



DEPARTAMENTO DE
ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES

Servidor de X
Estudo,
Projecto e
Implementação

FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Rua dos Bragas, 4099 Porto Codex – PORTUGAL

004(043)
COSa/SER

561

Elect.

António Manuel Cardoso da Costa

UNIVERSIDADE DO PORTO
Faculdade de Engenharia
BIBLIOTECA M
N.º 20558-07.
CDU 604(043)
Data 23 / 5 / 1995
N.º 34601

Servidor de X
Estudo,
Projecto e
Implementação

Faculdade de Engenharia da Universidade do Porto

Departamento de
Engenharia Electrotécnica e de Computadores

Fevereiro 1991

043M
C 87.5
2x. 3

Dissertação realizada sob a supervisão de

Tese submetida para satisfação parcial
dos requisitos do programa de mestrado em
Engenharia Electrotécnica e de Computadores

(Informática Industrial)

Dissertação realizada sob a supervisão de

Fernando Nunes Ferreira

Professor Associado do Departamento de Engenharia Electrotécnica e de Computadores
da

Faculdade de Engenharia da Universidade do Porto

Índice

1	Introdução	1
2	Objetivos	2
3	Justificativa	3
4	Revisão Bibliográfica	4
5	Metodologia	5
6	Resultados e Discussão	6
7	Conclusão	7
8	Bibliografia	8
9	Apêndice	9
10	Referências	10
11	Índice	11
12	Índice	12
13	Índice	13
14	Índice	14
15	Índice	15
16	Índice	16
17	Índice	17
18	Índice	18
19	Índice	19
20	Índice	20
21	Índice	21
22	Índice	22
23	Índice	23
24	Índice	24
25	Índice	25
26	Índice	26
27	Índice	27
28	Índice	28
29	Índice	29
30	Índice	30
31	Índice	31
32	Índice	32
33	Índice	33
34	Índice	34
35	Índice	35
36	Índice	36
37	Índice	37
38	Índice	38
39	Índice	39
40	Índice	40
41	Índice	41
42	Índice	42
43	Índice	43
44	Índice	44
45	Índice	45
46	Índice	46
47	Índice	47
48	Índice	48
49	Índice	49
50	Índice	50
51	Índice	51
52	Índice	52
53	Índice	53
54	Índice	54
55	Índice	55
56	Índice	56
57	Índice	57
58	Índice	58
59	Índice	59
60	Índice	60
61	Índice	61
62	Índice	62
63	Índice	63
64	Índice	64
65	Índice	65
66	Índice	66
67	Índice	67
68	Índice	68
69	Índice	69
70	Índice	70
71	Índice	71
72	Índice	72
73	Índice	73
74	Índice	74
75	Índice	75
76	Índice	76
77	Índice	77
78	Índice	78
79	Índice	79
80	Índice	80
81	Índice	81
82	Índice	82
83	Índice	83
84	Índice	84
85	Índice	85
86	Índice	86
87	Índice	87
88	Índice	88
89	Índice	89
90	Índice	90
91	Índice	91
92	Índice	92
93	Índice	93
94	Índice	94
95	Índice	95
96	Índice	96
97	Índice	97
98	Índice	98
99	Índice	99
100	Índice	100

Agradecimentos

- Ao Professor Nunes Ferreira, especialmente na revisão da tese.
- Ao Eng. João Paulo Pereira, durante a realização do trabalho.
- A minha mulher, pela paciência e apoio demonstrados...

I. Índice

I.	Índice	5
II.	Introdução	8
1.	Apresentação.....	8
2.	Perspectiva.....	9
III.	Software Gráfico.....	11
1.	Introdução.....	11
2.	Sistemas de Janelas	14
1.	O sistema do Apple Macintosh.....	15
A.	Introdução.....	15
B.	Breve história do <i>Desktop Computing</i>	17
C.	Arquitectura	29
D.	Modelo Gráfico	30
E.	Gestão de Janelas	31
F.	Interacção	32
G.	Conclusão.....	36
2.	O sistema MS-Windows.....	37
A.	Introdução.....	37
B.	Arquitectura	39
C.	Modelo Gráfico	40
D.	Gestão de Janelas	40
E.	Interacção	41
F.	Conclusão.....	41
3.	O sistema NeWS.....	42
A.	Introdução.....	42
B.	Arquitectura	42

C.	Modelo Gráfico	44
D.	Gestão de Janelas	45
E.	Interacção	46
F.	Conclusão.....	47
4.	O sistema X Windows.....	50
A.	Introdução.....	50
B.	Arquitectura	50
C.	Modelo Gráfico	55
D.	Gestão de Janelas	58
E.	Interacção	60
F.	Conclusão.....	62
IV.	Visão detalhada do Servidor X.....	64
1.	Introdução.....	64
2.	A camada DIX.....	67
3.	A camada OS.....	67
4.	A camada DDX.....	68
1.	<i>Input</i>	69
A.	Processando Acontecimentos	69
B.	Notificando o DIX	69
C.	Dispositivos de entrada	70
D.	Inicialização	70
E.	Teclado e códigos de teclas	70
2.	Ecrãs	70
A.	Requisitos fundamentais.....	71
B.	Visuais, Profundidades e Classes.....	72
C.	Mapas de cores	72
D.	Caracteres.....	73
E.	Desenháveis	73
F.	Janelas	74
G.	Contextos Gráficos	75

3.	Primitivas de Desenho	76
A.	Rotinas Spans	79
B.	Rotinas Glyph	79
C.	Rotina PushPixels	80
4.	Rotinas diversas	80
5.	Conclusões.....	80
V.	Experiências de Implementação de um servidor X para MS-DOS.....	82
1.	Introdução.....	82
2.	Arquitectura	82
3.	Experiência.....	84
4.	Conclusões.....	88
VI.	Servidor X para carta gráfica do Projecto ESTIMULO	90
1.	Introdução.....	90
2.	O servidor X.....	92
3.	Arquitectura	94
4.	Implementação	96
1.	Mecanismos básicos	96
2.	Comunicação - <i>Byte stream</i>	99
3.	Sistema operativo	104
4.	Rotinas gráficas	107
5.	Janela vídeo.....	111
5.	Análise.....	114
VII.	Conclusão	117
VIII.	Bibliografia.....	119

II. Introdução

1. Apresentação

A expressão software gráfico de base designa, de um modo muito genérico, todo o software que gera e manipula gráficos, e que, em geral, está funcionalmente muito próximo do hardware. São exemplos de software gráfico de base o algoritmo de **Bresenham**[BRES] para desenho de segmentos de recta, o algoritmo de clipping de **Cohen-Sutherland**[SUTH1] e o algoritmo **Z-Buffer** para remoção de faces e partes de faces escondidas e sombreamento de faces visíveis.

O desenvolvimento de hardware gráfico acarreta, normalmente, a implementação de uma camada de software que permite, assim, utilizar eficientemente as potencialidades desse hardware, de tal modo que qualquer utilizador o possa usar sem ter que conhecer, em demasia, as suas particularidades e idiossincrasias. Por exemplo, é vulgar, no domínio dos microcomputadores, ver publicitado um novo produto gráfico¹ especificando quais os **drivers** que são fornecidos conjuntamente e as aplicações que podem ser imediatamente executadas por seu intermédio, ou então bibliotecas de rotinas escritas numa qualquer linguagem de alto nível², às quais o utilizador só tem que adaptar os seus programas para que tudo fique funcional.

Esta tese de mestrado descreve o estudo, projecto e a implementação de uma camada de software para uma nova carta gráfica[JPP], com características inovadoras, que será posteriormente integrada numa estação gráfica funcionando em **UNIX**. A utilidade deste software³ será a de compatibilizar aplicações já existentes com o novo hardware.

¹geralmente cartas gráficas.

²a linguagem mais utilizada é C.

³tecnicamente um servidor para X Windows.

Este trabalho baseou-se no código desenvolvido em linguagem **C** pelos criadores do sistema **X Windows**[JONE] e na documentação respectiva[ANG1] [ANG2] [ROSE2], código ao qual foram feitas todas as alterações necessárias, de acordo com as características do hardware.

2. Perspectiva

Este trabalho está dividido em vários capítulos, os quais começam por uma abordagem geral do tema software gráfico até se chegar à tarefa específica de implementação, para melhor explanação dos assuntos envolvidos.

Assim, o capítulo III refere o que pode ser definido como software gráfico de base e apresenta uma perspectiva histórica deste tema. Em seguida, aborda-se o software gráfico mais sofisticado, sobretudo o que tem sido desenvolvido nos últimos 10 anos, baseado em conceitos tais como janelas e facilidade de uso⁴. Neste capítulo são descritos os sistemas de janelas mais vulgares actualmente, desde aqueles que funcionam em microcomputadores até aqueles que já trabalham em redes distribuídas de computadores e estações gráficas.

No capítulo IV refere-se com pormenor o sistema de janelas X Windows, a sua arquitectura e detalhes fundamentais. São analisadas as implicações do hardware da carta gráfica sobre o código fonte da versão 11 (*Release 3*) do servidor, problemas de implementação, soluções possíveis e aspectos inovadores a considerar.

O capítulo V descreve algumas experiências de implementação do servidor X Windows num microcomputador PC AT, equipado com uma carta gráfica monocromática de muito alta definição e uma carta de rede vulgar. Neste capítulo é descrita a arquitectura deste sistema e são analisadas possíveis estratégias de implementação.

No capítulo VI faz-se uma breve referência ao projecto **ESTIMULO**, dando-se particular ênfase à parte relativa à carta gráfica que está ser desenvolvida no seu âmbito. Para uma descrição pormenorizada do trabalho desenvolvido, o

⁴em inglês **user-friendliness**.

capítulo V subdivide-se em várias partes, de modo a contemplar a análise e a arquitectura do sistema gráfico e a implementação.

Finalmente, o capítulo VII descreve a situação actual deste trabalho e as conclusões que dele se podem retirar.

1. Introdução

III. Software Gráfico

1. Introdução

Com o advento dos computadores nos anos 40, o mundo científico ficou inundado com números e tabelas numéricas como nunca havia acontecido até então.

Os gráficos de computador evoluíram então como um meio alternativo de visualizar e interpretar essa imensa quantidade de informação. Assim, os computadores eram programados para converter os gigantescos volumes de informação numérica em histogramas, **pie charts**, gráficos de barras e outros tipos de representações gráficas. O que era na realidade novo neste modo tradicional de apresentar a informação numérica era a capacidade de aliar rapidez, versatilidade e precisão, fornecidas pelo computador. Este tipo de utilização foi-se refinando durante os anos 50, aparecendo na altura alguns sistemas capazes de produzir gráficos em **teletypes** e impressoras de linha, todavia sem qualquer vestígio de interacção dinâmica com os utilizadores[HARR].

Em 1963, Ivan Sutherland apresentou o seu sistema **Sketchpad**, cujo objectivo era a comunicação através de interacção⁵[SUTH2].

Para tal, Sutherland imaginava um utilizador sentado em frente de um terminal, interagindo dinamicamente com os gráficos desenhados pelo computador através de um **light pen** e de uma lista de símbolos (**menu**). Assim que esta ideia foi totalmente compreendida e assimilada, não demorou muito para que os grandes fabricantes de computadores ficassem interessados. O grande problema da altura era o estado incipiente do hardware.

⁵cujo título era "A Man-Machine Communication System".

No início dos anos 60, o único tipo de dispositivo de visualização existente era o ecrã de refrescamento por vectores (**vector refresh crt**). Neste dispositivo, um feixe de electrões desenha figuras no ecrã como conjuntos de segmentos de recta (vectores). Dado que o fósforo do ecrã só permanecia excitado durante uma fracção de segundo, era necessário redesenhar a imagem pelo menos 30 vezes por segundo para se obter uma imagem estável sem cintilação. A grande desvantagem deste tipo de tecnologia residia na sua complexidade do seu processamento gráfico, preço e número limitado de vectores que conseguia desenhar, dados os elevados requisitos de memória⁶.

Este problema foi parcialmente resolvido, mais tarde, com o aparecimento do ecrã de memória (**direct view storage tube**), que possuía um fósforo de elevada persistência, o que eliminava a necessidade de memória e permitia o desenho de figuras bastante mais complexas, embora o baixo nível de interacção gráfica e a ausência de cores fossem os seus grandes inconvenientes.

No fim dos anos 60 aumentou enormemente o potencial comercial da Computação Gráfica, ao qual a Microelectrónica não foi indiferente[HARR]. Os microprocessadores e as memórias de silício permitiram a divulgação dos terminais **raster** baseados em monitores de televisão. Este novo tipo de tecnologia permitiu o aparecimento de dispositivos não excessivamente caros, e pela primeira vez, a utilização ilimitada da cor. A imagem, num terminal raster, é construída como um arranjo rectangular de pontos⁷, cujas dimensões determinam a definição do terminal⁸.

Infelizmente a diversidade e a rapidez de desenvolvimento de novo e sofisticado hardware, nos últimos 20 anos, não foi acompanhada de um crescimento paralelo semelhante por parte do software gráfico. Assim, tornou-se vulgar nos fabricantes de hardware fornecer igualmente software para utilizar com os seus produtos. Mas no fundo o que todos estes fabricantes faziam não era senão reinventar a roda, na medida em que todos forneciam software com uma funcionalidade semelhante, mas desenvolvido em completo isolamento dos

⁶é necessário transformar em tempo real uma lista de vectores 3D.

⁷estes pontos são designados **pixels**.

⁸na altura, era vulgar 256 por 256 pontos com 1 bit de informação de cor em cada ponto. Actualmente, tem-se 1024x1024 com 24 bits por ponto.

restantes fabricantes, sem terem em conta as dificuldades que tal estratégia causava aos utilizadores de produtos de diferentes origens.

A partir de meados dos anos 70, toda uma nova gama de software gráfico começou a surgir, a maior parte do qual se auto-intitulava **machine-independent**, isto é, capaz de funcionar em qualquer computador com um esforço mínimo de conversão. Desde então, tem-se assistido a uma explosão no desenvolvimento de software gráfico sofisticado, capaz de interligar máquinas de capacidades tão diferenciadas como um microcomputador ou uma estação gráfica de alto desempenho, em ambientes de elevada abstracção e funcionalidade.

Com o aparecimento dos terminais raster com definições da ordem dos 512 por 512 pixel, monocromáticos ou com 256 cores, começaram a surgir programas que, para além dos vulgares modos de visualização alfanuméricos, apresentavam no ecrã zonas, geralmente rectangulares, de afixação de gráficos, imagens, etc. A estas zonas autónomas de afixação de informação desde o início se chamou **janelas**⁹.

Tornava-se assim possível ter, no mesmo ecrã, diversas zonas de interacção ou de troca de informação, o que, devido à sua natureza gráfica, facilitava a utilização e melhorava a aprendizagem (Figura 1).

2. Bases das Janelas

⁹em inglês **window**.

