Desenvolvimento de Aplicações Gráficas para o X Window System

Alfredo Kengi Kojima*
Paulo César Pereira de Andrade†
Carlos Augusto Moreira dos Santos‡

Mini-curso apresentado no Software Livre 2000
Porto Alegre, 5 de maio de 2000

*UFRGS, Instituto de Informática/Conectiva Informática. **kojima@windowmaker.org**
†Conectiva Informática. **pcpa@conectiva.com.br**
‡UFPel, Centro de Pesquisas Meteorológicas. **casantos@cpmet.ufpel.tche.br**

Este documento foi redigido usando apenas software livre.

Sumário

1 Introdução **5**
- 1.1 Histórico . . . 5
- 1.2 Arquitetura do X Window System . . . 6
  - 1.2.1 O modelo do sistema de janelas . . . 6
  - 1.2.2 Conceitos importantes . . . 7
- 1.3 Comunicação entre clientes . . . 9
  - 1.3.1 Envio de eventos . . . 9
  - 1.3.2 Atoms, properties, cut buffers e selections . . . 9
  - 1.3.3 Arrastar e soltar (drag-and-drop) . . . 10
- 1.4 Gerenciamento de janelas . . . 10

2 Desenvolvimento de aplicações **12**
- 2.1 Toolkits/Bibliotecas . . . 12
  - 2.1.1 Xlib . . . 12
  - 2.1.2 X Toolkit (Xt) . . . 13
  - 2.1.3 Athena (Xaw) . . . 13
  - 2.1.4 Motif . . . 14
  - 2.1.5 Gtk . . . 14
  - 2.1.6 Qt . . . 14
  - 2.1.7 Tcl/Tk . . . 15
  - 2.1.8 Outros toolkits . . . 15
- 2.2 Programação . . . 15
- 2.3 Exemplos de programas . . . 15
  - 2.3.1 Xlib . . . 15
  - 2.3.2 Xaw . . . 18
  - 2.3.3 Motif . . . 20
  - 2.3.4 Gtk . . . 21
  - 2.3.5 Qt . . . 21
  - 2.3.6 Tcl/Tk . . . 22

3 Documentação/URLs **23**

Referências Bibliográficas **24**

Apresentação

Este mini-curso pretende ser uma introdução à arquitetura do *X Window System* (X) e às técnicas de programação usadas no desenvolvimento de aplicações que o usem. O primeiro capítulo descreve os princípios que nortearam a concepção do protocolo X como ferramenta de suporte à criação de aplicações concorrentes e distribuídas. No segundo capítulo são apresentadas a interface para aplicativos da biblioteca *Xlib* e as técnicas básicas para autoria de aplicações orientadas ao tratamento de eventos. No terceiro são dados alguns exemplos de aplicações construídas por composição e mostrados exemplos de programas usando diversos toolkits de interfaces gráficas. Por fim, apresenta-se uma lista de bibliografia recomendada e endereços na Internet onde se pode encontrar maiores informações sobre o assunto.

1 Introdução[1]

1.1 Histórico

O X Window System [1] foi criado em 1984 no Massachusetts Institute of Technology (MIT) para dar suporte a dois projetos: o Projeto Athena [2] precisava de um sistama de janelas que pudesse ser usado em centenas de estações de trabalho com tela mapeada por bits e o Sistema Argus [3] precisava de um ambiente de depuração para aplicações multiprocessadas distribídas. Em 1988 o MIT formou um consórcio para prover suporte técnico e administrativo ao desenvolvimento subsequente do X Window System, financiado por seus membros, chamado MIT X Consortium. Em 1993 o MIT e os membros do consórcio decidiram que era melhor para seus interesses mudá-lo para fora do MIT e criaram uma organização independente e autônoma, chamada X Consortium, Inc., ao qual todos os direitos sobre X Window System foram dados pelo MIT em primeiro de janeiro de 1994.

Em 31 de dezembro de 1996 o X Consortium fechou suas portas e todos os direitos sobre o X Window System foram passados à Open Software Foundation. Em 1997, a Open Software Foundation deixou de existir, passando os direitos de todo o software desenvolvido por ela e do X Window System a uma nova organização chamada The Open Group. Nessa época o X Window System estava em sua versão 11, revisão 6.3 (X11 R6.3).

O X Consortium era sustentado financeiramente por contribuições de seus membros e não havia cobrança de taxas de licenciamento pelo uso dos padrões do X Window System e do código desenvolvido. Ao anunciar a revisão 6.4, em fevereiro de 1998, o Open Group quis, porém, adotar uma nova política de licenciamento, permitindo o uso livre do X Window System para fins não comerciais e de pesquisa, mas exigindo para sua comercialização o pagamento de uma taxa, proporcional ao número de cópias a distribuir. A justificativa para tal atitude era a necessidade de custear o desenvolvimento do software. Isso gerou protestos da comunidade de usuários de software livre, em especial o Projeto XFree86, que desenvolve a implementação do X Window System mais usada nos sistemas operacionais livres, como Linux e FreeBSD, e em sistemas comerciais como OS/2 e QNX. Havia o risco de se criarem duas "vertentes" do X: uma baseada na implementação do Open Group e outra na do XFree86, que decidiu continuar seu próprio rumo a partir da revisão 6.3, totalmente livre.

Em setembro de 1988 o Open Group decidiu voltar atrás em sua decisão anterior e retornou à política antiga de licenciamento, acabando com as restrições à cópia, modificação e distribuição do X. É importante observar, porém, que apesar de ser possível criar tantas variações quantas se queira do software distribuído pelo Open Group, o X Window System continua sendo um só, definido por uma especificação formal. Qualquer implementação que não siga à risca a especificação deve ser considerada, no mínimo, defeituosa.

[1] Este capítulo é baseado no artigo original de Scheifler e Gettys [1] e na documentação do X Window System.

1.2 Arquitetura do X Window System

O X possui arquitetura cliente-servidor, onde o servidor localiza-se na estação de trabalho do usuário, oferecendo "serviços" de acesso a dispositivos de entrada e saída, como teclado, mouse e vídeo gráfico. Clientes podem rodar local ou remotamente e conectam-se ao servidor através de uma rede, realizando a visualização de sua interface/dados no dispositivo de vídeo controlado pelo servidor.

Os serviços oferecidos pelo servidor incluem acesso transparente ao hardware gráfico (com primitivas de desenho 2D e criação/manipulação de janelas), mecanismos de comunicação entre clientes e entrada de dados por dispositivos como mouse e teclado.

Suporte a gráficos 3D não era um dos requisitos quando da implementação do sistema, mas o X Consortium chegou a padronizar uma extensão para suporte a modelagem 3D, baseada em PHIGS, chamada PEX (PHIGS extension for X), que nunca se tornou muito popular, apesar de alguns fabricantes terem-na incluído em seus servidores X. A extensão GLX, desenvolvida pela Silicon Graphics (SGI) para dar suporte ao padrão OpenGL, acabou se tornando a mais aceita no mercado. Em 1999 a SGI doou o código-fonte do GLX ao projeto XFree86.

