácil para o trabalho com componentes é começar com objetos (*widgets*) da interface de usuário. Por exemplo, barras de deslocamento, componentes gráficos e botões gráficos (para nomear

alguns). Mas não se esqueça que há muito mais em uma aplicação do que a interface de usuário. Você pode ter componentes que encapsulem recursos do sistema operacional, tal como acesso à rede, ou que encapsulem recursos de persistência, tais como componentes que acessam bancos de dados relacionais. Se você está construindo seus próprios componentes, tenha o cuidado de que eles façam somente uma coisa. Exemplificando, analise se, um componente de interface de usuário para edição de endereços de superfície é bastante reutilizável, porque você pode usá-lo em várias telas de edição. Um componente que edita endereços de superfície, endereços de e-mail e um número de telefone já não é tão reutilizável pois, não há muitas situações onde você queira todos esses três recursos simultaneamente. Em vez disso, é melhor construir três componentes reutilizáveis e usar cada um quando necessário. Quando um componente encapsula apenas um conceito, é um componente coeso. Por exemplo: num editor gráfico 2D pode-se futuramente criar novas ferramentas (componentes) que serão úteis no desenvolvimento de trabalhos " [7].

2.5 Conceitos de Orientação a Objetos

"A programação orientada a objetos tem como principal objetivo atingir um desenvolvimento interativo e incremental, criação rápida de protótipos, códigos reutilizáveis e extensíveis através do encapsulamento dos dados e uma modelagem do problema a ser solucionado. Donalt Knuth defende a seguinte adição: (Programas = Algoritmos + estruturas de dados). Muito mais do que a conotação matemática é importante observar que tanto dados quanto código, na programação, encontram-se em um patamar de mesmo nível " [5].

"Com isso, o conceito de encapsulamento de dados começa a sugerir a idéia de que dados e código não apenas se encontram em um mesmo nível mas também que ambos são a mesma coisa do ponto de vista de objetos. Porém, é importante lembrar que embora tratados de forma igual, apenas o código de um objeto pode manipular os dados contidos nele." [5] "Um segundo conceito importante em orientação a objetos (que engloba o anterior) acrescenta o mecanismo de herança. Isto representa a possibilidade de se reaproveitar classes já existentes para a elaboração de novas classes, sendo necessário para tanto apenas definir as diferenças. Este mecanismo de herança dá um grande impulso à POO (Programação Orientada a Objetos) visto que ele oferece recursos para reutilização de código, onde as classes representam grupos de elementos reais e os objetos representam elementos com existência física individual. A vantagem disso é justamente a modelagem e

a reutilização de códigos (classes) já testados, o que aumenta a confiabilidade do código e reduz o nível de manutenção" [5].

A programação orientada a objetos está apoiada em determinados conceitos para o bom desenvolvimento de projetos. Nas seções seguintes serão indicado com mais detalhes.

2.5.1 Objetos

"Na programação orientada a objeto, um objeto é qualquer coisa, real ou abstrata, sobre a qual armazena-se dados e operações que manipulam os dados" [5].

2.5.2 Classes

"Classe é a implementação de um tipo de objeto o qual contém uma estrutura de dados e métodos que especificam operações que podem ser feitas com aquela estrutura de dados" [5].

2.5.3 Métodos e Sobrecarga

"São as funções que manipulam sobre o objeto desejado, ou seja, especifica a maneira como as operações são codificadas no software. Sobrecarga é a capacidade de uma operação possuir mais do que um função-membro para sua execução" [5].

2.5.4 Herança

"A herança é a principal característica da programação orientada a objetos pois é através desse mecanismo que a classe mais especializada herda todos os métodos e atributos da classe mais geral" [5].

2.5.5 Encapsulamento

Todas as classes possuem duas interfaces, a Interface Pública e a Interface Privada. Interface Pública compreende todas as operações e propriedades que podem ser acessadas diretamente pelas demais classes. Interface Privada são aquelas operações e propriedades que podem ser acessadas apenas pela própria classe, sendo que a interface privada é protegida pelo encapsulamento.

O empacotamento de dados e operações bem como o, e o ocultamento desta informações, chama-e de encapsulamento, com os seguintes objetivos:

1. Proteger os dados de um objeto contra corrupção;

2. Protege os dados contra utilização arbitrária para que os conceitos não se tornem contraditórios.

"O encapsulamento é importante, porque ele separa o comportamento do objeto, da maneira como o objeto é implementado" [5].

2.5.6 Independência

"Cada instância criada é totalmente independente da outra, mesmo que venham da mesma classe. O Polimorfismo é a capacidade de assumir diversas formas. No contexto da programação orientada a objetos, a palavra diz respeito à capacidade de uma função-membro assumir diferentes formas com o mesmo nome. Utilizando o polimorfismo, o programa torna-se de fácil manutenção. Por exemplo, se posteriormente for necessário alterar alguma função somente ela será alterada. Todo o resto do código do programa permanecerá igual" [5].

