

**Universidade Federal de Santa Catarina**

**wxVisual Editor : um ambiente de programação visual  
de interface com o usuário para o framework wxWidgets**

**Diego Garcia Rodrigues**

Florianópolis - SC

2007 / 1

**Universidade Federal de Santa Catarina  
Departamento de Informática e Estatística  
Curso de Ciências da Computação**

**wxVisual Editor : um ambiente de programação visual  
de interface com o usuário para o framework wxWidgets**

**Diego Garcia Rodrigues**

Trabalho de conclusão de curso apresentado  
como parte dos requisitos para obtenção do  
grau de Bacharel em Ciências da Computação.

Florianópolis - SC

2007 / 1

Diego Garcia Rodrigues

wxVisual Editor : um ambiente de programação visual de interface com o usuário para o framework wxWidgets

**wxVisual Editor : um ambiente de programação visual de interface com o usuário para o framework wxWidgets**

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção do grau de Bacharel em Ciências da Computação.

---

Orientador: Prof. Dr. Leandro José Komosinski

Banca Examinadora:

---

Prof. Dr. rer.nat. Aldo von Wangenheim

---

Prof. Dr. Antônio Augusto Fröhlich

# Sumário

<b>Lista de Figuras</b>	<b>1</b>
<b>Lista de Tabelas</b>	<b>2</b>
<b>Resumo</b>	<b>4</b>
<b>Abstract</b>	<b>5</b>
<b>1 Introdução</b>	<b>6</b>
1.1 Contexto . . . . .	6
1.2 Descrição do Problema . . . . .	7
1.3 Objetivos . . . . .	8
1.3.1 Objetivos Específicos . . . . .	8
1.4 Metodologia . . . . .	9
1.4.1 Boas práticas ao eXtremo . . . . .	9
1.4.2 Código Aberto . . . . .	9
1.4.3 eXtreme Programming e UML . . . . .	10
<b>2 Estado da Arte</b>	<b>11</b>
2.1 Introdução . . . . .	11
2.2 Frameworks para criação de janelas em C++ . . . . .	11
2.2.1 Qt . . . . .	12
2.2.2 FLTK . . . . .	12
2.2.3 wxWidgets . . . . .	12
2.2.4 GTK+ . . . . .	13
2.2.5 A escolha: wxWidgets . . . . .	14
2.3 Ambientes Gráficos Existentes . . . . .	15
2.4 wxDevCpp . . . . .	15
2.5 DialogBlocks . . . . .	17

2.6	wxSmith . . . . .	18
2.7	XRCed . . . . .	19
2.8	Comparação entre as soluções . . . . .	20
2.9	Conclusão . . . . .	20
<b>3</b>	<b>Solução Proposta</b>	<b>22</b>
3.1	Introdução . . . . .	22
3.2	Soluções Possíveis . . . . .	22
3.3	Solução Escolhida . . . . .	23
3.4	Requisitos Iniciais para o desenvolvimento . . . . .	24
3.5	O Framework wxWidgets . . . . .	24
3.5.1	Uma aplicação em wxWidgets . . . . .	25
3.6	O Projeto Eclipse . . . . .	27
3.6.1	História do Eclipse . . . . .	27
3.7	O Visual Editor . . . . .	28
3.7.1	Uma visão funcional do Visual Editor . . . . .	28
3.7.2	Uma visão da arquitetura do Visual Editor . . . . .	31
<b>4</b>	<b>Desenvolvimento</b>	<b>33</b>
4.1	Introdução . . . . .	33
4.2	Levantamento de Requisitos . . . . .	33
4.3	Primeiros Passos . . . . .	34
4.4	O modelo de plug-ins do Eclipse . . . . .	34
4.5	Configurando a área de trabalho . . . . .	35
4.6	Extendendo o Visual Editor . . . . .	38
4.6.1	Implementação . . . . .	39
4.6.2	Resultado Obtido . . . . .	44
<b>5</b>	<b>Conclusão</b>	<b>49</b>
5.1	Proposição de trabalhos futuros . . . . .	50
	<b>Referências Bibliográficas</b>	<b>51</b>

# Lista de Figuras

2.1	Qt: Estável e Robusto, porém pago para aplicações comerciais . . . . .	12
2.2	FLTK: Leveza e Simplicidade . . . . .	13
2.3	wxWidgets: Robustez com licença livre . . . . .	14
2.4	GTK+: Popular e fácil de usar . . . . .	15
2.5	wxDevCpp: Possui a interface e usabilidade parecida com o Delphi . . . . .	16
2.6	DialogBlocks: Ótimo editor de janelas, mas sem um editor de código . . . . .	17
2.7	wxSmith: Solução ainda imatura . . . . .	18
2.8	XRCed: Não possui uma parte visual . . . . .	19
3.1	Resultado da aplicação mínima escrita em wxWidgets . . . . .	26
3.2	Visão funcional do visual editor . . . . .	29
3.3	Visão geral da arquitetura interna do visual editor . . . . .	31
4.1	Arquitetura resumida do Eclipse . . . . .	35
4.2	Configurações para o CVS . . . . .	36
4.3	Ambiente Inicial . . . . .	37
4.4	Selecionando o projeto de um plugin . . . . .	38
4.5	Nomeando o projeto . . . . .	39
4.6	Dependências do plug-in . . . . .	40
4.7	Relação dos containers com classes existentes . . . . .	41
4.8	Paleta Inicial com um botão . . . . .	42
4.9	Estrutura Interna de Geração de Código . . . . .	44
4.10	Seleção do menu para criar nova classe visual . . . . .	45
4.11	Seleção de um template para criar classes visuais . . . . .	46
4.12	Declaração dos atributos iniciais da classe . . . . .	46
4.13	Arquivos criados pelo wxVisual Editor . . . . .	47
4.14	Adição de botão provoca atualizações imediatas . . . . .	48

# Lista de Tabelas

2.1	Comparação entre soluções existentes e a solução desejável . . . . .	20
3.1	Requisitos iniciais para o desenvolvimento . . . . .	24

# Resumo

Este trabalho apresenta um ambiente visual de programação com o propósito de desenvolver interfaces com o usuário para o framework wxWidgets na linguagem C++, chamado wxVisual Editor. O wxVisual Editor dá ao usuário a possibilidade de construir interfaces com o usuário de maneira gráfica, que além de ser simples e amigável dá uma visão mais ampla da estrutura interna da interface que está sendo desenvolvida, mostrando a ligação hierárquica de seus componentes em forma de árvore de maneira intuitiva e apresentando todas as propriedades de cada componente de interação com o usuário presente no programa.

O wxVisual Editor foi desenvolvido sobre a plataforma Eclipse com dois plugins disponibilizados no site principal do projeto, o plugin Visual Editor e o plugin CDT. O plugin Visual Editor é na verdade um framework para o desenvolvimento de ambientes visuais de programação genéricos, o qual foi utilizado como base para a geração de código e interação com o usuário. O plugin CDT gerencia projetos em C e C++, cuidando de toda a interação do usuário com o compilador e tarefas básicas de uma ide C++, como completção automatizada de código e syntax highlighting.

Este projeto foi desenvolvido para ser um ambiente multi-plataforma para programação visual e disponibilizar aos programadores um ambiente onde eles possam facilmente desenvolver ou prototipar suas interfaces visuais, ou ainda integrá-las com código já existentes, de uma maneira rápida, ao contrário da maneira clássica de escrever toda a parte de janelas e diálogos com o usuário em um editor de texto. Seu desenvolvimento foi feito baseado no conceito de “Desenvolvimento Rápido de Aplicação” ([4]), no qual é enfatizado o reuso por meio de componentes, diminuindo o tempo total do desenvolvimento.

**Palavras-Chave:** eclipse, wxwidgets, rad, rapid application development, programação visual



# Abstract

This project presents a visual programming environment with the purpose of developing graphical user interfaces to the wxWidgets framework for the C++ programming language, called wxVisual Editor. The wxVisual Editor solution gives to its user the possibility of building user interfaces in a graphical manner, which, beyond being simple and friendly, gives a broader vision of the intern structure of the interface being developed, showing the hierarchical links of its components represented intuitively in the form of a tree and presenting the properties of each user interaction component present in the interface.

The wxVisual Editor solution was developpt over the Eclipse platform with the aid of two extra plugins available in the main site of the Eclipse project, the plugin Visual Editor and the CDT plugins. The Visual Editor plugin is, in fact, a framework to the development of generic visual programming environments, which was used as the base to the code generation and user interaction. The CDT plugin manages projects in C and C++, taking care of all compiler interaction and basic C++ IDE tasks, such as code completion and syntax highlighting.

This project was developed to be a multiplatform visual programming environment and to give the programmers an environment where they can easily develop or prototype their visual interfaces, or even integrate them with existing code fast, opposed the the classic way of writing all the code of window and dialogs creation and user interaction in a text editor. All this was done based in the concept of “Rapid Application Development” [[4]], in which is enfatized code reuse with components, reducing the total development time.

**Keywords:** eclipse, wxwidgets, rad, rapid application development, visual programming

# 1 Introdução

## 1.1 Contexto

Nas últimas décadas, o mercado de desenvolvimento de softwares sofreu mudanças extremas. Seu crescimento acelerado fez com que muitos investimentos fossem feitos em análises e estudos para melhorar o processo de desenvolvimento de softwares. Pesquisas foram feitas em muitas áreas, novas linguagens de programação surgiram e junto com estas novos paradigmas de programação, novos bancos de dados, frameworks, etc. Todos esses esforços foram feitos com um grande motivo, que é otimizar a tarefa de produzir softwares, diminuindo tempo e esforço gastos no desenvolvimento, pois quando se torna mais simples e eficiente a tarefa de desenvolver um software há um ganho significativo no lucro.

Segundo James Martin, chairman da James Martin Associates, Chicago, e expert de ferramentas CASE, “Interfaces gráficas com o usuário são extremamente importantes para o desenvolvimento de novas aplicações, estudo após estudo mostrou que usuários finais aprendem(a usar novas aplicações) mais rápido e fazem menos erros se a interface gráfica de interação com o usuário é bem projetada.“. James Martin ainda sugere que desenvolvedores deveriam sempre procurar por ambientes visuais para desenvolver interfaces gráficas com o usuário, estando essa ferramenta integrada ou não à uma ferramenta CASE. Além disso há estudos que mostram que uma média de 48% de código de uma aplicação é escrito para desenvolver a interface com o usuário e 50% do tempo requerido para desenvolver e manter toda a aplicação é devotado para a interface com o usuário [7].

O uso de ferramentas para automatizar essas tarefas pode drasticamente reduzir esses números. Por exemplo, o sistema MacApp da Apple relatou que teve seu tempo de desenvolvimento reduzido por um fator de quatro. Outro estudo descobriu que uma aplicação usando uma ferramenta para automatizar a construção da interface gráfica com o usuário teve 83% menos linhas de código e levou metade do tempo de aplicações escritas sem o uso dessas ferramentas [7]. As pesquisas iniciais sobre interfaces gráficas com o usuário foram conduzidas nos centros de pesquisa de Stanford, Xerox e MIT nos anos 70.

Sendo assim, é inegável que ferramentas que automatizem o processo de desenvolver interfaces visuais com o usuário são extremamente úteis e ajudam o desenvolvedor a ganhar tempo ao desenvolver suas aplicações, sejam elas simples bancos de dados para uma locadora ou uma aplicação mais avançada para segmentação de imagens de satélite para uma grande corporação. O uso desse tipo de ferramenta é altamente recomendado por metodologias de desenvolvimento de software como a RAD, que inicialmente pregou o uso de ambientes para o desenvolvimento rápido e que enfatiza um ciclo de desenvolvimento extremamente curto (entre 60 e 90 dias). O termo foi registrado por James Martin em 1991 na publicação de um livro homônimo ao modelo [4]. Como

o modelo RAD enfatiza a economia de recursos sem perda de qualidade necessário o uso de ferramentas que melhorem o desempenho da equipe de desenvolvimento. Ferramentas de construção de interfaces gráficas com o usuário são muito usadas neste modelo, temos como um grande representante delas o ambiente Delphi que escreve código automaticamente em Object Pascal enquanto seus componentes visuais são modificados.

Vários tipos de ambientes gráficos foram utilizados amplamente no passado, cada um com suas vantagens e desvantagens que serão discutidas nos capítulos a seguir. A busca por ambientes de programação visual foi assunto de vários livros e artigos científicos([8], [1], [6] ) e foi mais do que provado de que podemos ter uma interface altamente amigável que resulte em uma alta produtividade sem perder expressabilidade ou profundidade da percepção da aplicação que se está sendo desenvolvida.

## 1.2 Descrição do Problema

Como já foi apresentado, a criação de Interfaces Gráficas com o Usuário em aplicativos desktop é uma parte do desenvolvimento de aplicativos que afeta de uma maneira muito acentuada a satisfação do usuário final com o aplicativo. Porém, mesmo afetando bastante a qualidade do produto final não é uma parte do desenvolvimento onde queremos gastar muito tempo, logo a duração ideal é curta, pois não estamos lidando com a solução de um problema quando desenvolvemos interfaces gráficas e sim apenas com a apresentação final e interação de um usuário com uma solução que está representada de alguma maneira no nosso programa, podendo ser um modelo orientado a objetos, funcional, lógico...

Em contrapartida com a "ideal" curta duração do desenvolvimento temos uma quantidade enorme de código que deve ser gerada para que essa tarefa seja cumprida. A maior parte desse código não possui complexidade alguma, são meramente instanciações de objetos e chamadas de métodos que seguem praticamente um padrão toda vez que se desenvolve a interface gráfica com o usuário.

Levando em consideração esses fatos um aplicativo que ofereça ao desenvolvedor um ambiente para criar interface gráfica com o usuário com velocidade e qualidade é uma ferramenta muito útil, ainda mais quando ela está integrada à um ambiente poderoso para o desenvolvimento de aplicativos.

A idéia de utilizar um ambiente de desenvolvimento rápido, que ofereça geração automática de código, começou no final da década de 80 com a metodologia RAD a qual teve bastante sucesso e prestígio nos anos 90 (ver [4]).

Quando um desenvolvedor deseja escrever um programa que possua interface com o usuário hoje, ele possui várias opções de ambientes visuais de programação para o auxiliar nessa tarefa. Existem vários ambientes para o desenvolvimento de interfaces gráfica , mas eles são pobres, não possuem uma boa IDE para

a programação(falta de ferramentas que facilitam a programação como por exemplo integração automatizada com o CVS ou facilidades para a refatoração do código entre outros recursos que um programador necessita para aumentar sua produtividade e diminuir as tarefas mecânicas do seu cotidiano), o que faz com que o desenvolvedor tenha que usar um ambiente para o design de janelas e outro para programar seu sistema(lógica, camada de negócios), o que não é produtivo e ainda por cima é um processo repetitivo, pois a integração do código gerado pelo editor de janelas é feito de maneira manual, o que é um processo muito propenso a erros em sistemas grandes.

## 1.3 Objetivos

O wxVisual Editor foi inicialmente vislumbrado para disponibilizar aos programadores C++ uma maneira multiplataforma para desenvolver aplicações com interfaces gráficas com o usuário de maneira simples. Deveria ser uma solução que além de ser multiplataforma permitisse desenvolver interfaces gráficas multiplataforma, suprimindo a necessidade de reescrever a mesma interface para todas as plataformas onde ela deve ser disponibilizada. Isso por si só já é uma tarefa grande, mas como será apresentado nos próximos capítulos o uso de tecnologias já existentes torna essa tarefa trivial.

O principal requisito do projeto é ser um ambiente gráfico que possa manipular o código referente à criação de interfaces gráficas com acesso rápido às várias informações de seus componentes, sempre visando ser um ambiente que aumente a produtividade dos programadores que o utilizarão, não importando a experiência com programação que eles possuem, pois muitas vezes os desenvolvedores dessas interfaces serão especialistas em uma determinada área que não necessariamente engloba ciências da computação, como por exemplo design. Para termos usuários tão diversificados e também para atingir uma alta produtividade o ambiente gráfico bem projetado é um fator chave.

### 1.3.1 Objetivos Específicos

- Dar ao programador a visão da interface com o usuário durante o desenvolvimento(design time) em contrapartida como é feito hoje em dia(runtime).
- Possibilitar a melhor representação da interface que será gerada mesmo com as diferenças encontradas nas muitas plataformas onde o sistema irá rodar

## 1.4 Metodologia

Como todo projeto de desenvolvimento de software deve ser seguida uma metodologia bem determinada e bem escolhida. A metodologia de desenvolvimento escolhida para este projeto foi a metodologia chamada eXtreme Programming(XP).

O Extreme Programming é uma metodologia mais ágil que foi concebida para equipes pequenas e médias onde há um foco muito grande na simplicidade, comunicação, refactoring e padrões de código.

A simplicidade vem da preocupação em apenas entregar o que o cliente precisa. Nada é feito a mais ou a menos, desde o primeiro dia do desenvolvimento o objetivo é entregar exatamente o que o cliente pediu e como o cliente pediu. Isto faz com que o design seja simples e limpo.

