Programação para GTK/GNOME com C++

André Luís Gobbi Sanches
Grupo de Software Livre do Centro GeNESS

Em parceria com
PET - Ciências da Computação

7 de Outubro de 2003

Conteúdo

Lista de Figuras

Lista de Tabelas

Listings

Capítulo 1

Programação C++ para GNU/Linux

1.1 Compilando

Existem diversos compiladores para C++ disponíveis na plataforma GNU/Linux. Como exemplo podemos citar o GCC/G++ (GNU Compiler Collection) e o ICC (Intel C++ Compiler ©), entre outros. O GCC é provavelmente o mais popular, pois é o padrão da maioria das distribuições e o compilador oficial do GNU/Linux. Por essas razões, foi o compilador escolhido para testar os exemplos. A seguir será feita uma breve demonstração de como utilizar o GCC para compilar seus programas C++. Todos os exemplos foram testados com este compilador. Entretanto, os exemplos foram escritos em puro código ISO C++, e portanto os exemplos aplicam-se aos demais compiladores.

O conjunto de bibliotecas GTK+ possui uma versão para Windows ©, possibilitando o desenvolvimento de código de fácil portabilidade. No entanto, este texto abrangerá apenas o desenvolvimento para GNU/Linux, por ser seu foco. Para utilizar o Gtk+ no Windows é necessário obter as bibliotecas e escolher um compilador, como os distribuídos pela Microsoft e pela Intel.

1.1.1 Uso Básico

Para compilar programas C++ com o GCC, digite:

user@host:/**home**/user$ g++ −o <programa> <programa>.cc

Este comando gerará um executável chamado <programa>, a partir do arquivo <programa>.cc. Caso o programa a ser compilado esteja dividido em vários arquivos, todos devem ser listados:

```
user@host:/home/user$ g++ -o <programa> <arquivo1>.cc
                      <arquivo2>.cc <...>
```

Note que independentemente de quantos arquivos sejam usados, todo programa deve ter uma, e apenas uma, função main().

Em caso de omissão, o gcc liga os arquivos-objeto e gera o executável. Caso esse comportamento não seja desejado, é possível utilizar a diretiva -c:

```
user@host:/home/user$ gcc -c <arquivo>.cc
```

Com a diretiva -c, o arquivo-objeto <arquivo>.o será gerado. O nome do arquivo-objeto pode ser definido com a diretiva -o.

1.1.2 Ligando Bibliotecas

A maioria das instalações GNU/Linux oferecem uma vasta coleção de bibliotecas de funções, que geralmente são instaladas como pacotes -dev. Estas bibliotecas oferecem praticamente qualquer funcionalidade encontrada em aplicações livres para Windows, como reprodução de som, renderização de html, verificação ortográfica, etc. Uma das maiores vantagens do desenvolvimento de software para GNU/Linux é a possibilidade de estender a reusabilidade de código. Para utilizar uma biblioteca em seu programa, você deve ordenar explicitamente que o g++ a ligue ao seu programa, através da diretiva -l<nome da biblioteca>:

```
user@host:/home/user$ g++ -o <programa> <programa>.cc
                      -lm
```

No exemplo acima, a biblioteca libm (biblioteca de funções matemáticas) será ligada. Esta biblioteca oferece as funções trigonométricas e aritmética complexa, e portanto é muito utilizada.

A diretiva -l<nome> busca um arquivo chamado lib<nome>.a nos diretórios de bibliotecas do sistema e substitui a diretiva pela sua localização. O exemplo acima é equivalente a:

```
user@host:/home/user$ g++ -o <programa> <programa>.cc
                      /usr/lib/libm.a
```

As bibliotecas listadas na diretiva −l são buscadas nos diretórios listados no arquivos /etc/ld.so.conf. Quando este arquivo for alterado, o comando ldconfig deve ser chamado para atualizar a base de dados do sistema. O usuário pode definir diretórios adicionais para a pesquisas por bibliotecas definindo a variável de ambiente $LD_LINK_PATH ou através da diretiva −L:

user@host:/**home**/user$ export LD_LINK_PATH=˜/lib

ou:

user@host:/**home**/user$ g++ −o <programa> <programa>.cc
−lmylib −L˜/lib

Exportar as funcionalidades da sua aplicação através de bibliotecas pode ajudar a promover a reusabilidade, como é comum no desenvolvimento de software livre.

1.1.3 Inclusão de Cabeçalhos

A diretiva de pré-compilação #include <cabecalho.h> é utilizada para incluir cabeçalhos do sistema, enquanto que #include "cabecalho.h" deve ser utilizada para incluir cabeçalhos definidos pelo próprio desenvolvedor da aplicação. De forma similar às bibliotecas, os cabeçalhos a serem incluídos também são procurados nos diretórios padrão do sistema.

Para especificar um diretório com cabeçalhos, a diretiva −I<diretorio> deve ser utilizada. O usuário também pode definir a variável $C_INCLUDE_PATH (para código em C), $CPLUS_INCLUDE_PATH para código em C++, ou $CPATH para cabeçalhos que podem ser incluídos em ambas as linguagens.

1.2 O Programa pkg-config

O conjunto de bibliotecas que formam o GTK+/GNOME impõe alguma complexidade à compilação, exigindo que um grande número de diretivas −I, −L e −l sejam necessárias. Para facilitar a compilação de aplicações que utilizem estas bibliotecas, um programa foi desenvolvido que inclui automaticamente as diretivas necessárias para utilizar cada biblioteca. Este programa, distribuído junto com o ambiente GNOME, é chamado de `pkg-config` e sua sintaxe é:

```
user@host:/home/user$ pkg-config <opcoes>
        <biblioteca(s)>
```

Para mostrar as diretivas -I, -L e -l necessárias para utilizar o gtkmm, digite:

```
user@host:/home/user$ pkg-config gtkmm-2.0
        --cflags --libs
```

Em um *shell* bash, podemos incluir o resultado de uma execução do pkg-config utilizando a construção '<comando>', que substitui o <comando> entre as crases pelo conteúdo de sua saída padrão. Assim, para compilar um programa C++ que utiliza o gtkmm com o pkg-config digite:

```
user@host:/home/user$ g++ -o <programa>
        <programa>.cc pkg-config gtkmm-2.0
        --cflags --libs
```

1.2.1 Funcionamento do pkg-config

Para cada biblioteca que o programa pkg-config suporta, existe um arquivo .pc no sistema de arquivos. O pkg-config procura por arquivos .pc em todos os diretórios listados na variável $PKG_CONFIG_PATH. Para visualizar seu valor, digite:

```
user@host:/home/user$ echo $PKG_CONFIG_PATH
```

e para alterar seu valor:

```
user@host:/home/user$ export PKG_CONFIG_PATH=
                <diretorio>:<diretorio>:<...>
```

ou adicione esta linha ao seu arquivo ~/.bashrc para ser carregado automaticamente.

Caso um desenvolvedor queira que uma biblioteca seja suportada pelo pkg-config, ele deve desenvolver um arquivo .pc. A sintaxe do arquivo é bem clara, e como exemplo a listagem a seguir demonstra o arquivo pc da

biblioteca gtkmm. Neste caso, o gtkmm (e todo o GNOME) foi instalado no diretório /usr/garnome.

```
prefix=/usr/garnome
exec_prefix=/usr/garnome
libdir=/usr/garnome/lib
includedir=/usr/garnome/include

Name: GTKmm
Description: C++ wrapper for GTK+
Requires: glibmm-2.0 gdkmm-2.0 pangomm-1.0 atkmm-1.0
        gtk+-2.0
Version: 2.2.7
Libs: -L${libdir} -lgtkmm-2.0
Cflags: -I${includedir}/gtkmm-2.0 -I${libdir}/
        gtkmm-2.0/include
```

1.3 Otimização do Processo de Compilação

A compilação de programas pode levar desde alguns minutos até várias horas, dependendo do porte do programa e da complexidade da linguagem. Compiladores da linguagem C costumam ser bastante eficientes, porém mesmo a compilação de pequenos programas escritos em C++ costuma ser demorada. De forma a acelerar a compilação, geralmente o código-fonte é dividido em diversos arquivos, organizados de acordo com a estrutura funcional do programa. Assim, quando um arquivo é modificado, apenas ele é recompilado, e não todo o restante do programa. Caso algum arquivo dependa do arquivo modificado (através de #include, por exemplo), este arquivo também é compilado. Em geral, a ligação do programa também precisa ser refeita.

Para garantir que todos os arquivos afetados pelas alterações sejam recompilados, é necessário que todas as dependências entre arquivos sejam especificadas. Como exemplo de dependências, pode-se observar a figura 1.1, que demonstra o grafo de dependências de uma aplicação.

No exemplo o arquivo binário app depende dos arquivos-objeto interface.o e core.o. O arquivo interface.o depende de interface.c, o qual depende dos cabeçalhos interface.h e core.h. O arquivo core.o depende de core.c, que por sua vez depende do cabeçalho core.h.