Capítulo 3

Metodologia

O projeto foi desenvolvido no laboratório de computação de Lavras, propondo a modelagem de um Editor Gráfico 2D. Para isso, foram analisados alguns editores que já se encontram no mercado: *Corel Draw 9, xfig, Open Office*. Foram utilizados também: conhecimentos adquiridos na matéria de Computação Gráfica, consultas em [Foley, 1996] e no programa de projeção perspectiva e paralela sob orientação do professor Bruno. Inicalmente foram verificadas as características dos objetos presentes nestes editores, bem como os comportamentos. Paralelamente a essa análise foram surgindos relacionamentos possíveis do objeto em questão como os diversos tipos de objetos existentes no Editor Gráfico. A análise das características e comportamentos dos objetos dos editores 2D foi feito em editores de plataforma Windows e Linux, uma vez que trabalho tem como proposta desenvolver a modelagem de um editor 2D multiplataforma e extensível. Para satisfazer a extensibilidade proposta na modelagem do editor 2D, foram utilizados os conceitos de orientação a objeto, uma vez que os objetos são entidades independentes. [1] [2] [3]

[1] *Corel Draw 9.0* Editor Gráfico vetorial 2D para plataforma Windows.

[2] *Xfig 3.2* Editor Gráfico 2D para plataforma Linux.

[3] *Open Office* pacote de escritório com Editor Gráfico 2D para plataforma Linux.

Capítulo 4

Proposta de Implementação do Editor Gráfico

4.1 Descrição dos Objetos do Editor Gráfico

Os objetos do editor 2D devem ter métodos para inserir, consultar e alterar o estado interno.

4.1.1 Características Gerais

Analisado várias características de diferentes figuras, concluiu-se que os atributos **estilo da linha, cor da linha, espessura** estão presentes em todos os tipos de figura. Com relação ao estilo da linha, é útil mudar a linha de continua para tracejada. Dependendo do valor escolhido a linha será continua ou terá um espaçamento diferenciado. Além disso, tem um outro atributo que permite ao usuário aumentar ou diminuir o espaçamento das linhas tracejadas além das já disponíveis como padrão. Quanto a cor da linha, esse atributo e o atributo cor de preenchimento que facilita a implementação de uma ferramenta para pegar o valor contido tanto na variável a cor da linha como na variável cor de preenchimento. Essa cor pode ser aplicada em outra figura por meio de uma outra ferramenta. Assim, quando o usuário clicar na linha, será utilizado o valor contido na variável cor da linha. Por fim a espessura evita criar várias ferramentas para desenhar reta com espessuras diferentes.

4.1.2 Ponto

A classe ponto é a classe básica do editor gráfico, pois todas as outras classes dependem dela para existir. Toda figura por mais simples que seja, é formada por pontos, o qual tem como características **coordenadas x e y e identificador**. Quanto as coordenadas x e y, são importantes para definir a localização exata do ponto na tela. Além disso, a figura sendo formada por pontos (pixels) permite que seja selecionada os pontos que se deseja alterar. Com relação ao identificador, pode-se predefinir um valor para o atributo ficando assim fácil saber se a figura é um círculo e não é uma outra figura qualquer. Exemplos, predefinir 1 para quadrado, 2 para círculo. Toda vez que encontrar o numero 1, identificará o quadrado. E ao se deparar com o 2, identifica-se um círculo.

4.1.3 Reta

O objeto da classe reta é um objeto bem especializado. Dentro das características mais comuns dessa classe tem-se **cor da linha, tipo de ligação, modelo da flecha, tipo da flecha, espessura da linha, modo (tracejado, curva, reta) e tipo**. Com relação a cor da linha esse atributo e o atributo cor de preenchimento que facilita a implementação de uma ferramenta para pegar a cor da linha. Através de uma ferramenta pode-se aplicar essa cor armazenada em uma outra figura. Assim, quando o usuário clicar na linha, será utilizado o valor contido na variável cor da linha. Quanto ao tipo de ligação, esse atributo é útil pois permite ligar duas ou mais linhas a fim de ser criado um novo objeto do tipo seqüência de retas. Já o modelo da flecha tem 4 valores, os quais permitem ao usuário modificar as extremidades da reta. Se o usuário escolher o valor 1 para o atributo, é obtida a reta com as duas extremidades com flechas. Senão se o usuário escolher o valor 2 para o atributo é desenhado a reta sem flecha nas extremidades. Senão se o usuário escolher o valor 3 para o atributo, é desenhada a reta com flecha somente na extremidade da esquerda. Senão, se o usuário escolher o valor 4 para o atributo é desenhado a reta com flecha somente na extremidade da direita. Com relação ao tipo de flecha, permite ao usuário escolher alguns tipos de formatos de flechas. Esse atributo, bem como o anterior são importantes, pois se o usuário modificar o formato da flecha bem como indicar em qual ou quais extremidade(s) a flecha será feita, o usuário não perde tempo desenhado a reta e a flecha separadamente. Quanto ao atributo espessura da linha evita criar várias ferramentas para desenhar reta com espessuras diferentes. Com relação ao modo (tracejado, curva, reta), permite ao usuário passar de um tipo linha reta para linha curva ou vice-versa facilmente sem

ter que desenhar uma nova linha com o tipo desejado. Por fim o atributo tipo dá mais flexibilidade em formas de desenhar uma linha. O usuário pode desenhar um linha onde a mesma cresce nos dois sentidos, ou seja quando o usuário clicar com a setinha do mouse utilizando o botão esquerdo do mouse, segurar é arrastar o mouse, a linha será desenhada nos dois sentidos a medida que o usuário arrasta o mouse. Ele tem também a opção de desenhar a linha que cresce a partir do ponto indicado pelo clique do mouse até o ponto indicado pelo segundo clique do mouse. Existe também, a opção de desenhar linhas à mão-livre permitido ao usuário criar linhas com mais liberdade de formas.