A comunicação é considerada um dos pontos principais do XP. A comunicação acontece tanto entre os desenvolvedores quanto entre a equipe de desenvolvimento e os clientes. O cliente, por exemplo, deve estar sempre disponível, seja pessoalmente, seja por telefone ou até mesmo chat via internet. Isto é necessário para obter um feedback rápido e objetivo sobre o desenvolvimento do programa, para que não aconteça retrabalho ou, caso ocorra, seja o mínimo possível já que o cliente pode reclamar sobre algo que está errado no início do desenvolvimento da funcionalidade e não ao seu final como nas outras metodologias.

### 1.4.1 Boas práticas ao eXtremo

Uma pergunta bastante conhecida quando se começa a desenvolver utilizando a metodologia XP é ‘Como você programaria se tivesse tempo suficiente?’ (Kent Beck). Essa pergunta procura nos levar a responder coisas como: ‘Eu gostaria de fazer mais testes’, ‘Eu gostaria de projetar melhor a arquitetura’, ‘Mais qualidade’ ...

A resposta do XP para isso é: devemos levar todas as boas práticas ao extremo. Se testar é bom, todo o código será testado. Se projetar é bom, todos os desenvolvedores irão projetar diariamente. Se simplicidade é bom, sempre deixaremos o sistema com o projeto mais simples que suporte a funcionalidade atual (a coisa mais simples que possa funcionar). Se iterações de desenvolvimento curtas são boas e apresentam bons resultados, todas iterações serão curtas!

### 1.4.2 Código Aberto

Todo o código que será gerado neste projeto vai ser distribuído livremente na internet. O código será completamente aberto para que a comunidade possa, além de usufruir do sistema como uma ferramenta, possa também contribuir com a manutenção e criação de novas funcionalidades para ele.

### **1.4.3 eXtreme Programming e UML**

A metodologia eXtreme Programming sugere que haja o mínimo de documentação possível, mesmo que isso signifique abrir mão de diagramas UML e outras formas de documentar melhor o software.

Mesmo que seja sugerido que não seja usado, diagramas UML podem ser usados para o melhor entendimento do sistema (apesar disso já ser conseguido com a programação em dupla). Como o sistema a ser desenvolvido terá uma complexidade grande e não haverá a prática da programação em dupla, serão gerados diagramas UML. Esses diagramas ajudarão muito os desenvolvedores que desejarem contribuir com o projeto, ajudando no entendimento de como ele funciona.

# 2 Estado da Arte

## 2.1 Introdução

Dentre os problemas apresentados no capítulo anterior podemos destacar os dois principais, há a necessidade de encontrar uma maneira de escrever interfaces gráficas com o usuário de maneira multiplataforma na linguagem C++ e após escolhida a maneira como faremos isso deve-se pensar em como fazer um ambiente visual para programação que procuramos, como já foi explicitado.

Além de buscar soluções prontas para servirem como base para compor nosso ambiente visual de programação, deve-se também apresentar o que já há de disponível como solução, para avaliar a necessidade da implementação do projeto e também verificar a qualidade final do trabalho feito após ele estar completo.

O primeiro problema analisado será como produzir as interfaces gráficas de maneira multiplataforma. Como já deve ser de conhecimento geral existem inúmeras soluções já prontas, que era inclusive do conhecimento do autor ao início deste projeto. Logo a procura por uma maneira de escrever interfaces gráficas com o usuário em C++ é uma tarefa trivial, pois existem inúmeros frameworks desenvolvidos para C++ que cuidarão desta parte do problema. A maioria destes frameworks são bastante maduros e estáveis, com desenvolvedores de várias partes do mundo trabalhando continuamente para seu aprimoramento. Como há várias soluções, há apenas o trabalho de escolher dentre estes frameworks o mais adequado.

## 2.2 Frameworks para criação de janelas em C++

Nesta seção serão listados todos os frameworks e ao final será apresentado o framework escolhido juntamente com a justificativa. Nota que todos os frameworks apresentados são frameworks para criação de interfaces com o usuário multiplataforma para a linguagem C++.

A justificativa para usar um framework para criação de interfaces com o usuário já pronto pois o trabalho de implementar algo assim seria por si só um projeto que levaria muito tempo e demoraria muito para amadurecer e chegar no estágio de tudo que já se encontra hoje disponibilizado por grupos de todas as partes do mundo. Resumindo, estarei evitando o retrabalho e utilizando código limpo, altamente testado e com vários usuários já familiarizados e dispostos a usar o framework dentro do ambiente visual que será desenvolvido.

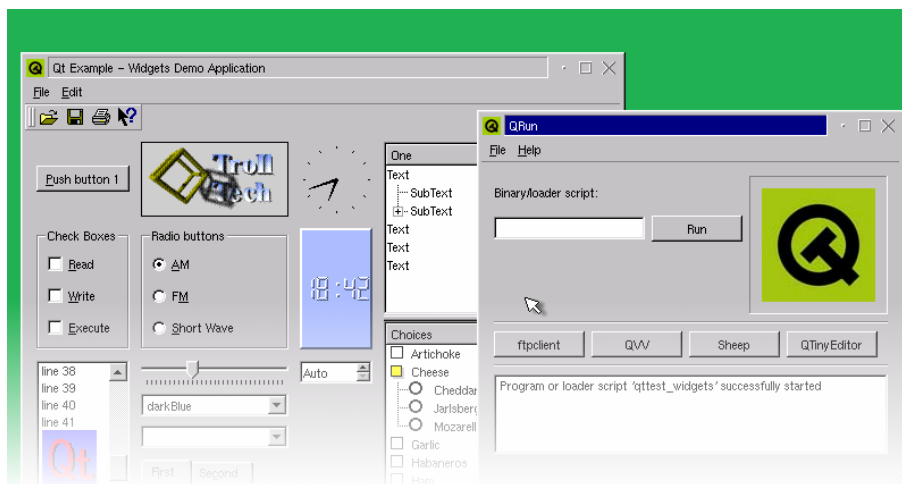


Figura 2.1: Qt: Estável e Robusto, porém pago para aplicações comerciais

### 2.2.1 Qt

**Descrição:** Qt(ou Qt3)(ver 2.1) é um framework desenvolvido pela empresa TrollTech, que possui vários clientes de peso, com a Motorola, Adobe e Google. Possui mais de 400 classes e vem junto com um ambiente visual para edição de interfaces próprio, este porém não possui todos os requisitos que o meu projeto visa cumprir.

**Características principais:** Robustez, Ambiente Visual Pronto

**Licença:** Qt está disponível sob a licença GPL, para desenvolver produtos de código fechado é necessário comprar uma licença.

### 2.2.2 FLTK

**Descrição:** FLTK(ver 2.2) é um framework relativamente novo que possui uma abordagem minimalista para a construção de interfaces gráficas com o usuário, não possuindo muitas soluções necessárias para desenvolver uma aplicação, como componentes para controlar impressão e rede.

**Características principais:** Leveza do framework

**Licença:** A licença é uma versão mais restrita do LGPL, onde para desenvolver aplicações de código fechado deve-se linkar dinamicamente o código da biblioteca obrigatoriamente.

### 2.2.3 wxWidgets

**Descrição:** wxWidgets(ver 2.3) é um framework que foi desenvolvido originalmente na universidade de Edinburgh. Seu propósito é escrever aplicações portáteis entre várias plataformas, dentre elas Linux, Windows



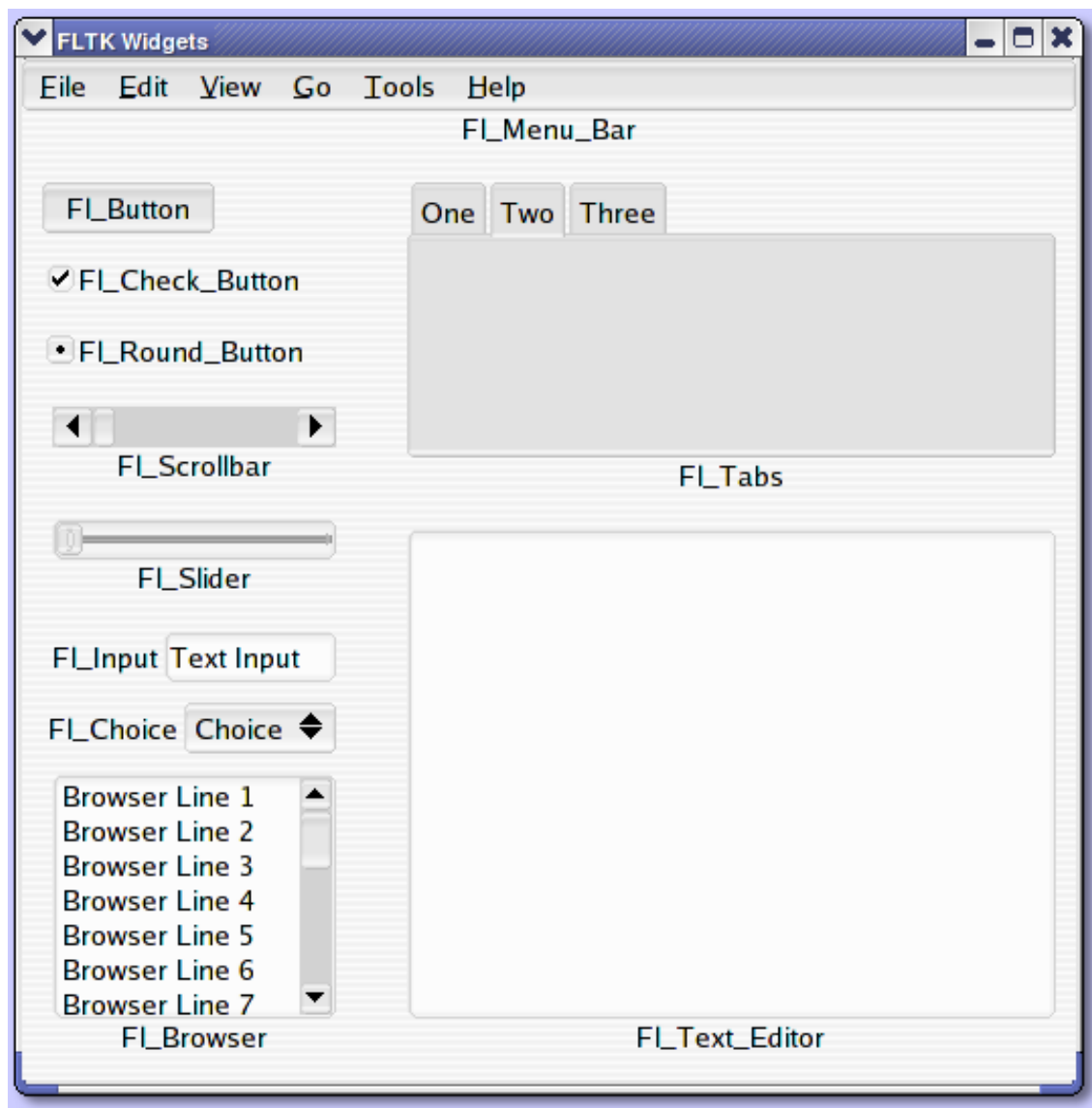


Figura 2.2: FLTK: Leveza e Simplicidade

e MacOS.

**Características principais:** Ótima estabilidade e robustez. Possui um livro publicado sobre como utilizá-lo

**Licença:** A licença é uma versão de LGPL mais aberta, onde não há restrições da distribuição das aplicações escritas com o uso do framework.

## 2.2.4 GTK+

**Descrição:** O GTK+(ver 2.4) era originalmente o Gimp Toolkit, desenvolvido para sistemas Unix na linguagem C. Para a integração com a linguagem C++ é utilizado um wrapper chamado GTKMM.

**Características principais:** Ótima estabilidade e robustez

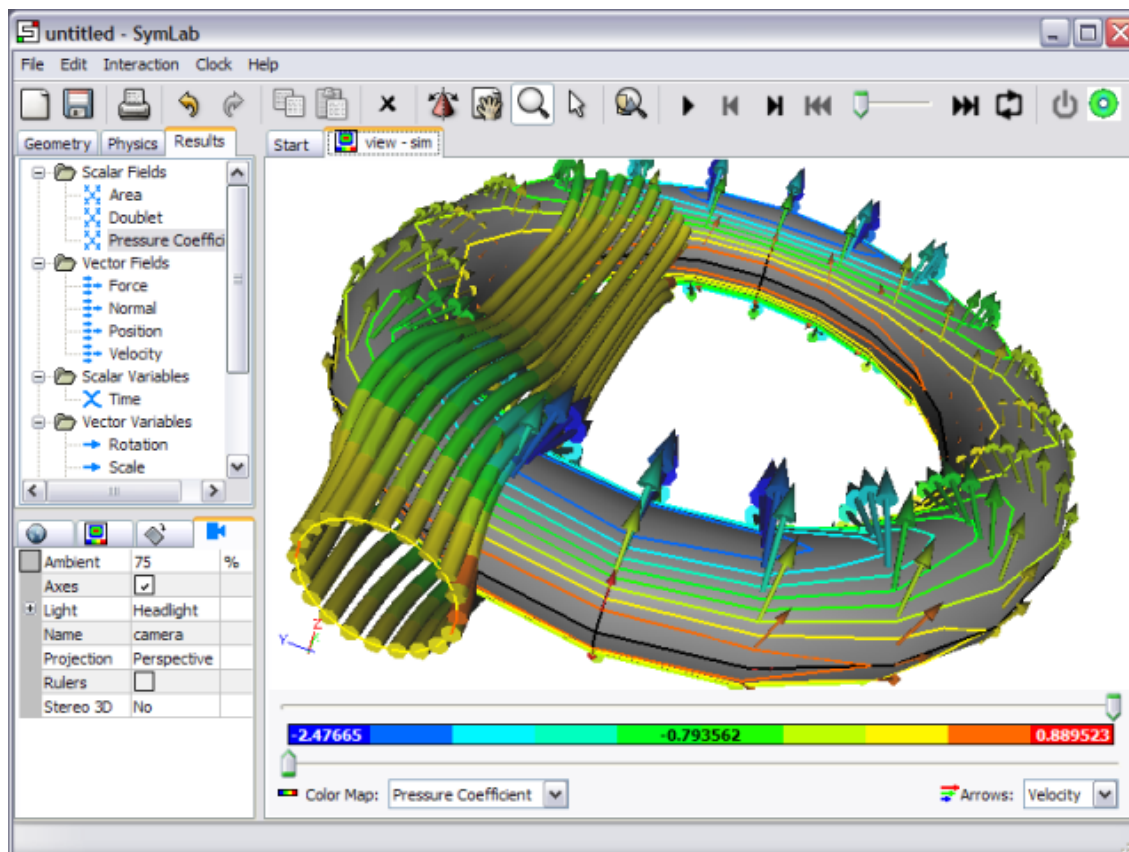


Figura 2.3: wxWidgets: Robustez com licença livre

**Licença:** A licença é GPL.

## 2.2.5 A escolha: wxWidgets

Todos os frameworks possuem vários pontos fortes, mas o framework escolhido foi o wxWidgets, por possuir ao mesmo tempo 20 anos de desenvolvimento, o que lhe deu uma grande robustez, uma licença que permite ao usuário fazer o que bem entender com sua aplicação, podendo vendê-la para obter lucro, e ser completamente multiplataforma, podendo disponibilizar até executáveis para dispositivos portáteis, chamado wxEmbedded.

Com o wxWidgets como escolha serão analisados os ambientes visuais de programação já disponíveis para, como já foi mencionado, validar a necessidade de um ambiente novo e também verificar a qualidade do produto final deste projeto.

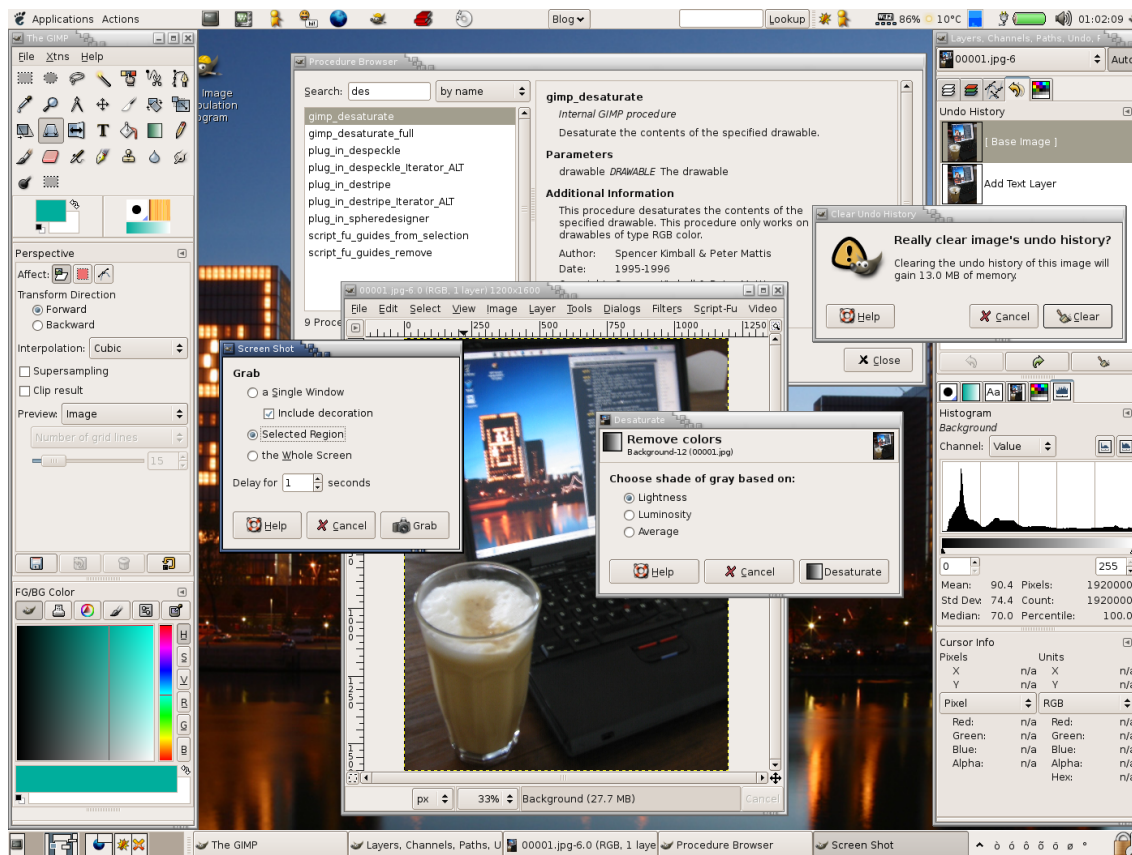


Figura 2.4: GTK+: Popular e fácil de usar

## 2.3 Ambientes Gráficos Existentes

Como já há uma escolha sobre qual framework será usado, resta procurar se já há uma solução para o problema apresentado. Para o desenvolvimento de janelas utilizando o framework wxWidgets já existem muitos ambientes, alguns gratuitos e alguns pagos. Dentre esses ambientes existentes há aqueles que são completamente visuais (wxDevCpp e DialogBlocks), os que possuem alguma ajuda gráfica mas não chegam a apresentar todo o desenvolvimento de uma maneira visual (wxSmith e wxDesigner) e outros que não possuem uma ajuda gráfica e apenas geram código a partir de especificações textuais da janela (XRCed). Esses ambientes possuem seus pontos fortes e fracos os quais serão apresentados a seguir.