1.2.1 O modelo do sistema de janelas

Gettys e Scheifler definem um sistema de janelas constituído de várias *interfaces*:

Interface de programação. Uma biblioteca de rotinas e tipos provida em uma linguagem de programação, para interação com o sistema de janelas. Há dois tipos de interface de programação:

Interface de baixo nível. Provê funções gráficas primitivas, tais como desenho de linhas, círculos e polígonos, desenho de texto, manipulação de cores e mapas de pixel. No X, a bibliotaca Xlib [4] provê essas funções primitivas, por meio de troca de mensagens com o servidor de display, cuja definição veremos mais adiante.

Interface de alto nível. Provê funções para apresentação de menus e outros elementos constituintes da interface. Existem diversas dessas interfaces para X, que no jargão da computação gráfica são chamadas *toolkits* (ver seção 2.1).

O X Window System não define uma interface de alto nível, mas possui uma biblioteca básica chamada X Toolkit Intrinsics (Xt) [5] que provê os mecanismos necessários à construção de uma interface, sobre a qual foram construídas duas interfaces bem conhecidas: X Athena Widgets [6] e Motif [7]. Outros toolkits como Tk [8], Xforms [9] e GTK [10] são construídos diretamente sobre a Xlib, sem usar o Xt.

Interface de aplicação. Define a interação mecânica entre a aplicação e o usuário, que é específica da aplicação. X, ao contrário de outros sistemas de janelas, não define regras para interação usuário/aplicação, embora o uso de um toolkit possa levar a aplicação a ter aparência e comportamento semelhantes às outras desenvolvidas com ele[2].

Interface de gerenciamento. Define a política de controle do posicionamento, tamanho e sobreposição das janelas sobre a área de trabalho. Este papel é desempenhado por uma aplicação chamada gerenciador de janelas (*window manager*). X não provê um gerenciador padrão, mas define um conjunto de atributos para cada janela chamado *window manager hints* (dicas), tais como os tamanhos máximo e mínimo que a janela deve ter. Gerenciadores de janela devem fazer uso dessas dicas para determinar as alterações que o usuário pode fazer na geometria

[2] O Open Group comercializa um pacote chamado Open Desktop, composto pelo X Window System, o toolkit Motif e o ambiente de trabalho Commonn Desktop Environment (CDE). Esse ambiente de trabalho é icluído na maioria dos sistemas operacionais Unix comerciais.

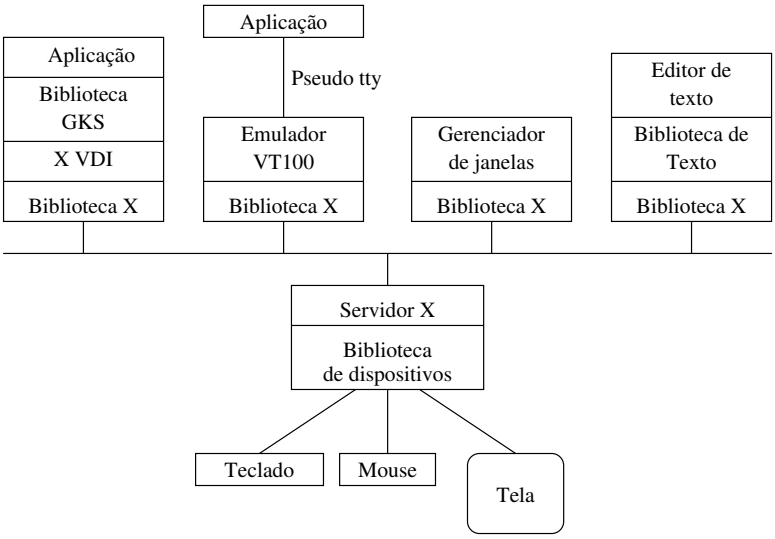

Figura 1.1: Estrutura de um sistema X [1]

das janelas. Uma conseqüência desta abordagem é que cada usuário pode usar o genenciador de janela que desejar, ao invés de ficar restrito àquele provido pelo sistema[3].

Gettys e Scheifler chamam de *interface com o usuário* a soma das interfaces de aplicação e de gerenciamento. Essa classificação define uma arquitetura em camadas e sugere independência entre *macanismo*, provido pelas camadas abaixo da interface de programação de baixo nível, inclusive, e *política*, provida pelas camadas acima desta. A figura 1.1 mostra como se relacionam os diversos elementos citados acima.

O X foi projetado de forma a fornecer mecanismos para que diferentes políticas de interface pudessem ser implementadas, ou seja, o X é como um *driver* de vídeo, teclado e mouse. Portanto, na sua forma pura, o X não possui uma interface com usuário. A função de prover tal interface é delegada aos clientes e bibliotecas especiais externos ao servidor X.

1.2.2 Conceitos importantes

Na terminologia do X Window System existem alguns termos aos quais são associados significados diferentes daqueles a que estamos acostumados:

Tela (screen). X possui suporte a múltiplos monitores ligados a uma mesma máquina ao mesmo tempo. Cada monitor ligado é denominado *screen*.

Display. Instância de um servidor X sendo executado em uma máquina. Geralmente um conjunto de hardware de vídeo, teclado e mouse mais o servidor que gerencia esses dispositivos. No X, múltiplas telas podem conviver, formando um display. O mouse pode passar de uma tela para outra, cruzando a fronteira entre elas. Na revisão 6.4, há uma extensão chamada Xinerama [12] que permite combinar várias telas, formando uma única tela virtual, de modo totalmente transparente às aplicações. Uma máquina dedicada que serve apenas como display, é chamade de *terminal X*.

[3] Se isso é bom ou não, é uma questão de gosto, e motivo de intermináveis controvérsias sobre qual o melhor window manager. Um dos autores deste documento, adivinhe qual :-), é o criador do Window Maker, mas outro é aficcionado pelo FVWM.

Janela (window). Pode ter dois sentidos dependendo do contexto:

- No nível mais baixo, é uma área retangular em uma das telas do display, na qual um ou mais clientes podem realizar operações de entrada e saída (desenhos, etc.). Normalmente apenas um cliente opera sobre cada janela, mas mais de um pode fazê-lo concorrentemente, se necessário. Uma janela pode ser movida, redimensionada e ter sua posição alterada em relação ao seu "empilhamento" e "aninhamento" com outras janelas.
- Em um nível mais alto, janela pode ter o mesmo significado comum que em ambientes como Windows(tm) (esse tipo de janela é conhecido como *janela de aplicação*, *toplevel window*, ou *toplevel shell*). Neste texto, o termo em inglês *Window* será usado quando houver possibilidade de confusão e o tipo de janela em questão for o primeiro (conceito de baixo nível).