4.1.4 Círculo

O objeto da classe cônica é um objeto bem especializado. As características principais são: **raio, diâmetro, cor da linha, espessura tipo de preenchimento, preenchimento padrão e identificador**. Quanto ao raio é importante pois, é necessário implementar um método que, quando aplicado no círculo, aumenta o seu diâmetro proporcionalmente em todas as direções, uma vez que é necessário ter um ponto de referência. Já o diâmetro é interessante pois, com ele pode-se saber a distância do raio até o contorno do círculo. O método mencionado acima altera o valor do atributo diâmetro. Com relação a cor da linha e espessura, suas vantagens são as mesmas citadas nos atributos gerais dos objetos. Para acrescentar o atributo espessura evita criar várias ferramentas para desenhar círculos com espessuras diferentes. No que diz respeito ao atributo tipo de preenchimento dá a opção para o usuário escolher o tipo de preenchimento mais adequado para a figura em questão. De outro modo, o atributo preenchimento padrão tem como objetivo facilitar o preenchimento rápido da figura. Dessa forma, o programa dispõe de tipos de preenchimento já prontos, os quais podem ser utilizados. Por fim, o com base no atributo identificador fica fácil saber se a figura é ou não um círculo.

4.1.5 Retângulo

O objeto da classe polígono possui características e métodos bem concretos. Classe concreta é aquele que se encontra na folha da árvore de subclasses, ou seja a classe já adquiriu um nível de especialização dos atributos e métodos muito grande. Os atributos principais são: **largura, altura, cor da linha, espessura e identificador.** Os atributos largura e altura podem ser utilizados por vários métodos matemáticos. Quanto a cor da linha, facilita a implementação de uma ferramenta para pegar a cor da linha. Essa cor coletada pode ser aplicada em uma outra figura. Assim,

quando o usuário clicar na linha, será utilizado o valor contido na variável cor da linha. Com relação ao atributo espessura, evita criar várias ferramentas para desenhar retângulos com espessuras diferentes. Por fim o identificador é útil para diferenciar essa figura das outras existentes.

4.1.6 Elipse

Os principais atributos do objeto elipse são: **2 raios, cor da linha, espessura, tipo de preenchimento, preenchimento padrão, cor de preenchimento e identificador**. Os atributos que definem os dois raios permitem implementar métodos para alterar as formas da elipse baseado na alteração dos raios. Com isso, pode-se ter por exemplo: elipse mais "gordinhas" ou mais "achatadas". Já o atributo cor da linha e o atributo cor de preenchimento facilitam a implementação de uma ferramenta para pegar a cor da linha. Essa cor coletada pode ser aplicada em uma outra figura. Assim, quando o usuário clicar na linha será utilizado o valor contido na variável cor da linha. Com relação a espessura permite fazer tipos de desenhos que seriam difíceis de desenhar, como por exemplo uma roda de carro mostrado uma coroa de cor preta indicando a parte da borracha da roda. De outra forma teria que ser feito dois círculo concêntricos e a parte intermediária ser colorida de cor preta. Quanto ao tipo de preenchimento, dá a opção para o usuário escolher o tipo de preenchimento mais adequado para a figura em questão. Com relação ao preenchimento padrão, facilitar o preenchimento rápido da figura. Dessa forma, o programa dispõe de tipos de preenchimento já prontos, os quais podem ser utilizados. De outro modo o atributo cor de preenchimento facilita a implementação de uma ferramenta para pegar a cor da linha. Essa cor coletada pode ser aplicada em uma outra figura. Assim quando o usuário clicar na cor de preenchimento será utilizado o valor contido na variável cor da linha. Por fim o identificador é útil para diferenciar essa figura das outras existentes.

4.1.7 Figura

Na classe figura se encontra todos os métodos relacionados com o menu do editor gráfico, pois essa contém características é comportamentos bem gerais do editor gráfico. Assim a classe figura é pai de todas as outras figuras. Os atributos mais importantes são: **nível e tamanho atual**. O atributo nível é necessário, pois permite implementar o método mover a figura para frente, para trás, etc. A idéia é criar um sistema parecido com camadas onde cada figura estaria em uma camada. Com relação ao atributo tamanho atual, permite implementar um método que restaurará

o tamanho anterior da figura. Por outro lado, os métodos são mudar escala, rotacionar, transladar, projetar paralelo. O método mudar escala é interessante pois permite implementar a ferramenta *zoom*. Obs A mudança de escala pode ser feita com um ponto fixo do lado oposto a ponto de alteração. Já o método rotacionar permite cria figuras que possuem pouca alteração entre si. Com relação ao transladar permite implementar movimento nas figuras ou ferramentas para recortar, copiar a figura de um local da tela para outro. Quanto ao método projetar permite criar efeitos de profundidade nas figuras. Por fim, o método inserirOrdenado é interessante, pois é útil para implementar o método innserir um objeto ponto. O método inserir um objeto ponto por sua vez permite ao usuário uma versatilidade maior de formas de figuras, uma vez que o usuário pode colocar o onto na onde ele deseja alterar a forma da figura.