## 2.4 wxDevCpp

O wxDevCpp (mostrado na figura 2.5) é uma extensão para o DevCpp da Bloodshed Software, que é mantido por um grupo independente. É completamente gratuito e possui o código aberto.

Há uma clara semelhança entre o wxDevCpp e o ambiente Delphi que curiosamente é o ambiente no qual

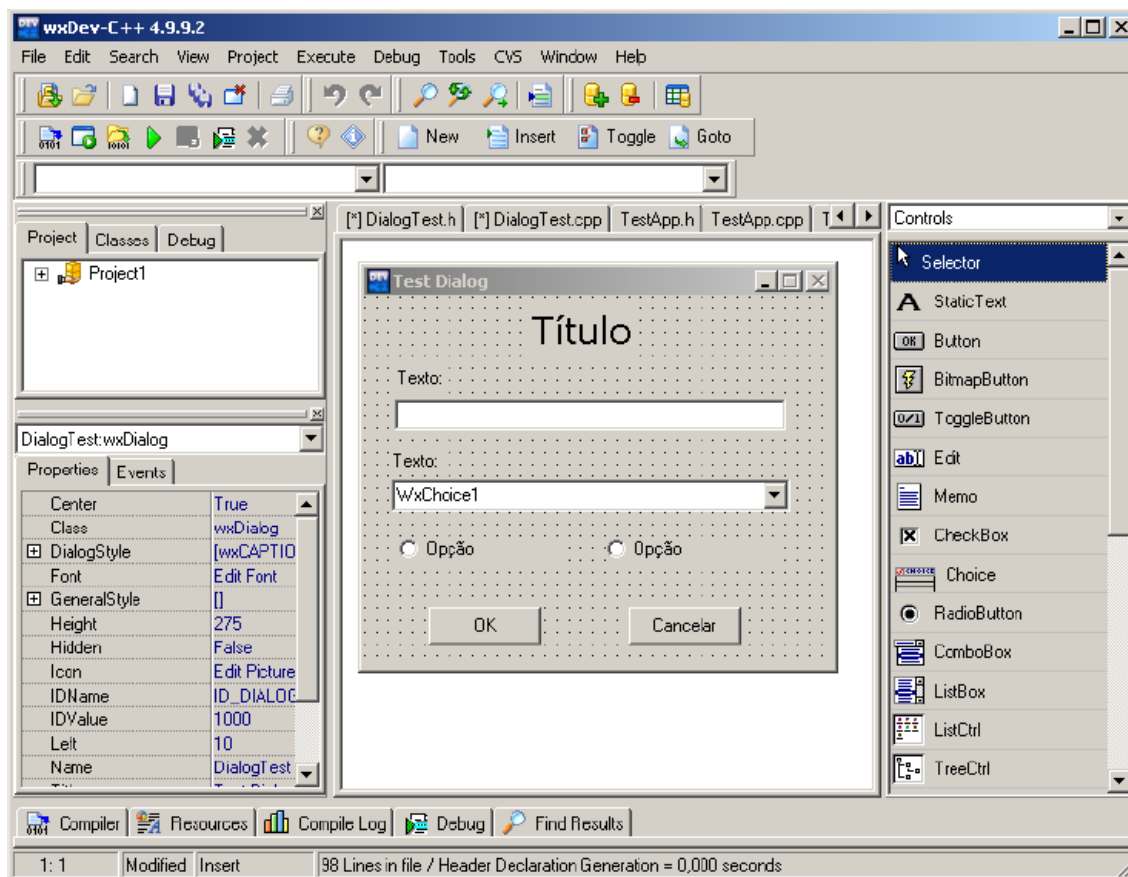


Figura 2.5: wxDevCpp: Possui a interface e usabilidade parecida com o Delphi

o DevCpp foi desenvolvido. Essa semelhança pode ser bastante agradável para quem já está acostumado ao ambiente Delphi, mas isso torna o uso do wxWidgets bastante limitado, uma vez que nem todas as suas funcionalidades podem ser utilizadas quando se está em um ambiente que é uma cópia do Delphi. Os Sizers, por exemplo, não podem ser utilizados utilizando o wxDevCpp.

Outro ponto fraco do wxDevCpp é a sua IDE. Como ele foi construído em cima do DevCpp ele acabou herdando várias falhas do seu antecessor. Falhas como não salvar o código gerado, travamentos, demora ao abrir ou ao salvar projetos maiores e indexador de código que consome muito processamento.

É um projeto que ainda não amadureceu e possui muitos bugs que realmente atrapalham o desenvolvimento, além da IDE não ser muito boa e ser específica para Windows, que é um sistema operacional relativamente caro e bastante decepcionante para os desenvolvedores de software (mesmo sendo muito bem visto por usuários leigos). Em resumo, é um editor visual que não funciona para projetos grandes.

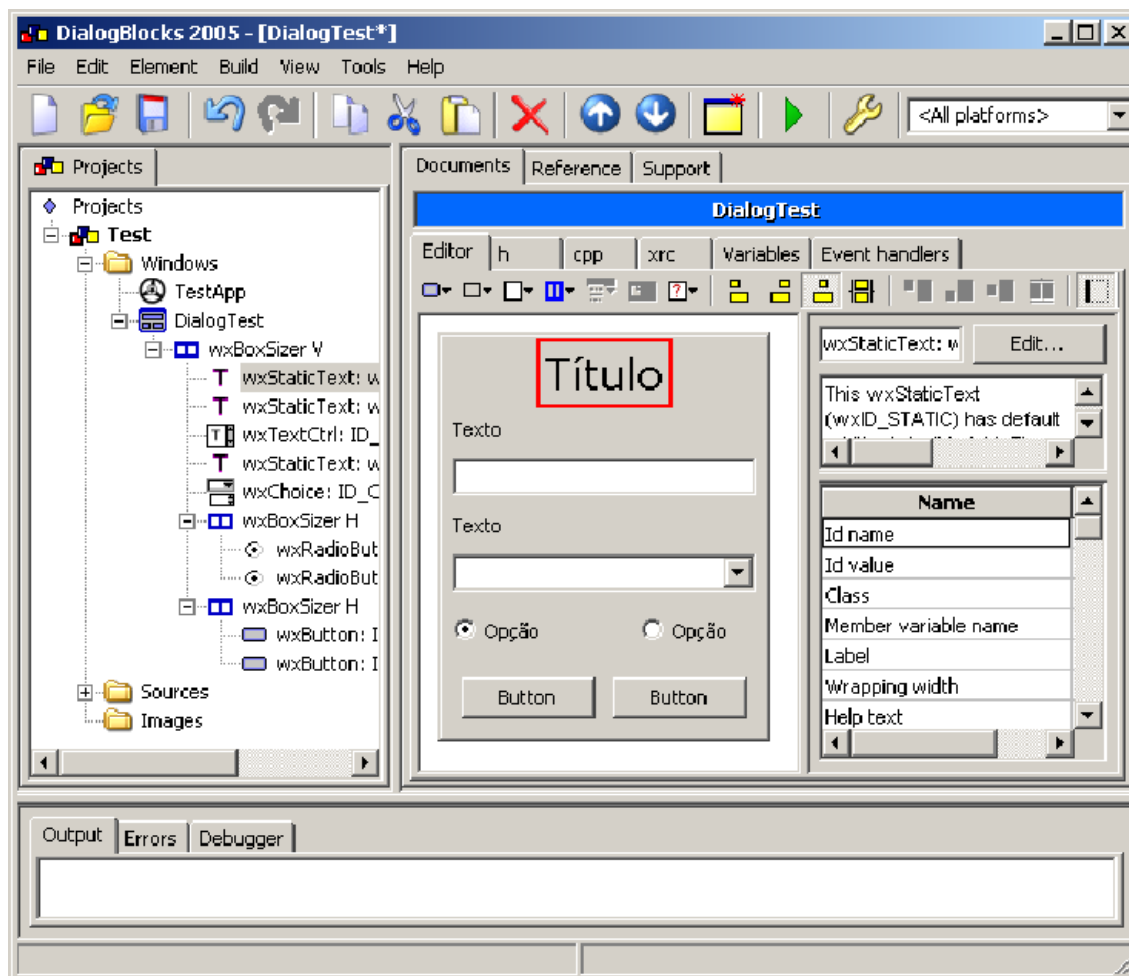


Figura 2.6: DialogBlocks: Ótimo editor de janelas, mas sem um editor de código

## 2.5 DialogBlocks

O DialogBlocks (mostrado na figura 2.6) é um sistema comercial desenvolvido pela Anthemion Software.

É um editor para wxWidgets bastante completo, possuindo ajuda visual para o desenvolvimento de janelas muito bom e que usufrui de todas as funcionalidades que o framework proporciona. Suporta com facilidade os Sizers de wxWidgets, mostrando hierarquicamente todos os componentes da tela em forma de árvore, que auxilia muito o desenvolvimento.

Além de usufruir muito bem dos componentes do framework ainda possui suporte para múltiplas línguas facilitando a internacionalização de software, podendo interagir com programas de internacionalização de terceiros, com o poEdit.

Com todas essas funcionalidades e suporte o DialogBlocks necessita apenas de um Pentium com 100Mhz, 32MB de RAM e 4MB de memória de vídeo.

A grande falha do DialogBlocks é que ele é apenas um editor de janelas. Ele não possui nenhum suporte para que se faça um programa completo nele, apenas janelas são construídas. Para fazer seu projeto você deverá escolher, além do DialogBlocks outro programa onde você poderá escrever a camada de negócios do seu projeto.

## 2.6 wxSmith

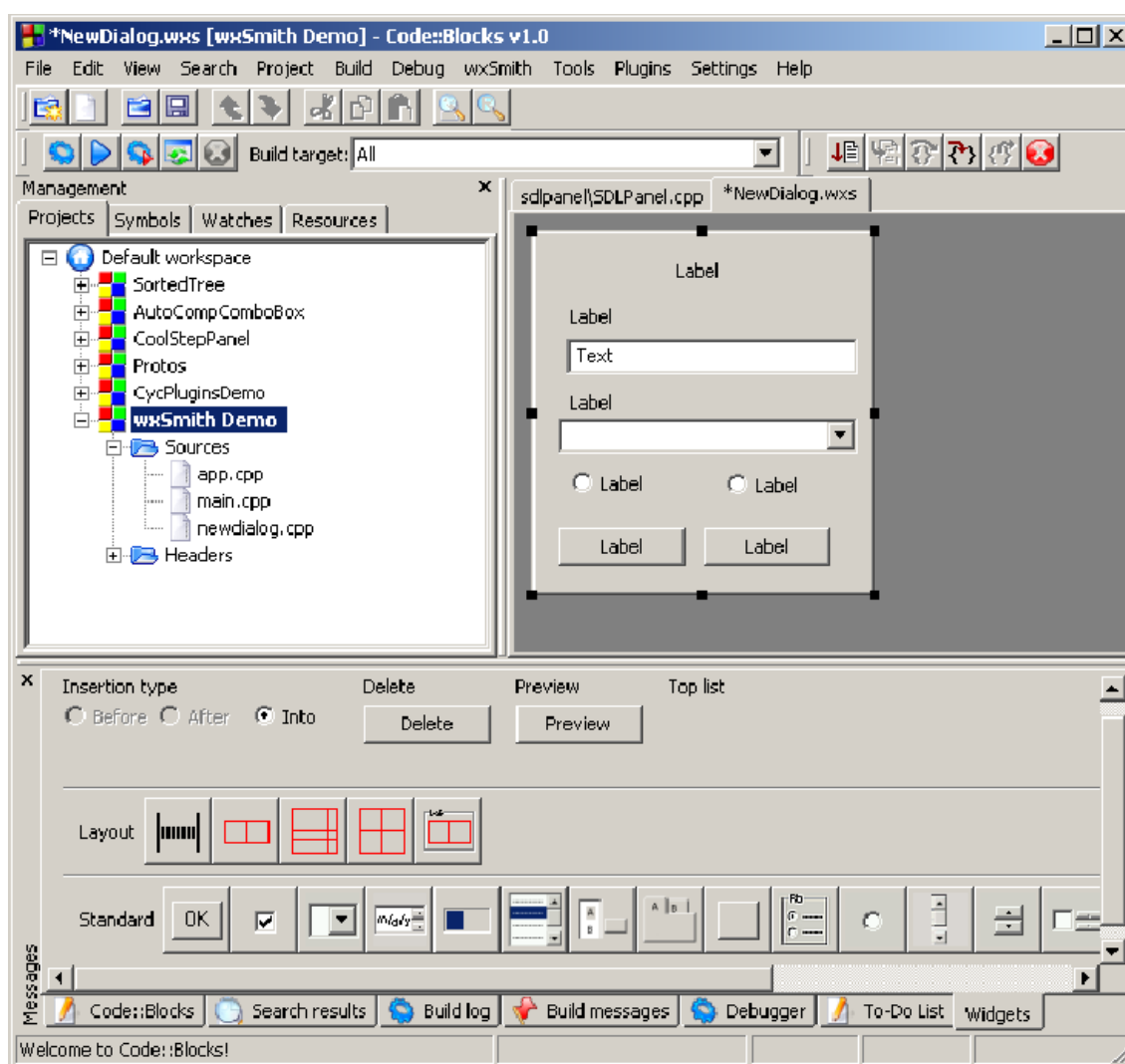


Figura 2.7: wxSmith: Solução ainda imatura

O wxSmith (mostrado na figura 2.7) é uma extensão para o ambiente CodeBlocks. Este ambiente, assim como o wxSmith, possui o código aberto e é completamente gratuito.

O ambiente CodeBlocks é bastante completo e possui várias facilidades que realmente ajudam o programador. Sua integração com CVS e Subversion para o controle de versões é muito bom. Possui também

completação automática de código e suporte para múltiplos compiladores e esquemas de compilação.

Exatamente ao contrário do DialogBlocks, o wxSmith possui um ambiente para programação muito bom, mas não oferece muito para a criação automática de janelas. Seu suporte é bastante primitivo e se limita a alguns poucos controles como botões e textos estáticos.