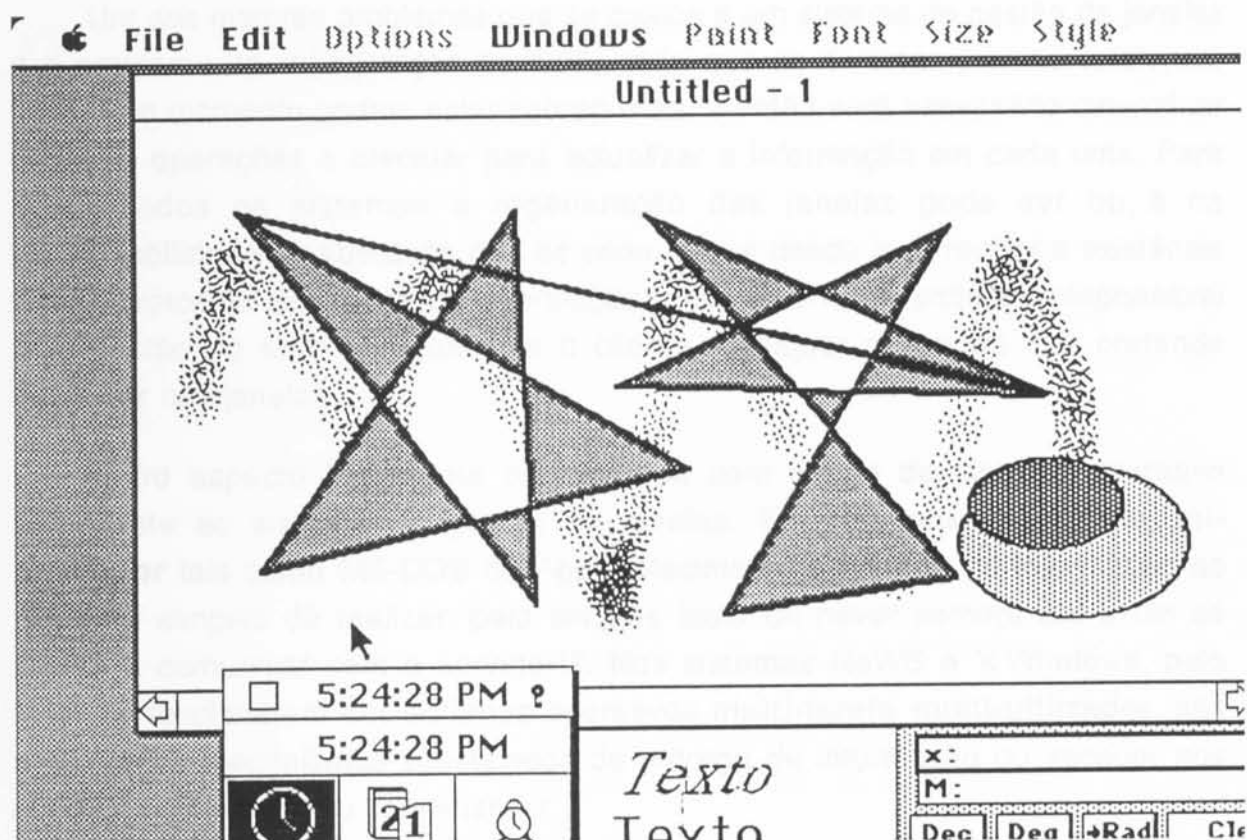


Figura 1

Macintosh - Exemplo de computador com sistema de janelas

2. Sistemas de Janelas

Os sistemas de janelas, vulgarizados desde o início dos anos 80, apresentam actualmente um grau de sofisticação bastante elevado, e embora as diferenças do ponto de vista gráfico sejam aparentemente menores, existem diferenças de implementação e de filosofia realmente importantes. Todos estes sistemas apresentam um conjunto básico de mecanismos que servem para criar, fechar, mover, destruir janelas, escrever texto, desenhar gráficos 2D, etc[GETT].

Um dos maiores problemas que se coloca a um sistema de gestão de janelas é o problema da regeneração do ecrã, dado que as diversas janelas existentes num dado momento podem estar sobrepostas, e então será necessário determinar quais as operações a efectuar para actualizar a informação em cada uma. Para quase todos os sistemas a regeneração das janelas pode ser ou é da responsabilidade da aplicação que as criou, o que desde logo requer a existência de um protocolo para troca de informação entre o servidor, entidade responsável pela gestão do terminal gráfico, e o cliente, qualquer programa que pretende desenhar nas janelas.

Outro aspecto importante relaciona-se com o tipo de sistema operativo subjacente ao sistema de gestão de janelas. Em sistemas **uni-tarefa uni-utilizador** tais como MS-DOS ou Apple Macintosh, a entrega de informação ao cliente é simples de realizar, pelo simples facto de haver sempre um e um só cliente a comunicar com o servidor¹⁰. Nos sistemas NeWS e X Windows, pelo facto de funcionarem em sistemas operativos **multi-tarefa multi-utilizador**, são necessários mecanismos sofisticados de entrega de informação do servidor aos clientes, do tipo filas ou semelhantes.

De todos os sistemas de gestão de janelas existentes, MS-Windows, Macintosh, Smalltalk, Andrew, Sun Windows, X Windows, NeWS e outros, alguns serão de seguida abordados, analisando-se a sua tecnologia e as suas características.

1. O sistema do Apple Macintosh

A. Introdução

O Macintosh é claramente um produto com uma linhagem notável, quer em termos de hardware, quer em termos de software. O seu sistema operativo foi

¹⁰embora o MS Windows possa ser **multi-tarefa**, os mecanismos de entrega de informação do servidor para os clientes são relativamente simples.

desenvolvido à volta da ideia de usar mecanismos gráficos sofisticados para simular fenómenos do mundo real, tais como **folders**¹¹, lápis, papel, etc. Esta ideia, geralmente denominada **desktop metaphor**¹², foi refinada ao longo de muitos anos de investigação nos laboratórios da Xerox, no Palo Alto Research Center (PARC).

Em fins de 1977, Alan C. Kay[KAY1], trabalhando na altura na Xerox PARC, escreveu:

"Em termos ideais, o computador pessoal deverá ser construído de tal forma que pessoas de todas as idades e estilos de vida poderão facilmente adaptar as potencialidades do computador às suas necessidades... Se o computador for verdadeiramente pessoal, adultos e crianças deverão conseguir que ele desempenhe tarefas úteis sem recorrerem à ajuda de peritos. As tarefas simples deverão ser simples, e as complexas deverão ser possíveis."

Nessa altura, o computador pessoal era pouco mais do que um conceito, embora hardware relativamente sofisticado, e não necessariamente caro, já existisse sob a forma do **Apple I** e do **Altair**.

No seu artigo de 1977 publicado na revista *Scientific American*, Kay descrevia a sua visão futurista de uma máquina, que embora do tamanho de um bloco de notas, possuía muita memória de massa, gráficos **bitmapped** de elevada resolução e software sofisticado orientado para a simulação de fenómenos reais. Ele antevia esta máquina como uma ferramenta universal que poderia simular, parecer ou mesmo actuar segundo uma perspectiva humana, quer ela fosse de um arquitecto, de um músico ou de um mecânico. Esta máquina, designada **Dynabook**, só em 1977 pode ser aproximadamente realizada, porque o software, até então, exigia hardware que excedia fisicamente as dimensões de uma tal máquina.

O trabalho de Kay na Xerox PARC, embora não produzisse uma máquina de acordo com a sua visão, preparou o terreno para alguns desenvolvimentos muito importantes na área dos computadores pessoais, incluindo as bases da tecnologia do Apple Lisa e do Macintosh. Analisemos então algumas das ideias e das

¹¹uma tradução possível será arquivo.

¹²metáfora ou paradigma do **desktop**.

peessoas que precederam o artigo de Kay, e que iniciaram assim a história do **desktop computing**.

B. Breve história do *Desktop Computing*

Simula e Flex - Em meados dos anos 60, dois investigadores do Norwegian Computing Center, em Oslo, desenvolveram uma linguagem a que chamaram Simula. Em contraste com outras linguagens, nomeadamente o Fortran, orientado para o cálculo científico, Simula foi especialmente criada para se desenvolverem programas que pudessem modelar fenómenos do mundo real. A contribuição mais importante do Simula foi a sua tratabilidade - era fácil escrever programas em Simula parecidos com a linguagem natural do problema, em contraste com as linguagens convencionais, que geravam geralmente programas obscuros e difíceis de entender.

No fim dos anos 60, Alan Kay e outro investigador colaboraram na construção de um computador pessoal que suportasse directamente gráficos e uma linguagem orientada para simulação. Este sistema, desenvolvido na Universidade de Utah, foi denominado FLEX, e as suas características mais importantes eram a simplicidade e a facilidade de uso. Kay baseou o software do FLEX nos aspectos mais atraentes do Simula, tendo chegado à conclusão que a simulação era o objectivo principal deste computador pessoal "ideal". A sua justificação era a de que um computador construído para a simulação poderia parecer e actuar de um modo mais familiar, e por consequência, ser mais facilmente dominado por um utilizador normal.

O hardware que suportava o FLEX era demasiado caro, em termos de anos 60, para ser considerado "pessoal", e o ambiente de software era, segundo as próprias palavras de Kay, "insuficientemente versátil para ser útil a uma vasta gama de utilizadores não-especializados". Ainda assim, o projecto FLEX foi importante na medida em que concretizava, na altura, uma noção radical de computador que poderia ser utilizado em actividades normais por indivíduos com mais de seis anos e que poderia ser facilmente transportado.

O nascimento do Rato - Na mesma altura, Douglas Englebart trabalhava no Stanford Research Institute (SRI) num projecto cujo objectivo era o aumento das

capacidades intelectuais humanas através dos computadores. Este projecto aparentemente muito utópico produziu duas inovações que influenciaram fortemente o trabalho de Kay e de outros. Uma foi o **rato**, uma pequena caixa com rodas que estava electronicamente ligada a um cursor no ecrã. A possibilidade de apontar desde há muito que tinha sido reconhecida pelos investigadores em design de computadores como uma alternativa desejável ao teclado, na medida em que era muito mais directa e menos ambígua. Embora as **light pens** permitissem apontar um local no ecrã e fazer o computador reconhecer essa acção, possuíam a desagradável característica de perderem a sua posição quando pousadas. O processo de abandonar o teclado, pegar na light pen e apontar era muito descontínuo e passível de cansar o braço, se executado muitas vezes seguidas. Ao invés, o rato permanecia operacional enquanto parado, e não se mostrava mais cansativo de operar do que o teclado. Além disso, não era muito caro o seu fabrico, e diversos estudos de Englebart[ENG] mostraram que o seu desempenho era superior do ponto de vista de produtividade.

Janelas - A segunda inovação de Englebart foi o uso de janelas, que dividiam o ecrã em zonas rectangulares que visualizavam diversos tipos de informação. O utilizador poderia apontar para uma qualquer janela através do rato para executar acções muito diversas. Este sistema baseava-se em dois importantes postulados sobre o modo como as pessoas trabalham com computadores:

- 1º as pessoas necessitam frequentemente de vários tipos de informação em paralelo¹³, afim de executarem eficientemente o seu trabalho.
- 2º é muito mais fácil apontar uma posição no ecrã do que especificá-la através de uma sequência de teclas - a capacidade de apontar permite a manipulação directa de objectos no ecrã em oposição à especificação indirecta pelo teclado.

Estes dois postulados foram extremamente importantes no desenvolvimento do **desktop metaphor**, como veremos adiante.

PARC e a criação do Smalltalk - Em 1971, Alan Kay transferiu-se para o Xerox PARC e fundou o Learning Research Group como um veículo para a pesquisa e desenvolvimento do projecto Dynabook. Embora o Dynabook fosse

¹³simultaneamente.

concebido como uma máquina para os anos 80, diversos protótipos seriam construídos em hardware intermédio à medida que este fosse mais pequeno e mais acessível. Kay possuía três importantes armas conceptuais no seu ataque à realização do computador pessoal: a filosofia da facilidade de uso e tratabilidade do Simula e do FLEX, janelas como dispositivo de visualização e o rato como método preferencial de acção entrada-saída.

Começou a explorar então no sentido de descobrir modos de funcionamento do computador que fossem compatíveis com a designada **physical metaphor**, isto é, fazer com que o computador se parecesse e actuasse como os objectos familiares da vida corrente, e assim fosse facilmente apreendido através do uso de intuição. A linguagem **Smalltalk** foi criada pouco após o aparecimento do LRG com o fim de fornecer o ambiente de software necessário para o seu trabalho.

Melhores Janelas - Alan Kay desenvolveu o conceito de janelas sobreponíveis¹⁴ como uma resposta aquilo que outro investigador havia denominado o dilema do preenchimento. Para muitos programas de computador, a sua execução acarreta a alocação de todos os recursos da máquina e impede o utilizador de usar outras capacidades do sistema. Isto continua a ser verdade, mesmo nos dias de hoje, o que quer dizer que a solução deste problema não é fácil de encontrar. Por exemplo, enquanto se escreve um memorando, pode ser útil visualizar gráficos noutra qualquer parte do ecrã; em muitos sistemas actuais, o uso do processador de texto exclui automaticamente o uso de outros programas tais como folha de cálculo, etc.

O trabalho de Englebart havia mostrado que as janelas eram um melhoramento irrefutável relativamente ao modo ecrã simples, porque permitiam actividades em simultâneo no mesmo ecrã, e assim reduziam o problema do preenchimento. No protótipo do SRI, as janelas eram pré-definidas, fixas e não se sobrepunham no ecrã, não sendo igualmente alteráveis pelo utilizador.

O paradigma de Kay apresentava uma melhoria notável sobre o sistema do SRI através do uso de janelas sobreponíveis, em que só uma das janelas estava "no topo" ou era completamente visível num dado instante, e esta janela podia-se sobrepôr ou obscurecer as restantes. Assim sendo, é como que se estas janelas existissem em planos possuindo uma profundidade espacial relativamente ao

¹⁴em inglês **overlapping windows**.

utilizador, o que permite representar mais informação, assim como uma pilha de folhas pode conter mais informação do que uma só folha.

As janelas funcionam melhor em conjugação com um dispositivo apontador, porque então o utilizador pode apontar para uma dada janela e facilmente torná-la visível. Se estas janelas puderem ser movidas, a informação que contêm pode ser convenientemente disposta de modo a tornar-se mais rentável para o utilizador; estas janelas iniciais, porém, eram fixas e colocadas em posição através do programa que as criava.

Kay achava que as janelas e o rato eram essenciais ao seu computador pessoal sofisticado pela necessidade de paralelismo e simultaneidade própria de uma ferramenta deste tipo, na medida em que as pessoas estavam acostumadas a pensar em várias coisas ao mesmo tempo enquanto trabalhavam, mudando de umas para outras em função das circunstâncias. O grande problema da altura era desenvolver hardware apropriado para testar estas ideias.

Alto, o computador pessoal - A Xerox iniciou o desenvolvimento do computador Alto mais ou menos no início de 1971, na intenção de ser uma máquina pessoal para investigação e estudo dos conceitos desenvolvidos pelo LRG[WADL]. O Alto foi o primeiro computador pessoal a incorporar realmente uma visualização com tecnologia bitmapped. Este tipo de visualização baseia-se numa malha muito densa de pontos, virtualmente indistiguíveis. Cada ponto está activado (produzindo luz) ou desactivado (escuro). Neste mapa de pontos, todos os tipos de informação, quer sejam texto ou objectos gráficos tais como rectângulos ou círculos são tratados como agrupamentos de pontos e devem ser produzidos por software. Em sistemas que não sejam bitmapped, o texto é geralmente produzido por um dispositivo hardware, e o seu estilo e tamanho não pode ser facilmente modificado. O conceito bitmap fornece o máximo de flexibilidade, na medida em que qualquer tipo de imagem pode ser sintetizada a partir de pontos. Especialmente no que se refere a texto, que pode ser representado numa variedade de tipos, estilos e tamanhos, limitados apenas pela versatilidade do software.