Caso o arquivo interface.h seja alterado, por exemplo, é necessário recompilar o arquivo-objeto interface.o, mas não o core.o. A recompilação de interface.o ativaria a ligação para gerar novamente o arquivo binário app.

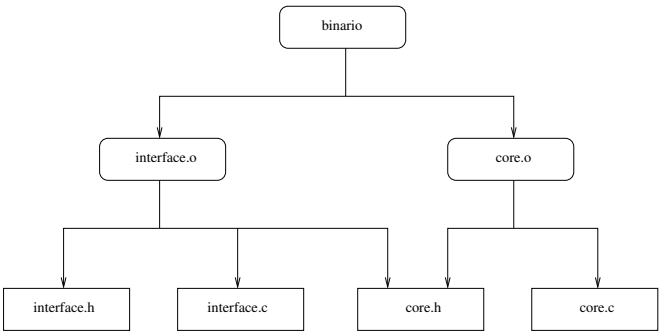

Figura 1.1: Grafo de dependências de uma aplicação

Este processo economiza tempo significativamente em programas grandes. A maioria das suítes de desenvolvimento oferecem um programa chamado make que, baseado nas dependências entre arquivos e nas instruções de compilação, otimiza e automatiza o processo de compilação conforme descrito acima.

1.3.1 Uso do Programa make

Para utilizar o programa make, é necessário criar um arquivo de configuração para o projeto, geralmente denominado Makefile ou makefile [1]. Neste arquivo, são especificados *alvos* (*targets*) de compilação, suas dependências, e os comandos necessários para compilá-los. As dependências podem ser arquivos ou outros *alvos* do make.Caso alguma das dependências de um *alvo* possua horário de modificação mais recente do que o *alvo*, os comandos do *alvo* são executados.

```
app : interface.o core.o
        gcc -o app interface.o core.o

interface.o : interface.h interface.c core.h
        @gcc -c interface.c #interface.c inclui os
                #cabecalhos

core.o : core.h core.c
        @gcc -c core.c      #core.c inclui core.h

clean :
        -@rm -f core.o
        -@rm -f interface.o
        -rm -f app
```

[1] GNUMakefile também pode ser utilizado para o make do projeto GNU

Como exemplo, a listagem apresenta o Makefile da aplicação apresentada na figura 1.1.

O primeiro *alvo* de compilação do arquivo é app, que depende de interface.o e core.o. O comando de compilação desta regra é gcc −o app interface.o core.o [2]. Comandos de compilação devem sempre ser precedidos por uma *tabulação*, caso contrário o make tentará interpretá-los como *alvos*. É possível especificar qualquer número de comandos para cada *alvo*, desde que todos sejam precedidos por *tabulações*. O make escreve todos os comandos na tela, exceto os precedidos pelo símbolo "@". A execução de comandos para um alvo pára quando um comando falha, a menos que o comando que falhou seja precedido pelo símbolo -". É possível utilizar -"e "@"no mesmo comando. O símbolo "#"especifica que o resto da linha é um comentário e deve ser ignorado pelo make. É costume incluir um *alvo* clean em arquivos makefile para eliminar os arquivos gerados pelos outros alvos. No exemplo acima, o *alvo* clean não produz e não depende de nenhum arquivo, e portanto sempre é executado. Os comandos das regras não necessariamente são comandos de compilação: qualquer comando válido do sistema pode ser utilizado. Assim, o make pode ser utilizado para automatizar qualquer tipo de tarefa, como por exemplo instalação. Vários sistemas complexos de instalação e gerenciamento de pacotes são baseados em makefiles: o sistema *gar*, o *portage* da distribuição GNU/Linux Gentoo e os *ports* do FreeBSD, por exemplo.

Para invocar o make, basta digitar:

make <alvo>

E o *alvo* especificado será processado. Caso nenhum alvo seja fornecido, o primeiro *alvo* do makefile será processado.

Também é possível definir variáveis que tornem um makefile flexível e configurável. A listagem a seguir é um exemplo.

```
CC = gcc
CFLAGS = ‘pkg−config gtkmm−2.0 −−cflags‘
LIBS   = ‘pkg−config gtkmm−2.0 −−libs‘
NAME = app

${NAME} : interface.o core.o
        ${CC} −o app interface.o core.o

interface.o : interface.h interface.c core.h
        @${CC} −c interface.c ${CFLAGS} ${LIBS}
```

[2] ld poderia ser usado ao invés de gcc, mas este foi escolhido por simplicidade

```
        #interface.c inclui os cabecalhos

core.o : core.h core.c
        @${CC} -c core.c      #core.c inclui core.h

clean :
        -@rm core.o
        -@rm interface.o
        -rm ${NAME}
```

Conforme pode ser observado no exemplo, para referenciar uma variável utiliza-se ${VARIAVEL}.

Capítulo 2

Desenvolvimento de Interface Gráfica

O conjunto de ferramentas GTK+ é baseado no padrão de projeto *Model-View-Controller*, que é a base de muitas bibliotecas modernas de interface gráfica. Inicialmente introduzido no Smalltalk 80, este padrão visa separar o objeto de aplicação (Model) da forma como ele é apresentado (View) e da forma como o usuário o controla (controller). Esta divisão é vantajosa pois código de interface é freqüentemente alterado e pode ser baseado em um *framework*. Trata-se de um modelo baseado em *eventos*, conforme descrito na seção 2.1.

2.1 Programação Orientada a Eventos

Na Programação Orientada a Eventos, toda ação iniciada pelo usuário (clique do *mouse*, digitação) ou pelo sistema (chegada de um pacote pela rede, incremento do relógio) é considerada um *evento*, que aciona um determinado tratador de *evento* (*event handler*). O *tratador do evento* a ser chamado é determinado pelo tipo de evento, e pelo *objeto* que o gerou.

O programador deve definir quais *eventos* são significativos para a aplicação e serão tratados, e qual função tratará cada *evento*. *Eventos* sem *tratadores* são ignorados pelo sistema.

2.2 O Modelo MVC

O modelo MVC define três entidades distintas:

- *Model (Modelo):* representação dos dados na aplicações. É responsável pela armazenagem dos dados e por oferecer as funções que os manipulam. O Modelo pode ser afetado por *eventos* do sistema (como o incremento do relógio), ou pelo resultado de cálculos.

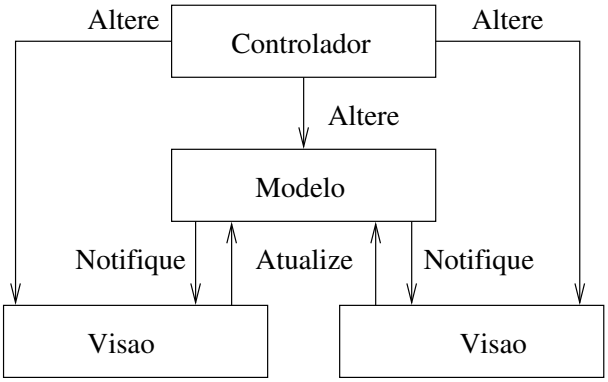

Figura 2.1: Modelo MVC

- *View (Visão):* representação visual dos dados. Constitui o que é mostrado ao usuário.
- *Controller (Controlador):* responsável por gerenciar os *eventos* iniciados pelo usuário, como um clique do *mouse*, por exemplo.

A interação entre estas três entidades pode ser vista na figura 2.1. Elas se relacionam através de troca de mensagens. Quando o usuário digita algum texto, um *evento* é gerado e tratado pelo *controlador* do objeto em questão (uma caixa de texto, por exemplo). Caso o *evento* altere os dados, o *controlador* deve acessar o *modelo* através de suas rotinas para manipulação dos objetos da aplicação. Caso o *evento* apenas altere a forma como os dados são mostrados na tela (maximizar uma janela, por exemplo), o *controlador* deve apenas acessar a *visão* em questão através das suas funções públicas de manipulação.

O conteúdo de um *modelo* pode ser apresentado por diversas *visões*. Um bom exemplo é o relógio do sistema, que pode ser apresentados por diversas aplicações ao mesmo tempo. O registro de hora atual do sistema operacional constitui um *modelo*, que mantém uma lista de quais *visões* apresentam seu conteúdo, a fim de notificá-las quando houver qualquer alteração. Ao receberem uma notificação, as *visões* acessam o *modelo* para atualizarem suas informações. Desta forma, é mantida a consistência entre um *modelo* e suas *visões*. Esta relação é conhecida como o padrão de projeto *Observer*.

Para ilustrar a relação entre *modelos*, *visões* e *controladores*, uma aplicação de calendário que permite alterar a data do sistema é mostrada na figura 2.2. É possível que diversas instâncias desta aplicação sejam executadas ao mesmo tempo, mas todas representam a mesma informação: a data do sistema. Trata-se de um caso de múltiplas *visões* para um mesmo *modelo*.