## 2.7 XRCed

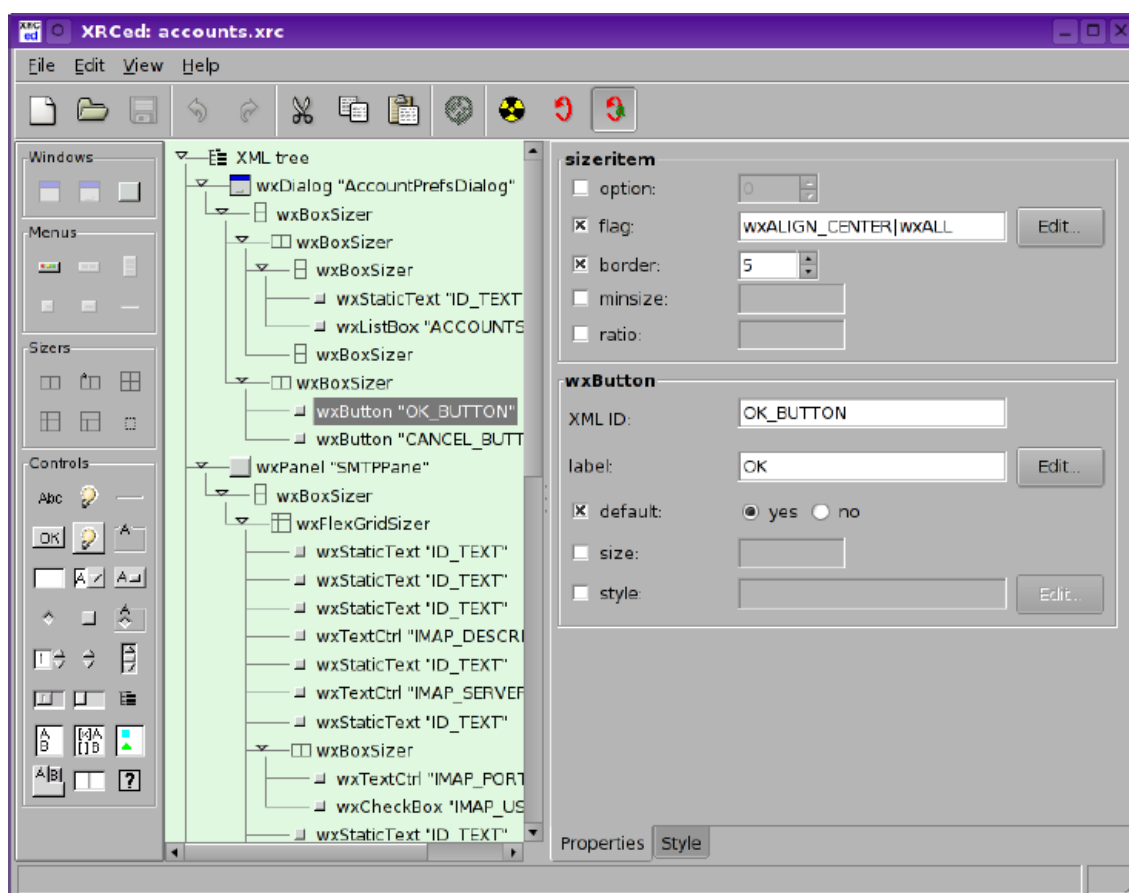


Figura 2.8: XRCed: Não possui uma parte visual

O XRCed (mostrado na figura 2.8) é um programa de código aberto e livre mantido por um grupo independente.

Para usá-lo o desenvolvedor tem acesso somente à uma árvore de componentes onde ele deve inserir seus objetos hierarquicamente, juntamente com suas configurações. Esse é um trabalho bastante chato e repetitivo, o que faz com que se pense se é realmente vale a pena usar o editor.

Este editor é bastante novo e possui ainda poucas funcionalidades, talvez no futuro apresente algo de novo, mas atualmente é apenas um projeto que perde para todos os outros sistemas encontrados.

## 2.8 Comparação entre as soluções

<i>Característica</i>	<i>wxDevCpp</i>	<i>DialogBlocks</i>	<i>wxSmith</i>	<i>XRCed</i>	<i>Desejável</i>
Geração de Código C++	Sim	Sim	Sim	Sim	<b>Sim</b>
Suporte a Eventos	Sim	Sim	Não	Não	<b>Sim</b>
Suporte a Imagens	Sim	Sim	Não	Não	<b>Sim</b>
Design com Sizers	Não	Sim	Sim	Sim	<b>Sim</b>
Design sem Sizers	Sim	Parcial	Parcial	Não	<b>Sim</b>
Ambiente para programação C++	Sim	Não	Sim	Não	<b>Sim</b>
Edição visual das janelas	Sim	Sim	Sim	Não	<b>Sim</b>
Ferramentas de controle de versão de código	Sim	Não	Sim	Não	<b>Sim</b>
Multiplataforma	Não	Não	Não	Não	<b>Sim</b>
Ferramentas de refatoração de código	Não	Não	Não	Não	<b>Sim</b>
Suporte a controle de projeto	Sim	Não	Sim	Não	<b>Sim</b>

Tabela 2.1: Comparação entre soluções existentes e a solução desejável

## 2.9 Conclusão

Hoje existem vários ambientes visuais o desenvolvimento de interfaces gráficas utilizando o framework wxWidgets, mas, como já foi apresentado, eles são pobres, não possuem uma boa IDE para a programação (falta de ferramentas que facilitam a programação como por exemplo integração automatizada com o CVS ou facilidades para a refatoração do código entre outros recursos que um programador necessita para aumentar sua produtividade e diminuir as tarefas mecânicas do seu cotidiano que podem ser facilmente automatizadas), o que faz com que o desenvolvedor tenha que usar uma IDE para o design de janelas e outro para programar seu sistema (lógica, camada de negócios), o que não é produtivo e ainda por cima é um processo repetitivo, pois a integração do código gerado pelo editor de janelas é feito de maneira manual, o que é um processo muito propenso a erros em sistemas Grandes.



Outros ambientes para desenvolvimento gráfico de janelas para C++ utilizando o WxDevCpp são pagos ou, assim como o WxDevCpp, não possuem uma IDE boa o bastante para fazer com que o programador realmente utilize o editor visual como uma IDE para o desenvolvimento da lógica da aplicação.

Como observado na tabela 2.1 e na descrição dos ambientes já disponíveis, o fator que mais fez falta aos ambientes é a presença de uma IDE completa pra desenvolvimento em C++ e não o ambiente visual de programação em si. Ou seja, a IDE é algo prioritário para o ambiente visual que será desenvolvido.

Como o desenvolvimento de uma IDE completa também é algo trabalhoso e existe um projeto de IDE universal, que engloba inúmeras linguagens de programação como Java, C, C++, Php, Pascal, Cobol, entre outras, além de ter um forte esquema de plugins que facilita sua Em contraposição à esses ambientes pobres para o programador existe um projeto de código aberto e livre mantido pela IBM que se chama Eclipse. Este projeto é uma plataforma de desenvolvimento que se baseia no uso de plugins. Utilizando um plugin chamado CDT, tem-se a disposição uma poderosa IDE para o desenvolvimento em C++. Esta IDE usufrui de todas as facilidades disponibilizadas pelo Eclipse que o programador tanto usa, como integração fácil com ferramentas de controle de versão como o CVS ou Subversion, busca inteligente no código, geração automatizada de documentação de código através de templates, formatador automático de código entre muitas outras.

# 3 Solução Proposta

## 3.1 Introdução

Tendo em vista todos os projetos para desenvolvimento automatizado para o wxWidgets há claramente a necessidade de um ambiente completo, onde o desenvolvedor possa fazer tudo que necessita em apenas um lugar.

O ambiente que deve ser oferecido deve conter várias facilidades para o desenvolvedor, assim como desempenhar bem a função de gerar código automaticamente para o wxWidgets e fornecer um ambiente de programação visual como proposto inicialmente.

## 3.2 Soluções Possíveis

A solução mais evidente (e também a mais trabalhosa), seria criar o ambiente de programação visual e as funcionalidades para o editor C++ do zero. Escrever todas as rotinas para colorir e compilar o código, mostrar os erros de sintaxe e linkagem, etc. Com total controle da aplicação pode-se ter até a ilusão de que será uma solução ótima, já que não dependeremos de código de terceiros para essa parte do desenvolvimento, mas indo por este caminho será feita muita coisa que outras pessoas já fizeram em várias ferramentas, e além do mais esse é o caminho que uma das soluções apresentadas seguiu: o wxDevCpp (mostrado na figura 2.5). Por tentar resolver o problema completamente o wxDevCpp não conseguiu ser nem um bom ambiente de programação visual nem um bom editor para código C++, é uma solução incompleta.

Como alternativa, há a possibilidade de usar alguma ferramenta já existente para cuidar da interação do usuário com o projeto C++, algo que já faça makefiles, coloque cor no código, tenha ferramentas para controle de versão e todas as outras necessidades apontadas na tabela 2.1. Essa alternativa pode ser tida à primeira vista como uma “saída preguiçosa”, mas fazendo uma boa escolha de qual ferramenta será utilizada como base para o desenvolvimento ela se mostra como a mais rápida, eficiente e que terá os melhores resultados, tanto a curto como a longo prazo.

### 3.3 Solução Escolhida

Como o desenvolvimento de uma IDE completa é algo trabalhoso foram feitas pesquisas para encontrar alguma IDE que pudesse ser usada e estendida para servir como base para o projeto. Foi encontrado um projeto de IDE universal, que engloba inúmeras linguagens de programação como Java, C, C++, Php, Pascal, Cobol, entre outras, além de ter um forte esquema de plugins que facilita sua extensão enormemente. Em contraposição aos ambientes pobres para o programador que foram apresentados no capítulo anterior existe um projeto de código aberto e livre mantido pela IBM que se chama Eclipse. Este projeto é uma plataforma de desenvolvimento que se baseia no uso de plugins para sua extensão. Utilizando um plugin chamado CDT, tem-se a disposição uma poderosa IDE para o desenvolvimento em C++. Esta IDE usufrui de todas as facilidades disponibilizadas pelo Eclipse que o programador tanto usa, como integração fácil com ferramentas de controle de versão como o CVS ou Subversion, busca inteligente no código, geração automatizada de documentação de código através de templates, formatador automático de código entre muitas outras, contemplando todos os itens da tabela 2.1 onde é mostrado a solução desejável. O único item não contemplado é a existência de um ambiente visual de programação para wxWidgets na plataforma Eclipse, que é exatamente o que este projeto se propõe a implementar.

A plataforma Eclipse é bastante madura e pode ser utilizada para que, através de um plugin, seja oferecido um ambiente visual de programação para wxWidgets de uma maneira visual e que explore todo o potencial do Eclipse. O projeto então será um plugin totalmente independente para a plataforma Eclipse, para disponibilizar aos usuários uma maneira automatizada e visual de desenvolver suas aplicações visuais, pois além dessa plataforma ser altamente robusta e eficaz para a programação na linguagem C++, ela oferece ao programador diversas facilidades necessárias ao conduzir um projeto relativamente grande e com particularidades próprias. Por exemplo, pode-se na plataforma Eclipse ao programar um sistema operacional em C++ já definir algum programa externo(e.g. QEmu) para emular uma máquina virtual de tal maneira que basta acessar um atalho no teclado que nosso sistema operacional seja compilado e lançado nesse programa externo. Mas mesmo possuindo tal robustez para o desenvolvimento na linguagem C++ a plataforma Eclipse ainda não possui várias facilidades para a linguagem, dentre elas a integração automática com ferramentas de documentação de código(e.g. Doxygen), com ferramentas de teste de unidade(e.g. CppUnit) e também não possui qualquer ambiente para programação visual em C++.

Entretanto, o Eclipse possui um plugin chamado Visual Editor(também conhecido pelas iniciais: VE), que é um framework para o desenvolvimento de ambientes visuais de programação genéricos. O Visual Editor tem como padrão a edição e geração de código na linguagem Java, mas sua documentação deixa explícito que pode ser usada por qualquer linguagem, por utilizar um modelo de descrição interno que independe da linguagem final se será gerada e compilada. Como se trata de um plugin do Eclipse que já se encontra em um estado

estável, será utilizado como base para a geração de código e interação com o usuário. dentro da plataforma Eclipse.

### 3.4 Requisitos Iniciais para o desenvolvimento

Como foi escolhido o Eclipse como base para o desenvolvimento e também alguns plugins iniciais, teremos como requisitos para o início do desenvolvimento do ambiente:

<i>Requisitos</i>
wxWidgets : Framework C++ que irá fazer com que as interfaces gráficas sejam criadas
Eclipse : Plataforma onde o ambiente visual de programação irá rodar
Plugin CDT : Irá gerenciar toda a interação com o compilador C++
Plugin Visual Editor : Irá gerenciar toda a interação do usuário com o ambiente visual
Plugin EMF : Plugin requerido como dependência do Visual Editor
Plugin GEF : Plugin requerido como dependência do Visual Editor
Plugin JEM : Plugin requerido como dependência do Visual Editor

Tabela 3.1: Requisitos iniciais para o desenvolvimento

Para que a solução proposta seja melhor compreendida, serão explicados cada requisito inicial que foi levantado até agora e mostrado na tabela 3.1.

### 3.5 O Framework wxWidgets

“wxWidgets é uma API C++ fácil de usar e de código aberto para escrever interfaces gráficas com o usuário que rodam em Windows, Linux, Unix, Mac OS X, e até Pocket PC dando suporte ao aspecto visual nativo de cada plataforma com virtualmente nenhum código adicional. (...) De AMD à AOL, Lockheed Martin à Xerox, os melhores desenvolvedores estão usando wxWidgets para economizar dinheiro, aumentar eficiência e alcançar novos mercados” [9], é assim que os desenvolvedores deste framework o descrevem.

Há 20 anos que este framework é desenvolvido e aprimorado e mesmo sendo um projeto consideravelmente antigo é bastante ativo, tendo versões novas com mais funcionalidades e correções de bugs bimestralmente e

com uma comunidade de desenvolvedores bastante grande.

Seu uso é relativamente simples e será ilustrado a partir de um exemplo minimalista retirado de [9].

### 3.5.1 Uma aplicação em wxWidgets

Toda aplicação em wxWidgets deve definir uma classe de aplicação estendendo a classe **wxApp**. Deve haver uma única instância dessa classe, que será a thread principal da aplicação. No mínimo a classe deve definir a função **OnInit()** que será chamada quando a aplicação é iniciada (equivalente à classe main)

A menor declaração de uma classe de aplicação que pode ser escrita é:

```
class MyApp : public wxApp
{
public:
    // Called on application startup
    virtual bool OnInit();
};
```

A implementação de **OnInit** geralmente cria uma janela, interpreta argumentos da linha de comando, inicializa dados da aplicação ou faz qualquer outra tarefa de inicialização requerida pela aplicação. Caso a função retorne **true** a aplicação iniciará e seu loop de execução será chamado. Caso retorne **false**, o wxWidgets irá limpar sua estrutura interna alocada na memória e o programa irá terminar sua execução.

Uma implementação simples de **OnInit** seria

```
bool MyApp::OnInit()
{
    // Create the main application window
    wxFrame *frame = new wxFrame(wxT("Minimal wxWidgets App"));

    // Show it
    frame->Show(true);

    // Start the event loop
    return true;
}
```

Isso cria uma instância da classe `wxFrame`(que é uma classe padrão do `wxWidgets` que define uma janela) e retorna `true` para iniciar o programa. O título da janela é passado usando uma macro `wxT()`. Essa macro é utilizada para converter automaticamente strings caso o programa tenha que usar alguma codificação diferente (como por exemplo a codificação Unicode) e também para facilitar a internacionalização da aplicação.

Deve-se também utilizar a macro `IMPLEMENT_APP` para definir que o nosso programa será um aplicativo `wxWidgets`, sendo necessário incluir no arquivo de implementação:

```
IMPLEMENT_APP(MyApp)
```

E também no arquivo de implementação a macro:

```
DECLARE_APP(MyApp)
```

Para que o `wxWidgets` saiba que a nossa aplicação é um `MyApp` e não um `wxApp`.



Figura 3.1: Resultado da aplicação mínima escrita em `wxWidgets`

O resultado deste código pode se visto na figura.3.1.

Como pode ser notado a arquitetura de uma aplicação em wxWidgets é altamente simples, o que facilitará a geração automática de código e visualização para a programação visual utilizando este framework.

## 3.6 O Projeto Eclipse

Segundo Erich Gamma(autor do livro Design Patterns) e Kent Beck(autor do framework JUnit para testes de unidade em java), “Eclipse é uma plataforma universal de ferramentas, um ambiente de desenvolvimento aberto e extensível para tudo e nada em particular. Eclipse representa uma das mais excitantes iniciativas vindas do mundo de desenvolvimento de aplicativos em um longo tempo e possui considerável suporte pelas companhias líderes no setor tecnológico.” [3]. Eclipse não é apenas um projeto, uma plataforma, é toda uma comunidade open source cujo objetivo é prover uma plataforma de desenvolvimento extensível(atraves de plugins) e frameworks para aplicativos. A fundação Eclipse não possui fins lucrativos e foi formada para avançar a criação, evolução, promoção e suporte da plataforma Eclipse e cultivar tanto a comunidade open source quanto um ecossistema de produtos, potencialidades e serviços.

As ferramentas da plataforma Eclipse dá aos desenvolvedores liberdade em um ambiente multi-linguagem, multi-plataforma(sistemas operacionais). A utilização de plugins facilita a criação, integração e utilização das ferramentas de software, economizando tempo e dinheiro. Os próprios desenvolvedores da plataforma Eclipse ficaram chocados com a magnitude do projeto que deveria ser desenvolvido, pois “eles queriam unificar todos seus ambientes de desenvolvimento em uma única base de código” [2].

De fato, esta plataforma é mais do que completa, e é aceita por muitos desenvolvedores como uma solução extremamente boa para quem procura um ambiente de desenvolvimento para virtualmente qualquer linguagem de programação e até para coisas que não envolvem programação, como editoração em Latex por exemplo.

Essa plataforma está disponível para Linux, HP-UX, AIX, Solaris, QNX, Mac OS X e Windows.

### 3.6.1 História do Eclipse

Em 2001 a Borland, IBM, MERANT, QNX Software Systems, Rational Software, Red Hat, SuSE, TogetherSoft e Webgain formaram o conselho ‘eclipse.org’. Ao final de 2003 já havia mais de 80 membros.

No dia 2 de fevereiro de 2004 Eclipse virou uma fundação sem fins lucrativos, o que fez com que a fundação Eclipse virasse um projeto independente(auto-suficiente) que levou a plataforma ao desenvolvimento que beneficiaram seus desenvolvedores e usuários. Toda tecnologia desenvolvida por esta comunidade é disponibilizada pela licença royalty-free chamada de ‘Eclipse Public License’.