A desvantagem está no facto do bitmap necessitar que o software cumpra rapidamente a sua tarefa, o que pode retardar o funcionamento de outros programas em execução na máquina. Assim sendo, os sistemas bitmapped devem incorporar processadores muito poderosos, ou então, software gráfico de base extremamente eficiente para a máquina possa ter alguma utilidade prática. Por

outro lado, uma vasta quantidade de memória rápida é necessária para armazenar a informação do ecrã. A tecnologia bitmapped é fundamental para qualquer sistema que utilize um rato e janelas, na medida em que estes sistemas são altamente visuais e exigem o desenho de rectângulos para as janelas e formas diversas de cursores em qualquer posição no ecrã. O preço elevado da memória foi a razão principal porque a tecnologia bitmapped não foi aplicada em computadores pessoais até aquela altura; com o preço da memória a baixar, estes sistemas começaram a invadir rapidamente o mercado.

O computador Alto custava aproximadamente 30000 dólares, embora nunca tenha estado comercialmente à venda. Foi construído apenas para investigação, e somente distribuído a universidades e dentro da Xerox. Possuía um rato com três botões e funcionava com um sistema operativo baseado em janelas, executando também um processador de texto denominado **Bravo**, o qual era controlado por combinações de teclas e botões. A maior parte dos computadores Alto continha entre 256 e 512 kbytes, o que no início dos anos 70 era realmente muita memória, e explicava em parte o seu preço elevado.

O processador de texto Bravo merece uma referência breve porque foi a primeira implementação consistente de um processador de texto WYSIWYG¹⁵. O processador de texto Bravo aproveitava a versatilidade do bitmap para apresentar uma imagem muito aproximada do que apareceria quando impresso, incluindo vários tipos de caracteres e tamanhos. Possuía, no entanto, algumas limitações, particularmente a utilização de "modos", o que será explicado adiante. Um outro programa que causou grande impacto em desenvolvimentos posteriores foi o programa de desenho **Draw**, que permitia aos utilizadores desenhar imagens e pintá-las com pincéis electrónicos no ecrã de tecnologia bitmap. Algumas das características deste programa influenciaram a concepção do programa **MacPaint** para o Macintosh.

Smalltalk - O computador Alto forneceu à equipa de Alan Kay no LRG a possibilidade de implementar um protótipo do Dynabook. Em 1972, esta mesma equipa concebeu e implementou a primeira versão da linguagem Smalltalk para servir de fundamento ao software do Dynabook[TESL]. A linguagem Smalltalk, tal como a linguagem Flex, foi largamente influenciada pelos conceitos do Simula, e foi projectada para providenciar um ambiente de programação poderoso, embora

¹⁵what you see is what you get - o que aparece no ecrã é praticamente igual ao que aparece numa impressora.

simples, de tal modo que pessoas de todas as idades pudessem moldar o Dynabook à medida das suas necessidades. A sua base filosófica era a de que um pequeno conjunto de princípios poderia ser aplicado coerentemente para construir sistemas mais elaborados, e que este processo poderia prolongar-se indefinidamente. O desenvolvimento do Smalltalk provavelmente contribuiu, como fenómeno individual, mais do que qualquer outro para o que é hoje em dia conhecido como a **desktop metaphor**. Embora o Smalltalk tenha nascido do desejo de aliar sofisticação e simplicidade, este modo de pensar não foi somente empregue no desenvolvimento da linguagem, mas também no aspecto da interface com o utilizador.

"Modos" - Em 1973, Larry Tesler juntou-se à equipa do LRG e a sua tarefa inicial foi abordar o dilema do preenchimento e a utilização de "modos" no desenvolvimento de software. Ele definia "modos" como sendo estados artificiais criados por um programa cujo único propósito era definir uma interpretação para a informação introduzida pelo utilizador. Por exemplo, alguns processadores de texto possuem um "modo de inserção" comutável - quaisquer caracteres introduzidos são inseridos entre os caracteres existentes no documento. Caso se abandone o "modo de inserção", os caracteres introduzidos substituirão os caracteres existentes¹⁶. Comutar diversas vezes entre os vários modos pode criar alguma confusão, porque às vezes não se sabe bem o que o computador irá fazer de seguida.

A esperança de Tesler no LRG era banir os "modos" para sempre na concepção de sistemas de escritório interactivos. O seu trabalho resultou num paradigma para a definição da interface com o utilizador, que foi incorporada no sistema Smalltalk em 1976, e numa versão mais evoluída, no **Apple Lisa Office System** em 1981. O modelo de Tesler faz uso extensivo do rato, tirando partido da natureza directa e explícita da acção de apontar objectos no ecrã. As suas regras básicas eram:

¹⁶é o "modo de substituição".

- 1ª A selecção precede o comando; o utilizador move o rato para o objecto, quer seja imagem ou texto, que pretende seleccionar, e só depois utiliza os botões do rato para optar definitivamente. Em seguida o utilizador pode escolher dentro de um conjunto de comandos aplicáveis, relacionados com a natureza do objecto seleccionado. Assim não existe nenhuma ambiguidade, pois só os comandos apropriados são mostrados e seleccionáveis.
- 2ª O texto introduzido substitui a selecção; a área seleccionada é visualizada com um caixilho negro à sua volta, e se um carácter for digitado após a selecção ter sido feita, esse carácter substitui a selecção. Digitar caracteres de texto nunca produz um comando, sendo este tipo de comportamento induzido pela interface.

Na interface do Smalltalk-76, os comandos eram escolhidos através de menus **pop-up**, estando um dos três botões do rato dedicado a produzir estes menus, que mais não são do que listas de comandos dentro de uma caixa algures no ecrã, sempre que este botão seja pressionado. O cursor associado ao rato pode ser então movido para o comando desejado, e a acção de libertar o botão sobre o comando escolhido executa-o. Deste modo, os comandos nunca precisam de ser memorizados ou sequer digitados. Alguns dos comandos disponíveis eram **Cut**, **Copy**, **Paste** e **Undo**, os quais podem ser usados para mover texto e imagens. Estes comandos estavam acessíveis por todo o sistema, coerentemente, sempre que eram apropriados.

A interface com o utilizador do Smalltalk-76 era na realidade a síntese de três grandes conceitos:

- 1º janelas sobreponíveis e simulação de fenómenos reais de Alan Kay;
- 2º interfaces sem "modos" de Larry Tesler;
- 3º a linguagem Smalltalk em evolução, construída a partir da aplicação coerente de princípios básicos.

Esta interface foi aplicada no computador Alto, ele também o resultado de uma sinergia que combinava importantes avanços tais como o rato e o bitmap. Como é frequente nestas situações de grande evolução tecnológica, o todo era claramente superior à soma das partes, na medida em que cada elemento inovador reforçava os restantes. A maior parte desta tecnologia apareceu

virtualmente inalterada sete anos mais tarde, nos computadores Lisa e Macintosh. Devido à natureza altamente iterativa do ciclo protótipo-teste-reconstrói do Smalltalk, este fornecia um excelente meio de investigação para as interfaces com o utilizador, o qual resultou numa interface extremamente eficaz e versátil.

O Projecto Star - Em 1976, foi iniciado na Xerox um projecto denominado **Star**, cuja finalidade era produzir uma estação gráfica bitmapped para fins comerciais. O hardware seria baseado no hardware do Alto, embora com um processador mais rápido e mais memória. Tanto o Alto como o Star foram desenvolvidos à volta da rede local **Ethernet**, criada pela Xerox. O projecto veio a beneficiar bastante do projecto Alto, essencialmente orientado para a investigação, e da imensidade de software experimental da comunidade Smalltalk escrito para essa máquina.

A Xerox investiu 30 anos-homem na concepção da interface com o utilizador para o Star[SMIT]. Novamente, a ênfase era dada à consistência, transparência¹⁷ e manipulação directa de objectos através da acção de apontar. Porém, uma nova faceta foi acrescentada pela equipa do Star, que, embora baseada em trabalhos anteriores, trouxe uma dimensão revolucionária ao projecto. O Star assimilou o conceito de Alan Kay da simulação da experiência dia-a-dia e criou uma nova simulação altamente sofisticada do ambiente mais familiar para a audiência prevista para esta máquina, o escritório. Símbolos representando objectos concretos, designados **Icons**¹⁸, foram introduzidos para descrever acções tais como enviar ou receber correio e escrever cartas. Estes símbolos estavam organizados num rectângulo cinzento abrangente, designado **Desktop**. Os ícones podiam ser movimentados sobre este desktop metafórico, e arranjados ao gosto do utilizador. Para seleccionar uma função particular ou para editar um documento, o utilizador escolhia e abria o ícone correspondente apontando com o rato, e esse ícone transformava-se numa janela. O objectivo do Star era, deste modo, conseguir que o utilizador fosse capaz de traduzir directamente os seus conhecimentos anteriores para o mundo do desktop, e assim minimizar o tempo de aprendizagem e a confusão.

Foi o sistema Star que deu origem à **desktop metaphor** tal como ela é hoje em dia entendida e o primeiro produto comercialmente viável após os longos anos

¹⁷no sentido de transparência de utilização, isto é, o computador mostra entidades que são imediatamente assimiláveis.

¹⁸ícones.

de investigação no LRG, embora o seu preço de 18000 dólares o afastasse do alcance de quase todos os potenciais utilizadores.

A tarefa de trazer o sonho de Alan Kay para a realidade concreta do dia-a-dia ainda não tinha sido completamente realizada. O computador Star foi comercializado em 1981, pouco tempo antes de Alan Kay abandonar a Xerox e se transferir para a Atari. Assim, o sonho magnífico do Dynabook continuava a ser pouco mais do que um sonho.

Apple Lisa - Pouco tempo antes da comercialização do Star, Larry Tesler abandonou igualmente a Xerox para ir trabalhar para a Apple Computer, na concepção de um novo produto ultra-secreto. A sua tarefa era a de supervisionar todo o software de aplicação para o Lisa, que era ou pretendia ser um sistema de escritório electrónico de elevado desempenho com ênfase na transparência e na facilidade de uso. A equipa do projecto Lisa beneficiou imenso do trabalho desenvolvido pela comunidade Smalltalk, criando uma máquina com gráficos bitmapped e um rato de um só botão, que, na sua perspectiva, causaria menos confusão do que a versão de vários botões anterior. Muitas das convenções adoptadas para a interface com o utilizador do Smalltalk-76 sobreviveram intactas no Lisa:

- interacção sem "modos"
- janelas sobreponíveis
- menus
- **Cut, Paste e Copy**
- filosofia de aplicação coerente de princípios básicos
- etc.

Para benefício da Apple, uma grande quantidade de esforço foi dispendido no desenvolvimento da interface com o utilizador para a tornar no que é hoje. Cada detalhe da interacção entre o utilizador e o sistema foi cuidadosamente estudado, na tentativa de eliminar situações ambíguas ou de indefinição. Um pouco ao modo do sistema Star, a implementação da interface com o utilizador do Lisa era governada por um documento extenso que definia todos os aspectos e atitudes adoptadas pelos implementadores para cada detalhe de interacção possível. Por exemplo, os leitores de discos amovíveis¹⁹ foram projectados para não permitir a remoção dos discos a não ser com o consentimento do software de

¹⁹em inglês **disk drives**.

sistema, tendo isto sido feito para prevenir a perda imprevista de informação, que poderia ocorrer se o disco fosse retirado enquanto o sistema estivesse a utilizá-lo.

O Apple Lisa reuniu o que havia de melhor na interface com o utilizador do Smalltalk-76 com a filosofia metafórica do Star, a qual recorria a ícones para transmitir informação funcional ao utilizador em vez de palavras. As janelas do Lisa eram dinamicamente ajustáveis em tamanho e posição pelo utilizador, e um formato padrão foi adoptado para a sua representação; a maioria delas permitia deslocamento horizontal ou vertical, avanço de página, assim como mover um indicador que marcava a posição absoluta na área de informação.

Aplicações do Lisa - Com a introdução do Lisa, a Apple anunciou sete aplicações que seriam integradas no ambiente de escritório electrónico:

- 1 - **LisaWrite**, um processador de texto
- 2 - **LisaCalc**, uma folha de cálculo
- 3 - **LisaDraw**, uma aplicação para desenho industrial
- 4 - **LisaGraph**, uma aplicação para geração de gráficos
- 5 - **LisaList**, uma aplicação de base de dados e de gestão de listas
- 6 - **LisaProject**, uma aplicação de gestão de projecto segundo o método PERT
- 7 - **LisaTerminal**, uma aplicação de emulação de terminais DEC VT100 e de telecomunicações.

Todas estas aplicações partilham um conjunto de princípios básicos acerca do que deve fazer o rato, como devem parecer as janelas, entre outros, cujo resultado é um tempo de aprendizagem extremamente pequeno para todas elas. A parte mais difícil da aprendizagem de um novo utilizador acontece então só durante a primeira aplicação; em seguida, o conhecimento adquirido pode ser aplicado com sucesso à grande maioria das situações que ocorram com outras aplicações. Todos estes programas apresentam um grau elevado de integração de funcionalidades como nunca tinha acontecido até à altura, e muita desta integração deveu-se, sem qualquer dúvida, à filosofia integrada do sistema Smalltalk, no qual sistema operativo e aplicações se tornam virtualmente indistinguíveis.

Embora aplaudido como um sucesso tecnológico, o Apple Lisa apresentava dois problemas muito sérios na altura da sua comercialização: software de

qualidade duvidosa²⁰ e um preço elevado²¹. Dado que o preço estava fora do alcance do utilizador caseiro, o qual seria o principal beneficiário da sua tecnologia, as empresas de desenvolvimento de software, tendo em conta o tempo necessário para transportar as aplicações para o ambiente complexo do Lisa, decidiram aguardar para ver, o que ainda isolou mais esta máquina.

Um computador de ligar à tomada - Na altura em que o projecto Lisa foi iniciado, a Apple começou igualmente a traçar planos para construir um computador pessoal de ligar à tomada, que deveria ser transportável, barato e fácil de usar. Um primeiro protótipo de hardware foi implementado com o processador de 8 bits 6809 da **Motorola**, mas rapidamente abandonado devido a um desempenho insatisfatório. No início de 1981, Steve Jobs, um dos co-fundadores da Apple, incentivou os engenheiros a trabalhar nesta nova máquina no sentido de produzirem os gráficos de elevado nível desenvolvidos pela equipa do Lisa[LEMM]. O protótipo seguinte baseou-se no processador 68000, e o objectivo era obter 70% do desempenho do Lisa apenas por 20% do seu preço, para isso transportando-se o núcleo gráfico **QuickDraw** do Lisa, originalmente desenvolvido por Bill Atkinson. Steve Jobs começou então a manifestar um grande interesse pessoal neste projecto, e conseguiu reunir uma equipa altamente talentosa e motivada para construir uma máquina, que segundo as suas palavras, iria ser "loucamente poderosa"²².

Apple Macintosh - Este projecto, designado Macintosh, foi a primeira tentativa séria, na história dos computadores pessoais, de trazer a tecnologia do desktop para o mercado dos utilizadores normais[WILL]. A Apple estava, de certo modo, mais apta a realizar este objectivo, por várias razões:

1º Tinham desenvolvido o primeiro computador de utilização simples generalizado, o **Apple II**.