Caso o relógio do sistema passe de uma hora para outra, todas as instâncias da aplicação serão automaticamente atualizadas, pois o *modelo*

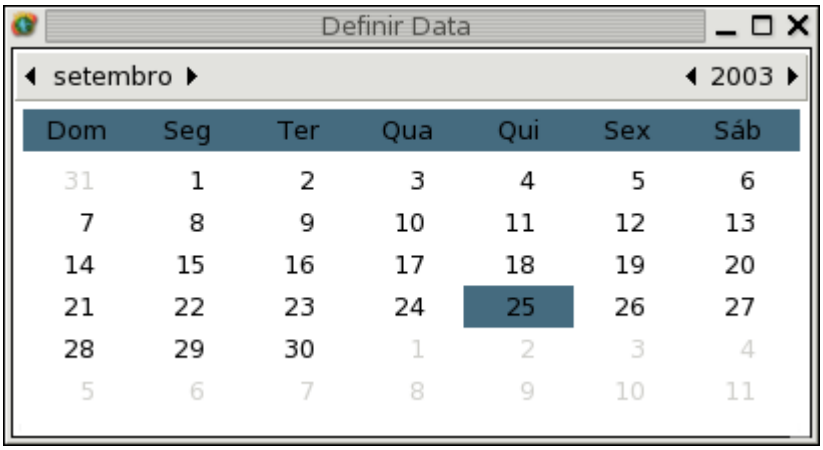

Figura 2.2: Aplicação de Calendário

notifica todas as instâncias de que elas devem se atualizar, e todas utilizam as chamadas do sistema para verificar a data atual.

Caso o usuário decida modificar a data, basta clicar em um dia qualquer uma das instâncias. O ambiente gráfico decidirá qual aplicação sofreu um *evento* pela posição do cursos do *mouse*, e acionará o *tratador do evento* clique desta posição. Também com base na posição do cursor, o tratador decidirá qual dia foi clicado, e assim qual a nova data. O *controlador* utilizará as funções do *modelo* para definir a nova data, e o *modelo* notificará todas as instâncias, que serão automaticamente atualizadas. Desta forma, a consistência entre todas as janelas de calendário do sistema será mantida.

Para melhor entendimento do modelo MVC, é recomendada a leitura de http://rd13doc.cern.ch/Atlas/Notes/004/Note004-7.html e http://www.enode.com/x/markup/tutorial/mvc.html.

2.3 O que é o GTK+

O Gtk+ é um conjunto de ferramentas para desenvolver interfaces GUI. O Gtk+ é um conjunto bastante completo, e possui uma grande quantidade de extensões. É prática comum que os projetos de software livre baseados em Gtk+ liberem seus componentes mais genéricos através de bibliotecas.

O Gtk+ é implementado na linguagem C, mas possui interfaces (*wrappers*) para diversas outras linguagens, como C++, Python, Perl, Eiffel, Ada, Ruby, etc, oferendo grande liberdade de escolha. O Gtk+ é desenvolvido especialmente para plataformas UNIX, mas possui também uma versão para Microsoft Windows ©.

O Gtk+ foi inicialmente desenvolvido para ser a base do aplicativo gráfico GIMP, mas foi adotado no desenvolvimento do ambiente gráfico GNOME, o que serviu para popularizá-lo.

Este texto focar-se-á na interface C++ do Gtk+, conhecida como GTKMM. Mais informações sobre o GTK+ podem ser obtidas em `http://www.gtk.org`, e também em `http://developer.gnome.org`. Informações específicas sobre GTKMM podem ser obtidas em `http://www.gtkmm.org`.

2.4 Arquitetura do GTK+

O Gtk+ foi projetado de forma modular, e portanto a biblioteca Gtk é baseada em outras 4 bibliotecas:

- *Glib:* biblioteca de utilitários.
- *Atk:* biblioteca que suporta acessibilidade.
- *Pango:* biblioteca que suporta internacionalização do Gtk+
- *Gdk:* biblioteca responsável pelas funcionalidades visuais. Interage com a Xlib.

Estas bibliotecas são básicas, e muitos programas Gtk+ raramente precisam acessá-las diretamente. A única exceção é a Glib, que define os tipos de dados que devem ser usados nos programas Gtk para facilitar o porte para diferentes arquiteturas.

Caso o programador utilize o GTKMM, ele deve utilizar também a biblioteca Glibmm, que oferece suporte a exceções e a classe Glib::ustring, que é similar a std::string mas é baseada em caracteres codificados no sistema *unicode* UTF-8. Ela suporta conversão automática de e para std::string, como o exemplo a seguir demonstra:

```
#include <string>
#include <gtkmm.h>
#include <iostream>

int main(int argc, char** argv) {
  std::string s1("Teste de õconverses");
  Glib::ustring u(s1);
  std::string s2(u);
  std::cout << s2 << std::endl;
}
```

Os tipos de dados oferecidos pela glib estão descritos em http://developer.gnome.org/doc/API/glib/glib-basic-types.html.

As dependências entre as bibliotecas podem ser observadas na figura 2.3.

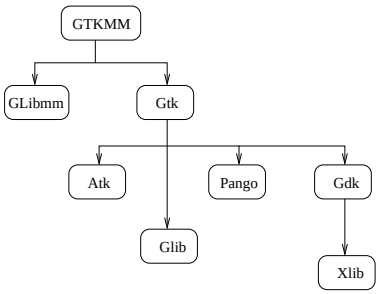

Figura 2.3: Arquitetura do Gtk+

2.5 Packing

Um aspecto importante de um *framework* GUI é a forma como a apresentação e organização dos componentes gráficos em uma janela é feita. Vários *frameworks* solicitam que o usuário especifique a posição exata de cada componente em relação à janela, em coordenadas cartesianas. Um exemplo deste tipo simples de organização é a suíte de desenvolvimento Delphi ©.

Outros *frameworks* requerem que o usuário informe apenas a organização lógica dos componentes e a posição relativa a outros componentes, e o posicionamento final é calculado pelos algoritmos do *framework*. Tanto o GTK+ como as GUIs Java mais populares (AWT e Swing) utilizam este método.

No GTK+, a área da janela é dividida em *caixas* (*boxes*), e os *componentes* (chamados de *widgets*) são colocados nas caixas. Cada caixa define sua forma de posicionar os componentes, e assim é definida a posição de cada componente. As caixas também são consideradas componentes e repassam as chamadas aos seus componentes, conforme o padrão de projeto *Composite*. Para definir a organização e o posicionamento de uma janela, basta definir as caixas que a compõem.

Componentes como janela e painéis são capazes de armazenar componentes. Eles podem armazenar diretamente um e apenas um componente. Este único componente pode ser uma caixa ou uma combinação de várias caixas, de forma que o número de componentes de uma janela não é limitado. Há componentes que não podem armazenar outros, como rótulos e botões.

Em GTK+, há basicamente 6 organizações de componentes que uma janela pode ter: Nenhuma Caixa (apenas um componente), Caixa Horizontal, Caixa Vertical, Tabela, Posicionamento Fixo e Combinações de Caixas. Cada uma das caixas é descrita nas seções a seguir. Os componentes que as representam na API do GTKMM são apresentados no capítulo 3.

A grande vantagem de utilizar *Packing* é a flexibilidade no dimensionamento das janelas. Como os componentes não tem tamanho fixo, caso o

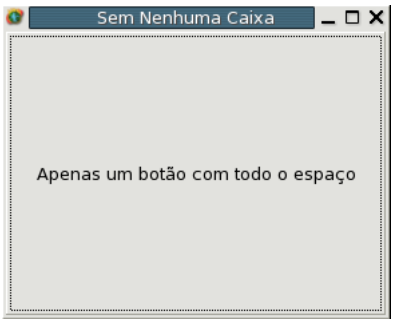

Figura 2.4: Nenhuma Caixa

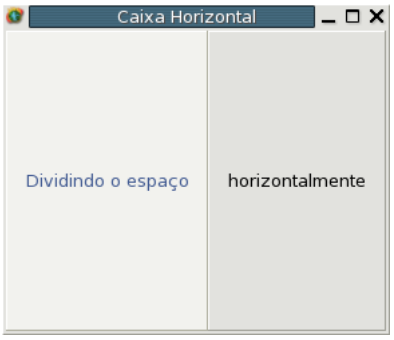

Figura 2.5: Caixa Horizontal

idioma da aplicação seja alterado, o tamanho dos componentes ainda será consistente. Esta capacidade facilita a internacionalização das aplicações desenvolvidas com Gtk+. Também facilita a ampliação ou redução de janelas, uma vez que a escala aplicada a janela também será aplicada a todos os seus componentes. O navegador Mozilla utiliza este recurso.