Com a mudança para uma corporação independente e sem fins lucrativos, um grupo organizador foi estabelecido para lidar com desenvolvedores comerciais e consumidores, instituições acadêmicas e de pesquisa, além de coordenar os projetos open source.

O Eclipse já possui 9 grandes projetos que englobam um total de mais de 50 subprojetos.

## 3.7 O Visual Editor

O Visual Editor(VE) é um dos grandes projetos da plataforma eclipse. Este projeto é mantido livre, open source e já teve contribuição de muitos desenvolvedores da comunidade Eclipse.

Este projeto é um editor visual de janelas e widgets em geral na plataforma Eclipse e possui originalmente implementações para os frameworks de Java Swing/JFC and SWT/RCP.

Os componentes disponibilizados permitem visualmente editar classes em java que possuem representação gráfica(como janelas e botões). O componente visual é integrado com o editor java do Eclipse, fazendo com que uma vez que a classe é editada sua representação visual é imediatamente atualizada na visualização do editor visual do VE.

Assim como quase todos os projetos do Eclipse o Visual Editor é disponibilizado através da ‘Eclipse Public License’

### 3.7.1 Uma visão funcional do Visual Editor

Janak Mulani and Sibylle Peter descrevem o plugin Visual Editor como sendo composto das seguintes partes( ver [5] ):

- Paleta de Componentes
- Editor Gráfico
- Editor de Código
- Tabela de Propriedades
- Árvore de Componentes

A figura 3.2 identifica as partes em uma janela Eclipse.



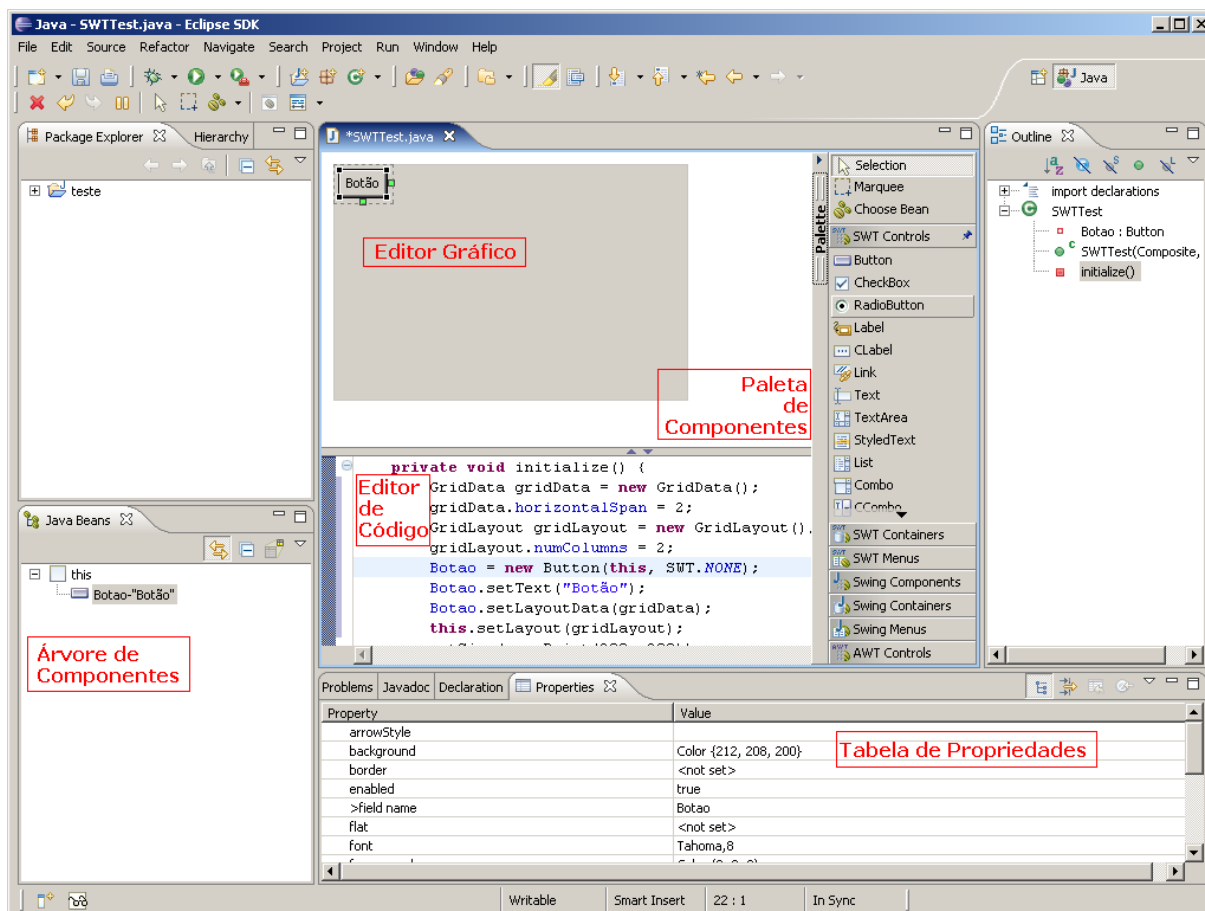


Figura 3.2: Visão funcional do visual editor

### 3.7.1.1 Paleta de Componentes

A paleta de componentes contém todos os componentes que podem ser visualizados e adicionados ao editor gráfico. O usuário deve selecionar o componente com um clique do mouse e jogar o componente dentro do editor gráfico selecionando a posição desejada e confirmando com outro clique do mouse. Ao mesmo tempo que o editor gráfico mostra a atualização visual o código pertinente ao componente que foi incluído é adicionado ao editor de código.

### 3.7.1.2 Editor Gráfico

O editor gráfico é um canvas que mostra uma pré-visualização da interface que se está desenvolvendo. Os componentes são mostrados em sua representação gráfica final. A cada modificação no editor gráfico é atualizado o código-fonte correspondente ao componente alterado. Os componentes podem ser modificados diretamente ou por meio de interação de menus popup. As mudanças no código-fonte também são mostradas no editor gráfico e por causa disso o editor gráfico não é apenas uma ferramenta de edição da interface, ele

também serve para mostrar o efeito de modificações no código fonte ao longo do desenvolvimento

### 3.7.1.3 Editor de Código

O editor de código é o centro de escrita/leitura de código-fonte referente às interfaces gráficas sendo desenvolvidas. Mudanças feitas no editor gráfico, árvore de componentes ou tabela de propriedades são refletidas no editor de código e vice-versa.

Este é a única parte do Visual Editor que é persistente, já que não é usado nenhum tipo de meta-dado para descrever a interface sendo desenvolvida. Logo, quando o Visual Editor é aberto, para determinar o estado inicial dos componentes o código-fonte deve ser interpretado e analisado. Essa análise cria todos os componentes com seus estados iniciais e relações entre eles. Todas as modificações que são feitas e refletidas no Editor Gráfico são apresentadas instantaneamente se o código for reconhecido como um padrão válido de código para a interface gráfica. A maioria das expressões deve ser avaliada com sucesso, caso o Visual editor falhe em avaliar alguma expressão ele exibe um aviso.

### 3.7.1.4 Tabela de Propriedades

A tabela de propriedades mostra todos os atributos referentes a um componente, assim como sua relação com os outros componentes(seu componente pai por exemplo). Esta tabela mostra as propriedades do componente que está selecionado no Editor Gráfico ou na Árvore de Componentes(que deve ser o mesmo). Sua alteração terá reflexos no Editor de Código.

### 3.7.1.5 Árvore de Componentes

A árvore de componentes mostra a estrutura hierárquica dos componentes. A seleção de um componente nesta árvore e a seleção do componente no editor gráfico estão sempre sincronizadas. A árvore tem a seguinte estrutura:

- cada nodo representa um componente
- cada nodo terá um ícone igual ao ícone da paleta de componentes que representa seu componente
- mostrará algumas vezes propriedades do componente ao lado do ícone

Essa árvore pode ser usada para reorganizar a hierarquia de componentes, facilitando a troca de pais ou reordenação de componentes de um certo pai. Pode-se também manipular algumas propriedades ou eventos de um componente com o auxílio de um menu popup.

### 3.7.2 Uma visão da arquitetura do Visual Editor

s Por detrás de cada parte funcional do Visual Editor existem inúmeros componentes arquiteturais que

## Arquitetura do Visual Editor em Alto Nível

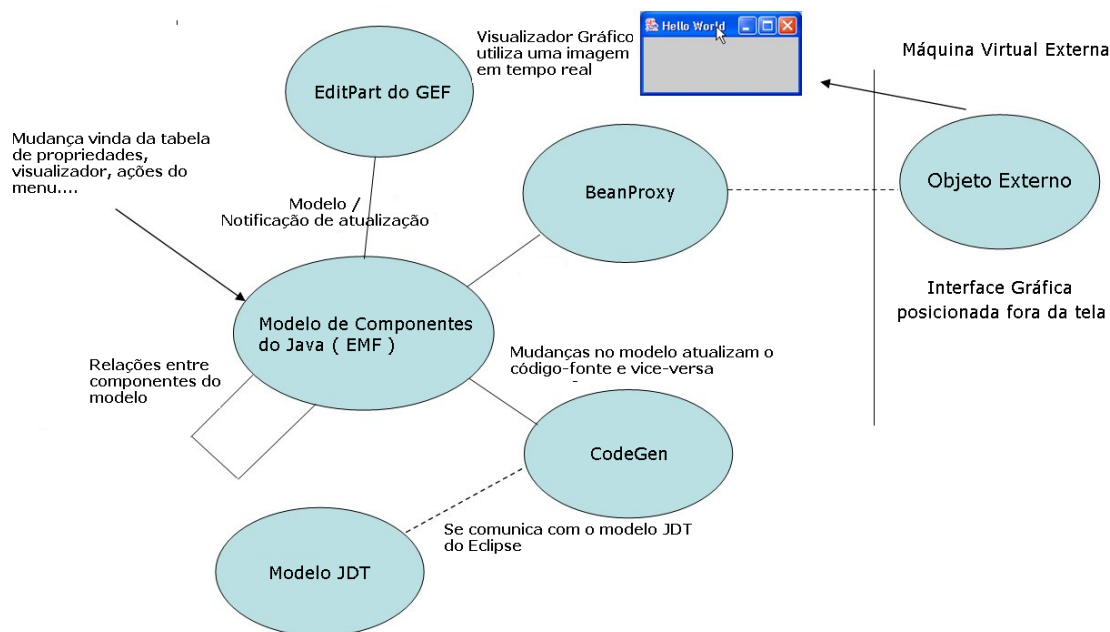


Figura 3.3: Visão geral da arquitetura interna do visual editor

implementam sua funcionalidade. Por exemplo, o Editor Gráfico é composto pelo framework GEF(Graphical Editor Framework) juntamente com o framework EMF(Eclipse Modelling Framework). O GEF é basicamente um framework que facilita a criação de um editor gráfico genérico, que pode ser um editor gráfico de imagens, dataflow ou, como no nosso caso, um editor de interfaces gráficas com o usuário. O GEF é usado como uma “tela de desenho” com funcionalidades de interação com o usuário, é um lugar onde podemos desenhar nossa interface gráfica e obtemos quase sem nenhum custo a interação do usuário com o editor, como captação de cliques e movimento do mouse. O EMF é um descritor de modelos genérico, ele é usado para descrever a interface gráfica como um modelo hierárquico de maneira neutra na forma de metadados. As janelas são descritas pelo EMF e cabe ao framework gráfico sendo implementado pelo Visual Editor à saber se desenhar no editor fornecido pelo GEF. No caso do padrão, que é java, eles conseguem ser mostrados na tela através de java beans(java.beans.\*) e reflexão(java.lang.reflect.\*). Esses dois padrões não podem ser utilizados diretamente pelo Visual Editor pelo fato de que sua linguagem padrão é Java, e o Visual Editor se propõe a ser independente de linguagem, ou seja, um modelo EMF pode ser também utilizado por outras linguagens, no caso deste projeto especificamente será utilizado C++ e o framework wxWidgets para representar o modelo EMF, já que o código fonte gerado será exatamente para o wxWidgets. Essa interação do GEF com o EMF está explicitada na

parte mais ao topo da figura 3.3. Nesta figura está representado que componentes do EMF podem ter relação entre si mesmos (como no caso de composição/especialização parecido com orientação a objetos) e também há a influência externa proveniente da mudança de propriedades, como por exemplo quando há uma Janela representada e ela tem seu tamanho modificado, isso acarreta em uma mudança no modelo de componentes EMF, pois seus metadados teriam que ser modificados, e essa modificação seria refletida no editor gráfico para mostrar ao usuário que sua mudança ocorreu com sucesso (uma vez que o usuário vê apenas a representação gráfica por estar em um ambiente visual de programação).

O modelo EMF também está diretamente relacionado com o CodeGen, que é o gerente de toda a geração de código-fonte. Vale a pena resaltar que mesmo havendo uma relação direta entre eles, o modelo EMF independe da geração de código. Caso nenhum código seja gerado, ou o código errado seja gerado para um dado componente do modelo sua representação em metadados e gráfica continuará correta. O trabalho do CodeGen é apenas sincronizar a representação que temos em metadados no código fonte (ele não se preocupa com a representação gráfica, pois assume que ela estará sempre correta assim como a representação gráfica assume que a geração de código está sempre correta).

Além de gerar o código, o CodeGen é encarregado da serialização e decodificação do código. O CodeGen possui um decodificador associado para cada expressão gerada no código fonte, esse decodificador é encarregado de decodificar uma e apenas uma expressão do código fonte e atualizar o modelo de metadados que está representado no EMF. Esse decodificador é utilizado ao carregar interfaces gráficas que foram salvas em um projeto. Há a necessidade de um decodificador pois nenhum metadado é salvo com o projeto (após a edição ambos os modelos EMF e GEF são descartados da memória), logo cabe ao CodeGen carregá-los.

O GEF, que é usado no editor gráfico, usa figuras leves para representar cada componente. Essas figuras estão descritas no pacote Draw2D do Eclipse. Cabe ao GEF organizar todas as figuras e montá-las de acordo com seu layout e ordem no editor gráfico, respeitando ordem de criação de componentes e hierarquia.

# 4 Desenvolvimento

## 4.1 Introdução

Até agora foi explicado explicitamente o problema para o qual este projeto propõe-se a apresentar uma solução, suas possíveis soluções e qual dentre estas que foi escolhida, além de apresentar detalhadamente todas as plataformas, ferramentas e frameworks que serão utilizados como base para o desenvolvimento, uma vez que o projeto não será escrito completamente do rascunho, e sim, serão utilizados inúmeros projetos que servirão de base, diminuindo o tempo de desenvolvimento e agregando mais valor ao produto final que será desenvolvido.

Neste capítulo será abordada a maneira de como deve ser desenvolvida a solução, levando em consideração tudo que já foi abordado até aqui. O foco principal do capítulo será no Visual Editor e como ele deve ser estendido para suportar uma linguagem de programação diferente da sua linguagem padrão que é Java.

## 4.2 Levantamento de Requisitos

Realizar tudo o que foi proposto até agora torna esta projeto bastante complexo, pois envolve programação para vários sistemas operacionais diferentes e para várias necessidades diferentes de aplicativos que serão gerados. No início do projeto houve a tentativa de fazer algo que funcionasse independente de plataforma e de uma maneira ótima, mas esse objetivo se mostrou inviável para o tempo disponível de menos de um ano de desenvolvimento. Logo os requisitos iniciais que foram escritos para o ambiente que se era desejado construir e disponibilizar deveriam ser reduzidos para que haja algo de concreto ao final do prazo para a apresentação final do projeto. Anteriormente os requisitos eram de um ambiente completamente independente de sistema operacional, que apresentasse diversos recursos para o programador e que abrangesse todas as funcionalidades do wxWidgets, usando o Eclipse como base para o desenvolvimento. Esse requisito foi reduzido a alguns sistemas operacionais alvo, Linux e Windows.

O framework wxWidgets também não será completamente suportado, isto é, não será explorada toda a sua potencialidade pelo editor visual, fazendo com que muito dos seus componentes para integração na interface gráfica com o usuário poderão ser utilizados da maneira clássica de editar o código fonte utilizando um editor de texto.

Mesmo tendo sido reduzido a abrangência do projeto, esses requisitos são o suficiente para a criação de muitos tipos de aplicações e com a sua implementação terminada há a possibilidade da inclusão de novos

componentes(visuais e não-visuais), plataformas e funcionalidades com o tempo.

### 4.3 Primeiros Passos

Como o projeto será feito sobre a plataforma Eclipse, um dos primeiros passos a se tomar deve ser saber como a plataforma Eclipse trata plug-ins, como funciona a estrutura interna do Eclipse para isso e, finalmente, como se deve desenvolver plug-ins para o Eclipse.