A lista contém um conjunto de pontos (objetos da classe ponto). A lista deve ficar na classe figura pois todo objeto instanciado da classe figura terá dentro de seu encapsulamento uma lista de pontos. Dessa forma, os objetos definidos teriam seus pontos independentes e seguros dentro do encapsulamento do objeto. Além disso, a programação orientada a objetos permite comunicação entre os objetos, permitindo com isso que figuras compartilham pontos em comum.

A classe figura tem também um ponteiro para o objeto da classe ponto, tal objeto é criado com novos valores toda vez que o usuário (clicar) com o esquerdo do mouse na aplicação. À medida que o usuário for clicando com o botão esquerdo, os pontos (vértices) da figura geométrica serão colocados em uma lista de objetos da classe ponto. Contudo, a figura será desenhada na tela quando o usuário clicar com o botão direito do mouse. A lista de pontos e o ponteiro temporário para classe ponto foram colocados na classe clFigura, porque os métodos InserirPonto e Imprimir é mais natural que fique na classe figura, uma vez que é nesta classe que ficam definidos todos os métodos relacionados com o menu do editor gráfico. Os métodos do menu do editor gráfico (ex: InserirPonto, Imprimir, rotacionar, transladar, etc) precisam ficar definidos na mesma classe que foi definido a lista de pontos e o objeto temporário, uma vez que esses métodos utilizam a lista de pontos e o objeto temporário da ponto. Além disso, a subclasse clAgrupamento precisa herda a lista de pontos, pois utiliza a mesma para guarda os pontos do objeto agrupado.

A classe clFigura tem que usar a classe matriz, pois a maioria do métodos da figura realizam as transformações geométricas baseado na operação com matrizes. Assim, quando o usuário na aplicação escolher mover (transladar) uma figura, será chamado o método Transladar da classe figura. Neste método, o programa pega

os pontos da lista de pontos que está encapsulado dentro do objeto selecionado e jogará para dentro do objeto matriz. A quantidade de deslocamento que o objeto deve sofrer é determinada no momento que o usuário soltar o mouse, depois de ter seguido os eventos (clicar + segurar / arrastar + soltar). A classe Figura tem também um método privado denominado int VerificarNumeroPontosFigura(), o qual é útil para determinar internamente o número de pontos de uma dada figura. Com isso, pode-se decidir entre preencher uma figura, se o número de pontos da figura é maior que dois ou não preencher, se o número de vértices da figura e menor ou igual a dois. Esse método é chamado depois do usuário ter desenhado a figura. O combobox contendo as opções (preenchimento, não preenchimento e preenchimento padrão), bem como o botão para cor de preenchimento que abre um janela com as cores disponíveis, só será habilitado (enabled) na janela propriedades do referente objeto caso o método interno tiver retornado um valor maior que dois logo depois de ter desenhado a figura. Baseado nisso é necessário ter mais um outro método interno chamado HabilitarBotaoPreenchimento(int);

Na classe Figura, há um método que pega a representação géometrica contida na lista é passa para a representação matricial. Para isso, o método pega a descrição geométrica do objeto que se encontra na lista de Pontos e passa como parâmetro para um método específico da biblioteca wxWindows que mostra a representação matricial da figura que o usuário quis desenhar.

4.1.8 Lista

A estrutura de dados responsável por conter a descrição geométrica da figura.

A classe Figura tem um objeto lista que é necessário para armazenar os objetos do tipo Ponto. Além disso a lista é útil pois estabelece uma relação entre os vértices da figura (ex: para indicar que o ponto p1 está ligado ao ponto p2, é inserido na lista primeiramente o ponto p1 e logo a seguir o ponto p2. Esse processo continua até que todos os pontos da figura estavam dentro da lista de Pontos).

4.1.9 Polígono

O objeto polígono da classe polgono é um tipo bem especializado. Esse objeto é bem comportado baseado nos critérios de lado e/ou ângulo.

4.1.10 Figura Aberto

A classe aberto é uma subclasse da classe Entidade. A classe Aberto tem 3 subclasses cônica, curvas, reta e seqüência de retas.

4.1.11 Cônicas

Os atributos principais da classe Cônica são **tipo de arco, raio e diâmetro**. O atributo tipo de arco permite implementar um método que dependendo do valor do atributo fechar o arco deixa-o na forma de um "queijo". Dependendo se for um outro valor abre o arco. Já o atributo raio é imprescindível, pois é necessário para implementar um método que quando aplicado no círculo aumenta o seu diâmetro proporcionalmente em todas as direções, uma vez que é necessário ter um ponto de referência. Por fim, o diâmetro é interessante, pois com ele pode-se saber a distância do raio até o contorno do círculo. O método mencionado acima altera o valor do atributo diâmetro.