A hierarquia de windows permite ao cliente estabelecer políticas de tratamento de eventos. O servidor pode ser instruído a remover um window de dentro de outro, colocando-o dentro de um terceiro. Este recurso, de adoção (*reparent*) é utilizado por vários aplicativos, especialmente pelo window manager (WM), uma aplicação especial que controla o posicionamento e redimencionamento das janelas dos aplicativos sobre a janela raiz (ver seção 1.4).

Área de trabalho. O conjunto formado pela janela raiz de uma tela e as janelas criadas sobre ela. O gerenciamento da área de trabalho cabe ao window manager. Somente um gerente pode controlar um display de cada vez, mas não existe a obrigatoriedade de se usar um determinado gerente.

Widget. Um elemento de interface gráfica. Exemplos são botões, *labels*, menus, caixas de texto, combobox etc. Janelas *toplevel* também são consideradas widgets. Para aumentar a confusão, um widget é geralmente construído usando-se janelas (windows).

Servidor. O programa que compartilha um ou mais Displays com os clientes por meio do protocolo X. O servidor é construído em duas camadas, uma dependente do hardware e outra independente. Portar o servidor para outro hardware implica em modificar apenas a primeira. Drivers de vídeo fazem parte do servidor X e somente ele tem acesso ao hardware. Fato curioso é que há um servidor especial, chamado Xnest, que é ao mesmo tempo cliente e servidor: é um cliente do servidor real, rodando como qualquer outro cliente, mas funciona como servidor para outros clientes conectados a ele, para os quais Xnest aparenta ser um servidor convencional. A operação é transparente, inclusive podendo um Xnest ser executado dentro (como cliente) de outro Xnest.

Cliente. Uma aplicação que usa um ou mais displays, comunicando-se com o(s) servidor(es) por meido do protocolo X. Uma aplicação pode ter mais de uma conexão aberta simultaneamente com o mesmo display, mas normalmente isso não é necessário. A aplicação não precisa rodar no mesmo computador que o display; na verdade a máquina onde roda a aplicação não precisa nem possuir vídeo, teclado ou mouse, bastando que tenha uma conexão de rede com o servidor X.

Gerente de Display (*display manager*). Aplicação responsável por fornecer autenticação de acesso a um ou mais displays. O gerente de display padrão do X é o programa XDM.

1.3 Comunicação entre clientes[4]

1.3.1 Envio de eventos

X provê mecanismos de comunicação não só entre cliente e servidor, mas também entre clientes. A rigor, um cliente não pode realmente enviar eventos para outro cliente, mas para as janalas que este possui no servidor, usando a função XSendEvent da Xlib. Clientes que desejem receber eventos enviados por outros deverão fazer uma requisição ao servidor, via XSelectInput, e a partir de então o servidor os enviará. Um cliente não precisa solicitar eventos para uma janela criada por ele, mas pode informar que não os deseja.

1.3.2 Atoms, properties, cut buffers e selections

Eventos permitem envio de mensagens, mas não são adequados ao compartilhamento de dados entre clientes. Para fazer isto existem *átomos*, *properties* e *selections*.

Átomos (atoms) são nomes únicos que os clientes podem usar para comunicar informação uns aos outros. Um átomo é simplesmente uma cadeia de bytes de tamanho arbitrário. Ao invés de enviar essas cadeias pela rede o cliente registra-as no nervidor, obtendo um rótulo único de 32 bits. Átomos também são usados para especificar tipos de dados e nomes de properties e selections. Existe um conjunto de átomos padrão, pré-registrados.

Propriedades (properties) são blocos de dados que um cliente "pendura" em uma janela, contendo quantidades arbitrárias de informação de um tipo escolhido. Ppropriedades são mantidas na memória do servidor, junto com as janelas, e podem ser criadas, destruídas, lidas e gravadas. A gravação/leitura é semelhante à que se faria em um arquivo.

Aplicações que compartilham dados via propriedades devem solicitar ao servidor, via XSelectInput, a notificação de alterações nas mesmas. A partir de então, sempre que houver alteração em uma propriedade o cliente será notificado por um evento do tipo PropertyNotify.

Áreas de recorte (cut buffers). A janela raiz de cada *screen* possui um grupo de propriedades identificadas pelos átomos pré-definidos XA_CUTBUFFER0 a XA_CUTBUFFER7. A Xlib possui funções para armazenar bytes nestas áreas de recorte, permitindo às aplicações implementar um sistema simples de recortar-e-colar. Um exemplo de aplicação das areas de recorte é o emulador de terminal Xterm que permite marcar uma parte do texto em uma janela e colá-la em outra.

Seleções (selections) são um mecanismo mais sofisticado de compartilhamento de dados. Uma seleção é tratada como um bastão em um sistema de revezamento. Apenas um cliente pode deter o bastão de cada vez. O detentor do bastão deve então atender a requisições dos outros clientes, convertendo os dados selecionados para um formato solicitado, armazenando o resultado em uma propriedade e notificando o solicitande da disponibilidade.

Seleções permitem que clientes armazenem os dados em seu próprio espaço de dados e seu uso exige que a aplicação a quem os dados foram solicitados saiba como convertê-los, se necessário, para o formato que o solicitante pede.

[4] As convenções para interação entre aplicações são descritas por Rosenthal e Marks [15]. Uma discussão extensa do tema foge ao escopo deste texto.

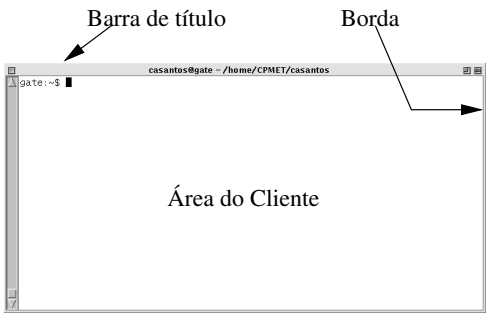

Figura 1.2: Janela de um cliente (xterm) dentro do contêiner criado pelo window manager.

1.3.3 Arrastar e soltar (drag-and-drop)

Arrastar e soltar, ou *drag-and-drop* (DnD) é técnica interativa baseada na metáfora de selecionar um objeto em uma janela e transportá-lo para outra, resultando em uma transferência dos dados entre as duas aplicações. X não provêm explicitamente tal recurso, pois ele está associado a uma política, mas as aplicações podem usar os mecanismos disponíveis para implementá-lo.

Há muitos protocolos de drag-and-drop desenvolvidos para X, dentre os quais destacam-se os três mais populares:

Motif. O toolkit Motif define um protocolo DnD baseado em mensagens entre clientes, properties e selections. Como exemplos de aplicações que usam esse protocolo, podemos citar o Netscape Navigator (versão para Unix) e os programas integrantes do CDE.