Primeiramente deve-se ressaltar mais uma vez que o Eclipse não é um programa, um aplicativo, qualquer. É uma plataforma que trabalha por meio de plug-ins para melhorar sua extensibilidade e para tonar a plataforma livre de linguagens de programação ou fins específicos. A plataforma Eclipse, resumidamente, é um pequeno kernel ao redor do qual temos vários plug-ins que dão funcionalidades à plataforma. Esse pequeno kernel em si não faz nada além de dar serviços para plug-ins. Esses plug-ins por sua vez que têm o trabalho de ser uma IDE Java, C++, um editor de textos ou um editor gráfico. Plug-ins podem também apenas fornecer serviços para outros plug-ins, como é o caso de vários plugins do Eclipse. Essa estrutura será explicada detalhadamente mais adiante.

Como se tornou um projeto aberto à comunidade, com pessoas de todo o mundo desenvolvendo e oferecendo soluções novas para a plataforma Eclipse, o desenvolvimento de novos plug-ins deve obedecer à algumas normas e padrões. Essas normas e padrões existem para facilitar a integração dos plug-ins, aumentando a reusabilidade de código de terceiros, além de facilitar o entendimento do código-fonte de plug-ins para que mais desenvolvedores possam ajudar nos projetos de plug-ins existentes, aumentando suas funcionalidades e, conseqüentemente, aumentando as funcionalidades da plataforma Eclipse. Esses padrões e normas podem ser vistos detalhadamente na seção de documentação localizada no site principal do projeto( <http://www.eclipse.org> ).

Sabendo como funciona a estrutura interna do eclipse e quais padrões de programação deve-se seguir ao desenvolver um plug-in última coisa que resta é saber exatamente como desenvolver um plug-in, por onde começar, quais ferramentas devem ser utilizadas, como testar o plugin e como distribuí-lo posteriormente. Estas questões serão abordadas nas próximas seções de modo a facilitar a compreensão da extensão do plug-in Visual Editor.

### 4.4 O modelo de plug-ins do Eclipse

Como pode-se ver na figura 4.1 internamente o Eclipse é formado por dois elementos centrais, o Java Development Tools e o Plugin Development Environment. O Java Development Tools fornece ferramentas

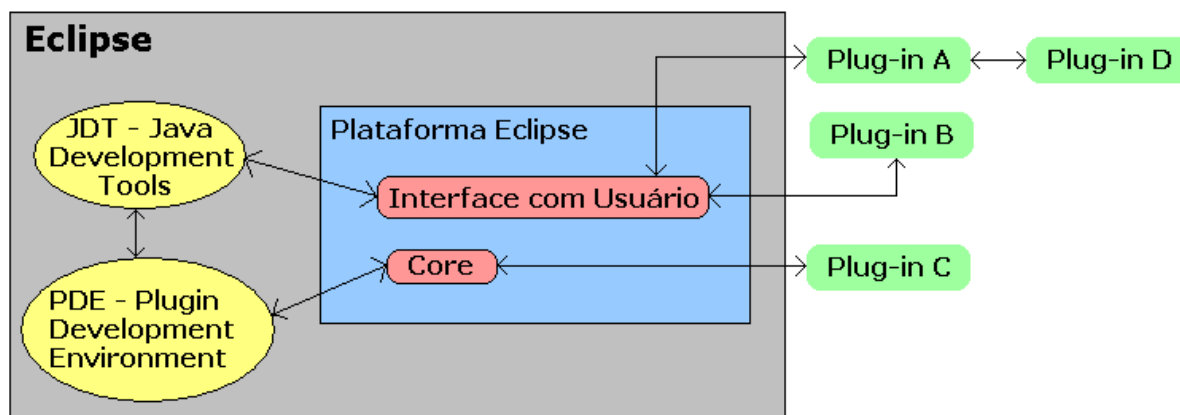


Figura 4.1: Arquitetura resumida do Eclipse

para o desenvolvimento final de plug-ins, além de adicionar a capacidade de gerenciamento de projetos na linguagem Java para o Eclipse. Essas capacidades podem servir como base para o desenvolvimento de outras ferramentas, facilitando a integração de novas linguagens e soluções à plataforma.

O Plugin Development Environment é a parte fundamental do Eclipse, pois dá suporte ao desenvolvimento de plug-ins, que são a razão de existência do Eclipse. Como tudo é feito por plug-ins todas as funcionalidades que o Eclipse suportar além do Java Development Tools deve ser feito a partir do Plugin Development Environment. Ele é formado por várias ferramentas no próprio Eclipse para desenvolver, testar, fazer debug, compilar e distribuir plug-ins para o Eclipse.

Logo, a grosso modo, essas duas partes do Eclipse fornecem as funcionalidades básicas para que seja possível que plug-ins sejam feitos e disponibilizados na plataforma. E como ilustrado na figura 4.1 plug-ins podem estender plug-ins já existentes (como no caso do “Plug-in D” que está estendendo o “Plug-in A”).

## 4.5 Configurando a área de trabalho

O que se fazer agora que há um objetivo bem definido é setar uma área de trabalho inicial com todas os requerimentos levantados na tabela 3.1. Com esses requerimentos cumpridos teremos uma área de trabalho inicial para começar a construção do ambiente visual de programação proposto.

O primeiro passo é baixar a plataforma Eclipse e instalá-la no sistema operacional que se está usando. A versão do eclipse que será utilizada é versão 3.2 que pode ser encontrada em <http://www.eclipse.org/>. A instalação do Eclipse é bastante simples, basta baixá-lo e descompactá-lo em um diretório qualquer e começar a usar, sem necessidade de qualquer configuração adicional. Ao abrir o Eclipse pedirá o caminho do workspace

que será utilizado. Um workspace é a área de trabalho onde ficarão todos os projetos os quais serão trabalhados. Como anteriormente pode-se escolher qualquer diretório sem maiores preocupações.

O próximo passo é instalar os plugins que são dependência para o Visual Editor, são eles o plugin GEF, EMF e JEM (como descrito na tabela 3.1). Todos esses plugins são encontrados no mesmo site do eclipse, por serem plugins oficiais mantidos pela IBM. A versão que será utilizada do GEF é a versão 3.2, a versão do EMF é a 2.2.0 e a versão do JEM será a 1.2.3 como requerido pelo Visual Editor. Para instalar cada um desses plugins basta baixá-los no site do Eclipse e logo em seguida descompactá-los na mesma pasta onde foi instalado o Eclipse.

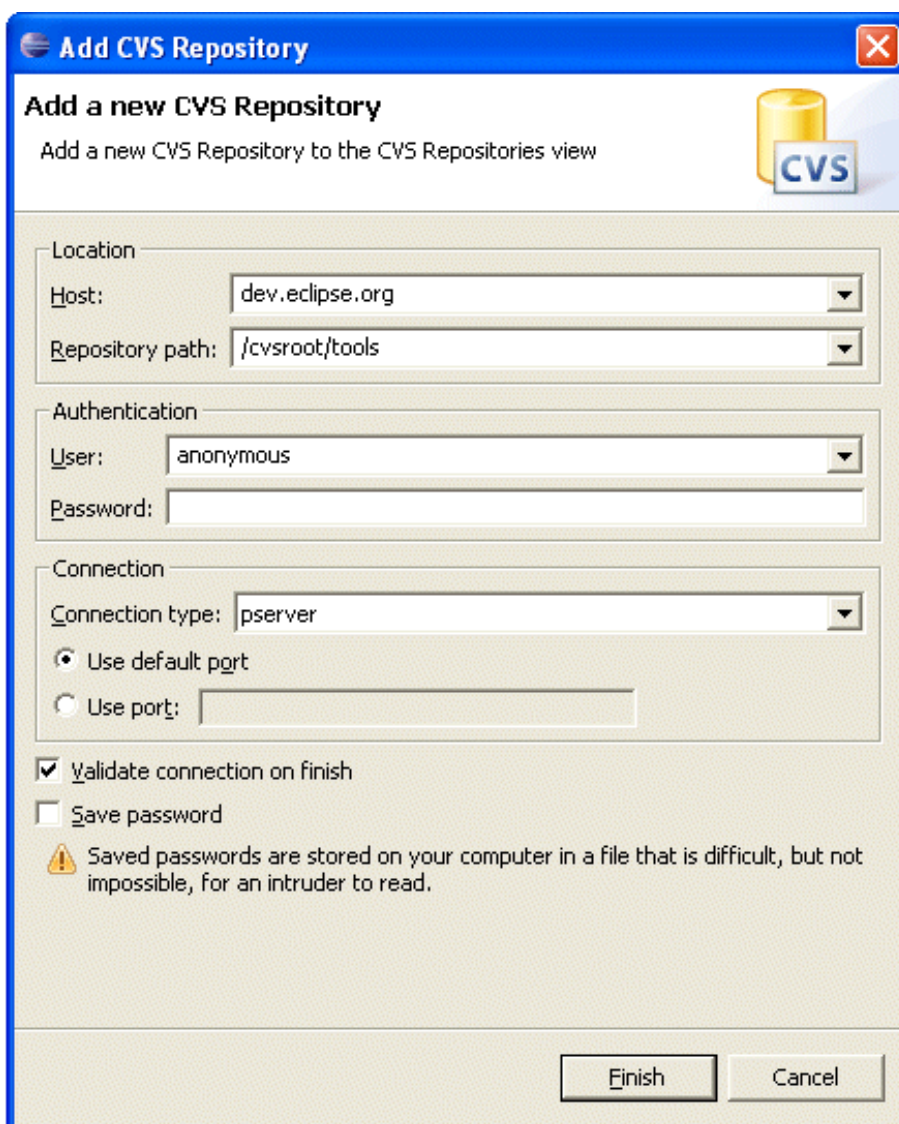


Figura 4.2: Configurações para o CVS

Com todas as dependências do Eclipse instaladas deve-se baixar o código-fonte do Visual Editor utilizando o servidor CVS do Eclipse. Pode-se fazer isso facilmente utilizando o próprio Eclipse, abrindo o menu “CVS Repositories”, clicando com o botão direito na tela apresentada e selecionando “New” – “Repository Loca-



tion...”. Após seguir esses passos será apresentada uma tela pedindo as informações referentes ao servidor CVS. No campo “Host” deve-se colocar “dev.eclipse.org”, no campo “Repository Path” colocar “/cvs-root/tools” e no campo “User” colocar “anonymous”. A tela final, após entrar com todas essas informações, pode ser vista na figura 4.2. Após isso será apresentada uma árvore com a estrutura dos projetos guardados no CVS. Os projetos “ve-dev-base” e “ve-dev-others” devem ser baixados.

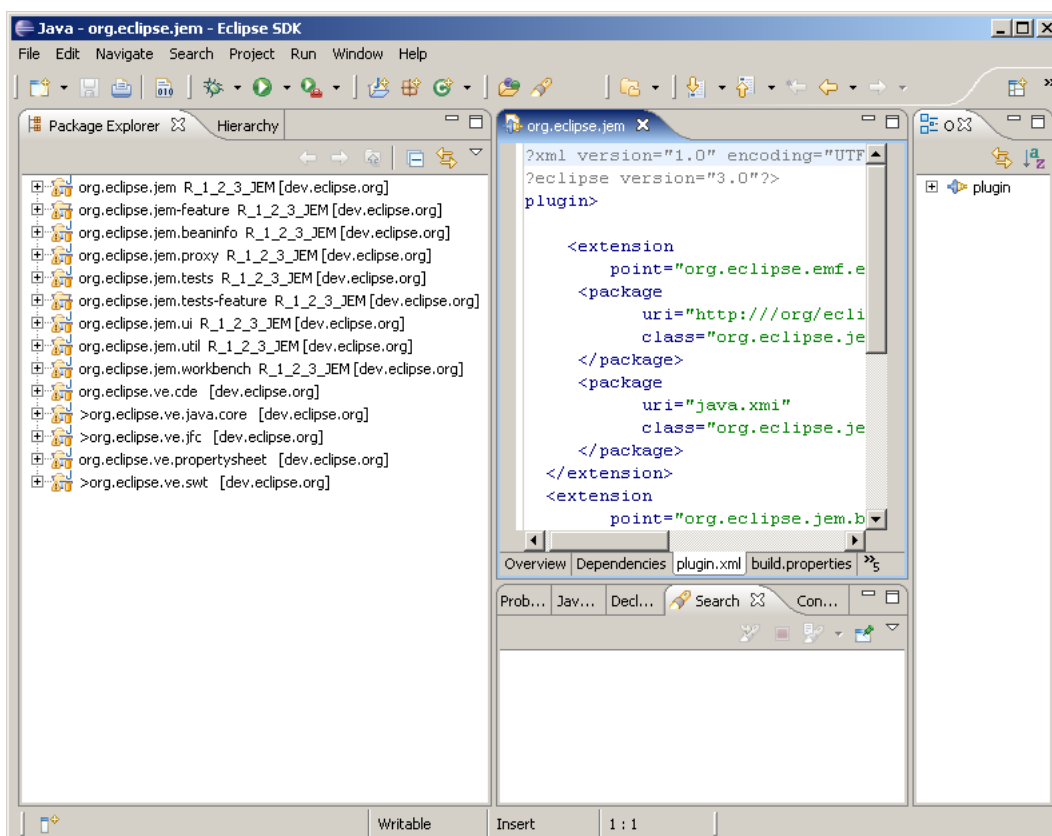


Figura 4.3: Ambiente Inicial

Após setado o ambiente corretamente deve-se obter uma área de trabalho no Eclipse como mostrada na figura 4.3. Para testar se o ambiente está configurado corretamente basta selecionar o subprojeto “org.eclipse.ve.swt”, clicar com o botão direito sobre ele e no menu “Run As...” escolher a opção “Eclipse Application”. Isto deve lançar uma nova instância do Eclipse, onde o projeto do plugin estará disponível e nesse primeiro momento ele dará a funcionalidade de criar interfaces visuais com o usuário para a linguagem java apenas.

Com o ambiente corretamente inicializado, configurado e testado tudo está pronto para o desenvolvimento do ambiente de programação visual para interfaces gráficas com o usuário na linguagem C++ utilizando o wxWidgets.

## 4.6 Extendendo o Visual Editor

Para estender o Visual Editor a primeira ação a se tomar é criar um novo subprojeto na área de trabalho. Esse subprojeto deve ser um novo plug-in vinculado à todos os projetos existentes na área de trabalho inicial, pois estes serão a base do ambiente que será desenvolvido.

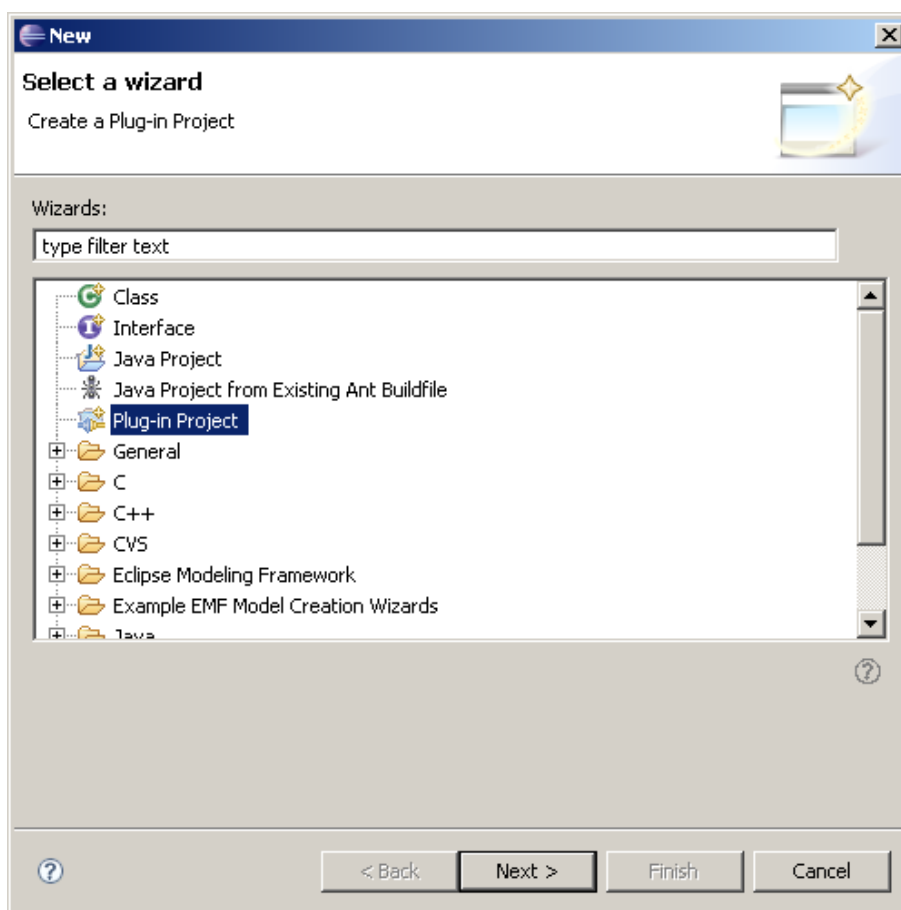


Figura 4.4: Selecionando o projeto de um plugin

Para criar um novo projeto de plugin basta selecionar no menu de novos projetos a opção “Plug-in Project”(como demonstrado na figura 4.4). Logo em seguida será pedido o nome do projeto, por padrão o nome do projeto deve ser igual ao identificador dele na plataforma Eclipse, esse identificador será utilizado por novos plug-ins que desejarem estender ou fazer referência a este plug-in. foi escolhido o nome “org.eclipse.ve.wxwidgets” para seguir o padrão de nomenclatura utilizado já no Visual Editor(figura 4.5).