2º Tinham organizado uma empresa cujo objectivo era a de levar os computadores a casa das pessoas, aos escritórios e às escolas de toda a América.

²⁰alguns **bugs** eram catastróficos.

²¹o preço de venda inicial, incluindo aplicações, era aproximadamente 10000 dólares.

²²literalmente *insanely great*.

O Lisa foi uma tentativa falhada e quase poderia ser qualificado como desastre se a Apple não tivesse disso retirado os devidos ensinamentos. Era necessário salvar os 200 homem-ano de investimento em software através da produção de uma máquina acessível, e nessa perspectiva, a decisão de transportar o núcleo gráfico do Lisa directamente para o Macintosh foi fundamental. Por outro lado, o Apple II continuou a ser um sucesso financeiro, e permitiu à Apple o luxo de investir 20 milhões de dólares numa fábrica altamente automatizada e tecnologicamente sofisticada capaz de produzir 1 milhão de Mac's por ano.

O Macintosh foi concebido desde o início para ser uma máquina de produção em massa - sem opções de hardware, conectores de expansão e com um número muito reduzido de circuitos integrados. Estes factores, aliados à elevada quantidade de peritos disponíveis do projecto Lisa, deram ao Macintosh uma vantagem inicial importante. Os alvos da Apple eram os "25 milhões de trabalhadores da informação", pessoas cujos empregos envolviam a manipulação e a comunicação de informação. Steve Jobs antevia cada um possuindo dois aparelhos na sua secretária: um telefone e um Macintosh.

Outros aspectos importantes na concepção do Macintosh foram o de ocupar uma área pequena e de ser suficientemente leve para poder ser facilmente transportado entre casa, escola e escritório. O produto final pesava cerca de 10 kg e ocupava uma área de aproximadamente 400 cm², valores óptimos quando se pensa em colocar um **IBM PC** numa secretária ou em levá-lo para casa.

O software do Macintosh teria necessariamente que ser uma versão reduzida do software do Lisa, dado que este possuía apenas 128 kbytes de memória em comparação com 1 megabyte do Lisa. Apesar disso, as convenções básicas envolvendo janelas, o rato e os menus foram mantidas, incorporando os princípios do Smalltalk-76, embora noutro contexto. A principal diferença entre o sistema operativo do Lisa e o do Macintosh estava na capacidade de executar em multi-tarefa, disponível no Lisa e não existente no Macintosh devido a limitações de memória. Ainda assim, esta diferença era relativamente transparente para a maioria das aplicações, pelo que ninguém teria grandes problemas em transportar software do Lisa para o Macintosh.

Com a introdução do Macintosh, a Apple anunciou a sua nova família de máquinas 32 bit, o que envolveu uma remodelação do Lisa em três novos

modelos, mais baratos e de melhor desempenho, e que podiam partilhar software com o Macintosh.

Terá o Macintosh realizado o sonho de Alan Kay da máquina pessoal de infinitas capacidades[KAY2]? Obviamente, não. O Macintosh, embora tenha trazido a tecnologia bitmap para o domínio público, é apenas um passo mais na direcção do Dynabook.

C. Arquitectura

O Macintosh possui uma arquitectura relativamente simples, e o seu sistema operativo é uni-processo e uni-utilizador, inicialmente devido ao facto de possuir apenas 128 kbytes de memória, ao passo que o Lisa era multi-processo e uni-utilizador, com 1 megabyte de memória. Apesar de ser uni-processo, é possível em certas circunstâncias executar determinadas aplicações enquanto se está a meio de uma outra. Este tipo de aplicações é designado **Desk Accessories**²³, e embora duas aplicações "normais" não possam executar simultaneamente, as aplicações tipo Desk Accessory podem, dado serem tratadas de maneira especial e ocuparem uma zona de memória particular[MAC1] [MAC2] [MAC3].

O sistema operativo está construído sobre um sistema de ficheiros hierárquico. Só este sistema existe de base e toda a informação tem que ser fornecida pelo utilizador; é como se existisse um arquivo com bastantes dossiers vazios e rótulos prontos a usar, mas com uma diferença importante - uma secretária diligente sempre pronta a tomar conta de tudo, apropriadamente designada **Finder**.

O que a maioria dos computadores pessoais fornece em termos de sistema de ficheiros não se afasta em quase nada do Macintosh, isto é, a capacidade de produzir ficheiros e armazená-los em disco. Sistemas operativos "tradicionais", tais como o UNIX e o MS-DOS, fornecem igualmente comandos para copiar, remover e executar outras operações em ficheiros. Este tipo de operação é geralmente a mais difícil de aprender para um utilizador iniciado, porque não tem qualquer semelhança com nada que ele conheça; um conjunto totalmente novo de

²³traduzido por acessórios de secretária.

conceitos e ordens tem que ser aprendido do nada. O Macintosh reduz drasticamente este período de aprendizagem, no que respeita às operações sobre ficheiros, através da integração destas operações no ambiente desktop, o qual é principalmente visual e depende essencialmente de acções como apontar. O manuseamento de ficheiros é assim substituído pela manipulação directa e visual de objectos no ecrã.

O gestor do sistema de ficheiros do Macintosh, designado **File Manager**, está implementado como um sistema organizado ao byte que trata os ficheiros como sequências ordenadas de bytes, como outros sistemas mais antigos. Porém, o Macintosh acrescenta uma novidade que é considerar os ficheiros formados por duas entidades logicamente distintas, denominados **forks**. Cada ficheiro possui um **resource fork**, isto é, um campo de recursos, contendo todos os recursos necessários para ser válido, sobretudo se for uma aplicação, e um **data fork**, um campo de dados que contém a informação propriamente dita. Num ficheiro aplicação, o campo de recursos contém simultaneamente os recursos e o código²⁴.

D. Modelo Gráfico

O modelo gráfico do Macintosh baseia-se no paradigma do **RasterOp**²⁵, e o seu núcleo gráfico foi implementado sobre o núcleo que Bill Atkinson havia desenvolvido para o Lisa, denominado **QuickDraw**. Este núcleo não é um programa que qualquer utilizador possa invocar, mas antes um conjunto muito vasto e versátil de rotinas para realizar tarefas de índole gráfica. Este núcleo reside em memória ROM e permite às aplicações desenhar todas as entidades gráficas que fazem parte, normalmente, da interface com o utilizador, tais como janelas, menus e caixas de diálogo. Para além disso, o QuickDraw fornece o esqueleto de aplicações como o **MacPaint** ou o **MacDraw**, basicamente editores gráficos. Muito do trabalho pesado de manipular gráficos é executado pelo QuickDraw sobre supervisão da aplicação. Este constrói imagens a partir de pontos sobre o bitmap e pode realizar várias transformações sobre essas imagens, tais como translação, rotação e escalamento. Por exemplo, um programa define o seu

²⁴considerado como um recurso para certos fins.

²⁵todas as operações gráficas são realizadas através da manipulação de blocos de pixels com operações booleanas à mistura.

ambiente gráfico relativamente ao QuickDraw através do que é designado um **GrafPort**. Um GrafPort associa uma imagem em bits na memória com um sistema de coordenadas qualquer. Vários GrafPort's podem usar a mesma imagem, e definir diversos sistemas de coordenadas associados a essa imagem. Sempre que um desenho é criado num dado sistema de coordenadas, esse desenho pode ser facilmente movido no ecrã alterando a origem desse sistema de coordenadas relativamente à imagem de bits. Esta acção pode ser realizada através de uma única invocação de uma rotina do QuickDraw, o que permite dar uma ideia das capacidades poderosas e abstractas do QuickDraw.

Todas as rotinas do Toolbox do Macintosh utilizam o QuickDraw para manipular imagens no ecrã²⁶.

E. Gestão de Janelas

A gestão de janelas do Macintosh está a cargo de um gestor capaz de realizar qualquer tarefa com janelas e que, conjuntamente com o QuickDraw, contribui fundamentalmente para dar o aspecto tão peculiar ao desktop do Macintosh. Este gestor realiza um trabalho notável no manuseamento de informação associada às janelas. Ele actua como um intermediário entre as aplicações e o QuickDraw em todas as operações complexas que envolvam manipulação de janelas. O esforço de concepção e de funcionalidade postos neste gestor fazem com que seja relativamente simples para as aplicações obter um estilo de apresentação comum, o que convém realçar que era um dos grandes objectivos da Apple.

Quase todas as aplicações para o Macintosh são estruturadas de um modo muito semelhante ao seu mais alto nível. As acções do teclado, do rato e do disco são dadas a conhecer a uma aplicação através da entidade **Event**. Ou seja, o sistema operativo monitoriza constantemente os três dispositivos (e mais, caso existam) e quando o utilizador executa qualquer acção relevante com qualquer deles, o sistema operativo toma conta dessa ocorrência, armazenando-a num Event. Uma lista ordenada de Events é gerida pelo sistema operativo; qualquer

²⁶o ambiente de software do Macintosh não existiria talvez se não fosse o talento de Bill Atkinson, que conseguiu comprimir o código de 160 kbytes em **Pascal** no Lisa para 20 kbytes em **Assembly** na ROM do Macintosh.

aplicação que necessite de informação de entrada de um dos dispositivos é uma aplicação **Event-Driven**²⁷, isto é, possui, no seu nível mais elevado, um ciclo que repetidamente pergunta ao sistema operativo o que está a acontecer, e depois age em conformidade com isso. É conveniente lembrar que o utilizador pode, na maioria das aplicações, fazer o que bem entender com o rato, desde seleccionar um **Desk Accessory**, tornar outra janela activa, etc. A aplicação geralmente não pode exigir ao utilizador que introduza informação pelo teclado ou execute uma acção especial com o rato, pois tal constituiria um modo²⁸ de funcionamento. As principais excepções são as caixas de alerta, as quais só aparecem quando um erro acontece. Consequentemente, as aplicações têm que estar constantemente prontas para tudo, e serem capazes de lidar com uma gama variada de acontecimentos. O modelo subjacente a este funcionamento é consistente, e existem rotinas de sistema apropriadas para lidar com acontecimentos não directamente relacionados com a aplicação. Por exemplo, se o utilizador pressiona o botão do rato noutra janela que não a correntemente activa, a aplicação receberá a notificação de um acontecimento tipo **Deactivate Event**, o qual indica que a janela corrente deverá tornar-se inactiva, e a outra janela deverá ser redesenhada em frente de todas as outras. Embora o trabalho de refrescar a nova janela activa seja efectuado pelo gestor de janelas, a aplicação tem que detectar esse acontecimento e invocar as rotinas apropriadas.

F. Interacção

Uma característica única do ambiente de software do Macintosh é a de todas as aplicações partilharem o mesmo tipo de convenções no que diz respeito à interface com os utilizadores, o que foi propositadamente intenção da Apple.

O software de sistema do Macintosh está basicamente dividido em duas partes, o sistema operativo e um conjunto de ferramentas para realizar a interface com o utilizador, denominado **User Interface Toolbox**, o qual permite manipular janelas, menus, desenhar bitmaps, etc. Em termos conceptuais, o Toolbox é como uma máquina virtual, no sentido de esconder os detalhes de funções complexas, tais como gestão de janelas, das aplicações. As operações básicas necessárias a

²⁷comandada por acontecimentos.

²⁸a palavra modo deve ser entendida com o significado apresentado anteriormente.

uma aplicação para ser conforme ao standard de interface com o utilizador da Apple estão todas presentes no Toolbox, e tudo o que a aplicação tem de fazer é fornecer as estruturas de controlo de mais alto nível para que tudo funcione.

As diversas partes componentes do User Interface Toolbox são:

- **Gestor de Recursos** - O gestor de recursos permite a criação e a manipulação dos objectos com que uma aplicação trabalha - janelas, menus, texto, etc. O código pode ser compilado separadamente, sem que os objectos tenham que estar definidos, excepto a informação mais geral acerca de cada um. Assim, esses objectos podem ser criados com editores de recursos ou compiladores e adicionados ao código depois da ligação e antes da execução. Por exemplo, se uma alteração tiver que ser feita num título de um menu, ele poderá ser modificado e posteriormente re combinado com o código, em vez de se ter de recompilar todo esse código. Este mecanismo pode acelerar enormemente o processo de desenvolvimento dado que isola a construção da interface com o utilizador da criação do código. A principal motivação para se criar o resource manager foi a de tornar a escrita de programas independente de qualquer linguagem, seja Inglês, Francês ou outra. Uma aplicação deverá armazenar as suas mensagens textuais como recursos; se tiver que ser convertida para outra linguagem, somente o ficheiro de recursos terá que ser modificado, ou seja, traduzido. Isto permite, pelo menos em teoria, a penetração do Macintosh em quaisquer mercados mais rapidamente do que as outras máquinas, desde que as empresas de software adiram ao conceito do recurso.
- **Font Manager** - o gestor de caracteres permite que as aplicações tratem os conjuntos de caracteres como entidades lógicas que podem ser incluídas em recursos. O conjunto especifica o modo como os caracteres devem ser desenhados, o que inclui quer o tamanho quer variações estilísticas. Diversos conjuntos pré-definidos estão disponíveis numa variedade de tamanhos; além disso, o gestor de caracteres é capaz de escalar um conjunto para um dado tamanho caso não exista o tamanho desejado.

- **QuickDraw** - este núcleo gráfico é por muita gente considerado a obra-prima do Macintosh. Com o QuickDraw, uma aplicação pode realizar gráficos muito sofisticados e escrever texto através de simples invocações de rotinas. Existem primitivas para manipular pontos, linhas, rectângulos, rectângulos arredondados, elipses, polígonos e regiões arbitrárias - conjuntos de pontos. Estas figuras elementares podem ser combinadas, preenchidas com padrões e sobrepostas com ou sem clipping. Uma janela é facilmente criada utilizando o conceito **GrafPort**, através da definição de uma região rectangular do ecrã que representará o ecrã lógico da aplicação. Na realidade, é o QuickDraw que permite ao Macintosh possuir o ambiente desktop e uma interface com o utilizador graficamente tão complexos. O gestor de janelas, o gestor de menus e outros invocam o QuickDraw para satisfazer as suas necessidades. Uma característica muito importante do QuickDraw é a sua capacidade de manipular regiões, conjuntos arbitrários de pontos que não correspondem a nenhuma forma elementar. Assim, qualquer forma pode ser definida como uma região e manuseada independentemente. As regiões podem ser facilmente movidas, aumentadas, encolhidas e intersectadas com outras regiões. Outras entidades importantes são o cursor, que define o aspecto que o ponteiro associado ao rato toma, e a caneta, que define o tamanho e o aspecto do que é desenhado no ecrã. Estas são completamente independentes. A propriedade mais importante e desejável do QuickDraw é provavelmente a sua velocidade. O QuickDraw é suficientemente rápido para que as aplicações não precisem de realizar elas próprias os seus gráficos para ganhar tempo, e o MacPaint é um óptimo exemplo dessa estratégia.
- **Gestor de Eventos** - O gestor de acontecimentos permite às aplicações terem conhecimento de acontecimentos causados pelo teclado, pelo rato ou devido à actividade do disco.
- **Gestor de Janelas** - descrito anteriormente.