Offix. Offix foi um pacote desenvolvido por César Augusto Rorato Crusius, contendo um conjunto de aplicativos úteis, que mais tarde evoluiu para um toolkit gráfico completo. Crusius desenvolveu um protocolo de comunicação entre aplicações que permite drag-and-drop. A biblioteca DND é distribuida livremente e o protocolo é usado por um grande número de aplicações.

Xdnd. Xdnd é um protocolo recente, mas que está se tornando padrão de fato entre os pacotes de software livre, sendo adotado por toolkits como GTK, Qt e JX. O toolkit LessTif (ver seção 2.1.4) também tem suporte a Xdnd, apesar de ser um clone de Motif.

Esses "padrões" são incompatíveis entre si e a falta de um padrão real é, provavelmente, o motivo principal da relativa escassez de aplicações para X Window que suportem drag-and-drop.

1.4 Gerenciamento de janelas

O uso de um WM não é obrigatório, mas é muito útil porque implementa as políticas de atribuição do foco (a janela em foco recebe os eventos de teclado) e redimensionamento e sobreposição de janelas. O window wanager faz isso com base em um conjunto convenções estabelecidas entre ele e os clientes, usando mecanismos como properties (ver seção 1.3.2) acoplados a cada janela, trocas de mensagens e eventos. Há um conjunto padrão de *hints* (dicas) e mensagens, definidos pelo ICCCM (Inter-Client Conventions Manual), mas alguns toolkits estendem as convenções definidas por ele, adicionando novos *hints*.

Quando uma aplicação deseja criar uma janela *toplevel*, cria-se um Window descendente direto da janela raiz. Quando a aplicação requisitar o *mapping* (um Window permanece invisível até que seja requisitado o *mapping* deste) desta janela ao servidor X, o gerenciador de janelas interceptará esta requisição e realizará diversas operações, tais como:

- verificação de propriedades colocadas no próprio Window da aplicação, contendo informações relativas à forma como ele deve ser tratado pelo gerenciador de janelas. Exemplos dessas propriedades são WM_NORMAL_HINTS, que fornece informações sobre a geometria da janela, como o tamanho máximo/mínimo; WM_NAME que contém um texto a ser mostrado ao usuário como sendo o título da janela e WM_HINTS que fornece informações como uma imagem a ser usada em representações iconizadas da janela. O utilitário xprop pode ser usado para visualizar o conjunto de propriedades definidas para uma janela;
- criação de uma borda ou *contêiner* por meio do qual se poderá realizar as operações de redimensionamento/reposicionamento, visualizar o título da janela e outros. A aparência desta borda pode ser controlada através do uso dos *hints* acopladas à janela;
- *reparenting* da janela para dentro da borda (a janela passa a ser descendente da borda, em vez da janela raiz).

2 Desenvolvimento de aplicações

2.1 Toolkits/Bibliotecas

Toolkits e/ou bibliotecas implementam interfaces de programação para X. Não existe toolkit padrão para interfaces gráficas e o X Consortium nunca impôs um padrão, o que resultou na grande proliferação desse tipo de biblioteca[1]. Há dezenas de toolkits diferentes, que vão de simples cunjuntos de widgets usadas em uma única aplicação até bibliotecas sofisticadas e cheias de recursos. Não há como tratarmos de todos eles neste texto, por isso nos restrigiremos àqueles mais comuns, apresentando um programa-exemplo para cada um.

2.1.1 Xlib

Xlib [4] é a API de mais baixo nível disponível para se programar aplicações para X (a não ser que a aplicação utilize o protocolo X diretamente para se comunicar com o servidor). Ela foi definida como padrão pelo X Consortium e é uma biblioteca em linguagem C, que faz a comunicação com o servidor X e oferece funções com correspondência direta, ou quase, com o protocolo X. Possui funções para se abrir uma conexão com um display, criação, destruição e manipulação de janelas, desenho, recebimento de eventos etc.

Todos (ou quase todos) os toolkits utilizam a Xlib na sua implementação, oferecendo um certo nível de abstração e funções de mais alto nível que permitem, além da construção de interfaces com o usuário, tratar de outras necessidades da aplicação, tais como configuração, tratamento de erros e acesso a serviços do sistema operacional.

Apesar da Xlib poder ser utilizada na escrita de aplicações, isso não é aconselhável para aplicações complexas ou que possuam interfaces não triviais com usuário, pois além de ser muito trabalhoso gerará aplicações com interfaces inconsistentes com a das outras. Por outro lado, programas que realizam operações "de baixo nível", tais como gerenciadores de janelas e processadores de imagens, terão obrigatoriamente que usar as funções da Xlib. Exemplos de aplicações comuns escritas exclusivamente com Xlib são o procesador de imagens Xv e o emulador de terminal rxvt.

Uma abordagem bastante comum em aplicações complexas é usar um toolkit para construir a interface e funções da Xlib para operações de baixo nível. Neste caso deve-se tomar cuidado para que as chamadas feitas à Xlib não interfiram com a política implementada pelo toolkit[2]. Exemplos de aplicações desse tipo são o gerenciador de janelas twm, que usa Xt e Xlib, e o emulador de terminal xterm, que usa Xaw e Xlib.

[1] Em meados da década de 80, quando as interfaces gráficas começaram a se tornar populares, houve a chamada "guerra de interfaces", na qual grandes fabricantes tentaram estabelecer um padrão para a indústria. Os principais concorrentes eram o OpenLook, da Sun e AT&T, e o Motif, da OSF. No fim o Motif venceu e tornou-se a interface padrão nos sistemas Unix comerciais, mas por ser proprietária não atraiu a simpatia de muitos desenvolvedores de software livre.

[2] Exemplo desse tipo de interferência é o uso da função XNextEvent para receber eventos fora do laço de tratamento de eventos do toolkit

2.1.2 X Toolkit (Xt)

Xt, ou X Toolkit Intrinsics [5] foi criado com a intenção de prover recursos básicos para a escrita de toolkits gráficos. Xt não oferece widgets completos, mas fornece os mecanismos necessários para se criar uma biblioteca que os possua.

Xt implementa um modelo de Programação Orientada a Objetos (POO), mas como é implementado em C, uma linguagem exclusivamente procedural, isso é feito por meio de uma série de convenções e técnicas de programação. Talvez esse seja o motivo pelo qual a programação para Xt seja tão desafiadora (ou assustadora) a princípio: é necessário que o programador tenha um bom domínio da linguagem de programação e desenvolva uma rigorosa disciplina de trabalho.

Como vantagem do modelo adotado, Xt e os toolkits derivados dele são extremamente flexíveis e possuem recursos únicos dentre os demais. Uma aplicação Xt pode ter não só sua aparência mas também seu comportamento (até certo ponto, pelo menos) modificados pelo usuário sem que seja necessário alterar uma única linha do código-fonte do programa. E isso pode ser feito com um programa que esteja rodando! O mecanismo por meio do qual isso é feito chama-se *Xt resources* e consiste uma das características mais inusitadas deste toolkit.