4.1.12 Curvas

Os atributos mais importantes dessa classe são: **cor da linha, tipo de ligação, modelo da flecha, tipo de flecha e espessura da linha**. O atributo cor da linha e o atributo cor de preenchimento facilita a implementação de uma ferramenta para pegar a cor da linha. Essa cor coletada pode ser aplicada em uma outra figura. Assim, quando o usuário clicar na linha, será utilizado o valor contido na variável cor da linha. Já o atributo tipo de ligação é útil pois permite ligar duas ou mais linhas a fim de ser criado um novo objeto do tipo seqüência de retas. Com relação ao modelo da flecha o atributo pode assumir 4 valores, os quais permitem ao usuário modificar as extremidades da reta. Se o usuário escolher o valor 1 para o atributo, é obtida a reta com as duas extremidades com flechas. Senão, se o usuário escolher o valor 2 para o atributo, é desenhada a reta sem flechas nas extremidades. Senão se o usuário escolher o valor 3 para o atributo é desenhado a reta com flecha somente na extremidade da esquerda. Senão, se o usuário escolher o valor 4 para o atributo, é desenhado a reta com flecha somente na extremidade da direita. Quanto ao tipo da flecha permite ao usuário escolher alguns tipos de formatos de flechas. Esse atributo, bem como o atributo anterior são importantes, pois se o usuário modificar o formato da flecha ou indicar em qual ou quais extremidade(s) a flecha será feita. Assim, o usuário não perde tempo desenhado a reta e a fecha separadamente. Quanto ao atributo espessura da linha evita criar várias ferramentas para desenhar reta com espessuras diferentes. Com relação ao

modo (tracejado, curva, reta), permite ao usuário passar de um tipo linha reta para linha curva ou vice-versa facilmente sem ter que desenhar uma nova linha com o tipo desejado. Por fim o tipo (bezier, simples, etc) dá mais flexibilidade em formas de desenhar uma linha. O usuário pode desenhar um linha, onde a mesma cresce nos dois sentidos.

4.1.13 Entidade

Essa classe possui todas as características do objetos não agrupados. Os principais atributos são **intensidade de preenchimento, cantoArredondado, preenchimento padrão, número de objetos, descrição do objeto, cor de preenchimento, cor da linha e lista de pontos**. Em primeiro lugar o atributo intensidade de preenchimento permite implementar um método para alterar o intensidade de preenchimento. Com isso, dá maior versatilidade no tipo despreenchimento, bem como, nas possibilidades de cor. Dessa forma evita ter um número muito grande de cores e consequentemente sobrecarregar o editor gráfico. O atributo preenchimento padrão, tem como objetivo facilitar o preenchimento rápido da figura. Dessa forma, o programa dispõe de tipos de preenchimento já prontos, os quais podem ser utilizados. O atributo cantoArredondado permite alternar rapidamente entre figura com cantos arredondados para figuras sem canto arredondado. O atributo número de objetos é usado para saber se a figura é não um objeto agrupado. O usuário desenha a primeira figura e depois a segunda. Feito isso, o usuário seleciona a duas figuras é as agrupa. Nesse momento, a variável número do objeto agrupado passa a valer dois. Futuramente, não precisar ficar verficando se a figura é um objeto agrupado, basta verificar o valor da variável número de objetos. Quanto o atributo descrição do objeto, tem como objetivo informar o usuário algumas características e comentário a respeito da figura. Com relação a cor de preenchimento, facilita a implementação de uma ferramenta para pegar a cor da linha. Essa cor coletada pode ser aplicada em uma outra figura. Assim, quando o usuário clicar na cor de preenchimento será utilizado o valor contido nessa variável. A vantagem do atributo cor da linha é a mesma citada nas características gerais dos objetos. Por fim, o atributo lista de objetos é responsável para organizar e armazenar os pontos das figuras do editor gráfico. Como exemplo, existe os métodos dividir e unir figuras, para alternar entre figura com canto para figuras com cantos arredondados.

Na classe Entidade fica os atributos e os métodos gerais dos objetos das classes Aberto, Poligono e Texto. À medida que for descendo na árvore os objetos vão sendo especializados (adquirindo características e métodos novos). No entanto, as subclasses, ao se especializar, não deixam de ter características e métodos das

respectivas classes pai. Por exemplo, um quadrado é um polígono fechado, o qual na árvore de classes fica abaixo da classe Entidade e, consequentemente, herda as características e os método gerais de uma figura. Por sua vez, essa classe é filha da classe clFigura. Com isso, herda uma lista de pontos, bem como, todas os outros métodos e atributos contidos na classe clFigura.

4.1.14 Texto

A classe Texto é um tipo especial da classe Entidade, pois não se enquadra nas formas clássicas de polígonos, nem no tipo de curvas (não tem atributos do tipo raio, centro, etc). O objeto texto é também um objeto bem concreto. O atributo mais importante dessa classe é o atributo **texto**, o qual permite inserir um texto desejado pelo usuário. Esse atributo é útil para armazenar a string digitada pelo usuário. Ele pode ser usado por um método que altera o tipo de letra digitada pelo usuário.

Desenvolvimento

A seguir tem-se alguns tipos de ferramentas presente na barra de ferramentas do editor gráfico.

Ferramenta de criação de segmentos de retas;
Ferramenta de criação de elipses;
Ferramenta de criação de polígonos.
Ferramenta de criação de Texto
Ferramenta de criação de caixas.
Ferramenta de criação de curvas.
Seleção de figuras;
Manipulação de cores, tanto de preenchimento quanto de bordas (linhas) e cor do fundo;

4.2 Funcionamento das Ferramentas do Programa

O funcionamento das ferramentas seguem em geral as seguintes idéias abaixo. Para exemplificar, serão analisados os eventos necessários para a criação de Segmentos de Reta.

4.2.1 Criação de Segmentos de Reta

As possíveis formas que o programador pode considerar para desenhar uma reta.