2.5.1 Nenhuma Caixa

Este é o estilo mais simples de posicionamento. Não é necessário adicionar qualquer componente de caixa, mas apenas o componente desejado. A figura 2.4 mostra um exemplo em que apenas um botão é adicionado. O botão foi configurado para expandir-se e ocupar toda a área da janela, mas também é possível definir que ele deve ter um tamanho fixo ou apenas o suficiente para mostrar seu texto.

2.5.2 Caixa Horizontal

O componente Caixa Horizontal permite dividir uma área em colunas. O número de colunas é definido na criação da caixa (é possível alterar o número depois), assim como o sentido de posicionamento: da esquerda para a direita

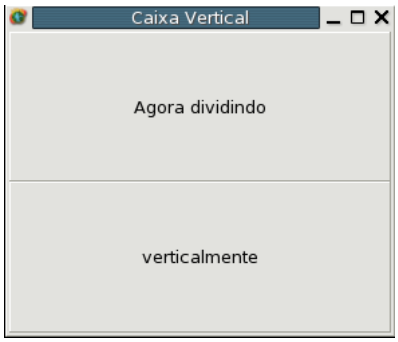

Figura 2.6: Caixa Vertical

Figura 2.7: Organização em Tabela

ou vice-versa. Os componentes serão adicionados neste sentido, pela ordem de adição. Também é possível especificar a posição de um componente, como em uma lista. Um exemplo desta caixa pode ser visto na figura 2.5

2.5.3 Caixa Vertical

O componente Caixa Vertical é muito similar ao Caixa Horizontal, exceto por dividir a área vertical e não horizontalmente. Sua interface também é similar, e portanto também pode ser tratado como uma lista. Um exemplo de Caixa Vertical pode ser visto na figura 2.6.

2.5.4 Tabela

A caixa Tabela divide a área em linhas e colunas, conforme a figura 2.7. Os componentes são posicionados em células, e podem ocupar uma ou mais células. Uma aplicação deste tipo de caixa pode ser observada na figura 2.8. Esta janela poderia ser criada com uma caixa vertical com 2 linhas, cada uma contendo uma caixa horizontal. Ambas as caixas horizontais possuiriam

Figura 2.8: Exemplo de Uso da Caixa Tabela

Figura 2.9: Combinação de Caixas Horizontais e Verticais

Figura 2.10: Posicionamento Fixo

um rótulo, um divisor horizontal e uma caixa de texto. Mas neste caso, seria necessário garantir o alinhamento dos divisores horizontais, o que a caixa Tabela já faz. A figura 2.9 demonstra o problema do alinhamento.

2.5.5 Posicionamento Fixo

Nesta caixa, a posição e o tamanho de cada componente é especifica pelo programador em coordenadas cartesianas, utilizando um pixel como unidade. A figura 2.10 exemplifica esta caixa. O GTK+ não gerencias o posicionamento nesta caixa, e portanto o programador deve alterar o posicionamento manualmente quando o tamanho de algum componente é modificado. Esta caixa dificulta a tradução de aplicações onde as expressões nos diferentes idiomas possuem dimensões adversas.

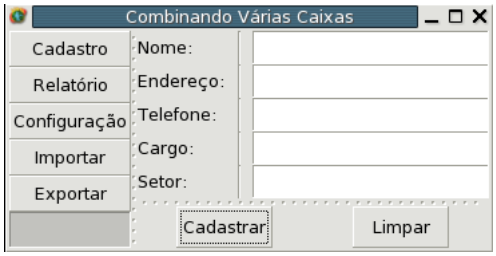

Figura 2.11: Combinação de Caixas

2.5.6 Combinações de Caixas

Dificilmente uma caixa oferece ao usuário exatamente o posicionamento de componentes desejado. Entretanto, ao combinar as diversas caixas oferecidas, é possível especificar praticamente qualquer posicionamento. Como toda caixa é um componente, é possível posicionar caixas dentro de caixas. Desta forma, o programador tem grande flexibilidade. A figura 2.11 mostra uma aplicação onde diversas caixas são utilizadas.

Capítulo 3

A API GTKMM

A API (*Application Programming Interface*) GTKMM oferece uma versão C++ da biblioteca GTK+ para C. Assim, ela oferece ao programador todas as funcionalidades necessárias para o desenvolvidos de aplicativos gráficos para as plataformas que suportam o GTK+, como vários sistemas UNIX e o Microsoft Windows. O ambiente gráfico GNOME é baseado no GTK+, e oferece seus componentes como extensões. Entretanto, vários destes componentes requerem o ambiente GNOME (ou parte dele, como o Bonobo) para funcionarem corretamente.

Neste capítulo, os principais componentes da API GTKMM serão apresentados. Este texto não visa descrever toda a API, mas apenas preparar o leitor para entender a documentação própria dos componentes. Assim, apenas os principais componentes básicos serão apresentados. A maioria dos restantes são variações ou composições destes componentes básicos, e portanto não será difícil para o leitor aprender a utilizá-los conforme a necessidade.

A documentação completa da API GTKMM pode ser encontrada em http://www.gtkmm.org/gtkmm2/docs/, incluindo diversas extensões.

3.1 Primeiro Programa

Para introduzir o leitor a programação com GTKMM, um programa simples é apresentado na listagem a seguir. Este programa apenas cria um objeto Gtk::Window, que representa uma janela.

```
#include <gtkmm.h>

int main(int argc, char **argv) {
  Gtk::Main main(argc, argv);

  Gtk::Window janela;
```

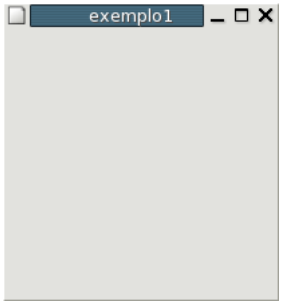

Figura 3.1: Execução do primeiro exemplo

```
  Gtk::Main::run(janela);
}
```

Este programa possui apenas 3 linhas de código. A primeira cria uma aplicação Gtk+. Este passo é necessário para repassar ao Gtk os argumentos do programa. Esta função removerá de argv todos os argumentos reconhecidos, tanto pelo Gtk+ quanto pela Xlib.

O comando seguinte cria o objeto janela, da classe Gtk::Main. O último comando ordena ao Gtk+ que mostre e ative a janela. Esta é a janela principal da aplicação, e o quando ela for fechada o programa será encerrado. Note que todos os símbolos do GTKMM estão definidos no espaço de nomes Gtk, e portanto devem ser precedidos por Gtk, a menos que o espaço de nomes seja incorporado no escopo.

3.2 Compilação

Para testar o primeiro programa, salve-o com o nome exemplo1.cc e compile-o com o comando:

```
user@host:/home/user$ g++ -o exemplo1 exemplo1.cc
  `pkg-config gtkmm-2.0 --cflags --libs`
```

Para executá-lo, digite:

```
user@host:/home/user$ ./exemplo1
```

Após este comando, a janela mostrada na figura 3.1 deve aparecer na tela.

3.2.1 Automatizando a Compilação

Para facilitar a compilação dos exemplos, pode-se utilizar o seguinte Makefile:

```
FILE=$<$nome do exemplo$>$
LIBS=gtkmm-2.0
CFLAGS=`pkg-config ${LIBS} --cflags --libs`
RUN=/bin/true

${FILE}: ${FILE}.cc
  g++ -o ${FILE} ${FILE}.cc ${CFLAGS} &&
    ${RUN} && (./${FILE} &)

clean:
  -rm ${FILE}
```

É preciso definir a variável FILE com o nome do exemplo a ser compilado, sem extensão (ex: exemplo1) e a variável LIBS com o nome das bibliotecas a serem ligadas. O programa pkg−config é descrito na seção 1.2 na página 8 e o uso de Makefiles na seção 1.3 na página 10.

3.3 Tratamento de Sinais

Conforme descrito na seção 2.1, no modelo MVC é preciso especificar rotinas que respondam aos eventos gerados pelo usuário. Na terminologia Gtk+ os eventos são chamados de *sinais* (*signals*). O suporte ao tratamento de sinais do GTKMM é fornecido pela biblioteca libsigc++.

3.3.1 A Biblioteca libsigc++

A biblioteca libsigc++ oferece um tratamento muito flexível de eventos através de classes altamente parametrizadas. Entretanto, a base de seu funcionamento está em três classes:

- `SigC::Signal:` esta classe representa os eventos. Cada componente define seus eventos como objetos desta classe, permitindo a vinculação de tratadores para cada evento.
- `SigC::Slot:` esta classe representa os tratadores de eventos. O programador deve declarar as funções que tratarão os eventos, e encapsulá-las em objetos SigC::Slot. Apenas objetos desta classe podem tratar os eventos representados pela classe SigC::Signal.

- `SigC::Object:` classes cujos métodos são tratadores de eventos (métodos SigC::Slot) devem sempre derivar desta classe. Logo, esta classe é a base dos *controladores* (ver seção 2.2). Assim, quando um objeto da classe é destruído, todos os seus tratadores são desvinculados automaticamente. Todos os componentes de GTKMM são derivados de Sig :: Object.