Xt define classes básicas de widget que, *per se*, não implementam nenhuma política de interface. Elas existem para que se criem *subclasses*, estas sim determinantes de uma política. O restante do Xt consiste de funções para:

- inicialização do toolkit e da aplicação;
- criação, manipulação e destruição de widgets;
- adição e remoção de *callbacks* e *actions* (resposta a eventos) a widgets;
- gerenciamento de Xt Resources;
- tratamento de eventos, *timeouts* e sinais.

2.1.3 Athena (Xaw)

O "X Athena Widget Set" (Xaw) [6] foi criado durante as pesquisas do Projeto Athena (daí o nome) como um exemplo de uso do Xt. Xaw nunca foi um toolkit completo, para uso profissional, mas acabou tornando-se muito utilizado poque era distribuído junto com a implementação de referência do X, apesar de nunca ter sido adotado como um padrão pelo X Consortium. Como uma consquência de sua origem como mero exercício acadêmico Xaw carece de muitos recursos encontrados em outros toolkits, como documentação detalhada ou uma grande coleção de widgets, mas suas duas principais defiências são:

1. não implementar uma política de *foco de entrada* (*input focus*) para os eventos de teclado. Não há como identificar qual dos widgets receberá os eventos correspondentes à digitação (normalmente é o widget que está sob o cursor do mouse);
2. como consequência do problema anterior, não há como percorrer a hierarquia de widgets usando exclusivamente o teclado;
3. os widgets não se diferenciam muito visualmente e não há como distinguir uma caixa de entrada de texto de um botão, por exemplo.

Como alternativa para resolver esses problemas, ou pelo menos o último deles, surgiram diversas variantes de Xaw: Xaw3D, Xaw95, neXtaw, XawM, etc.. Algumas aplicações também tentam resolver os problemas por meio de truques de programação, mas como regra geral podemos dizer que Xaw não se presta para o desenvolvimento de aplicações grandes. Para isso existe Motif.

2.1.4 Motif

Motif é um toolkit, desenvolvido pela antiga OSF, hoje Open Group, concebido desde o início como uma ferramenta de desenvolvimento profissional, ao contrário do Xaw. Os fabricantes de Unix o adotaram interface gráfica padrão, mas ele não é distribuído livremente como o X ou Xaw, e sim comercializado, o que restringiu bastante o número de aplicações Motif que rodam em sistemas operacionais livres[3].

Um aspecto importante do Motif é que apesar de ser um software comercial sua especificação é aberta: os manuais de referência são publicados pela editora Prentice Hall e podem ser comprados por qualquerm um. Isso permitiu que um grupo começasse a desenvolver um clone livre, sob licença GPL, chamado LessTif [18], que já imita com bastante sucesso as características do Motif versão 1.2.

A interface com o usuário provida por Motif é inspirada no sistema DEC Windows, da Digital, no padrão CUA, da IBM, e no Windows (pré-95), da Microsoft.

2.1.5 Gtk

Gtk é uma biblioteca de widgets escrita em C, com inspiração em orientação a objetos, a exemplo do X Toolkit. Devido a essa característica, a programação nela pode-se tornar um tanto tediosa, pois é necessário emular POO "à mão", por meio de uma disciplina de programação. A razão é histórica: Gtk foi criado para substituir Motif como toolkit do GIMP (GNU Image Manipulation Program) e como consequência disso existe uma grande influência de Xt e Motif sobre Gtk, como se pode perceber tanto na sintaxe quanto na semântica das funções disponíveis.

Existem interfaces para uso do Gtk em diversas linguagens, além do C, incluindo Python, C++, Objective-C e Guile (um dialeto de Lisp). Naturalmente o suporte nativo à orientação a objetos da linguagem é aproveitado, quando existe. Como o Gtk esconde toda a interface de baixo nível, o programador não precisa saber nada de X Window para usá-lo, se ficar restrito aos recursos do toolkit. Isso permitiu que se criasse uma versão para o Microsoft Windows, também.

As aplicações mais conhecidas que usam Gtk são o próprio GIMP e o ambiente GNOME (GNU Network Object Model Environment). Existe um construtor de interfaces gráficas para Gtk, chamado Glade.

2.1.6 Qt

Qt é um toolkit em C++, orientado a objetos e multiplataforma, desenvolvido pela Trol Tech. A versão para X Window é distribuída livremente, mas houve uma controvérsia há algum tempo por problemas de licenciamento, uma vez que as versões para outras plataformas são comerciais.

A interface do Qt é fortemente influenciada pelo Windows 95, da Microsoft, mas o toolkit implementa um modelo de comunicação entre widgets chamado "signals and slots", no qual os sinais gerados por um widget podem ser conectados a um slot existente noutro, o que dispensa a criação de funções *callback* para responder aos sinais.

As aplicações mais conhecidas que usam Qt são o ambiente KDE (K Desktop Environment) e o navegador Opera. Existem ambientes de desenvolvimento para ele, como KDevelop e Qtez, que incluem ferramentas de construção de interface.

[3] Vendedores independentes de software como Metrolink, Xi Graphics e Apps2go, comercializam implementações de Motif que rodam em Linux e BSD, a preços bastante acessíveis (pelo menos para desenvolvedores profissionais).

2.1.7 Tcl/Tk

Tcl é uma linguagem de scripts bastante simples e Tk, o seu toolkit gráfico. Programas escritos em Tcl/Tk podem ser rodados em uma variedade de plataformas, além de X Window, incluindo Windows e Macintosh. A linguagem não é apropriada para a escrita de aplicações grandes, porém é bastante vantajosa para escrita de pequenos programas e prototipação, por ser de fácil uso.

A interface peovida pelo Tk é idêntica, ou pelo menos muito semelhante, à do Motif, mas a implementação é completamente diferente.

Tk pode ser usado também com outras linhguagens de programação, além de Tcl, e há interfaces para C, Python, Perl e Scheme (outro dialeto de Lisp). Há um modelador de interfaces para Tcl/Tk chamado Visual Tcl (VTCL).

2.1.8 Outros toolkits

Além dos toolkits mencionados anteriormente existem muitos outros de que não trataremos aqui por falta de espaço ou porque não temos conhecimento o bastante sobre eles.

2.2 Programação

Um programa para X, realiza os seguintes passos, em linhas gerais:

1. Inicialização (conexão com servidor X)

   Durante a inicialização, o programa fará o tratamento de opções de linha de comando, conexão com o servidor X (através de sockets UNIX ou TCP/IP, no caso do Linux) e alocação de estruturas internas. Tudo isto é feito internamente pelo toolkit, caso seja utilizado um.

2. Montagem da interface gráfica

   Após aberta a conexão com o servidor, o programa pode criar e montar sua interface (janelas, menus, botões etc.) utilizando os widget oferecidos pelo toolkit, ou manualmente, usando-se as primitivas do Xlib.