- **Control Manager** - Este gestor comanda os diversos tipos de controlo que o utilizador pode realizar com o rato: botões, **check boxes**, barras de deslizamento, etc. Estes itens podem ser tratados como recursos, definidos separadamente da aplicação, e usados e manuseados através deste gestor. Ele avisará sempre que o ponteiro do rato estiver dentro de uma área que possa provocar o disparo de uma actividade, e comandará a visualização do controlo de acordo com as suas características.
- **Gestor de Menus** - O gestor de menus trata da parte mais complexa da gestão dos menus **pull-down**, substituindo a aplicação. Os menus, como muitos outros aspectos do desktop, podem ser tratados separadamente como recursos, e isto tanto inclui a barra de menus no topo do ecrã como os itens individuais no menu. Estes recursos são definidos quando a aplicação é invocada e passam a ser controlados pelo gestor de menus. Assim, quando uma qualquer acção do rato é despoletada na zona da barra de menus, a aplicação transfere o controlo para o gestor, que se encarrega de visualizar através de menus pull-down os itens relevantes e determinar qual a escolha feita. De seguida, a aplicação retoma o controlo afim de processar a escolha realizada, etc.
- **TextEdit** - Geralmente existindo em ROM, este bloco fornece a funcionalidade básica das operações de edição **Cut, Copy e Paste**, assim como a selecção de texto e a sua visualização. Todas as operações envolvendo texto são executadas em relação estreita com o QuickDraw, pelo que a tarefa da aplicação fica grandemente aliviada.
- **CoreEdit** - Este bloco fornece operações de edição de texto mais sofisticadas do que o TextEdit, e constitui uma excelente ferramenta para aplicações orientadas para o processamento de texto.
- **Gestor de Diálogos** - Permite a uma aplicação a criação de caixas de diálogo com gráficos e texto, por exemplo, de uma maneira consistente. As caixas de diálogo e as caixas de alerta são recursos que podem ser editados por programas apropriados, e incluídos em tempo de execução pelo gestor de recursos. Este gestor invoca o gestor de janelas para criar e controlar a janela de diálogo, e o QuickDraw para visualizar os ícones e outros objectos gráficos.

- **Scrap Manager** - Este gestor controla a transferência de informação previamente assinalada com Cut, Copy ou Paste entre aplicações. Esta informação é mantida na área de trabalho deste gestor, conhecida em termos de utilizador por **Clipboard**. Diversos tipos de informação podem ser manipulados, tais como texto e gráficos.
- **Desk Manager** - Este gestor permite às aplicações invocar os **Desk Accessories**, mini-aplicações acessíveis a partir do menu Apple. Estas são programadas de tal modo que possam coexistir com as aplicações normais, na medida em que não interferem com o código e os dados dessas aplicações. Quando o utilizador invoca um Desk Accessory, a aplicação requisita os serviços do Desk Manager para o lançar e para controlar o rato durante a sua execução. Esta é a única maneira pela qual diversas aplicações podem ser executadas em concorrência no Macintosh²⁹; normalmente, apenas uma aplicação pode ser carregada e executada num dado instante. Estas aplicações miniaturizadas não são realmente aplicações no sentido exacto da palavra, mas antes um tipo especial de **drivers** de entrada/saída, visto ocuparem áreas de memória que não podem ser utilizadas pelas aplicações propriamente ditas.

G. Conclusão

Desde o seu aparecimento em 1984 que o Macintosh se tornou um sucesso, e novas e muito mais poderosas versões foram surgindo ao longo destes anos. Mas o que realmente explica esse sucesso é sem dúvida a sua interface com o utilizador, que se tornou mundialmente conhecida e respeitada, e que representou, com a sua introdução num computador de custo relativamente baixo³⁰, um grande avanço em termos do computador pessoal ideal, especialmente em ambientes de escritório electrónico.

Embora a Apple já possua um sistema operativo tipo UNIX para o Macintosh, capaz de funcionar com outros sistemas de gestão de janelas (**X Windows**), ela

²⁹pelo menos nas versões do sistema operativo até inícios de 1990.

³⁰em Janeiro de 1984, início da comercialização do Macintosh, este era mais barato do que um IBM PC.

está nitidamente empenhada em manter e aperfeiçoar o seu sistema, para o qual existem milhares de aplicações, todas com o mesmo estilo coerente de interface com o utilizador, o que leva à conclusão que a aposta que a Apple fez no início da década de 80 foi claramente ganha.

2. O sistema MS-Windows para MS-DOS

A. Introdução

O computador pessoal tipo IBM XT/AT³¹ com MS-DOS é claramente o maior sucesso de vendas da década de 80 e o maior responsável pela massificação do "computador", na medida em que um grande número de novos utilizadores completamente inexperientes o comprou sem recear o seu desconhecimento e com enorme curiosidade em aprender a tirar partido dessa nova máquina.

No início, o tipo de interface com o utilizador desta nova máquina era realmente muito elementar, dado que toda a interacção com o sistema era realizada unicamente através do teclado e a informação era visualizada num ecrã alfanumérico. Aliás, o MS-DOS pouco mais era do que uma versão ultra-miniaturizada do UNIX, em função do hardware disponível na altura. Ainda assim, desde os primeiros tempos que o MS-DOS, para além de toda funcionalidade normalmente exigida a um sistema operativo, possuía um "pequeno" núcleo de rotinas para a visualização de texto e de gráficos, independentemente da carta gráfica instalada, o que dava a possibilidade de se criarem aplicações relativamente portáteis, em termos gráficos[JAMS].

Mas ao contrário do Apple Macintosh, que possuía um núcleo gráfico indiscutivelmente poderoso e versátil, este núcleo do MS-DOS era muito elementar, pois incluía apenas desenho de texto com tamanhos fixos, avanço de página, desenho de pontos individuais, etc, e principalmente, era extremamente lento. Como consequência, todos os programadores rapidamente se

³¹IBM XT/AT deve ser entendido no seu sentido mais lato, incluindo todos os compatíveis...

desinteressaram deste núcleo e passaram a aceder directamente aos recursos gráficos da máquina, o que implicava mais esforço individual e, mais grave ainda, duplicação inútil de esforços. São exemplos disso:

- As bibliotecas gráficas que acompanharam a maior parte dos compiladores de linguagens de alto nível, todas diferentes e incompatíveis entre fabricantes;
- Alguns mini-sistemas de gestão de janelas, normalmente muito incompletos e com falhas importantes.

Analisando a evolução do MS-DOS ao longo destes anos, especialmente em termos gráficos, conclui-se que algumas modificações foram realmente importantes e beneficiaram nitidamente este sistema operativo:

- Possibilidade de configuração do sistema no arranque através de ficheiros especiais, sem se ter que mexer no código do sistema operativo.
- Introdução de **device drivers** para hardware diverso, instalados aquando do arranque do sistema.
- Suporte imediato de várias cartas gráficas de características muito diversificadas, tais como **CGA**, **EGA**, **VGA**, **Hercules**, etc, a nível do bloco básico de entrada/saída (**BIOS**) do sistema operativo.

Mas o suporte destas várias cartas gráficas continuava a não passar do nível mais elementar, texto e ponto. E assim, todas as aplicações existentes para MS-DOS tipicamente utilizavam todos os recursos do sistema operativo, excepto no que dizia respeito a gráficos, em que ninguém queria usar as poucas rotinas existentes e incrivelmente lentas, e cada aplicação tinha o seu arsenal de rotinas mais sofisticadas e, muitas vezes, diversas versões para diferentes cartas gráficas.

É claro que o facto do IBM PC ser um sistema aberto³², ao passo que o Macintosh era claramente fechado, complicava grandemente as coisas, pois a proliferação de novas cartas gráficas para o IBM PC cedo se manifestou, e quase todas elas eram significativamente diferentes entre si. Frequentemente, o fabricante de uma nova carta gráfica limitava-se a fornecer um **driver** alfanumérico

³²aberto significa ter acesso aos detalhes da arquitectura (circuitos, barramentos, etc) e poder-se facilmente adicionar hardware externo.

para o MS-DOS, e alguns **drivers** gráficos para as aplicações comerciais mais populares. Assim, tornava-se imperioso impôr alguma ordem nesta grande confusão de ambientes gráficos incompatíveis. Surgiu então o sistema de gestão de janelas MS-Windows para o IBM PC com MS-DOS, o qual representava um grande passo em frente em termos de interface consistente com o utilizador³³ nesta máquina. Porém, era importante que o sistema MS-Windows fosse compatível com aplicações que não estivessem preparadas para ele, pois de outro modo as suas possibilidades de sucesso seriam muito reduzidas. Este objectivo foi elegantemente alcançado, o que ajudou, sem qualquer dúvida, o MS-Windows a impôr-se.

Na sua versão mais recente (MS-Windows 3.0), este sistema de gestão de janelas já apresenta uma sofisticação muito elevada, e graficamente é muito parecido com o gestor de janelas MOTIF para X Windows!

B. Arquitectura

O sistema MS-Windows está basicamente dividido em vários módulos, que repartem entre si as diversas tarefas de gestão. Embora o MS-Windows seja uni-processo, isto é, só executa uma aplicação de cada vez, apresenta uma inovação importante que é ser **multi-tarefa** - permite ter várias aplicações prontas em "memória" para execução imediata, visualizadas através de ícones sobre o seu desktop.

Os seus blocos fundamentais são:

- **Kernel** - Este bloco assenta directamente sobre o MS-DOS, e liga o sistema operativo propriamente dito (e os recursos subjacentes) à máquina conceptual de diversas janelas.
- **GDI (Graphics Device Interface)** - Este bloco manipula e executa os comandos gráficos relevantes de um modo transparente em relação ao hardware.

³³convém lembrar que, simultaneamente, a Microsoft lançou versões para MS-Windows de todas as aplicações comerciais de sucesso, na altura.

- **SysEdit** - Comanda todos os aspectos de edição de texto, manipulação de tipos de caracteres, etc.
- **User** - Controla todos os aspectos da interface com o utilizador.

De um modo semelhante ao Macintosh, o MS-Windows possui uma "secretária" continuamente de serviço, pronta a ajudar o utilizador a invocar novas aplicações, alterar o sistema, etc, designada **Program Manager** (na realidade, existe outro programa associado à interface com o utilizador para suprir algumas lacunas do **Program Manager** e que o complementa, o **Task Manager**).

C. Modelo Gráfico

O modelo gráfico do MS-Windows baseia-se igualmente no paradigma do RasterOp, e a sua implementação encontra-se distribuída pelos blocos **GDI** e **User**. Este núcleo 2D já se pode qualificar verdadeiramente como um núcleo gráfico, na medida em que apresenta uma funcionalidade sofisticada e muito diversificada.

O bloco **GDI** serve de esqueleto para um editor gráfico bastante evoluído, embora muito parecido com o MacPaint, denominado **PaintBrush**³⁴.

D. Gestão de Janelas

No MS-Windows, todas as janelas são rectangulares, uniformidade esta que simplifica grandemente muitos aspectos de implementação e permite desempenhos razoáveis, atendendo a que este pode estar a executar em hardware pouco sofisticado.

Toda a gestão está centralizada num módulo, capaz de criar novas janelas, movê-las, aumentá-las, etc, e que dialoga com outros módulos para realizar operações tais como transferir zonas de ecrã para disco, etc.

³⁴de um modo similar, a aplicação **NotePad** baseia-se praticamente sobre o módulo **SysEdit**.

Em sistemas com processador 80386, o MS-Windows pode executar em multi-processo, e então já é possível ter várias aplicações a actuar "simultaneamente" nas suas respectivas janelas, inclusivé aplicações totalmente desconhecedoras do MS-Windows, ou mesmo do BIOS do MS-DOS (com acesso directo à memória vídeo). Isto implica obviamente uma gestão de janelas com um grau de sofisticação elevado.

E. Interacção

Um dos grandes objectivos do MS-Windows 3.0 é a de criar um estilo de interface comum para todas as aplicações, o que na verdade até hoje ainda não aconteceu, e inclusivé aproveitar esta mudança para preparar os utilizadores para o **Presentation Manager** do OS/2. Para realizar esta política de uniformização, e tendo em conta os bons resultados que acarreta, existe um Toolbox para desenvolvimento de novas aplicações, cujo objectivo é o de acelerar e simplificar esse processo, e integrá-lo harmoniosamente na filosofia do MS-Windows.

Comprovando mais uma vez a liderança tecnológica que a Apple exerce, o MS-Windows possui semelhanças importantes tais como:

- Definição de entidades e acções como recursos, separados fisicamente da aplicação propriamente dita, e manipuláveis através de um compilador de recursos. São exemplos as janelas, os menus, títulos de janelas e caixas de diálogo.
- Gestão independente de caracteres, quer os existentes no sistema, quer através da adição de novos tipos.
- Núcleo gráfico.

F. Conclusão

Cerca de seis anos após o lançamento comercial do Macintosh, com a sua inovadora e revolucionária interface com o utilizador, surge uma versão

verdadeiramente competitiva para o IBM PC, e que utiliza muita funcionalidade de outros sistemas já existentes, nomeadamente o próprio Macintosh e o MOTIF para X Windows.

Com o aparecimento de máquinas MS-DOS cada vez mais potentes, sem as limitações dos 640 kbytes do DOS e outras, e nas quais o MS-Windows poderá funcionar com um bom desempenho, o tempo dos programas com acesso directo ao hardware e do esforço redundante poderá estar a findar, e talvez esteja a iniciar-se a fase de grande expansão do MS-Windows, que assim dará um novo fôlego ao sistema operativo MS-DOS, cada vez mais cercado pelo UNIX³⁵.

3. O sistema NeWS para estações gráficas SUN

A. Introdução

O NeWS é um sistema de gestão de janelas distribuído e extensível que pretende fazer uma abordagem de longo prazo no contexto das interfaces com o utilizador e tecnologia de visualização. Não é mais uma tentativa para construir um sistema em cima de algo já existente, mas sim uma tentativa para definir um novo nível de tecnologia[SUN1]. Uma característica única do NeWS é o uso flexível do conceito de extensão. A extensibilidade de um sistema é a chave para integrar eficientemente janelas num ambiente distribuído. O desempenho é aumentado através da interacção sofisticada entre clientes e servidor, a comunicação é acelerada por meio de mecanismos de compressão de informação próprios de cada aplicação, aspectos semânticos são totalmente controlados por uma autoridade centralizadora, e aspectos de interface com o utilizador tornam-se mais simples de implementar.

B. Arquitectura

³⁵hoje em dia já existem estações gráficas UNIX/X Windows com desempenhos razoáveis por preços semelhantes aos dos PC's mais poderosos.

O sistema NeWS baseia-se num novo tipo de comunicação entre processos³⁶. A comunicação entre processos é geralmente implementada através do envio de mensagens de um processo para outro sobre um meio de comunicação qualquer. Estas mensagens são normalmente uma mistura de comandos e parâmetros. Por outro lado, estas mensagens podem ser entendidas como programas escritos numa linguagem muito simples. O que poderá acontecer se esta linguagem for enriquecida e se tornar uma linguagem de Turing?³⁷. Então, aplicações deixam de comunicar através do envio de mensagens, mas sim através de programas, o que acaba por ter efeitos importantes em termos de compressão de dados, desempenho e flexibilidade.