Os eventos de cada componente devem ser declarados como *atributos* (ou *propriedades*) do componente, e objetos da classe SigC::Signal. Através do método:

SigC::Connection SigC::Signal::connect(SigC::Slot);

O método connect() conecta um tratador (SigC::Slot) a um evento (SigC::Signal). O método retorna um objeto SigC::Connection, que pode ser usado para desconectar o tratador de eventos.

A biblioteca libsigc++ oferece funções para encapsular métodos de classes e funções globais em objetos SigC::Slot. Estes construtores são, respectivamente:

```
SigC:: slot (void funcao(void));          // para funcoes globais
SigC:: slot (objeto , void metodo(void)); // para metodos
```

Caso a primeira função seja usada, quando o evento acontecer, o sistema chamará funcao(). Caso a segunda seja usada, o sistema chamará objeto.metodo(). Os métodos passados também podem ser estáticos ou virtuais. Todos os símbolos da biblioteca libsigc++ pertencem ao espaço de nomes SigC. Esta biblioteca é incluída juntamente com o GTKMM, pois é base para todos os componentes. Para utilizar apenas a biblioteca libsigc++ é preciso incluir o cabeçalho <sigc++−1.2/sigc++/sigc++.h> e passar "libsigc++−1.2"como parâmetro para o pkg−config.

O exemplo a seguir ilustra o tratamento de eventos com GTKMM. Para demonstrar o funcionamento de eventos, é preciso utilizar mais um componente, Gtk::Button, que representa um botão. O código do exemplo é mostrado a seguir:

```
#include <gtkmm.h>

class Controlador : public SigC::Object {
  Gtk::Window& janela;
```

```
public:
  Controlador(Gtk::Window& _janela) :
    janela(_janela) {
    Object();
    janela. set_title ("Cliques: " );
  }

  void botao_pressionado() {
    janela. set_title (
        janela. get_title () + "+");
  }
};


int main(int argc, char **argv) {
  Gtk::Main main(argc, argv);

  Gtk::Window janela;
  Controlador controlador(janela);
  Gtk::Button botao("Clique-me");
  janela.add(botao);
  botao.show();

  botao. signal_clicked (). connect(
      SigC:: slot (controlador,
      &Controlador::botao_pressionado) );

  Gtk::Main::run(janela);
}
```

Neste exemplo, uma classe Controlador é definida para tratar os eventos da janela principal. Um botão é criado e adicionado à janela. Este botão possui o evento signal_clicked, que é conectado ao método tratador botao_pressionado da classe Controlador. Como este método não é estático, é necessário especificar um objeto (controlador).

Neste exemplo, apenas o evento signal_clicked é tratado. Os demais eventos serão ignorados, pois o sistema define um tratador nulo em caso de omissão.

Alguns eventos podem passar alguns argumentos para seus tratadores. É necessário sempre observar na documentação qual a interface esperada do tratador para cada evento. A documentação de todas as classes do GTKMM pode ser acessada em http://www.gtkmm.org/docs/gtk/class_index.html.

explicit Window (GtkWindowType type=); Cria uma nova janela e define seu tipo
void set_default_size (gint width, gint heigth); Define o tamanho padrão da janela
void set_position (GtkWindowPosition position); Define a posição padrão da janela
void set_user_resizeable (gboolean setting); Define se o usuário pode ou não redimensionar a janela
void add (Widget& p0); Agrega o componente p0
void remove (Widget& p0); Remove o componente p0

Tabela 3.1: Principais métodos de Gtk::Window

Nas seções a seguir, os principais componentes do GTKMM são apresentados. Nenhum componente será descrito em detalhes. A função deste texto é apenas introduzir o usuário à API. Para detalhes, veja a documentação do GTKMM no *link* acima.

3.4 Gtk::Window

Conforme foi apresentado anteriormente, o componente Gtk::Window representa uma janela. É derivado da classe abstrata Gtk::Bin, que é a base para todos os componentes que podem possuir outros componentes, e fornece os métodos necessários para gerenciar o componente agregado.

Os principais métodos para gerenciar objetos Gtk::Window estão listados na tabela 3.1. Caso o leitor não encontre o método necessário nesta tabela, deve buscar na documentação da classe e de suas bases.

3.5 Componentes Básicos

Os componentes básicos do GTKMM serão descritos a seguir. Estes componentes são suficientes para pequenas aplicações, e constituem a base dos demais.

3.5.1 Gtk::Label

O componente Gtk::Label (rótulo) é utilizada para mostrar uma seqüência de caracteres em uma janela. Os principais métodos deste componente estão listados na tabela 3.2

explicit Label (const nstringi& label=0); Cria um rótulo
string get_text () const; Retorna o texto do rótulo
void set_justify (GtkJustification jtype); Define o posicionamento do texto
void set_text (const string& str); Define o texto do rótulo

Tabela 3.2: Principais métodos de Gtk::Label

Button (); Cria um botão sem rótulo
explicit Button (const string& label); Cria um botão e define seu rótulo
emitable signal void clicked (); Evento disparado quando o botão é clicado
emitable signal void enter (); Evento disparado quando o cursor entra no botão
emitable signal void leave (); Evento disparado quando o cursor sai do botão

Tabela 3.3: Principais métodos e eventos de Gtk::Button

3.5.2 Gtk::Button

Gtk::Button é o componente base de botões do GTKMM. Todos os outros botões derivam desta classe. Todo botão possui um rótulo e dispara eventos. A tabela 3.3 mostra os principais métodos e eventos oferecidos por todos os tipos de botões.

A classe Gtk::Button é concreta e pode ser instanciada. Além disso, o botão pode conter um outro componente.

3.5.3 Gtk::ToggleButton

O componente Gtk::ToggleButton representa um botão que possui dois estados: pressionado e solto. Oferece os métodos e eventos apresentados na tabela 3.4, em adição aos oferecidos por Gtk::Button. Naturalmente, a definição dos *construtores* é ToggleButton(...). Este componente possui duas propriedades *booleanas*. A primeira é o seu estado (*active*), que define se o botão está pressionado ou não. A segunda é o seu modo (*modo*), que define se o estado do botão deve alterar sua aparência ou não.

bool get_active () const; Retorna o estado do botão
bool get_mode () const; Retorna se o estado está sendo mostrado
void set_active () const; Define o estado (lig/desi) do botão
void set_mode () const; Define se o estado deve ser mostrado
emitable signal void toggled (); Evento disparado quando muda de estado

Tabela 3.4: Principais métodos de Gtk::ToogleButton

RadioButton (Group& groupx); Cria um botão de rádio pertencente ao grupo
RadioButton (Group& groupx, const string& label); Cria um botão pertencente ao grupo e define seu rótulo
Group group (); Retorna o grupo ao qual o botão pertence
void set_group (Group p0); Define o grupo do botão

Tabela 3.5: Principais métodos de Gtk::RadioButton

3.5.4 Gtk::CheckButton

O componente Gtk::CheckButton comporta-se como o seu componente base, o Gtk::ToggleButton, mas difere na aparência. Consiste de um pequeno botão quadrado e um rótulo externo. O estado do botão é mostrado pela presença ou ausência de um marco no quadrado.

Este componente não define qualquer método ou evento adicional.

3.5.5 Gtk::RadioButton

O componente Gtk::RadioButton se comporta como um Gtk::CheckButton exclusivo em um grupo de botões. O programador define um grupo de botões onde apenas um pode estar ligado em cada tempo. Quando um botão do grupo é marcado, todos os restantes são desmarcados. Este componente especializa o Gtk::CheckButton, oferecendo os métodos apresentados na tabela 3.5.

Entry ();
Cria uma caixa de texto vazia
string get_text () const;
Retorna o conteúdo da caixa de texto
void set_text set_text(const string& text);
Define o conteúdo da caixa de texto
void set_visibility(bool visible);
Define se a caixa deve ser visível ou não

Tabela 3.6: Principais métodos de Gtk::Entry

3.5.6 Gtk::Entry

O componente representa uma caixa de texto de apenas uma linha. Este componente não é derivado de Gtk::Button, e portanto não possui qualquer método ou evento associado a botões. Seus principais métodos estão listados na tabela 3.6.

3.5.7 Separadores

A API GTKMM oferece dois componentes de separação: Gtk::HSeparator (horizontal) e Gtk::VSeparator (vertical). A função destes componentes é traçar uma linha separando os demais componentes na posição desejada. São componente visuais que não possuem muitos métodos para manipulação e não geram eventos.

3.6 Caixas

Conforme explicado na seção 2.5 (página 18, em Gtk+ os componentes devem ser organizados em caixas, que podem ou não estar contidas em outras caixas. A interface das caixas de Gtk+ é bastante simples e a maior dificuldade no posicionamento está em entender o conceito de *Packing* e conseguir imaginar e descrever a organização desejada através das caixas disponíveis. Os conceitos apresentados na seção 2.5 e um pouco de prática devem ser suficientes para a compreensão.