3. Loop de tratamento de eventos.

   Finalmente, o programa entra em um loop, onde fica esperando por eventos vindos do servidor, como pressionamento de teclas ou botões do mouse, movimento do mouse ou requisição do servidor para redesenhar o conteúdo de suas janelas. Em toolkits, estes eventos são primeiro tratados internamente e dependendo do caso (quando o mouse é clicado em um widget botão, por exemplo) são repassados ao programa através de *callbacks* ou mecanismos similares.

2.3 Exemplos de programas

2.3.1 Xlib

O programa a seguir é um exemplo de programa excrito em C usando exclusivamente as funções da Xlib. Para compilá-lo, em Linux:

```
% gcc -c -I/usr/X11R6/include hello-Xlib.c
% gcc -L/usr/X11R6/lib hello-Xlib.o -o hello-Xlib -lX11
```

```
 1  #include <X11/Xlib.h>
 2  #include <X11/Xatom.h>
 3  #include <X11/Xutil.h>
 4  #include <X11/keysym.h>
 5  #include <X11/cursorfont.h>
 6
 7  #include <stdio.h>
 8
 9  void
10  button_callback(XEvent ev, GC gc)
11  {
12    XDrawImageString(ev.xany.display, ev.xany.window,
13     gc, ev.xbutton.x, ev.xbutton.y,
14     "Hello!", strlen("Hello!"));
15  }
16
17  void
18  redraw_callback(XEvent ev, GC gc)
19  {
20    XWindowAttributes attr;
21
22    (void) XGetWindowAttributes(ev.xany.display, ev.xany.window, &attr);
23    XDrawLine(ev.xany.display, ev.xany.window, gc,
24      0, 0, attr.width - 1, attr.height - 1);
25    XDrawLine(ev.xany.display, ev.xany.window, gc,
26      0, attr.height - 1, attr.width - 1, 0);
27  }
28
29  void
30  configure_callback(XEvent ev, GC gc)
31  {
32  }
33
34  int main(int argc, char *argv[])
35  {
36    Display *dsp;
37    Window root, win;
38    GC gc;
39    XEvent ev, nextev;
40    KeySym key;
41    XSetWindowAttributes xswa;
42    XClassHint classhint;
43    Atom wm_delete_window;
44    Atom wm_protocols;
45    int screen;
46    unsigned long foreground, background;
47    int i;
48    char text[10];
49    int done;
50
51  /***** Inicialização ********************************************************/
52
53    dsp = XOpenDisplay(NULL);
54    if (dsp == NULL) {
55      fprintf(stderr, "XOpenDisplay falhou!\n");
56      exit(1);
57    }
58    screen = DefaultScreen(dsp);
59    root = DefaultRootWindow(dsp);
60    background = WhitePixel(dsp, screen);
61    foreground = BlackPixel(dsp, screen);
62
63    xswa.background_pixel = background;
```

```
64    xswa.border_pixel = foreground;
65    xswa.backing_store = WhenMapped;
66    xswa.event_mask = ButtonPressMask | KeyPressMask |
67      ExposureMask | StructureNotifyMask;
68    win = XCreateWindow(
69      dsp,/* display */
70      root,/* janela pai */
71      200, 300,/* posição (x,y) */
72      350, 250, 5,/* largura, altura, largura da borda */
73      CopyFromParent,/* profundidade */
74      InputOutput,/* classe */
75      CopyFromParent,/* visual */
76      CWBackPixel | CWBorderPixel | /* atributos a estabelecer */
77      CWBackingStore | CWEventMask,
78      &xswa);
79
80    classhint.res_name = "hello";
81    classhint.res_class = "Hello";
82    XmbSetWMProperties( /* "hints" para o window manager */
83      dsp, win,/* display, janela */
84      "Hello, cruel world!",/* título da janela */
85      "Hello",/* título do ícone */
86      argv, argc, None, None, &classhint
87      );
88
89    /* WM_DELETE_WINDOW é a mensagem do window manager, mandando terminar */
90    wm_delete_window = XInternAtom (dsp, "WM_DELETE_WINDOW", False);
91    wm_protocols = XInternAtom (dsp, "WM_PROTOCOLS", False);
92    (void) XSetWMProtocols (dsp, win, &wm_delete_window, 1);
93
94    XDefineCursor(dsp, win, XCreateFontCursor(dsp, XC_man));
95    gc = XCreateGC(dsp, win, 0, 0);
96    XSetBackground(dsp, gc, background);
97    XSetForeground(dsp, gc, foreground);
98
99    XMapRaised(dsp, win);
100
101 /***** Laço de tratamento de eventos ************************************/
102
103   done = False;
104   while (!done) {
105     XNextEvent(dsp, &ev);
106     switch (ev.type) {
107     case Expose:
108       /* Comprime uma sequência de Exposes, para evitar repetidas
109        * repinturas, já que para um redimensionamento da janela o
110        * servidor X envia vários Exposes consecutivos.
111        */
112       while (
113 (XEventsQueued(ev.xexpose.display, QueuedAfterReading) > 0) &&
114 (XPeekEvent(ev.xexpose.display, &nextev), (nextev.type == Expose))
115 ) {
116 XNextEvent(dsp, &ev);
117       }
118       redraw_callback(ev, gc);
119       break;
120     case MappingNotify:
121       XRefreshKeyboardMapping(&ev.xmapping);
122       break;
123     case ConfigureNotify:
124       /* Reconfiguracoes da janela (alteracao de tamanho, etc).
125        * Se ConfigureNotify é seguido de Expose, a janela foi redimensionada,
126        * caso contrário foi apenas movida.  No primeiro caso é necessário
```

```
127         * recalcular a geometria do conteúdo da janela.
128         */
129        if (
130  (XEventsQueued(ev.xexpose.display, QueuedAfterReading) > 0) &&
131  (XPeekEvent(ev.xexpose.display, &nextev), (nextev.type == Expose))
132  ) {
133       configure_callback(ev, gc);
134  }
135        break;
136      case ButtonPress:
137        button_callback(ev, gc);
138        break;
139      case KeyPress:
140        XLookupString(&ev.xkey, text, 10, &key, NULL);
141        switch (key) {
142  case XK_Q: /* fim */
143  case XK_q:
144    done = True;
145    break;
146        } /* switch(ev.xkey) */
147        break; /* case KeyPress */
148      case ClientMessage:
149        if (ev.xclient.message_type == wm_protocols &&
150            ev.xclient.data.l[0] == wm_delete_window)
151  done = True;
152        else
153  XBell (dsp, 0);
154        break; /* case ClientMessage */
155      } /* switch(ev.type) */
156    } /* while(!done) */
157
158  /***** Término ********************************************************/
159
160    XFreeGC(dsp, gc);
161    XDestroyWindow(dsp, win);
162    XCloseDisplay(dsp);
163    return 0;
164  }
```