Após a criação do projeto a seleção das dependências deve ser feitas no arquivo “MANIFEST.MF” que é criado automaticamente para este tipo de projeto. Este arquivo contém as informações mais básicas sobre o plug-in, como suas dependências, seu nome, sua versão, seus pacotes, etc. . . Tendo isso pronto falta implementar as estruturas internas básicas que compõe o Visual Editor mudando seu aspecto de linguagem Java para C++.

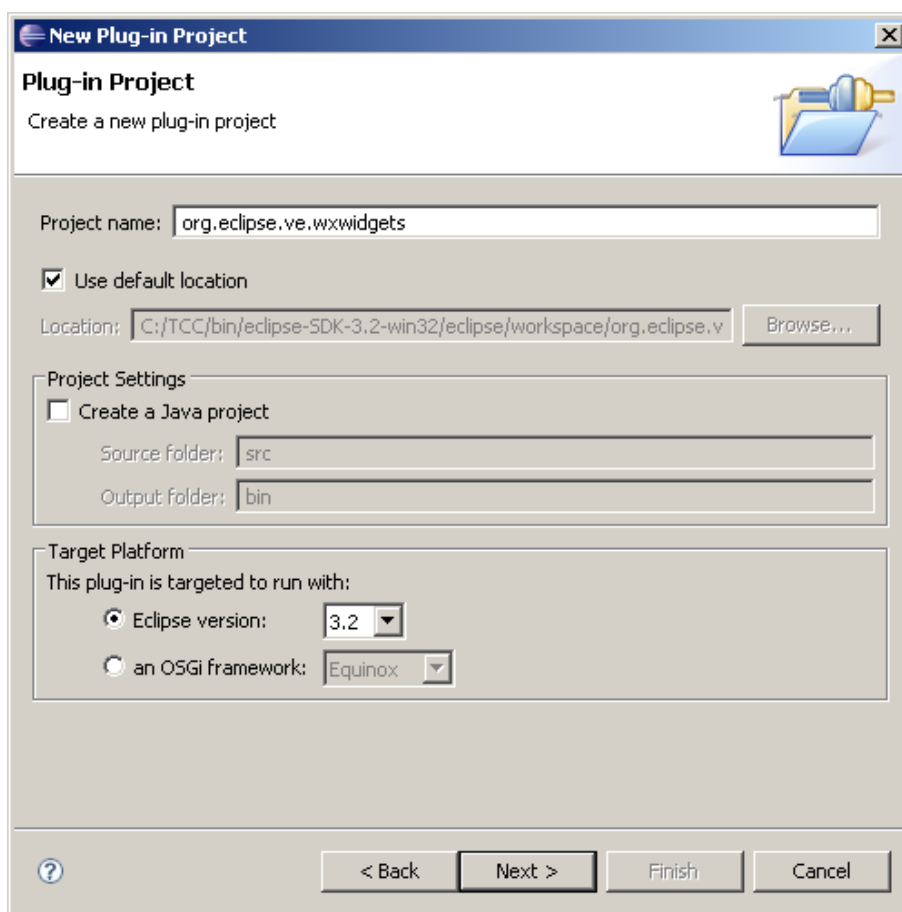


Figura 4.5: Nomeando o projeto

### 4.6.1 Implementação

O início da implementação se dá com a redefinição dos valores padrão do Visual Editor através da criação dos arquivos:

- **plugin.xml** Define as extensões que serão feitas e dependências relacionadas a cada extensão.
- **palette/wxwidgetspalette.xmi** Define como será a nova paleta de componentes (na realidade apenas adiciona os componentes do novo framework separado pelo seu nome)
- **.override** Os arquivos terminados em “.override” definem as extensões feitas a componentes existentes, adicionando relações ou controladores.

#### 4.6.1.1 O arquivo plugin.xml

O arquivo plugin.xml que deve ser criado deve definir como extensões um container para projetos C++, um inicializador próprio para esse container, declarar classes que auxiliarão o layout no editor gráfico e adicionar

## Dependencies

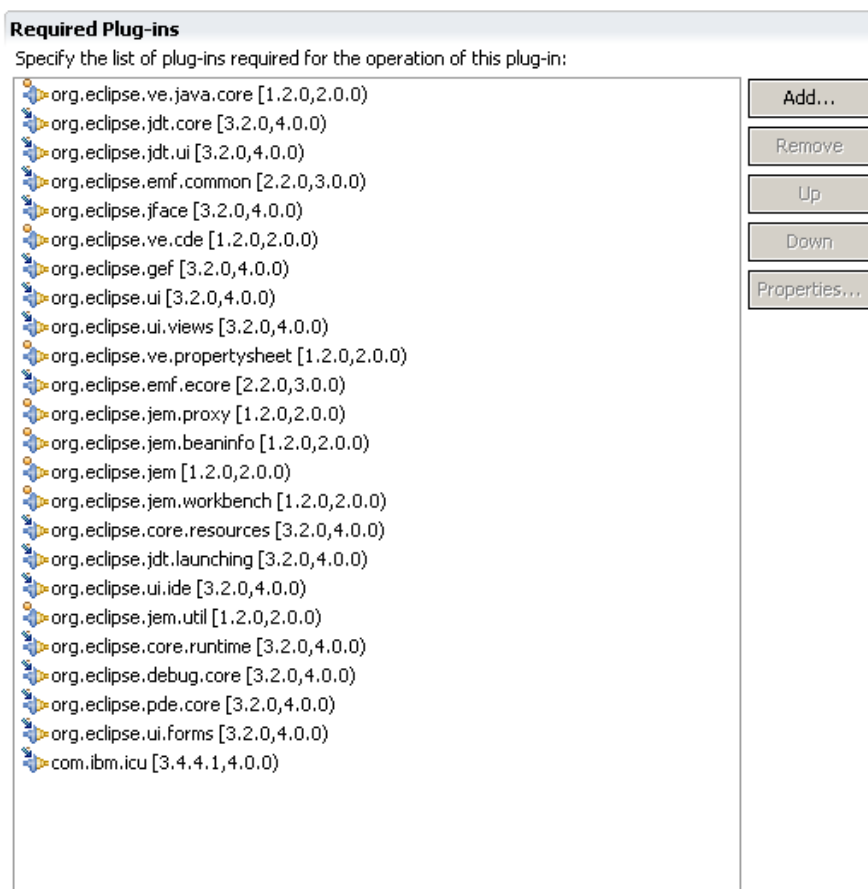


Figura 4.6: Dependências do plug-in

as opções para criação de classes visuais nos menus de criação de arquivos para C++.

O container será simplesmente uma página em uma caixa de diálogo que pedirá o nome da classe visual se irá ser implementada, ela será definida por duas classes Java, a classe “wxWidgetsContainerWizardPage” e a classe “wxWidgetsContainerWizardContent”. A figura 4.7 mostra quais classes existentes podem ser estendidas para fornecer a base para essas duas classes que serão o container definido no arquivo plugin.xml.

Para definir no arquivo essa relação usa-se o seguinte código:

```
<extension
  point="org.eclipse.jdt.ui.classpathContainerPage">
  <classpathContainerPage
    name="%wxContainerPageName"
    class="org.eclipse.ve.internal.wxwidgets.wxWidgetsContainerWizardPage"
    id="WX_CONTAINER">
  </classpathContainerPage>
```

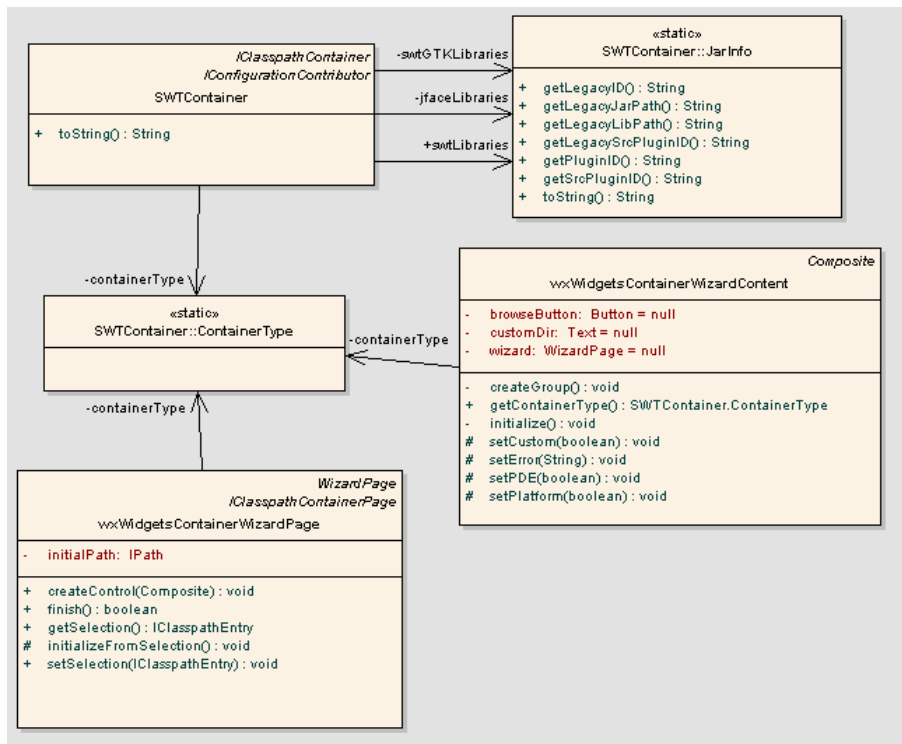


Figura 4.7: Relação dos containers com classes existentes

```
</extension>
```

O código define que a partir de uma classe base do pacote “org.eclipse.jdt.ui” iremos definir a classe. Essa classe será a responsável pela inicialização das classes visuais que o usuário poderá manipular. No caso teremos inicialmente apenas janelas (wxForms) e caixas de diálogo(wxDialogs). Para ser um container qualificado além de usar a classe existente do Visual Editor como base também deve-se implementar a interface “IClasspathContainerPage”.

No arquivo plugin.xml deve também ser definida nossa paleta de componentes que terá como container o container já definido previamente no mesmo arquivo.

```

<extension
    point="org.eclipse.ve.java.core.contributors">
    <palette
        container="WX_CONTAINER"
        categories="palette/wxwidgetspalettexmi"
        plugin="org.eclipse.ve.wxwidgets">
    </palette>
</extension>
  
```

#### 4.6.1.2 A Paleta de componentes

Como definido no arquivo `plugin.xml`, a paleta de componentes estará no arquivo “`palette/wxwidgetspalette.xmi`”. Este arquivo definirá todos os componentes visuais e não-visuais manipuláveis dentro do ambiente visual de programação. No arquivo `xmi`, cada componente será definido por tags `children`.

Cada componente pode ser categorizado, além de estar na categoria principal como um componente `wxWidgets`, em subcategorias como “Componentes de Menu”, “Componentes de Painéis Visuais”, etc.

Um componente para ser válido e instanciável deve ter descrito no arquivo de paleta:

- **icon16Name** Esse é o ícone que será mostrado na paleta de componentes
- **entryLabel** Essa é a legenda que será mostrada como nome do componente. Efetivamente é o que designará um componente para o usuário final do plug-in.
- **objectCreationEntry** Define qual classe será instanciada quando o componente for selecionado na paleta.
- **nameincomposition** Valor do nome da variável que será gerada no código fonte para este componente.

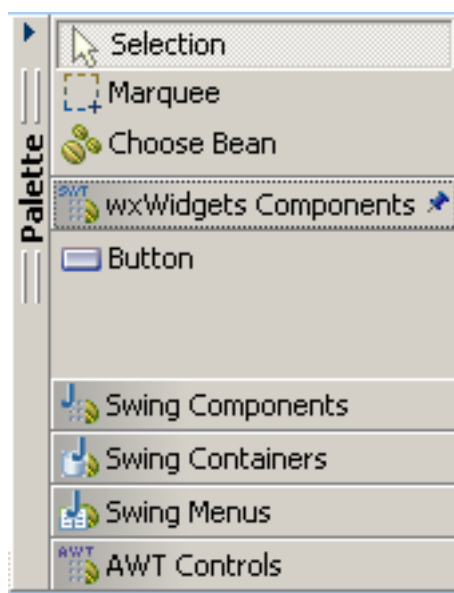


Figura 4.8: Paleta Inicial com um botão

Após definir todos esses requisitos para um componente temos como resultado sua representação na paleta de componentes e por consequência a possibilidade de utilizá-lo (ver figura 4.8). Note que na figura há o componente definido na paleta `wxwidgetspalette.xmi`, mas ainda aparece a possibilidade de escolher componentes do Swing e AWT (que são frameworks para java).

### 4.6.1.3 O Editor Gráfico

No wxVisual Editor, deve haver uma classe raiz para todas as classes gráficas como requerimento do Visual Editor, o qual está sendo estendido. A classe do Visual Editor que deve ser estendida para fornecer as funcionalidades do Editor Gráfico é a “org.eclipse.gef.editparts.AbstractGraphicalEditPart”. Todas as classes visuais em wxWidgets vão ser subclasses dessa extensão ou vão implementar sua própria “AbstractGraphicalEditPart”.

Os métodos mais importantes do editor gráfico são:

- **createFigure()** Método que dá uma imagem para o Editor Gráfico e seta o ambiente para ser desenhado. O layout manager se encarrega de desenhar as imagens na ordem correta.
- **getAdapter()** Retorna um adaptador de modelo para necessidades específicas, como uma interface IPropertySource para a tabela de propriedades, IActionFilter para controle das ações com componentes, IVisualComponent para componentes. . .
- **activate()** Ativa como listener no editor gráfico.
- **deactivate()** Desativa como listener no editor gráfico.

### 4.6.1.4 O uso de um BeanProxyAdapter

A primeira questão que essa seção deve levantar é: “O que é um BeanProxyAdapter?”. A resposta é simples, um BeanProxyAdapter é uma classe que trata de fazer o “meio de campo” entre o que está sendo mostrado no Visual Editor e o que realmente aquela classe é.

Exemplificando, quando temos um botão no Editor Gráfico do visual editor ele têm três formas: sua forma gráfica no Editor Gráfico, sua representação em metadados internos do Visual Editor e sua representação fora do Eclipse, onde o botão está instanciado em uma aplicação “escondida”. O que um BeanProxyAdapter faz é transformar os metadados nessa instância que deve ficar invisível para o usuário do Visual Editor, mas que é usada pelo Editor Gráfico para se ter uma “foto” de como é o componente realmente. Assim o Editor Gráfico consegue mostrar como realmente é o botão colocando sua foto no Editor.

### 4.6.1.5 Geração de Código

A geração de código no Visual Editor se dá por várias partes. A parte mais importante são as classes decodificadoras. As classes decodificadoras agem como mediadores entre o código fonte e o modelo interno do visual editor(que mais tarde é traduzido para uma representação gráfica no editor gráfico). Cada componente

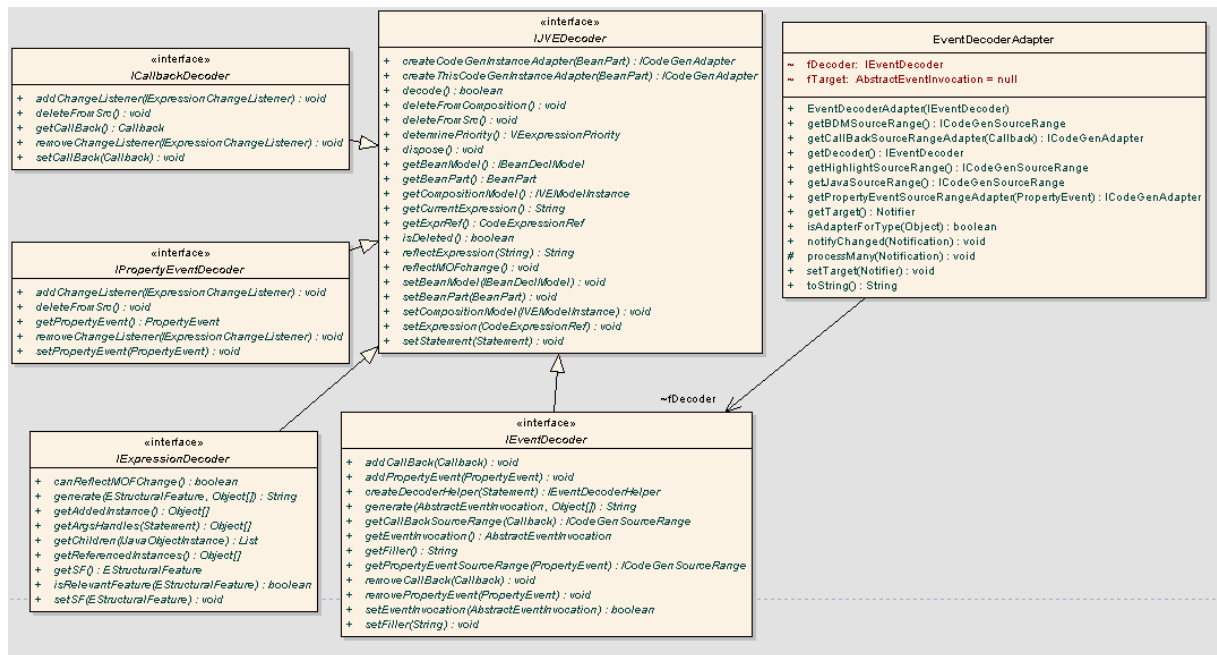


Figura 4.9: Estrutura Interna de Geração de Código

gráfico deve ter seu decoder, que devem obrigatoriamente implementar a interface “org.eclipse.ve.internal.java.codegen.java.IJVEDecoder” direta ou indiretamente.