Nas seções a seguir as caixas disponíveis em Gtk+ e seus métodos são apresentados.

3.6.1 Gtk::Container

A classe Gtk::Container é a base dos componentes que podem conter outros componentes, sendo caixas ou não. Alguns componentes podem conter apenas um componente, enquanto que outros (na maioria caixas) podem

void add (Widget& p0); Adiciona o componente
void remove (Widget& p0); Remove o componente
void set_focus_child (Widget& p0); Repassa o foco para o componente

Tabela 3.7: Principais métodos de Gtk::Container

conter vários. O método add (...) adiciona um componente sem especificar a posição. As caixas oferecem métodos mais especializados que devem ser usados.

A tabela 3.7 apresenta os principais métodos desta classe. O exemplo a seguir demonstra como adicionar um botão a uma janela.

```
#include <gtkmm.h>

int main(int argc, char **argv) {
        Gtk::Main main(argc, argv);

        Gtk::Window janela;
        janela. set_title (
            "Exemplo de Gtk::Container");
        Gtk::Button botao("Botao simples");
        janela.add(botao);
        botao.show();

        Gtk::Main::run(janela);
}
```

3.6.2 Gtk::Box

As caixas derivadas da classe Gtk::Box oferecem métodos mais avançados para adicionar componentes. Ao adicionar um componente, é possível especificar se ele pode ser expandido e se deve ocupar toda a área disponível na caixa, ou manter seu tamanho padrão. Em caso de omissão, ambas as propriedades serão verdadeiras. Também é possível especificar um valor *padding*, ou seja, distância dos outros componentes. É preciso especificar se o componente será adicionado ao início ou ao final da caixa. O método pack_start, que adiciona um componente ao início da caixa tem a seguinte interface:

void pack_start(Widget& child, **bool** expand=**true**,

bool fill =**true**, guint padding=0);

O método pack_end possui a mesma interface e adiciona um componente ao final da caixa.

Os componentes Gtk::VBox e Gtk::HBox derivam da classe Gtk::Box e posicionam seus componentes no sentido vertical e horizontal, respectivamente. Ambos oferecem a mesma interface. O exemplo a seguir demonstra seu uso:

```
#include <gtkmm.h>

int main(int argc, char** argv) {
  Gtk::Main main(argc, argv);

  Gtk::Window janela;
  janela. set_title (
      "Exemplo de Gtk::HBox e Gtk::VBox");
  janela. resize (300, 200);
  Gtk::Button botao1("1"), botao2("2"),
      botao3("3"), botao4("4"), botao5("5");
  Gtk::HBox hbox;
  Gtk::VBox vbox;
  janela.add(vbox);
  vbox.pack_start(hbox,true,true,10);
  hbox.pack_end(botao1);
  hbox.pack_end(botao2);
  hbox.pack_end(botao3);
  vbox.pack_start(botao4, false , false ,10);
  vbox.pack_start(botao5, false , false ,10);

  botao1.show();
  botao2.show();
  botao3.show();
  botao4.show();
  botao5.show();
  vbox.show();
  hbox.show();
  Gtk::Main::run(janela);
}
```

3.6.3 Gtk::Table

O componente Gtk::Table representa a tabela apresentada na seção 2.5.4 (página 20). Para adicionar componentes a esta caixa, é necessário especi-

Table (gint rows=1, gint columns=1, gint homogeneous=FALSE); Construtor da tabela
void resize (guint rows, guint columns); Redimensiona a tabela
void set_homogeneous (bool homogeneous);
Define se os componentes devem ser igualmente espaçados entre si

Tabela 3.8: Principais métodos de Gtk::Table

ficar quais células serão ocupadas por ele. Um mesmo componente pode ocupar mais de uma célula. O método para adicionar componentes é o attach, que recebe como parâmetros as coordenadas das células que o componente ocupará. Como parâmetros opcionais, é possível especificar se os omponentes podem ser expandidos vertical ou horizontalmente e a distância dos outros componentes. O método attach possui a seguinte interface:

```
void attach(Widget& child, guint left_attach, guint right_attach,
    guint top_attach, guint bottom_attach, gint xoptions=
    (GTK_FILL|GTK_EXPAND), gint yoptions=(GTK_FILL|GTK_EXPAND),
    guint xpadding=0, guint ypadding=0);
```

Outros métodos do componente estão listados na tabela 3.8. O exemplo a seguir ilustra o uso da caixa Gtk::Table:

```
#include <gtkmm.h>

int main(int argc, char** argv) {
  Gtk::Main main(argc, argv);
  Gtk::Window janela;
  Gtk::Table tabela(3,2);
  janela. set_title ("Exemplo de Gtk::Table");
  janela. resize (300,200);
  janela.add(tabela);
  Gtk::Label labNome("Nome"), labEndereco("Endereco");
  Gtk::Entry entNome, entEndereco;

  tabela.attach( labNome , 1, 2, 1, 2 );
  tabela.attach( labEndereco , 1, 2, 2, 3 );
  tabela.attach( entNome , 2, 3, 1, 2 );
  tabela.attach( entEndereco , 2, 3, 2, 3 );
```

Fixed ();
Cria uma caixa de posições fixas
void put (Widget& widget, gint16 x, gint16 y);
Adiciona um componente na posição determinada
void move (Widget& widget, gint16 x, gint16 y);
Move um componente para a posição determinada

Tabela 3.9: Principais métodos de Gtk::Table

```
  janela.show_all();
  Gtk::Main::run( janela );
}
```

3.6.4 Gtk::Fixed

A caixa Gtk::Fixed é usada quando o programador deseja especificar a posição exata de cada componente. Ela fornece um método para colocar um componente em uma posição determinada, e outro para movê-lo (além dos métodos de Gtk::Container), conforme pode ser observado na tabela 3.9. O exemplo a seguir demonstra o uso do componente Gtk::Fixed:

```
#include <gtkmm.h>
#include <string>

int main(int argc, char** argv) {
  Gtk::Main main(argc, argv);
  Gtk::Window janela;
  Gtk::Fixed fixed;
  janela. set_title ("Exemplo de Gtk::Fixed");
  janela. resize (300,200);
  janela.add(fixed);
  Gtk::Label labNome("Nome"), labEndereco("Endereco");
  Gtk::Entry entNome, entEndereco;

  fixed.put(labNome, 10, 10);
  fixed.put(labEndereco, 10, 100);
  fixed.put(entNome, 120, 10);
  fixed.put(entEndereco, 120, 100);

  janela.show_all();
  Gtk::Main::run( janela );
}
```

Apêndice A

Bibliotecas GTK+/GNOME

AtkMM

Descrição:	C++ wrapper for atk
Versão:	2.2.7
Dependências:	glibmm-2.0 atk

Atk

Descrição:	Accessibility Toolkit
Versão:	1.4.0
Dependências:	gobject-2.0 gmodule-2.0

audiofile

Descrição:	audiofile
Versão:	0.2.3
Dependências:	Nenhuma

Bonobo Activation

Descrição:	Object activation framework for GNOME
Versão:	2.4.0
Dependências:	glib-2.0 gmodule-2.0 ORBit-2.0

camel

Descrição:	the Evolution mail library
Versão:	1.4.5
Dependências:	gal-2.0 >= 1.99.8

cspi

Descrição:	Accessibility Technology software simple client library
Versão:	1.3.7
Dependências:	libspi-1.0

eel

Descrição:	Eazel Extensions Library
Versão:	2.4.0
Dependências:	gconf-2.0 gdk-pixbuf-2.0 glib-2.0 gmodule-2.0 gnome-vfs-2.0 gthread-2.0 gtk+-2.0 libart-2.0 libgnome-2.0 libgnomeui-2.0 libxml-2.0 gail libglade-2.0

Epiphany Browser

Descrição:	GNOME Web Browser
Versão:	1.0
Dependências:	gtk+-2.0, libxml-2.0, libgnomeui-2.0, bonobo-activation-2.0, libbonoboui-2.0 >= 2.1.1, ORBit-2.0, libglade-2.0, gnome-vfs-2.0, gnome-vfs-module-2.0, gconf-2.0

esound

Descrição:	esound
Versão:	0.2.32
Dependências:	audiofile

evolution-addressbook

Descrição:	libraries needed for Evolution addressbook backends
Versão:	1.4.5
Dependências:	gconf-2.0 libbonobo-2.0 gal-2.0 >= 1.99.8 libgnome-2.0 camel = 1.4.5

evolution-calendar

Descrição:	libraries needed for Evolution calendar backends
Versão:	1.4.5
Dependências:	libgnome-2.0 libbonobo-2.0 gal-2.0 >= 1.99.8 gnome-vfs-2.0

evolution-shell

Descrição:	libraries needed for Evolution shell components
Versão:	1.4.5
Dependências:	libgnome-2.0 libgnomeui-2.0 libbonoboui-2.0 gal-2.0 >= 1.99.8