A linguagem de programação POSTSCRIPT³⁸ é utilizada precisamente deste modo[ADOB]. O que os criadores do POSTSCRIPT pretendiam não era mais do que comunicar com impressoras[WARN]. Assim, eles transmitem programas POSTSCRIPT para uma impressora, que são posteriormente decodificados por um processador interno, e esta decodificação faz com os gráficos apareçam na página. A capacidade de definir novas funções permite o alargamento e a alteração das capacidades da impressora.

Esta ideia tem implicações importantes no domínio dos sistemas de gestão de janelas, dado que define um mecanismo poderoso de tornar o sistema muito mais flexível e fornece soluções óbvias para problemas tais como desempenho e sincronização. Por exemplo, se se pretende desenhar uma grelha, deixa de ser necessário transmitir um conjunto vasto de comandos de linha para o servidor, enviando-se então um programa simples que desenhe essa grelha.

As extensões do sistema NeWS são conformes às primitivas do sistema POSTSCRIPT, que é uma linguagem simples, possui um modelo gráfico coerente, e muito importante, é compatível com muitas impressoras actuais[SUN1].

Basicamente, o NeWS é um único processo, que actua como gestor e que contém um interpretador de linguagem POSTSCRIPT³⁹. Dentro deste processo de gestão estão contidos vários processos elementares que executam programas POSTSCRIPT. Estes processos elementares diferem dos processos UNIX normais na

³⁶em inglês **interprocess communication**.

³⁷no sentido de poder executar qualquer programa executável numa máquina de Turing.

³⁸definida por John Warnock e Charles Geschke da Adobe Systems.

³⁹o sistema NeWS foi totalmente concebido e implementado pela SUN Microsystems.

medida em que partilham entre si o espaço de armazenamento de dados, pelo que a criação de novos processos não implica um custo de processamento elevado e é bastante rápida.

Os clientes NeWS comunicam com o servidor através de canais de bytes⁴⁰, em que cada canal possui geralmente um processo elementar dentro do servidor NeWS que o controla. A informação é transmitida entre os clientes, que existem algures na rede, e os processos do servidor, que existem dentro do servidor. Estes processos podem executar acções no ecrã, receber informação de dispositivos tais como teclado ou rato e comunicar com outros processos NeWS.

Embora o NeWS tenha sido implementado à volta da linguagem POSTSCRIPT, tudo o que o NeWS fornece não é mais do que um conjunto de mecanismos. Políticas de interface com o utilizador implementam-se através dos programas POSTSCRIPT. Por exemplo, no sistema NeWS não existe nenhuma política de colocação de janelas no ecrã, embora haja mecanismos para criar janelas e posicioná-las, dadas as coordenadas pretendidas. A escolha das coordenadas ficará a cargo de um programa POSTSCRIPT.

O que é vulgarmente considerado como a interface de utilização de um sistema de janelas está totalmente fora de consideração para o sistema NeWS, pois essa interface inclui coisas tais como "como é que os títulos das janelas serão desenhados", ou "será possível alargar a janela pressionando o botão esquerdo do rato por cima do canto superior direito da fronteira da janela?". Estes aspectos de interface são resolvidos implementando programas apropriados em POSTSCRIPT.

C. Modelo Gráfico

Os gráficos em NeWS são baseados no modelo **stencil-paint**, segundo a descrição no artigo de Warnock e Wyatt[WARN] e na linguagem POSTSCRIPT[ADOB].

⁴⁰em inglês **byte streams**.

Um **stencil** é uma região delimitada por uma linha fronteira infinitamente fina, linha essa composta por splines definidos num sistema de coordenadas não-inteiro. **Paint** designa qualquer cor pura ou textura, eventualmente outra imagem, que pode ser aplicada à superfície de desenho. Qualquer tinta é sempre filtrada através de um stencil antes de ser aplicada na superfície de desenho. E isto é o tudo acerca do modelo gráfico do sistema NeWS. Assim, por exemplo, linhas podem ser definidas através de stencils longos e estreitos.

Algumas das características mais atraentes deste modelo gráfico são a sua simplicidade e abstracção. Por exemplo, um conjunto de caracteres pode ser definido de diversas maneiras: bitmaps, vectores e stencils. No caso do sistema NeWS, o uso de stencils não pressupõe nenhum conhecimento acerca dos pixels envolvidos, ou se existem mesmo pixels. Mesmo a extensão do sistema para lidar com o anti-aliasing não afecta em nada o modelo, pois a utilização de um modelo muito abstracto implica automaticamente um elevado grau de independência relativamente a quaisquer dispositivos.

Embora a especificação deste modelo seja simples e elegante, a sua implementação é complexa. Por exemplo, o mecanismo de criação de um stencil permite a utilização de segmentos de recta, arcos e outras curvas de maior complexidade para a especificação da sua fronteira, ou ainda, os stencils podem ser usados para clipping ou para enchimento de áreas, o que exige a determinação da intersecção das suas fronteiras.

O sistema NeWS implementa curvas através de splines cónicos, os quais servem para criar formas, existindo um conjunto muito vasto de algoritmos para manipular essas formas e que obedece ao modelo **stencil-paint**.

D. Gestão de Janelas

No sistema News a superfície básica de desenho é o **canvas**, termo escolhido para evitar a confusão semântica com o termo janela. Assim, um canvas é uma superfície onde qualquer imagem pode ser desenhada, podendo a superfície ser opaca ou transparente e ter qualquer forma. Os canvases existem num sistema de coordenadas a 2.5 dimensões, isto é, podem ser sobrepostos.

A implementação do canvas depende fortemente do software do modelo gráfico referido acima.

E. Interacção

No sistema NeWS, uma qualquer acção de entrada de informação é designada acontecimento. O conceito de acontecimento inclui acções tais como botões sendo pressionados, botões sendo libertados e movimentos do rato, e são implementados como mensagens entre processos. Os acontecimentos são distinguidos através da sua localização (janela, posição), da acção concreta e do objectivo dessa acção. É importante realçar que a associação entre acontecimento e acção é deliberadamente vaga e fácil de modificar[SUN1].

As acções a serem executadas quando se dá um acontecimento são especificadas de um modo muito geral, através da linguagem POSTSCRIPT. O despoletar do acontecimento provoca o envio de uma mensagem para um processo NeWS, que será responsável pela decisão de o que fazer com ele.

Os procedimentos da linguagem POSTSCRIPT controlam muito mais do que a mera interpretação das teclas pressionadas. Podem estar envolvidos no rastreio do cursor, na construção de caixilhos à volta das janelas, na implementação de menus, etc.

Uma aplicação de um cliente compõe-se de duas partes: uma escrita em POSTSCRIPT e existindo dentro do sistema NeWS, e a outra fora do NeWS e comunicando com ele através de um canal de bytes. Isto conduz à existência de vários níveis onde a interface com o cliente pode existir:

- no nível mais baixo, o programador escreve programas POSTSCRIPT e lida com um universo inteiramente POSTSCRIPT. Gestores de menus e políticas de gestão de janelas são exemplos de objectos geralmente implementados a este nível. O sistema NeWS dispõe de convenções que definem uma interface orientada ao objecto para janelas, menus, etc, e embora os objectos herdem um conjunto de características básicas, estas podem ser completamente redefinidas.

- no nível acima, o programador escreve programas em C, ou noutra qualquer linguagem, que vão gerar programas em POSTSCRIPT. Note-se que a este nível o programador explicitamente sabe da existência do universo POSTSCRIPT. Emuladores de outros sistemas de gestão de janelas são normalmente implementados a este nível. O sistema NeWS fornece um pré-processador de C que permite a programadores em linguagem C atravessar facilmente as fronteiras para o POSTSCRIPT⁴¹.
- no nível mais elevado, a existência do POSTSCRIPT e dos mecanismos de envio de informação estão completamente camuflados por uma camada de interface, geralmente construída com as ferramentas do nível anterior. O NeWS aparece então como um conjunto de rotinas.

F. Conclusão

Em muitos aspectos, o sistema de janelas NeWS representa um grande avanço na tecnologia dos sistemas de janelas actuais. Noutros, é muito semelhante. De seguida resumir-se-ão estas diferenças e semelhanças, relacionando-as com os objectivos fundamentais do NeWS.

No que respeita à independência relativamente a dispositivos, a maioria dos sistemas de janelas em rede, tais como Sun Windows⁴²[SUN2] e X Windows[ROSE1], é baseada no paradigma **RasterOp** e num espaço de coordenadas definido ao pixel. Com a evolução do hardware, estes pressupostos tornar-se-ão menos adequados:

- este modelo entra em conflito com o uso de hardware avançado de transformação e sombreamento⁴³, na medida em que insiste que os clientes decomponham as suas operações gráficas num nível tão elementar que inutiliza os benefícios do hardware. Os sistemas de janelas actuais só tiram partido do hardware sofisticado quando os clientes tomam medidas especiais para esse fim, o que é paradoxal.

⁴¹ferramentas semelhantes existem para outras linguagens.

⁴²é um sistema de janelas próprio da Sun Microsystems, relativamente antigo.

⁴³em inglês **rendering**.

- este modelo implica que os clientes conheçam o tamanho individual de cada pixel no ecrã, dado que o único sistema de coordenadas que possuem é fornecido pelo hardware. Até recentemente, isto não era um problema grave, porque os pixels nos ecrãs actuais estavam dentro de uma margem de mais ou menos 30% de uma resolução de 80 pontos por polegada. Mas para resoluções de 300 pontos por polegada ou superiores, tudo se complica.
- este modelo não se adapta versatilmente ao uso da côr⁴⁴. Além disto, expõe claramente as diferenças entre os três modos vulgares de representação de côr em terminais: preto e branco a 1 bit, 256 côres a 8 bits e 16 milhões de côres a 24 bits, devendo os clientes invocar operações especiais para cada modo.

O modelo gráfico mais sofisticado do NeWS permite que o hardware avançado beneficie todos os clientes, sem que tenham que tomar medidas especiais para esse efeito. Além disso, não precisam de se preocupar com o espaço de coordenadas do hardware, pois podem definir o seu próprio espaço.

No que respeita à côr, os clientes não têm que se preocupar com quaisquer dependências em relação a um modelo particular ou em relação a hardware relacionado com côr. O sistema NeWS permite que os clientes especifiquem côr através de tripletos RGB ou HSI⁴⁵, e fará todos os esforços para visualizar as côres correctamente no ecrã. Em terminais **full-color**, visualizará a côr correcta; em terminais com mapas de côres limitados, seleccionará a côr mais aproximada para a visualização; em terminais monocromáticos ou em níveis de cinzento, utilizará técnicas tipo **half-toning** ou **dithering** conforme necessário. Assim consegue-se que os clientes, com um esforço mínimo, obtenham resultados válidos numa vasta gama de dispositivos.

Um problema de independência semelhante também se coloca no que diz respeito aos dispositivos de entrada, para os sistemas de janelas actuais. Estes suportam um conjunto fixo de dispositivos, e o cliente tem que estar necessariamente informado disso. No sistema NeWS, cada cliente define o seu

⁴⁴basta pensar que a maioria das funções booleanas entre pixels coloridos gera resultados inúteis, e muitas vezes pirotécnicos.

⁴⁵vermelho, verde e azul ou luminosidade, saturação e brilho.

próprio conjunto de dispositivos, e mesmo dispositivos novos e imprevistos podem ser acomodados facilmente.

No importante aspecto da portabilidade, mesmo que os sistemas de janelas actuais não expusessem tanto o hardware subjacente, ainda assim muitos não seriam portáveis entre diferentes máquinas. Sistemas que são implementados como um superconjunto do núcleo do sistema operativo, tal como o Sun Windows, evidenciam claramente esse problema. O sistema News, tal como o X Windows, é um processo servidor ao nível dos utilizadores. O sistema X Windows já foi transportado para uma gama razoavelmente vasta de máquinas com relativamente pouco esforço, embora o NeWS tenha sido projectado desde o início para ser totalmente portátil, pelo menos em ambientes UNIX.

Em termos de flexibilidade, alguns sistemas de janelas impõe um único estilo coerente de interface com os utilizadores, de que o Macintosh é um bom exemplo. Este facto acarreta grandes benefícios, quer para o utilizador, quer para o programador, na medida em que todas as aplicações se comportam da mesma maneira.

Por outro lado, outros sistemas evitam explicitamente a imposição de qualquer estilo de interface. O sistema NeWS aborda os dois lados deste problema, isto é, o servidor não depende, em qualquer detalhe, da interface com o utilizador; dado que está escrita em POSTSCRIPT, pode ser facilmente alterada. Porém, o ambiente POSTSCRIPT global acarreta a herança de muitas características, o que garante uma certa consistência para todos os clientes.

Finalmente, é evidente nos dias de hoje que as janelas podem ser fornecidas como um serviço de rede, disponíveis para clientes localizados em qualquer ponto, possuindo um desempenho mais do que suficiente para a maioria das tarefas. O sistema NeWS implementa esta funcionalidade, mas elevando-a um degrau acima dos restantes sistemas. A capacidade de escrever partes de uma interface com o utilizador necessitando de resposta instantanea, tal como **rubber-banding**, na linguagem POSTSCRIPT e de execução pelo servidor sem intervenção do cliente, resolve um dos problemas mais importantes de desempenho dos sistemas actuais. Igualmente, a capacidade de enviar pela rede interpretadores de protocolos para aplicações muito específicas permite uma muito melhor utilização da largura de banda limitada da rede.

4. O sistema X Windows

A. Introdução

O sistema de gestão de janelas X Windows é um sistema portátil desenvolvido pelo Projecto **Athena** do *Massachusetts Institute of Technology*. Este sistema controla a visualização em estações gráficas e fornece um ambiente standardizado e flexível a muitos tipos de aplicações eminentemente gráficas. Assim, aplicações que utilizem o X para operar no ecrã de uma estação gráfica podem ser facilmente executadas noutras estações de fabricantes diferentes[ROSE1].

O laboratório de Computação Gráfica e o Projecto Athena do MIT, grandes centros de investigação, utilizavam diversas estações gráficas de capacidades muito diferenciadas e de fabricantes diferentes. Afim de ultrapassar o problema de reprogramar cada aplicação para cada sistema, foi desenvolvido o sistema X Windows. Este sistema veio a dar ao MIT a capacidade de programar aplicações para estações gráficas utilizando uma interface coerente e portátil em relação ao hardware de cada máquina. Na altura o Projecto Athena era subsidiado pela **IBM** e pela **Digital Equipment Corporation (DEC)**, e estes dois fabricantes anteviram o potencial deste sistema e decidiram apoiá-lo objectivamente.

Em Janeiro de 1987, diversos vendedores anunciavam os seus esforços conjuntos para standardizar e promover o X. Hoje em dia, quase todos os vendedores de estações gráficas aceitaram o sistema X Windows como a "melhor" interface para o hardware das suas máquinas[JONE].

B. Arquitectura

O sistema X Windows é formalmente definido em termos de um protocolo de rede para interligação de displays: as estruturas de dados e o significado das mensagens enviadas sobre essa conexão. Cada interação entre uma aplicação e a estação gráfica eventualmente reduz-se a uma ou mais mensagens enviadas sobre o protocolo[SCHE1].

As estruturas de dados e a sua funcionalidade constituem o que se designa por sistema de janelas básico. Todo o ambiente X é composto por camadas desenvolvidas sobre este núcleo (Figura 2).