1. Clicar e soltar com o esquerdo do mouse para determinar o primeiro extremo da reta. É necessário colocar uma marca "x","+", etc para indicar o começo da linha. Isso é importante, pois o usuário tem noção da inclinação que a reta ficará baseado no ponteiro do mouse e na marca feita para o primeiro extremo antes determinar o segundo extremo da reta. Além disso, a marca permite que o usuário visualize melhor o começo da reta e verifique se o seu começo da mesma apareceu no local exato que o usuário pretendia colocá-la.

2. Clicar e soltar com o esquerdo do mouse para determinar o segundo extremo da reta. Contudo, isso pode sobrecarregar o botão esquerdo do mouse pois a maioria das funções de um editor gráfico já utilizam o botão esquerdo do mouse. Dessa forma é interessante escolher o evento *mouse down* do mouse para determinar o primeiro extremo. A partir daí o usuário segura o botão esquerdo do mouse e o desloca o mouse até a posição desejada. Feito isso, o segundo extremo da reta é determinado soltando o botão esquerdo do mouse (evento *mouse up*). Essa estratégia economiza um acionamento do botão esquerdo do mouse.

   Uma outra alternativa é utilizar o botão direito do mouse para desenhar a reta.

   Além disso é bom colocar um rastro no formato de uma linha ligando a marca do primeiro extremo da reta com o ponteiro do mouse, a qual cresce ou diminui à medida que o usuário desloca o mouse. Assim, o usuário teria mais noção de como a reta ficaria antes de realmente inseri-la na área de desenho do editor gráfico.

   Uma outra melhoria seria a opção do usuário apagar a reta que está sendo desenhada. Por exemplo: o usuário já determinou o primeiro ponto da reta (inseriu a marca), porém o início da reta na saiu no ponto que o usuário desejava. Neste contexto, se o usuário soltar o botão esquerdo do mouse a reta será desenhada. Porém se o usuário não quer que isso aconteça, terá que escolher a ferramenta borracha para apagar a reta desenhada para depois tentar desenhá-la novamente, resolvendo assim o problema. Há outra alternativa mais viável:

1. Para determinar o primeiro extremo (insere a marca) o usuário poderia clicar com o botão esquerdo do mouse. À medida que o usuário desloca o mouse a linha é desenhada. Dessa forma, o segundo clique do botão esquerdo do mouse é para confirmar a reta se tudo estiver de acordo com que o usuário

espera ou clicar o botão direito do mouse para apagar a reta que está sendo desenhada. No caso, de a escolha do início da reta não ter saído do jeito que o usuário queira. Após selecionada a ferramenta de criação de segmentos de reta, o usuário deverá clicar no início do segmento, levar o ponteiro do mouse até o fim do segmento desejado e somente depois soltar o botão do mouse. Entre a definição do início e do fim do segmento, o programa mostrará onde ficaria o segmento quando desenhado entre a posição inicial e a posição atual.

4.2.2 Estrutura de Dados Utilizada

1. Lista de clPontos (otimização da memória).
2. Matriz de float.

É interessante escolher uma matriz de float pois muitos valores passados tem casas decimais. Contudo é necessário estabelecer o número n de casas decimais utilizadas a fim de evitar erros de arredondamentos.

4.3 Comentário dos Objetos do Editor Gráfico

4.3.1 Aplicação

A aplicação tem objetos para as classes folhas da árvore, uma vez que as folhas da árvore são objetos bem concretos (bem especializados). Como as folhas herdam todos os métodos e atributos das classes acima, então dentro de cada folha está encapsulado uma série de características e métodos herdados mais os da própria classe folha. Portanto quando a aplicação usa as classes folhas, na verdade, ela pode utilizar métodos e atributos de classes mais acima da folha. Os objetos quadrado, retângulo não precisam ser subclasses da classe polígono, bem como os objetos círculo e elipse serem subclasses da classe cônica, pois tanto a classe polígono como a cônica já contém todos os atributos é métodos suficientes para estes objetos poderem existir, uma vez que tanto a classe polígono como a classe cônica contém métodos e atributos específicos alem dos atributos herdados das respectivas classe acima. Ao invés disso, é mais interessante implementar o quadrado, retângulo, círculo e elipse no próprio botão da ferramenta que se encontra na aplicação, ou seja, por exemplo: quando o usuário clicar na ferramenta quadrado para desenhar quadrado o programa pegará a coordenadas (x, y) que o usuário determinou ao clicar na área de desenho (ponto 1) bem com as coordenadas (x, y) quando

o usuário soltar o botão esquerdo do mouse (ponto 4). Assim, com o ponto do lado superior esquerdo (ponto 1) e como o ponto do lado inferior direito (ponto 4) fica fácil determinar os pontos dos lados inferior esquerdo (ponto 3) e superior direito (ponto 2). Depois de ter terminar os quatro pontos do quadrado, é chamado o método da classe polígono que insere os pontos na lista de pontos.