2.3.2 Xaw

O próximo exemplo usa o X Toolkit e Xaw. Observe-se a simplificação das três etapas (inicialização, criação da interface e loop de eventos). Como Xaw não provê um tratamento padrão para a mensagem WM_DELETE_WINDOW gerada pelo window manager, é preciso incluir o código necessário, à semelhança do que foi feito na versão Xlib. Para compilá-lo em Linux:

```
% gcc -c -I/usr/X11R6/include hello-Xaw.c
% gcc -L/usr/X11R6/lib hello-Xaw.o -o hello-Xaw -lXaw -lXt -lX11
```

```
1  /*
2   * hello-Xaw.c
3   */
4
5  #include <X11/Intrinsic.h>
6  #include <X11/StringDefs.h>
7
8  #include <X11/Xaw/Command.h>
9
10 static Atom wm_delete_window;
11 static Atom wm_protocols;
12
```

```
13 static void WMProtocols(Widget w, XEvent *ev, String *params, Cardinal *n)
14 {
15     if (ev->type == ClientMessage &&
16         ev->xclient.message_type == wm_protocols &&
17         ev->xclient.data.l[0] == wm_delete_window) {
18 XtAppSetExitFlag(XtWidgetToApplicationContext(w));
19     }
20 }
21
22 static void
23 QuitCallback(Widget w, XtPointer cdata, XtPointer cbs)
24 {
25     XtAppSetExitFlag(XtWidgetToApplicationContext(w));
26 }
27
28 XtActionsRec actions[] = {
29     {"WMProtocols", WMProtocols}
30 };
31
32 String fallback[] = {
33     "*button.background: rgb:c0/c0/c0",
34     "*button.font: -b&h-lucida-bold-r-normal-*-*-240-*-*-p-*-*-*",
35     "*button.label: Hello, world!",
36     NULL
37 };
38
39 int
40 main(int argc, char *argv[])
41 {
42     XtAppContext app;
43     Widget top, button;
44
45     XtSetLanguageProc(NULL, (XtLanguageProc)NULL, NULL);
46
47     top = XtVaAppInitialize(
48     &app,/* Application context */
49     "XHello",/* Application class */
50     NULL, 0,/* command line option list */
51     &argc, argv,/* command line args */
52     fallback,/* for missing app-defaults file */
53     NULL); /* terminate varargs list */
54
55     XtAppAddActions(app, actions, XtNumber(actions));
56     XtOverrideTranslations(top,
57     XtParseTranslationTable ("<Message>WM_PROTOCOLS: WMProtocols()"));
58
59     button = XtVaCreateManagedWidget(
60     "button",/* arbitrary widget name */
61     commandWidgetClass,/* widget class from Label.h */
62     top,/* parent widget */
63     NULL); /* terminate varargs list */
64     XtAddCallback(button, XtNcallback, QuitCallback, NULL);
65
66     XtRealizeWidget(top);
67     wm_delete_window = XInternAtom(XtDisplay(top), "WM_DELETE_WINDOW", False);
68     wm_protocols = XInternAtom(XtDisplay(top), "WM_PROTOCOLS", False);
69     (void) XSetWMProtocols(XtDisplay(top), XtWindow(top),
70    &wm_delete_window, 1);
71
72     XtAppMainLoop(app);
73     return 0;
74 }
```

2.3.3 Motif

O programa a seguir é a versão Motif do anterior. Observe-se a semelhança entre os dois e o fato de que Motif já incorpora o tratamento da mensagem WM_DELETE_WINDOW, o que torna desnecessária boa parte do código existente na versão Xaw. Para compilá-lo em Linux é preciso ter a biblioteca Motif (ou LessTif):

```
% gcc -c -I/usr/X11R6/include hello-Xm.c
% gcc -L/usr/X11R6/lib hello-Xm.o -o hello-Xm -lXm -lXt -lX11
```

```
 1 /*
 2  * hello-Xm.c
 3  */
 4
 5 #include <Xm/Xm.h>
 6 #include <Xm/PushB.h>
 7
 8 static void
 9 QuitCallback(Widget w, XtPointer cdata, XtPointer cbs)
10 {
11     XtAppSetExitFlag(XtWidgetToApplicationContext(w));
12 }
13
14 String fallback[] = {
15     "*button.background: rgb:c0/c0/c0",
16     "*button.fontList: -b&h-lucida-bold-r-normal-*-*-240-*-*-p-*-*-*",
17     "*button.labelString: Hello, world!",
18     NULL
19 };
20
21 int
22 main(int argc, char *argv[])
23 {
24     XtAppContext app;
25     Widget top, button;
26
27     XtSetLanguageProc(NULL, (XtLanguageProc)NULL, NULL);
28
29     top = XtVaAppInitialize(
30     &app,/* Application context */
31     "XHello",/* Application class */
32     NULL, 0,/* command line option list */
33     &argc, argv,/* command line args */
34     fallback,/* for missing app-defaults file */
35     NULL); /* terminate varargs list */
36
37     button = XtVaCreateManagedWidget(
38     "button",/* arbitrary widget name */
39     xmPushButtonWidgetClass,/* widget class from PushB.h */
40     top,/* parent widget */
41     NULL); /* terminate varargs list */
42     XtAddCallback(button, XmNactivateCallback, QuitCallback, NULL);
43
44     XtRealizeWidget(top);
45
46     XtAppMainLoop(app);
47     return 0;
48 }
```

2.3.4 Gtk

O programa a seguir é um a versão Gtk do mesmo exemplo anterior. Observe-se que o código não utiliza nenhuma funcionalidade específica do X, portanto pode ser compilado em Windows também. Observe-se também a semelhança estrutural com o exemplo anterior, que demostra a influência de Motif sobre o Gtk de que falamos anteriormente. Para compilá-lo em Linux:

```
% gcc hello-Gtk.c -o hello-Gtk `gtk-config --cflags --libs`
```

```
 1  /* inclusão dos cabeçalhos do Gtk */
 2
 3  #include <gtk/gtk.h>
 4
 5  /* função que será chamada quando a janela receber o sinal "destroy" */
 6  void quit_hello(GtkWidget *widget, gpointer data)
 7  {
 8      gtk_main_quit ();
 9  }
10
11  int main(int argc, char *argv[])
12  {
13      GtkWidget *window, *button;
14
15      /* Inicialização do Gtk: conexão com o servidor X, tratamento de opções */
16      /* de linha de comando padrão e outras inicializações internas.         */
17      gtk_init(&argc, &argv);
18
19      /* criacao de uma janela principal */
20      window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
21
22      /* conecta o sinal "destroy" à função quit_hello */
23      gtk_signal_connect (GTK_OBJECT (window), "destroy",
24                          GTK_SIGNAL_FUNC (quit_hello), NULL);
25
26      /* criacao de um button */
27      button = gtk_button_new_with_label("Hello, world!");
28
29      /* Quando o botão receber o sinal "clicked", chama a função
30       * gtk_widget_destroy, passando como parâmetro o objeto window */
31      gtk_signal_connect_object (GTK_OBJECT (button), "clicked",
32                                 GTK_SIGNAL_FUNC (gtk_widget_destroy),
33                                 GTK_OBJECT (window));
34
35      /* o button é colocado dentro da janela principal */
36      gtk_container_add(GTK_CONTAINER(window), button);
37
38      /* faz com que o widget seja visivel */
39      gtk_widget_show(button);
40
41      /* faz com que a janela seja visivel na tela */
42      gtk_widget_show(window);
43
44      /* entra em um loop infinito de processamento de eventos */
45      gtk_main();
46  }
```