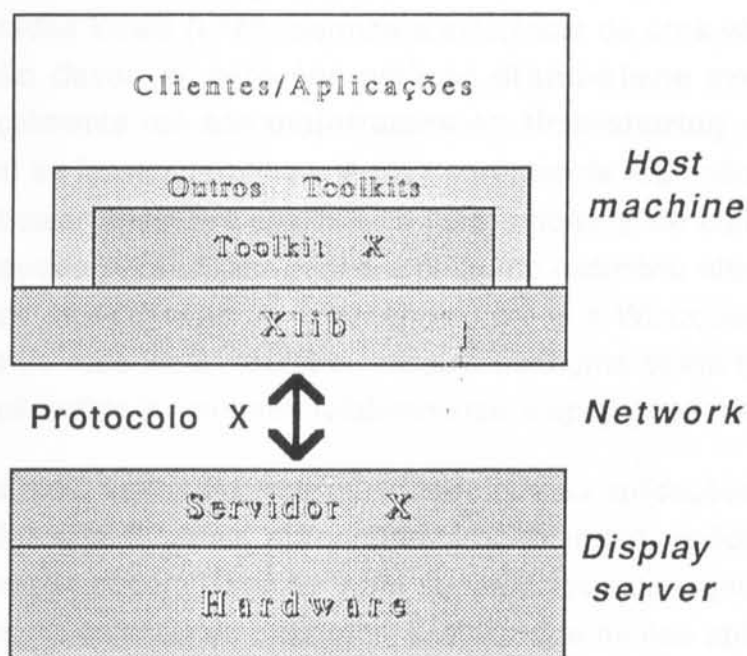


Figura 2
Arquitetura do X Windows

O sistema de base comunica com o mundo exterior através do protocolo de rede X, o qual pode trabalhar para um só processador ou para vários. A existência de um protocolo de rede permite assim ter uma grande independência relativamente ao hardware e confere transparência ao X.

Como se vê na Figura 2, o protocolo de rede é a única interface com o sistema de janelas de base. Não existem quaisquer tipos de mecanismos de acesso privilegiado; o protocolo é utilizado para todas as operações. Isto quer dizer, por exemplo, que aplicações tipo gestor de sessão ou gestor de janelas são

consideradas pelo X apenas meras aplicações, e não um tipo especial de software de sistema com privilégios.

O objectivo fundamental do X Windows é o de criar um ambiente de funcionamento transparente em termos de rede e independente do hardware para aplicações destinadas a executar em estações gráficas. Este ambiente suporta janelas sobreponíveis numa variedade de estações monocromáticas ou a côr.

A transparência de rede significa que aplicações X executando numa máquina podem visualizar gráficos no ecrã dessa máquina ou no de outra qualquer máquina. Muito da potencialidade do X resulta da sua integração no ambiente de redes locais (LAN); permite a existência de uma variedade de estilos de computação desde as estações gráficas **stand-alone** executando as suas aplicações localmente até aos **mainframes** em **time-sharing** utilizando diversas estações como se fossem terminais. A transparência de rede não favorece nenhum estilo em particular, mas sim aglutina-os num ambiente de computação em rede flexível e adequado para utilizar desde o pequeno escritório até ao vasto contexto universitário de investigação. A experiência com o X Windows demonstrou que, em ambientes de rede local, pouca ou mesmo nenhuma perda de desempenho se verifica nas aplicações X em rede, relativamente a aplicações meramente locais.

Por outro lado, esta transparência implica que as aplicações possam executar no processador que fôr mais apropriado. Por exemplo, aplicações que exijam cálculos intensivos podem executar num supercomputador ligado à rede, em vez de executarem na estação do utilizador. É óbvio que muitas aplicações tais como processadores de texto, emuladores de terminais, gestores de janelas, etc, executam localmente na mesma máquina onde residem processador e ecrã. Não é necessário alterar nenhum programa X para o fazer executar sobre a rede noutra qualquer máquina.

Para um utilizador, uma estação gráfica X "parece" estar ligada simultaneamente a muitas máquinas. Assim, é possível ter várias aplicações a visualizar informação simultaneamente no ecrã de uma estação e a receber dados a partir desta. Inversamente, uma única aplicação X pode visualizar gráficos em várias janelas em diversas máquinas diferentes, ao mesmo tempo.

Todas as aplicações X são portáteis, isto é, lidam unicamente com o sistema X, e desconhecem os detalhes do hardware de visualização de cada máquina. Desde que uma aplicação seja capaz de estabelecer conexão com uma estação

gráfica, fica então habilitada a utilizar todas as capacidades do sistema de janelas de base dessa estação. Na medida em que o hardware está camuflado pelo protocolo de rede X, uma aplicação não depende em nada dos recursos gráficos existentes em cada estação gráfica.

Geralmente, uma aplicação utiliza pedidos sem retôrno sobre o protocolo para instruir a estação sobre o que deve fazer. Por exemplo, operações gráficas tais como desenho de linhas são pedidos sem retôrno: quando a rotina Xlib correspondente é invocada, o Xlib formata convenientemente um pedido para o protocolo com a informação necessária, e a seguir armazena-o num **buffer**. Quando apropriado, o buffer é despejado e o Xlib envia os pedidos para a estação, que então desenha a linha pretendida. Na medida em que os pedidos sem retôrno podem ser armazenados num buffer e dado que a aplicação não tem que esperar pela sua execução, desempenhos razoavelmente elevados podem ser obtidos mesmo sobre linhas de transmissão de baixo débito. Este mecanismo de buffer de pedidos é, em princípio, transparente para as aplicações.

Nem todos os pedidos originados por aplicações são do tipo "envia e esquece" dos pedidos sem retôrno. Alguns pedidos, especialmente os que se destinam a inquirir informação sobre a estação, necessitam de um mecanismo de retôrno. Embora a aplicação invoque uma rotina Xlib como no caso de pedidos sem retôrno, o Xlib envia imediatamente esse pedido sobre o protocolo e bloqueia-se, à espera da resposta do servidor da estação. Logo que a resposta surge, o Xlib torna a informação acessível através dos mecanismos apropriados (como resultado de uma função ou parâmetros). Finalmente, a aplicação retoma a execução. Desde logo se conclui que não é possível usar buffers para pedidos com retôrno; sempre que uma aplicação emite um pedido com retôrno, o Xlib tem que esvaziar previamente o buffer⁴⁶, transmitir e esperar[SCHE2].

O X Windows permite a várias aplicações estarem simultaneamente activas numa estação. As aplicações individuais podem executar ou parar conforme necessário durante uma sessão do utilizador. As aplicações executam independentemente entre si - uma aplicação não necessita de se informar sobre as janelas das outras aplicações para desenhar nas suas próprias, nem de as desactivar de alguma forma para realizar as suas acções.

⁴⁶afim de se obter a informação realmente correcta.

As aplicações X, durante a sua execução, têm necessidade de criar e destruir recursos, os quais são essenciais no modo como estas interagem com a estação gráfica[JONE]. Os recursos permitem às aplicações a transmissão de informação de estado (posição de janelas, côres, etc) para a estação uma só vez, e depois o seu uso repetido. No X, existem os seguintes tipos de recursos:

Janelas - na acepção normal.

Graphic Contexts - O recurso contexto gráfico é utilizado para armazenar atributos gráficos de visualização tais como cor de fundo, largura de desenho de linha, estilo de linha, tipo de caracteres, etc. Cada pedido para geração de gráficos tem que especificar obrigatoriamente um contexto gráfico.

Fonts - Este recurso contém toda a informação necessária para visualizar texto numa janela, incluindo tamanho, estilo e outros atributos.

Color Maps - Os recursos mapa de cores traduzem a informação gráfica que a aplicação desenha nas janelas em cores no ecrã.

Pixmaps - Os pixmaps, que no fundo são memória gráfica não-visível, são semelhantes às janelas, no sentido em que as aplicações podem gerar gráficos neles. Diferem das janelas na medida em que os utilizadores não os podem ver directamente. A sua principal utilidade é permitir copiar informação entre janelas, outros pixmaps e as aplicações. Podem ser também usados para armazenar padrões para fundos, cursores, etc.

Cursores - Este recurso descreve o aspecto do ponteiro no ecrã, o qual pode ser movido no ecrã através do rato. As aplicações podem criar cursores com formas e cores arbitrárias, e podem associá-los com janelas.

Apesar das suas diferenças, todos os tipos de recursos X partilham algumas características:

- Todos os recursos residem no servidor X da estação gráfica⁴⁷. As aplicações criam-nos, destroem-nos e alteram o seu conteúdo através de pedidos à estação. A finalidade dos recursos é permitir às aplicações controlar explicitamente o estado da estação.
- Todos os recursos são partilháveis entre as aplicações que estejam a executar numa estação. Cada recurso numa estação possui um identificador específico. Qualquer aplicação que conheça o identificador de um qualquer recurso pode utilizá-lo, alterá-lo ou destruí-lo. Ao invés, recursos não são partilháveis entre estações. Identificadores com um significado particular numa estação não serão válidos noutra estação.
- Os recursos são criados através de pedidos X sem retorno, o que aumenta bastante o desempenho.
- Os recursos são transitórios, no sentido em que não são apropriados para armazenar informação entre sessões.
- Os recursos são entidades computacionalmente simples e eficientes.

C. Modelo Gráfico

O sistema de base do X Windows oferece um conjunto vasto de operações gráficas bitmapped. As aplicações e os toolkits de alto nível utilizam as operações gráficas do X para realizar os gráficos sofisticados no ecrã.

O modelo gráfico do X é igualmente o RasterOp. Eis algumas das capacidades gráficas do X Windows:

- O X organiza os ecrãs numa hierarquia de janelas sobreponíveis. Cada aplicação pode utilizar as janelas de que necessitar, alterando-as, movendo-as e empilhando-as conforme deseje.

⁴⁷servidor X designa o programa residente na estação que traduz os comandos do protocolo em operações gráficas no ecrã.

- O X fornece diversas operações gráficas sofisticadas. Estas são executadas imediatamente; a estação não armazena as operações numa lista para posterior execução⁴⁸.
- Todas as operações gráficas são orientadas ao bitmap, e especificadas em termos de janelas, pelo que assim as aplicações realizam os seus gráficos dentro das janelas sem terem que se preocupar com a sua localização no ecrã.
- É possível desenhar texto de elevada qualidade gráfica, mesmo até com tipos de caracteres incomuns (kanji, cirílico, grego, etc).
- O sistema X funciona com diversos géneros de estações gráficas a côres, assim como com estações a preto e branco.

Sempre que uma operação gráfica é invocada, é necessário especificar dois identificadores de recursos X:

1º um objecto de destino, que pode ser uma janela ou um pixmap;

2º um contexto gráfico (GC), que determina os atributos gráficos a usar durante a operação.

A utilização de atributos gráficos a partir de recursos GC's é um dos conceitos principais do modelo gráfico X. Primitivas gráficas invocadas por uma aplicação são processadas por um **pipeline** gráfico - um conjunto de operações que converte as primitivas gráficas em pixels no ecrã. Os valores dos atributos dos GC's controlam precisamente este pipeline.

Durante a execução de uma aplicação, podem ser criados diversos tipos de pipelines definidos pelos valores armazenados nos atributos dos GC's. Aquando da invocação de uma operação gráfica, bastará então especificar o pipeline apropriado (GC) para essa operação. A versão 11 do X Windows utiliza os GC's deste modo para reduzir a transmissão sobre a rede. Ao invés, a versão 10 do X Windows obrigava a especificar atributos tais como largura e côr da linha, por exemplo, explicitamente para cada operação de desenho de linha invocada; experiências demonstraram que era relativamente raro as aplicações alterarem os

⁴⁸este mecanismo, que o X Windows não utiliza, é designado **display list**.

atributos gráficos em cada primitiva. Assim, em vez de as aplicações especificarem os atributos por primitiva, repetida e redundantemente, especificam-nos indirectamente através de um GC por primitiva.

O pipeline gráfico do sistema X Windows é formado pelos seguintes blocos (Figura 3):

- **Seleção de pixels** - O bloco inicial actua sobre um objecto gráfico tipo linha, arco, polígono. etc. Ele determina analiticamente quais os pixels a alterar afim de visualizar a forma do objecto. A maioria das primitivas gera pixels do tipo **foreground**⁴⁹, embora algumas também gerem pixels de **background**. Cada um destes pixels vai ser transferido para o bloco seguinte. Aliás, a maioria dos atributos dos GC's controla o modo como os pixels são seleccionados, para cada tipo particular de primitiva. Por exemplo, o desenho de linhas é controlado por atributos tais como largura, estilo de linha, estilo de junção, tipo de ponteadado, etc.
- **Patterning** - Este bloco aplica um padrão de pixels coloridos aos pixels resultantes do bloco anterior, através dos seguintes atributos:
 - Foreground** - cor de desenho;
 - Background** - cor de fundo;
 - Stipple** - identificador de um pixmap contendo pixels de valor 0 ou 1 que servirá para colorir os pixels, aplicando-se a cor de desenho onde o pixel correspondente do pixmap for 1 e a cor de fundo ou nenhuma onde for 0;
 - Tile** - identificador de um pixmap contendo pixels com cores;
 - Fill style** - estilo de enchimento dos pixels.
- **GC Clipping** - Após as cores terem sido aplicadas às primitivas gráficas, este bloco despreza todos os pixels que fiquem fora da região especificada, caso exista. O mecanismo de clipping pode funcionar com uma máscara de bits ou através de uma lista de áreas rectangulares.
- **Window Clipping** - Este bloco despreza todos os pixels que estejam fora da janela ou do pixmap de destino, ou que doutro modo seriam desenhados fora da janela de destino.

⁴⁹pixels opostos em relação ao fundo.

- **Raster** - O bloco final combina os pixels sobreviventes com os pixels já existentes no objecto de destino, através de funções lógicas aplicadas aos pixels e de máscaras de planos.