Na interface IJVEDecoder tudo acontece basicamente no método “decode()”, que deve pegar o código-fonte, decodificar e atualizar o modelo interno do Visual Editor. Para fazer isso são utilizados vários “Decoder Helpers” que são pequenos decodificadores especialistas em uma determinada área. Apesar de ter o nome de decodificador, essas classes também geram o código fonte final, pois elas decodificam o modelo interno de metadados do Visual Editor em uma representação gráfica.

## 4.6.2 Resultado Obtido

Após estender o Visual Editor para obter o wxVisual Editor obteve-se como resultado um editor de interfaces gráficas para C++ com capacidades multiplataformas como o previsto. A próxima seção tratará de mostrar como esse ambiente deve ser usado e as possibilidades que ele oferece, mostrando passo-a-passo o que deve ser feito e em qual ordem.

### 4.6.2.1 Usando o ambiente desenvolvido

Inicialmente deve-se iniciar um projeto C++ a partir do menu “New...”. Este projeto será gerenciado normalmente pelo plugin CDT que cuidará de toda a interação com o compilador e projeto C++. Logo em



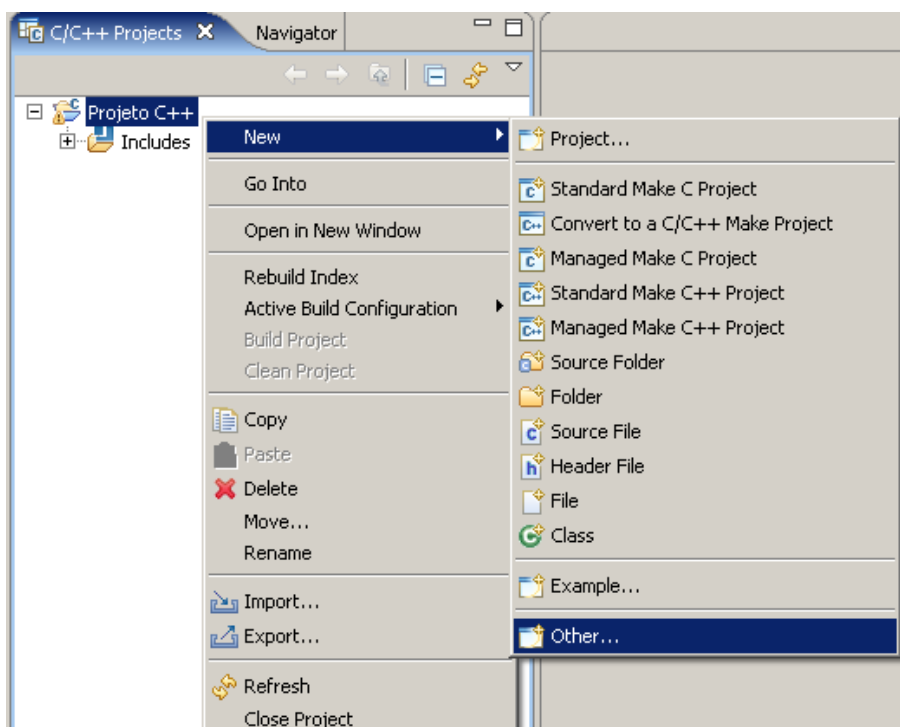


Figura 4.10: Seleção do menu para criar nova classe visual

seguida deve-se adicionar uma classe Visual wxWidgets, para isso deve-se clicar com o botão direito do mouse sobre o projeto C++(ou pasta dentro do projeto ) e selecionar a opção “New...” – “Other...”(ver figura 4.10).

Após a seleção da opção “Other...” irá aparecer uma janela com as várias opções de templates prontos para serem criados. Deve ser selecionado o template “wxWidgets Visual Class” que está na pasta de templates para C++(ver figura 4.11).

Quando for selecionado o template “wxWidgets Visual Class” serão solicitados os atributos iniciais da classe visual. Inicialmente esses atributos são apenas o nome da classe(que dará também nome ao arquivo onde a classe será implementada) e seu tipo. Seu tipo pode ser tanto uma caixa de diálogo(New wxWidgets Dialog) quanto uma janela(New wxWidgets Form). Após selecionar as opções com os valores corretos o botão “Finish” será habilitado(ver figura 4.12).

Automaticamente serão criados automaticamente o arquivo de implementação da classe visual e uma entrada para ele no makefile já existente no projeto(ou no caso de não existir um makefile no projeto é criado um makefile novo) como mostra a figura 4.13.O seguinte código será gerado no editor de código fonte para o arquivo de implementação da nova classe visual:

```
#ifdef __BORLANDC__
    #pragma hdrstop
```

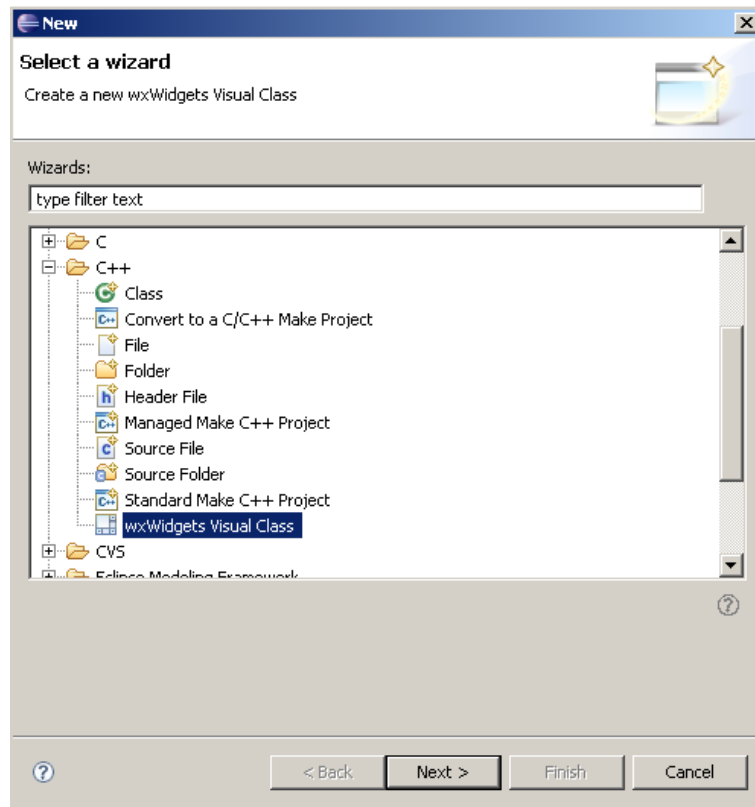


Figura 4.11: Seleção de um template para criar classes visuais

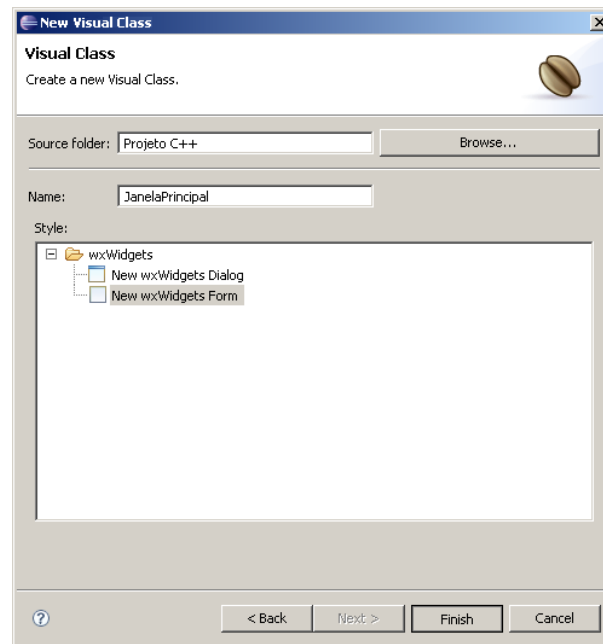


Figura 4.12: Declaração dos atributos iniciais da classe

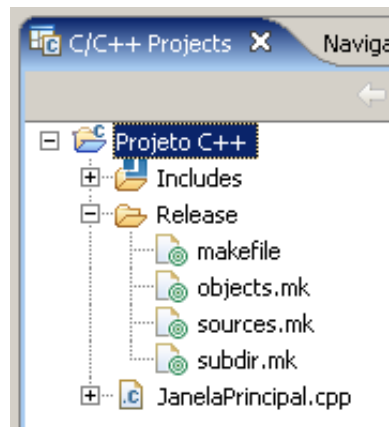


Figura 4.13: Arquivos criados pelo wxVisual Editor

```
#endif
```

```
#ifndef WX_PRECOMP
```

```
    #include <wx/wx.h>
```

```
#endif
```

```
class JanelaPrincipal: public wxFrame
```

```
{
```

```
    public:
```

```
        JanelaPrincipal(wxFrame *frame, const wxString& title);
```

```
        ~JanelaPrincipal();
```

```
    private:
```

```
        DECLARE_EVENT_TABLE();
```

```
};
```

```
BEGIN_EVENT_TABLE(JanelaPrincipal, wxFrame)
```

```
END_EVENT_TABLE()
```

```
JanelaPrincipal::JanelaPrincipal(wxFrame *frame, const wxString& title)
```

```
    : wxFrame(frame, -1, title)
```

```
{
```

```
}
```

```

JanelaPrincipal::~MyFrame()
{
}

```

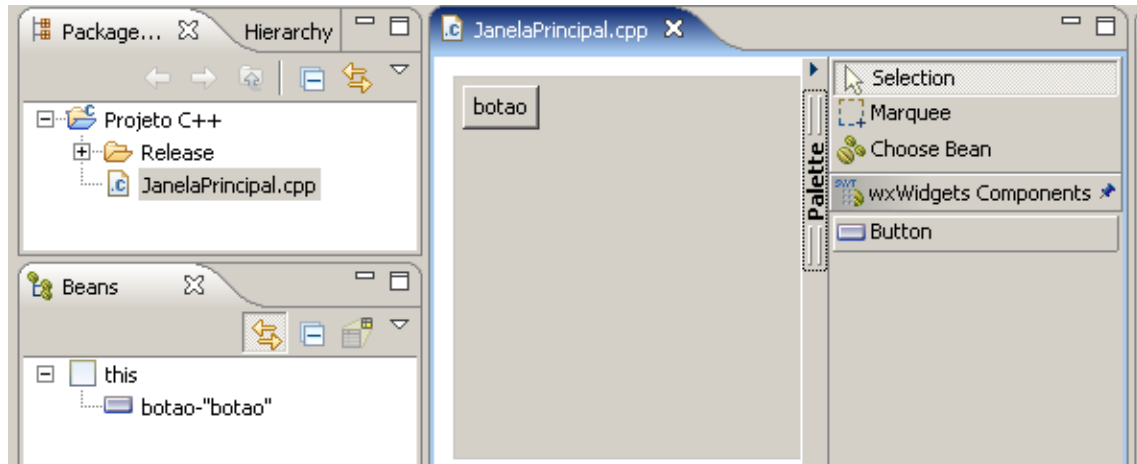


Figura 4.14: Adição de botão provoca atualizações imediatas

Ao mesmo tempo será mostrada uma janela em branco no editor visual, pronta para ter mais componentes adicionados dentro dela. Nesta primeira implementação do editor visual apenas o componente de botão está disponível. Para adicionar um botão basta selecioná-lo na paleta de componentes e arrastá-lo para a janela recém criada. Tanto o código fonte quanto a visualização da janela serão atualizados imediatamente. Assim como um botão é colocado na árvore de relacionamento entre componentes visuais como filho direto de “this” que no caso é uma janela (pois a classe visual que estamos editando foi setada como uma janela), como mostrado na figura 4.14.

Com o código-fonte em mãos deve-se apenas selecionar o projeto C++ e chamar um “Build” para ter um arquivo executável para a plataforma onde se está programando. São suportadas as plataformas Windows e Linux, futuramente podem haver extensões para MacOS e outros sistemas operacionais.

## 5 Conclusão

O projeto wxVisual Editor apresentou uma solução para aumentar a produtividade no desenvolvimento de softwares. O projeto teve como produto final uma ferramenta incompleta e que como as demais apresentadas deixa a desejar em muitos pontos. Esse resultado deve-se ao fato de que um dos projetos escolhidos como base para o desenvolvimento não teve todo o rendimento esperado. O Visual Editor, que foi a grande base do projeto apresentou várias falhas ao ter seu código estendido para suportar uma linguagem completamente diferente da sua linguagem padrão, mesmo tendo uma linguagem de descrição neutra em relação à linguagem de programação.

Um dos melhores aspectos do Visual Editor é também um dos seus grandes defeitos, a ausência de metadados guardados para descrever a interface na qual se está trabalhando. Essa ausência é o que torna necessário o decodificador de código fonte, que funciona de uma maneira esplêndida para interfaces simples, parece ser realmente uma idéia genial, mas após uma reflexão mais calma percebe-se que essa decodificação é altamente propensa a erros quando se constrói uma interface que possui muitos componentes com relações mais complexas que devem ser mudadas no editor de código e não no editor visual.

Outro ponto fraco do Visual Editor é seu código fonte, que é altamente extenso e de difícil compreensão aliado a uma total ausência de documentação, o que afasta desenvolvedores e foi um dos motivos do projeto ficar meses parado sem ter desenvolvedores para mantê-lo.

Mas deixando de lado os problemas encontrados com o framework Visual Editor, este projeto propôs-se a reduzir o tempo de desenvolvimento de interfaces gráficas com o usuário, mas realmente, foi percebido que um ambiente gráfico de programação não consegue essa tarefa mágica de, ao mesmo tempo que reduz o tempo de desenvolvimento, produzir um código fonte limpo e que seja amigável à manutenção de alguém que não utiliza um ambiente gráfico. O código que é gerado automaticamente, por mais empenho que se tenha tido ao fazer o gerador de código, é sempre algo sujo e que se torna ilegível aos programadores.

Logo a grande conclusão tida com o projeto é de que ambientes de programação visual para geração de interfaces não são uma solução muito boa para diminuir custos no desenvolvimento de softwares, pois restringem a maneira de como lidamos com as interfaces gráficas com o usuário como também podem restringir a arquitetura do nosso programa.

## 5.1 Proposição de trabalhos futuros

Para o aperfeiçoamento da idéia inicial deste projeto, que era de auxiliar no desenvolvimento de interfaces gráficas com o usuário, propõe-se que se faça uma boa pesquisa de outras alternativas à utilização de ambientes visuais de programação.

Deve-se pesquisar alguma alternativa que seja incorporada diretamente ao bom e velho editor de textos usado para escrever o código-fonte, esquecendo toda essa idéia de programação visual que para construção de interfaces gráficas não dá bons resultados.

Uma sugestão seria uma paleta de componentes que em vez de ser utilizada em um editor gráfico possa ser utilizada diretamente no código fonte, podendo-se por exemplo adicionar um botão dentro de uma janela apenas arrastando o componente para a declaração da janela, como o código seria atualizado instantaneamente o programador poderia limpar o código de maneira mais rápida e continuar seu trabalho. Essa paleta seria muito útil na hora de escrever os layouts, podendo oferecer uma gama de layouts pré-definidos para seus usuários.

Poderia também ser pesquisada alguma maneira de que a hierarquia entre componentes gráficos sejam mostrados em forma de árvore ao programador que utiliza o editor de textos da mesma maneira que são mostrados para quem usa o editor gráfico, pois essa árvore de componentes é uma das características mais utilizadas quando se faz uma interface gráfica.

Uma última sugestão é alguma maneira de mostrar um 'preview' da interface que se está construindo. Poderia ser feita uma interpretação (talvez feita de maneira grosseira para levar menos tempo que compilar o programa) que mostrasse rapidamente uma boa aproximação do resultado da interface gráfica na qual se está trabalhando.

# Referências Bibliográficas

- [1] Shi-Kuo Chang. *Principles of Visual Programming Systems*. Prentice Hall, New York, 1 edition, 1989.
- [2] Eric Clayberg and Dan Rubel. *Eclipse: Building Commercial Quality Plug-ins*. Addison-Wesley, New York, 2 edition, 2006.
- [3] Erich Gamma and Kent Beck. *Contributing to Eclipse: Principles, Patterns, and Plugins*. Addison-Wesley, New York, 1 edition, 2003.
- [4] James Martin. *Rapid Application Development*. Macmillan Coll Div, New York, 1 edition, 1991.
- [5] Janak Mulani and Sibylle Peter. Implementation of ulc visual editor for eclipse. *Visual Editor Contribution Articles*, 2005.
- [6] Brad A. Myers. *Taxonomies of visual programming and program visualization*. J. Visual Languages and Computing, New York, 1 edition, 1990.
- [7] Brad A. Myers. User interface software tools. *ACM Transactions on Computer-Human Interaction*, 1995.
- [8] N.C. Shu. *Visual Programming*. Van Nostrand Reinhold, New York, 1 edition, 1988.
- [9] Julian Smart, Kevin Hock, and Stefan Csomor. *Cross-Platform GUI Programming with wxWidgets*. Prentice Hall PTR, New York, 1 edition, 2005.