2.3.5 Qt

Esta é a versão Qt do exemplo. Observe-se o uso do mecanismo de signals e slots para conectar o sinal “clicked” do botão ao slot “quit” da aplicação. Para compilá-lo em Linux:

```
% c++ -c -I/usr/include/qt -I/usr/X11R6/include hello-Qt.c++
% gcc -L/usr/X11R6/lib hello-Qt.o -o hello-Qt -lX11 -lqt
```

```
 1  #include <qapplication.h>// defines QApplication
 2  #include <qpushbutton.h>// defines QPushButton
 3  int main( int argc, char *argv[] )
 4  {
 5      QApplication app( argc, argv ); // create application object
 6
 7      QPushButton  hello( "Hello, world!" ); // create a push button
 8      app.setMainWidget( &hello ); // define as main widget
 9      app.connect( &hello, SIGNAL(clicked()),// clicking the button
10   &app, SLOT(quit()) ); //   quits the application
11      hello.show(); // show button
12      return app.exec(); // run main event loop
13  }
14
```

2.3.6 Tcl/Tk

Nosso último exemplo é em Tcl/Tk. Como se pode ver pelo tamanho do programa, o Tk oferece uma interface de muito alto nível. Para rodá-lo em Linux, basta usar o comando

```
% wish hello-Tk
```

```
 1  #!/usr/local/bin/wish8.0
 2  #
 3  # hello-Tk
 4  #
 5
 6  # define o título da janela
 7  wm title . "Hello, world"
 8
 9  # cria um botão que quando pressionado executa o comando "exit"
10  button .hello -text "Hello, world!" -command exit
11
12  # empacota o botão
13  pack .hello
14
15  # o laço de eventos é implícito (e tudo mais :-)
```

3 Documentação/URLs

Documentação sobre a Xlib, Xt e Xaw pode ser encontrada na forma de páginas de manual normalmente instalados em sistemas com o X. Manuais em PostScript® podem ser obtidos via FTP anônimo em:

- `ftp://ftp.x.org/pub/R6.4/xc/docs/hardcopy/X11/`
- `ftp://ftp.x.org/pub/R6.4/xc/docs/hardcopy/Xt/`
- `ftp://ftp.x.org/pub/R6.4/xc/docs/hardcopy/Xaw/`

Muitas referências sobre sobre Motif pode ser encontrada em a partir dos seguintes endereços na WWW:

- Home-page do projeto LessTif: `http://www.lesstif.org/`
- The Motif Developer: `http://www.motifzone.com/tmd`
- Motif FAQ: `http://www.rahul.net/kenton/mfaq.html`

Documentação sobre Gtk, assim como as bibliotecas, podem ser encontradas na própria home-page do Gtk (há ponteiros para tópicos relacionados, como o Glade e GNOME):

- `http://www.gtk.org/`

Documentação sobre Qt, assim como suas bibliotecas, podem ser encontradas no site da Troll Tech:

- `http://www.trolltech.com/`

Sobre Tcl/Tk a referência básica é o livro de John Ousterhout (veja na bibliografia). Além dele é possível encontrar muitas referências a partir da página WWW da Scriptics (uma empresa especializada em Tcl/Tk, pertencente ao próprio Ousterhout):

- `http://www.scriptics.com/`

Além dessas referências, muitas outras sobre X Window podem ser encontradas a partir do Yahoo:

- `http://dir.yahoo.com/Computers_and_Internet/Software/Operating_Systems/X_Window_System/`

Referências Bibliográficas

[1] Scheifler, Robert; Gettys, Jim. The X Window System. ACM Transactions on Graphics, New York, ACM, 5 (2): 79-109, april, 1986.

[2] Balkovich, E.; Lerman, S.; Parmelee, R. P. Computing in higher education: The MIT Athena experience. Communications of the ACM, New York, ACM, 28 (11): 1214-1224, novembro, 1985.

[3] Liskov, B.; Scheifler, Robert. Guardians And Actions: Linguistic support for robust, distributed programs. ACM Transactions on Progamming Languages Systems, New York, ACM, 5 (3): 381-404, julho, 1983.

[4] Gettys, James & Scheifler, Robert. Xlib — C language X interface. Maynard, Digital Equipment Corporation/X Consortium, 1994.

[5] McCormack, Joel; Asente, Paul; Swick, Ralph; Converse, Donna. X Toolkit Intrinsics — C Language Interface. Digital Equipment Corporation/X Consortium, 1994.

[6] Peterson, Chris D. Athena Widget Set — C Language Interface. Maynard, Digital Equipment Corporation/X Consortium, 1994.

[7] Open Group Desktop Technologies — Motif. Disponível via WWW em <http://www.camb.opengroup.org/tech/desktop/motif/>.

[8] Ousterhout, John. Tcl and the Tk Toolkit Reading, Addison-Wesley, 1994.

[9] ZHAO, T. C.; Overmars, Mark. Forms Library: A Graphical User Interface for X. Disponível via WWW em <http://bragg.phys.uwm.edu/xforms>.

[10] Mattis, Peter; Kimball, Spencer; MacDonald, Josh. The GIMP toolkit. Disponível via WWW em <http://www.gtk.org>.

[11] Scheifler, Robert. X Window System Protocol. Cambridge, X Consortium, 1994.

[12] The Open Group. X Window System, Version 11, Release 6.4: Release Notes. Open Group, 1998.

[13] Tanembaum, Andrew S. Computer Networks. Upper Saddle River, Prentice Hall, 1996.

[14] Jones, Oliver. Introduction to the X Window System Englewood Cliffs, Prentice Hall, 1990.

[15] Rosenthal, David; Marks, Stuart W. Inter-Client Communication Conventions Manual. Sun Microsystems/X Consortium, 1994.

[16] The F? Virtual Window Manager home page. Disponível via WWW em <http://www.fvwm.org>

[17] Ghostview and GSview.Disponível via WWW em <http://www.cs.wisc.edu/~ghost>

[18] Lesstif. Disponível via WWW em <http://www.lesstif.org>.

[19] Taylor, Lloyd. Client/Server Frequently Asked Questions. Disponível via WWW em <http://www.abs.net/~lloyd/csfaq.txt>.

[20] Tanembaum, Andrew S. Sistemas Operacionais Modernos. Rio de Janeiro, Prentice Hall do Brasil, 1995.