Fontconfig

Descrição:	Font configuration and customization library
Versão:	2.2.1
Dependências:	Nenhuma

fribidi

Descrição:	Unicode BiDirectional algorithm library
Versão:	0.10.4
Dependências:	Nenhuma

Gail

Descrição:	GNOME Accessibility Implementation Library
Versão:	1.4.0
Dependências:	atk gtk+-2.0 libgnomecanvas-2.0

GAL

Descrição:	GNOME Application Library
Versão:	1.99.9
Dependências:	gtk+-2.0 libxml-2.0 libglade-2.0 libgnomeprint-2.2

GAPI

Descrição:	GObject .NET API Wrapping Tool
Versão:	0.10
Dependências:	Nenhuma

gconf

Descrição:	GNOME Config System.
Versão:	2.4.0.1
Dependências:	ORBit-2.0

gDesklets Core

Descrição:	GNOME Desktop Utilities written in Python, Core Package
Versão:	0.21
Dependências:	gnome-python-2.0 >= 1.99.14 pygtk-2.0 >= 1.99.14 gdk-2.0 gtk+-2.0

GDK

Descrição:	GIMP Drawing Kit ($target target)
Versão:	2.2.4
Dependências:	gdk-pixbuf-2.0 pangoxft pangox

GDKmm

Descrição:	C++ wrappers for GLib, GTK+, and Pango
Versão:	2.2.7
Dependências:	glibmm-2.0 pangomm-1.0 gdk-2.0

GdkPixbuf

Descrição:	Image loading and scaling
Versão:	2.2.4
Dependências:	gobject-2.0,gmodule-2.0

GdkPixbuf Xlib

Descrição:	GdkPixbuf rendering for Xlib
Versão:	2.2.4
Dependências:	gobject-2.0,gmodule-2.0,gdk-pixbuf-2.0

GDK

Descrição:	GIMP Drawing Kit ($target target)
Versão:	2.2.4
Dependências:	gdk-pixbuf-2.0 pangoxft pangox

gedit

Descrição:	gedit
Versão:	2.4.0
Dependências:	libgnomeui-2.0 libglade-2.0 libgnomeprintui-2.2 eel-2.0 gtksourceview-1.0

GIMP

Descrição:	GIMP Library
Versão:	1.3.20
Dependências:	glib-2.0

GIMP_UI

Descrição:	GIMP User Interface Library
Versão:	1.3.20
Dependências:	gimp-1.3 gtk+-2.0 >= 2.2.0

GKrellM

Descrição:	Extensible GTK system monitoring application
Versão:	2.1.19
Dependências:	gtk+-2.0 >= 2.0.0

GLib

Descrição:	C Utility Library
Versão:	2.2.3
Dependências:	Nenhuma

GLibmm

Descrição:	C++ wrapper for GLib
Versão:	2.2.7
Dependências:	gobject-2.0 sigc++-1.2

GModule

Descrição:	Dynamic module loader for GLib
Versão:	2.2.3
Dependências:	glib-2.0

Gnet

Descrição:	A network compatibility layer library
Versão:	1.1.9
Dependências:	Nenhuma

gnome-desktop-2.0

Descrição:	Utility library for loading .desktop files
Versão:	2.4.0
Dependências:	gtk+-2.0 libgnomeui-2.0 libstartup-notification-1.0

gnome-mag

Descrição:	Gnome Screen Magnifier
Versão:	0.10.3
Dependências:	libbonobo-2.0 gtk+-2.0

gnome-mime-data

Descrição:	Base set of file types and applications for GNOME
Versão:	2.4.0
Dependências:	Nenhuma

Gnome Pilot

Descrição:	Não disponível
Versão:	2.0.10 (pilot-link version 0.11.5)
Dependências:	libgnome-2.0 libgnomeui-2.0

GNOME-python

Descrição:	Python bindings for GNOME libraries
Versão:	2.0.0
Dependências:	Nenhuma

gnome-speech

Descrição:	GNOME text to speech infrastructure
Versão:	0.2.7
Dependências:	libbonobo-2.0

gst

Descrição:	Gnome System Tools
Versão:	0.27.0
Dependências:	Nenhuma

gnome-vfs

Descrição:	The GNOME virtual file-system libraries
Versão:	2.4.0
Dependências:	bonobo-activation-2.0,gthread-2.0,gmodule-2.0

gnome-vfs-module

Descrição:	The GNOME virtual file-system module include info
Versão:	2.4.0
Dependências:	bonobo-activation-2.0,gthread-2.0,gmodule-2.0

gnome-window-settings-2.0

Descrição:	Utility library for getting window manager settings
Versão:	2.4.0
Dependências:	gtk+-2.0 libgnomeui-2.0

gnopernicus

Descrição:	Accessibility Technology Software
Versão:	0.7.0
Dependências:	cspi-1.0 gtk+-2.0 libglade-2.0 libgnome-2.0 libgnomeui-2.0 libxml-2.0

GObject

Descrição:	GLib Type, Object, Parameter and Signal Library
Versão:	2.2.3
Dependências:	glib-2.0

Gok

Descrição:	GNOME On-screen Keyboard
Versão:	0.8.1
Dependências:	libgnomeui-2.0 cspi-1.0 libspi-1.0 libbonobo-2.0 atk gtk+-2.0 gail libwnck-1.0 esound

GStreamer

Descrição:	Streaming-media framework
Versão:	0.6.3
Dependências:	glib-2.0, gobject-2.0, gmodule-2.0, gthread-2.0, libxml-2.0

GStreamer control library

Descrição:	Dynamic parameters for plug-ins
Versão:	0.6.3
Dependências:	gstreamer-0.6

GStreamer GConf Library

Descrição:	Streaming media framework, GConf support library
Versão:	0.6.3
Dependências:	gstreamer-0.6

GStreamer Media-Specific Libraries

Descrição:	Streaming-media framework, media-specific libraries
Versão:	0.6.3
Dependências:	gstreamer-0.6

GStreamer Play Library

Descrição:	Streaming-media framework, play libraries
Versão:	0.6.3
Dependências:	gstreamer-0.6 gstreamer-control-0.6

GThread

Descrição:	Thread support for GLib
Versão:	2.2.3
Dependências:	glib-2.0

GTK+

Descrição:	GIMP Tool Kit ($target target)
Versão:	2.2.4
Dependências:	gdk-$target-2.0 atk

gtk-doc

Descrição:	API documentation generator
Versão:	1.1
Dependências:	Nenhuma

gtk-engines-2

Descrição:	GTK+ Theme Engines
Versão:	2.2.0
Dependências:	gtk+-2.0

gtkhex

Descrição:	GtkHex - A hex display widget.
Versão:	1.0.0
Dependências:	gail atk gtk+-2.0

GTKmm

Descrição:	C++ wrapper for GTK+
Versão:	2.2.7
Dependências:	glibmm-2.0 gdkmm-2.0 pangomm-1.0 atkmm-1.0 gtk+-2.0

Gtk#

Descrição:	Gtk# - GNOME .NET Binding
Versão:	0.10
Dependências:	Nenhuma

gtksourceview

Descrição:	GTK+ 2.0 Source Editing Widget
Versão:	0.6.0
Dependências:	gtk+-2.0 libxml-2.0 libgnomeprint-2.2

GTK+

Descrição:	GIMP Tool Kit ($target target)
Versão:	2.2.4
Dependências:	gdk-$target-2.0 atk

gucharmap

Descrição:	GTK+ Unicode Character Map
Versão:	1.0.0
Dependências:	gtk+-2.0 glib-2.0 libgnomeui-2.0 libgnome-2.0

libart

Descrição:	LGPL version of the libart library
Versão:	2.3.16
Dependências:	Nenhuma

libbonobo

Descrição:	libbonobo
Versão:	2.4.0
Dependências:	glib-2.0 ORBit-2.0 bonobo-activation-2.0

libbonobui

Descrição:	libbonoboui
Versão:	2.4.0
Dependências:	glib-2.0 ORBit-2.0 libxml-2.0 libbonobo-2.0 libgnomecanvas-2.0 libgnome-2.0

libcroco

Descrição:	a CSS2 Parsing Library in C.
Versão:	0.3.0
Dependências:	glib-2.0

libgail-gnome

Descrição:	GNOME Accessibility Implementation Library for gnomeui and libbonoboui
Versão:	1.0.2
Dependências:	atk gtk+-2.0 libbonoboui-2.0

libgda

Descrição:	GDA (GNOME Data Access) library
Versão:	0.91.0
Dependências:	glib-2.0 libxml-2.0 libxslt