Os objetos quadrado, retângulo, círculo e elipse não precisam estar presentes na árvore de classes do editor gráfico. A invés disso, implementamos tais objetos nos respectivos botões presentes na barra de ferramenta do editor gráfico por meio de métodos similares ao do método quadrado. Para exemplificar o que foi dito nos dois parágrafos anteriores segue a descrição abaixo:

```
x1 = x superior
x4 = x inferior
y1 = y superior
y4 = y inferior


//Método existente na aplicação.

void Quadrado()
{
clPoligono polígono;

int x1;
int y1;
int x4;
int y4;

//evento (\emph{mousedown}) - clique do botão esquerdo do mouse.
x1 = GetX();
y1 = GetY();

//evento (\emph{mouse up}) -  solta o botão esquerdo do mouse.
X4 = GetX();
Y4 = GetY();

Poligono$.$InserirPonto(x1,y1); //ponto 1
```

```
Poligono$.$InserirPonto(x4,y1); //ponto 2
Poligono$.$InserirPonto(x1,y4); //ponto 3
Poligono$.$InserirPonto(x4,y4); //ponto 4
}
```

```
UmQuartoCirculo(int x, int y, int value)
{
writePixel(x, y, value)
writePixel(x, y, value)
writePixel(x, y, value)
writePixel(x, y, value)
writePixel(x, y, value)
}
```

O método acima foi tirado de [4]

Quando na aplicação o usuário escolher a ferramenta para desenhar o quadrado, será usada a classe Poligono, a qual tem dentro de si, via herança, todos os métodos e atributos necessários para definir um quadrado. Por exemplo, se o usuário depois de desenhar o quadrado quiser alterar sua posição na tela (transladar), o método da classe figura que está dentro do objeto Poligono será chamado para transladar os pontos da lista de pontos do quadrado. Para isso, o método usará a classe Matriz para transladar todos os pontos da lista colocando de volta na lista os pontos atualizados.

4.3.2 Agrupamento

A classe agrupamento tem como atributos duas variáveis inteiras:

int CantoSuperiorEsquerdo int CantoInfeirorDireito

Descrição da Árvore

A classe clFigura usa a classe Ponto pois todo objeto das suas subclasses precisam ter como atributo objetos do tipo Ponto, os quais são definidos como tendo atributos x e y e métodos para manipula-los. Além disso é necessário encapsular as coordenadas x e y do ponto antes de inserir na lista de Pontos. Além disso, construir um objeto Ponto antes de inserir na lista e útil pois se for necessário modificar o estado interno do objeto de um ponto da figura fica fácil. A classe

clFigura precisa ter duas subclasse Agrupamentos e Entidade. Essas duas classes surgiram, pois comparando as características dos objetos agrupados e dos objetos simples nota-se que tais classes tem atributos em comum. Esses atributos em comum foram colocados na classe pai das duas que no caso é a Figura.

Entretanto essas duas subclasses da classe pai (Figura) têm também características específicas que faz com elas se especializem. Até agora a árvore tem uma classe base Ponto uma subclasse Figura. A classe Matriz que é usada pela classe Figura. A classe Lista que é usada pela Figura para armazenar os pontos geométrico dentro do encapsulamento do objeto. Duas subclasses Agrupamento e Entidade da classe Figura. Por sua vez, a Entidade tem duas as seguintes subclasses, a saber: Polígono, Texto, caixa e Aberto, isso porque, todas essas subclasses tem preenchimento, mesmo a Aberto possui preenchimento transparente. Verificando um texto é constatado, que as letras de um texto são fechadas, contudo não é um polígono pois não é observado características regulares que definem o texto como sendo um polígono (ex: ângulos retos, ângulos iguais, lados iguais, etc). O texto também não se enquadra na subclasse curvas, pois não apresenta características como raio, diâmetro, etc.

Sendo assim, os representantes dessa classe são: quadrado, retângulo, etc. Considerando agora a classe Aberto é constatado que existem quatro subclasses da classe Aberto: cônica, curvas, reta, Segmento de reta. A classe reta representa as retas simples. A classe Segmento de reta são linhas simples conectadas umas com as outras onde o final da primeira e início da segunda e assim sucessivamente até que o usuário dê um clique duplo ou clique com o botão direito do mouse. A classe Curvas representa os objetos curvilíneos abertos. A classe Curvas permite desenhos do tipo: linhas a mão-livre, linhas em forma de ondas, etc. Os objetos da classe cônica são círculo, elipse.

Baseado nas caractéristicas e métodos dos objetos acima descritos e comentados uma possível relacionamento entre tais objetos é a seguinte: 4.1.

$\longrightarrow$ Indica que uma subclasse herda os métodos e os atributos de uma classe pai. [1]

$\multimap$ Indica que um objeto usa uma classe [1]