Libglade

Descrição:	a library for dynamically loading GLADE interface files
Versão:	2.0.1
Dependências:	gtk+-2.0 libxml-2.0

libgnome

Descrição:	libgnome
Versão:	2.4.0
Dependências:	glib-2.0 ORBit-2.0 libbonobo-2.0 gconf-2.0 gnome-vfs-2.0

libgnomecanvas-2.0

Descrição:	libgnomecanvas
Versão:	2.4.0
Dependências:	libart-2.0 pango pangoft2 gtk+-2.0

libgnomecups

Descrição:	GNOME CUPS Library
Versão:	0.1.6
Dependências:	glib-2.0 gobject-2.0

libgnomecups

Descrição:	GNOME CUPS Library
Versão:	0.17
Dependências:	libgnomecups-1.0 gtk+-2.0

libgnomedb

Descrição:	libgnomedb
Versão:	0.91.0
Dependências:	gtk+-2.0 >= 2.0.0 libgda >= 0.11.2 libgnomeui-2.0 >= 1.103 bonobo-activation-2.0 libbonoboui-2.0 libglade-2.0

libgnomeprint-2.0

Descrição:	libgnomeprint
Versão:	1.116.1
Dependências:	libart-2.0 glib-2.0 libxml-2.0 libbonobo-2.0 pango

libgnomeprint-2.2

Descrição:	libgnomeprint - Printing library for gtk+ based applications
Versão:	2.3.1
Dependências:	libart-2.0 glib-2.0 gmodule-2.0 gobject-2.0 libxml-2.0 pango

libgnomeprintui-2.0

Descrição:	libgnomeprintui
Versão:	1.116.0
Dependências:	libgnomeprint-2.0 libgnomecanvas-2.0

libgnomeprintui-2.2

Descrição:	libgnomeprint - Printing library for gtk+ based applications
Versão:	2.3.1
Dependências:	libgnomeprint-2.2 libgnomecanvas-2.0

libgnomeui

Descrição:	libgnomeui
Versão:	2.4.0.1
Dependências:	libgnome-2.0 libgnomecanvas-2.0 gtk+-2.0 gdk-pixbuf-2.0 libart-2.0 gconf-2.0 libbonoboui-2.0

libgsf-1

Descrição:	A library for reading and writing structured files (eg MS OLE and Zip)
Versão:	1.8.2
Dependências:	glib-2.0 gobject-2.0 libxml-2.0

libgsf-gnome-1

Descrição:	GNOME specific extensions to libgsf
Versão:	1.8.2
Dependências:	libgsf-1

libgtkhtml

Descrição:	libgtkhtml
Versão:	2.4.0
Dependências:	gtk+-2.0 gdk-pixbuf-2.0 libxml-2.0

libgtkhtml

Descrição:	libgtkhtml
Versão:	3.0.8
Dependências:	libgnomeui-2.0 >= 1.112.1 libgnomeprint-2.2 >= 2.2.0 libgnomeprintui-2.2 >= 2.2.1 libglade-2.0 >= 2.0.0 gal-2.0 >= 0.0.2

libgtop

Descrição:	Portable System Access Library
Versão:	2.0.5
Dependências:	glib-2.0

libIDL

Descrição:	IDL parsing library
Versão:	0.8.2
Dependências:	glib-2.0

libmetacity-private

Descrição:	Metacity internals shared
Versão:	2.6.1
Dependências:	gtk+-2.0

libmrproject

Descrição:	libmrproject
Versão:	0.9.1
Dependências:	glib-2.0 gmodule-2.0 gobject-2.0 libxml-2.0 libgsf-1

libmusicbrainz

Descrição:	The Musicbrainz Client Library.
Versão:	2.0.2
Dependências:	Nenhuma

libnautilus

Descrição:	A library to create Nautilus components
Versão:	2.4.0
Dependências:	eel-2.0 bonobo-activation-2.0 libbonobo-2.0 libbonoboui-2.0

libpanel-applet-2

Descrição:	libpanel-applet-2
Versão:	2.4.0
Dependências:	gtk+-2.0 libgnomeui-2.0 libbonoboui-2.0

librsvg

Descrição:	library that renders svg files using libart and pango
Versão:	2.4.0
Dependências:	glib-2.0 gdk-pixbuf-2.0 libart-2.0 libxml-2.0 pangoft2 libgsf-1 libcroco

libspi

Descrição:	Accessibility Technology software library
Versão:	1.3.7
Dependências:	libbonobo-2.0 atk gtk+-2.0

libstartup-notification

Descrição:	Startup notification library
Versão:	0.5
Dependências:	Nenhuma

libwnck

Descrição:	Window Navigator Construction Kit library
Versão:	2.4.0.1
Dependências:	gtk+-2.0 libstartup-notification-1.0

libxine

Descrição:	The xine engine library
Versão:	1.0.0
Dependências:	Nenhuma

libxklavier

Descrição:	libxklavier library
Versão:	0.71
Dependências:	libxml-2.0

libXML

Descrição:	libXML library version2.
Versão:	2.5.11
Dependências:	Nenhuma

libxslt

Descrição:	XSLT library version 2.
Versão:	1.0.32
Dependências:	libxml-2.0

libzvt-2.0

Descrição:	libzvt
Versão:	2.0.1
Dependências:	gtk+-2.0 libart-2.0

linc

Descrição:	ORBit2 network transport library
Versão:	1.1.1
Dependências:	glib-2.0 gobject-2.0 gthread-2.0

libloudmouth

Descrição:	libloudmouth
Versão:	0.13.2
Dependências:	glib-2.0

Mono

Descrição:	Mono Runtime
Versão:	0.26
Dependências:	glib-2.0 gmodule-2.0

Mozilla Gtk Embedding Widget

Descrição:	Mozilla Embedding Widget for Gtk+
Versão:	1.5b
Dependências:	mozilla-xpcom = 1.5b

JavaScript

Descrição:	The Mozilla JavaScript Library
Versão:	1.5b
Dependências:	mozilla-nspr = 1.5b

NSPR

Descrição:	The Netscape Portable Runtime
Versão:	1.5b
Dependências:	Nenhuma

NSS

Descrição:	Mozilla Network Security Services
Versão:	1.5b
Dependências:	mozilla-nspr = 1.5b

Mozilla Plug-In API

Descrição:	Mozilla Plug-In API
Versão:	1.5b
Dependências:	mozilla-xpcom = 1.5b

XPCOM

Descrição:	The Mozilla Cross Platform Component Library
Versão:	1.5b
Dependências:	mozilla-nspr = 1.5b

ORBit-2.0

Descrição:	High-performance CORBA Object Request Broker.
Versão:	2.8.1
Dependências:	glib-2.0 gmodule-2.0

ORBit-CosNaming-2.0

Descrição:	High-performance CORBA Object Request Broker - Naming Service.
Versão:	2.8.1
Dependências:	ORBit-2.0

ORBit-idl-2.0

Descrição:	ORBit-2.0 IDL Compiler Backend Interface
Versão:	2.8.1
Dependências:	libIDL-2.0

ORBit-imodule-2.0

Descrição:	ORBit runtime typelib generation service.
Versão:	2.8.1
Dependências:	ORBit-2.0 libIDL-2.0

Pango FT2

Descrição:	Freetype 2.0 font support for Pango
Versão:	1.2.5
Dependências:	pango

Pangomm

Descrição:	C++ wrapper for Pango
Versão:	2.2.7
Dependências:	glibmm-2.0 pango

Pango

Descrição:	Internationalized text handling
Versão:	1.2.5
Dependências:	glib-2.0,gobject-2.0,gmodule-2.0

Pango Xft

Descrição:	Xft font support for Pango
Versão:	1.2.5
Dependências:	pango

Pango X

Descrição:	X Window System font support for Pango
Versão:	1.2.5
Dependências:	pango

PyGTK

Descrição:	Python bindings for GTK+ and related libraries
Versão:	2.0.0
Dependências:	gobject-2.0

PyORBit

Descrição:	Python bindings for ORBit2
Versão:	2.0.0
Dependências:	ORBit-2.0 >= 2.4.4

Render

Descrição:	Render extension headers
Versão:	0.8
Dependências:	Nenhuma

Rhythmbox

Descrição:	Rhythmbox
Versão:	0.5.3
Dependências:	Nenhuma

SigC++

Descrição:	Type-safe signal and callback system for C++
Versão:	1.2.5
Dependências:	Nenhuma

libsoup

Descrição:	a glib-based HTTP library
Versão:	1.99.23
Dependências:	glib-2.0

vte

Descrição:	Vte terminal widget.
Versão:	0.11.10
Dependências:	xft >= 2.0 glib-2.0 gobject-2.0 gtk+-2.0 >= 2.2 fontconfig pangoxft >= 1.1.0 pangox

Xft

Descrição:	X FreeType library
Versão:	2.1.2
Dependências:	fontconfig

Xrender

Descrição:	X Render Library
Versão:	0.8.3
Dependências:	Nenhuma