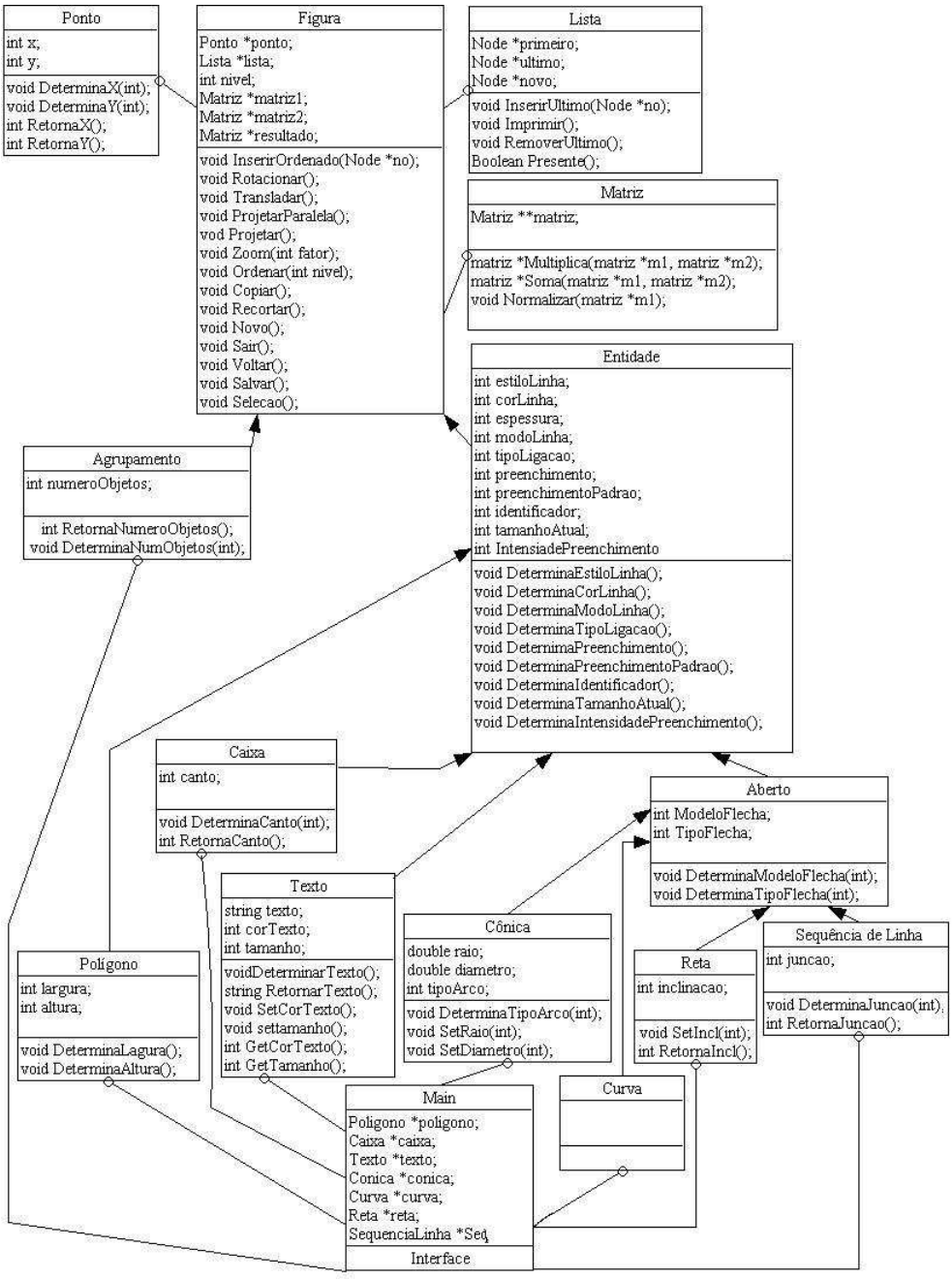

Figura 4.1: Descrição da árvore de classes do Editor Gráfico

Capítulo 5

Conclusão

O trabalho de modelagem de um editor gráfico 2D à luz da orientação a objeto é um passo fundamental para a sua implementação. No trabalho foram propostas as características e métodos mais importantes das classes envolvidas em um editor gráfico, bem como um possível relacionamento entre essas classes.

A modelagem de classes foi proposta de forma que preserva a compatibilidade com a maioria dos programas de edição de imagens existes no mercado.

Foram propostas também exemplos de ferramentas no editor gráfico. Entre as principais ferramentas do editor gráfico pode-se citar: ferramenta quadrado para desenhar quadrado na tela; ferramenta retângulo para desenhar um retângulo na tela; ferramenta elipse para desenhar uma elipse na tela; ferramenta reta para desenhar reta na tela e assim por diante.

A modelagem do editor gráfico serve, como uma diretriz na implementação do editor gráfico, evitando assim, que o programador se perca em divagações e possíveis erros esclarecidos na modelagem do editorr gráfico.

Foi constatado e utilizado um quantidade relativamente pequena de classes que se relacionam por uso ou herança pode-se implementar uma grande quantidade de recursos no editor gráfico.

Referências Bibliográficas

[1] BOOCH, Grady, James Rumbaugh, Ivar Jacobson: Tradução de Fábio Freitas da Silva. *UML, guia do usuário*, Rio de Janeiro: Campus, 2000.

[2] DEITEL, H.M. Deitel, P.J; trad. Carlos Arthur Lisboa e Maria Lucia Lang Lisboa *C++ Como Programar*, Porto Alegre: Bookman, 2001.

[3] FERREIRA, Aurélio Buarque de Holanda. IN: Minidicinário Aurélio. Rio de Janeiro: Nova Fronteira, 1977.

[4] FOLEY, James D.. *Computer Graphics primnciples and pratice* 2nd, ed, 1991.

[5] KNUTH, Donald.. *Orientacao a objetos*, url: `http://www.conceitosinfo.hpg.ig.com.br/orient\protect\T1\textunderscoreobjetos.htm`

[6] ROCHA, Heloísa Vieria da BARANAUSKAS, Maria Cecília Calami *Design e avaliação de interfaces humana computador*, São Paulo, IME_USP, 2000, 242p.

[7] SCOTT, Ambler. UMLBrasil. url: `http://www.uml.com.br/arquivos/tutoriais/umavisaorealisticadareutilizacao.doc`