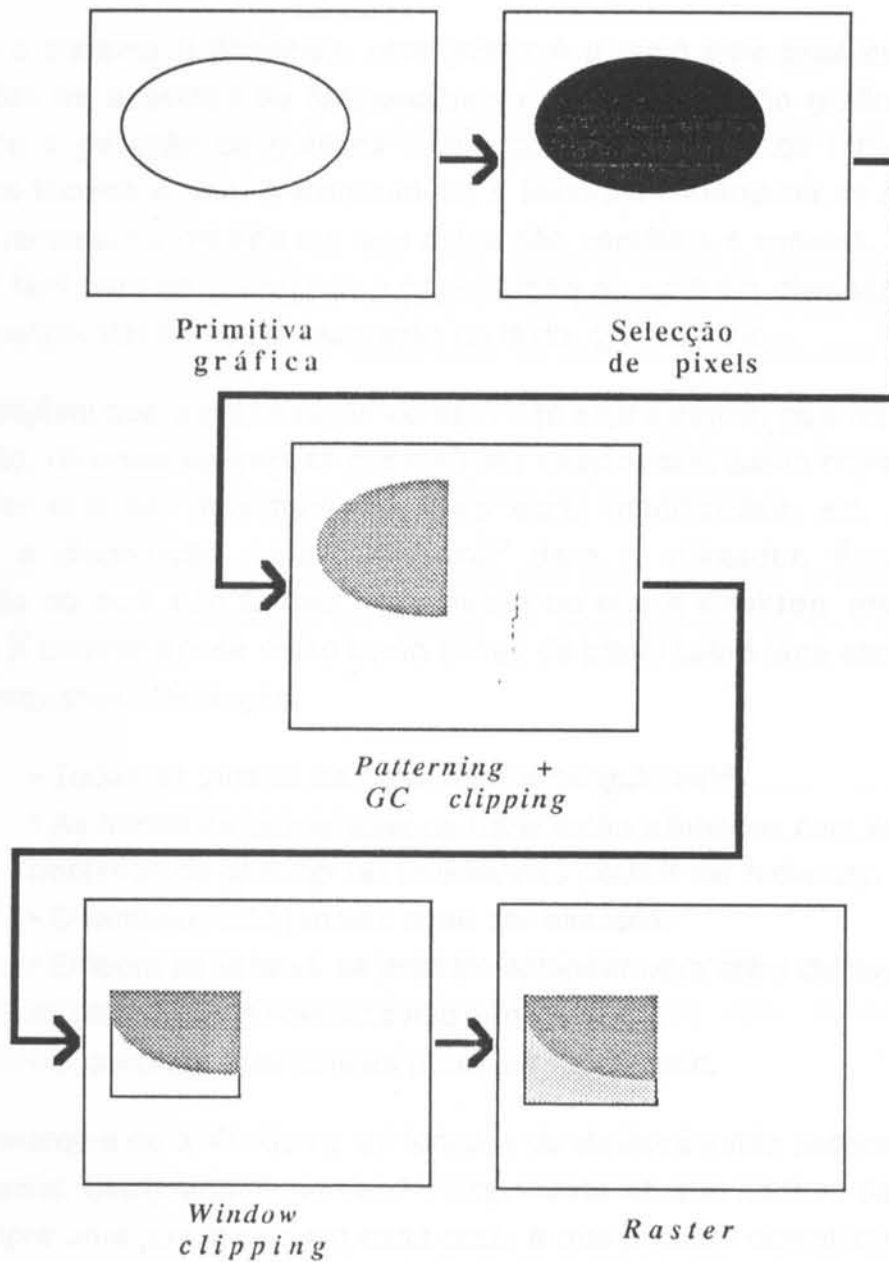


Figura 3
Pipeline gráfico do X Windows

O ecrã de uma estação gráfica é actualmente sofisticado e permite a visualização de uma grande quantidade de informação; para tirar partido destas capacidades, um utilizador deve poder organizar esta informação visualmente. O X Windows cumpre este objectivo através da divisão do ecrã em janelas[ROSE1].

Para o sistema X Windows, uma janela é o meio pelo qual este organiza quase todos os aspectos de funcionamento de uma estação gráfica, incluindo obviamente a geração de gráficos assim como a entrada de informação dos dispositivos teclado e rato. A estrutura do X encoraja claramente as aplicações a utilizarem janelas, na medida em que estas são versáteis e simples. Elas tornam possível e fácil para as aplicações a organização do ecrã em diversas áreas para fins apropriados, tais como apresentação de texto, gráficos, etc.

É desejável que a organização do ecrã não seja estática, pois no decorrer de uma sessão, diversas aplicações poderão ser executadas, assim como as janelas poderão ter que ser movimentadas pelo ecrã, modificadas, etc, afim de se conseguir a disposição mais "agradável" para o utilizador. Este modo de organização do ecrã não é mais nem menos do que a **desktop metaphor**. As janelas do X comportam-se muito como folhas de papel sobre uma secretária, com algumas pequenas diferenças:

- Todas as janelas de base são rectangulares⁵⁰.
- As fronteiras das janelas de base estão alinhadas com as fronteiras do desktop (as janelas não podem ser rodadas).
- O tamanho das janelas pode ser alterado.
- Embora as janelas se possam estender para além do desktop, as partes fora do desktop não são visíveis.
- O conteúdo das janelas pode ser modificado.

Na hierarquia do X Windows, as funções do desktop estão associadas a uma janela especial, designada janela raiz⁵¹, a qual está situada no topo da hierarquia. Existe sempre uma janela raiz em cada ecrã, e que o cobre completamente. Este tipo de janelas, ao contrário das outras, não pode ser movido ou removido do ecrã⁵².

⁵⁰através de extensões, é possível ter janelas que não sejam rectangulares.

⁵¹na designação do X, **root window**.

⁵²em termos de paradigma do desktop, a janela raiz corresponde ao desktop, e todas as janelas aparecem sobre ela.

As capacidades da hierarquia do X Windows vão muito além do que se consegue fazer com folhas de papel numa secretária, isto é, as janelas podem ser imbricadas de formas muito diversas e sofisticadas. A hierarquia de janelas do X pode ser controlada de duas maneiras:

- Profundidade, que define o número de níveis de imbricação de janelas.
- Largura, que determina o número de janelas "irmãs" em cada nível hierárquico.

A maioria das aplicações possui uma janela principal e várias outras janelas imbricadas nesta janela principal. Mas o X não obriga as aplicações a utilizarem uma profundidade e largura de hierarquia de janelas pré-definidas. Aliás, nada obriga as aplicações a possuírem uma só janela principal; estas podem escolher o modo como as janelas se sobrepõem em cada nível da hierarquia, quer movimentando-as, quer alterando a sua ordem de empilhamento.

As aplicações podem igualmente desenhar texto, gráficos, imagens, etc, nas janelas e solicitar informação de quaisquer acontecimentos em cada janela, individualmente.

E. Interacção

O X Windows fornece diversos mecanismos sofisticados para realizar interacção com as aplicações[JONE]. Eis algumas das capacidades de manipulação de informação de entrada do X:

- Os utilizadores geram informação através do teclado e do rato; esta é distribuída às aplicações sob a forma de acontecimentos.
- Os acontecimentos permitem que as aplicações descubram alterações na hierarquia da árvore de janelas, etc.
- O sistema de base organiza os acontecimentos convenientemente de forma que cheguem às aplicações apropriadas.

Quando várias aplicações executam na mesma máquina, é frequente necessitarem de comunicar entre si. O X Windows permite a troca de informação dos seguintes modos:

- Mecanismos tipo **cut and paste** para partilhar dados simples e não estruturados.
- **Properties**, entidades que permitem armazenar dados estruturados e que podem ser associadas a janelas.
- **Selections**, as entidades mais sofisticadas e complexas para interagir e transmitir quaisquer tipo de dados.

A interface com o utilizador, no seu sentido mais lato, é a combinação da funcionalidade fornecida pelo sistema de janelas de base com as ferramentas de desenvolvimento (os Toolkits), com os gestores de sessão e de janelas e com a aplicação do utilizador. O dogma fundamental do X Windows é que o sistema de janelas de base só fornece mecanismos, e não política. O modo como a interface responde às solicitações dos utilizadores fica a cargo de outras entidades. Por exemplo, alguns utilizadores preferirão uma política em que as janelas coexistirão lado a lado sem se sobrepôr, modo que se designa **tiling**. O modo tiling é perfeitamente possível em X; os aspectos da política que lidam com a configuração das janelas no ecrã são controlados por um gestor de janelas, logo para se ter tiling basta escolher um gestor de janelas orientado para o tiling. Mas se os mecanismos de visualização de janelas exigissem que estas fossem colocadas no modo tiling, imagine-se a dificuldade que se teria para implementar uma política do tipo menu pop-up. A versatilidade é justamente uma das características mais interessantes do sistema de base X, dado que fornece mecanismos suficientes para implementar uma gama muito diversificada de políticas de interface.

Um princípio geral do X é o de que uma aplicação nunca recebe informação de entrada que não tenha sido solicitada. Assim, a informação (isto é, os acontecimentos) só é entregue aos clientes que a tenham explicitamente requisitado.

A maioria das aplicações X não necessita de aceder ao protocolo directamente, mas fá-lo através de interfaces de programação, sendo a de baixo

nível⁵³ denominada Xlib. A maioria das aplicações utiliza as interfaces de mais alto nível na medida em que escondem melhor as complexidades do protocolo.

Como quaisquer outros programas, as aplicações X têm que ser compiladas e "linkadas" para cada combinação de processador e sistema operativo em que irão funcionar. Por exemplo, é necessário fazê-lo quando se transporta uma aplicação de um ambiente DEC VAX/Ultrix para um ambiente DEC VAX/VMS. Os fabricantes de estações gráficas fornecem versões pré-compiladas da biblioteca Xlib e outras, as quais podem ser "linkadas" com o código das aplicações.

F. Conclusão

Desde o aparecimento da versão 10 do X Windows que este se têm vindo progressivamente a impôr como o sistema de janelas standard da actualidade, e este facto deve-se a vários factores, todos eles importantes:

- 1º O X Windows é um sistema portátil e aberto. Apesar de ter sido desenvolvido sobre o sistema operativo UNIX em linguagem C e inicialmente suportar um pequeno número de estações gráficas, já existem versões para outros sistemas operativos e para muitos modelos de estações gráficas. Sem dúvida que a decisão de tornar o código do X, quer da interface de programação Xlib, quer do código esqueleto do servidor de estações gráficas, acessível a toda a gente muito contribuiu para este sucesso.

⁵³baixo nível aqui não se refere à linguagem da interface, mas sim ao nível de localização na hierarquia do X.

- 2º Não impõe qualquer tipo de interface com o utilizador. Pelo facto do tratamento da informação gerada pelos dispositivos de entrada, pelas aplicações, etc, ser de baixo nível, é possível implementar praticamente qualquer estilo de interface baseada em janelas. Aliás, um dos aspectos mais "paradoxais" da programação de aplicações em X através do Xlib, para programadores inexperientes, é o tratamento de baixo nível da informação (existem 33 tipos de acontecimentos possíveis) num sistema que é evoluído. Só que é precisamente a disponibilidade desta informação de baixo nível que permite implementar os mecanismos necessários de modo a que qualquer aplicação possa definir as suas próprias convenções. Basta constatar a diversidade de aplicações sofisticadas de gestão de janelas X existentes (DEC Windows, MOTIF, etc), de gestão de sessão (X/OpenLook, XView, etc) e muitas outras para se ter uma noção do trabalho de investigação e de desenvolvimento que o X originou.
- 3º A independência em relação ao hardware. Existem hoje em dia sistemas X a executar sobre hardware de visualização normal (por exemplo, cartas gráficas EGA e VGA para IBM PC), assim como a executar sobre estações gráficas com hardware gráfico de elevado desempenho (DEC Vaxstation 3520, IBM RS6000, etc). Qualquer aplicação X pode executar em qualquer máquina que funcione sobre X Windows. Para aplicações que necessitem de elevados desempenhos, e sem perder de vista a independência em relação ao hardware, existem mecanismos que permitem tirar partido de características sofisticadas do hardware gráfico, caso existam e o servidor X de estação as possa utilizar.
- 4º Ambiente de computação verdadeiramente distribuído. A facilidade com que se pode trabalhar em rede utilizando o X Windows, executando aplicações numas máquinas e visualizando gráficos noutras, independentemente dos sistemas operativos subjacentes, redes de transmissão, etc, é igualmente uma das razões para o seu sucesso.
- 5º Modelo gráfico versátil. O seu modelo 2D permite realizar todo o tipo de gráficos sofisticados sem praticamente impôr quaisquer limites, o que não quer dizer que seja o melhor modelo possível para um sistema deste tipo⁵⁴.

⁵⁴confrontar com o último capítulo do sistema NeWS.

IV. Visão detalhada do Servidor X

1. Introdução

De todos os sistemas de janelas existentes hoje em dia, sem dúvida que aquele que garante a maior portabilidade e difusão das aplicações é o X Windows⁵⁵, pelo que se torna estrategicamente importante, ao desenvolver-se uma nova carta gráfica, compatibilizá-la com este sistema. O objectivo deste trabalho é o de compatibilizar uma nova carta gráfica, em desenvolvimento, com software já existente; nessa perspectiva, a escolha do X Windows fica perfeitamente justificada:

- é um standard de *facto*
- existe uma grande quantidade de software
- é um sistema aberto e documentado
- o servidor X disponível para desenvolvimento é estruturado de modo a poder ser utilizado em novas implementações.

Este capítulo descreve o sistema X Windows em termos do hardware e software que existe no servidor da estação ou do terminal gráfico[ANG2].

O servidor X faz a gestão das janelas e de comandos gráficos elementares para o utilizador em função das diversas aplicações que este lançou em execução. Estas aplicações podem estar a ser executadas em qualquer máquina interligada na rede (Figura 4).

⁵⁵basta dizer que o X Windows funciona em quase todos os "sabores" de UNIX (Ulrix, HP-UX, etc), em VMS, em MS-DOS, etc.

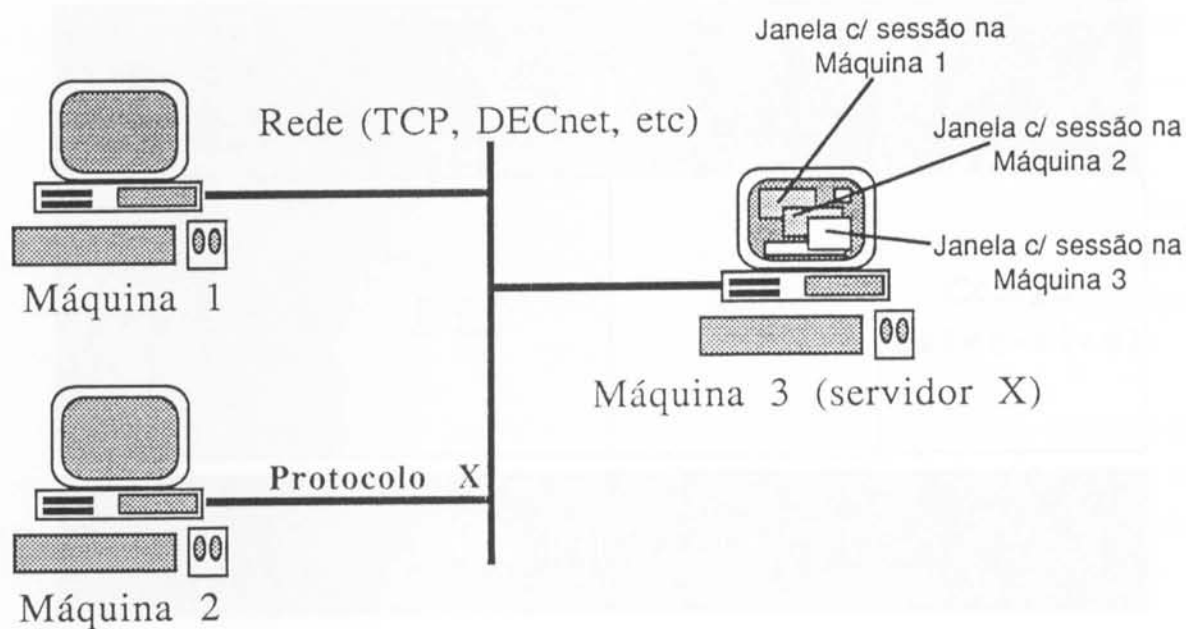


Figura 4
Funcionamento distribuído do X Windows

Basicamente, o servidor realiza 3 funções:

- responde aos pedidos do protocolo feitos pelos clientes (essencialmente comandos de gráficos e de texto)
- envia informação dos dispositivos de entrada (acções de teclas e de rato) e outros acontecimentos para os clientes
- mantém as conexões existentes.

Para implementar esta funcionalidade básica, o código do servidor está organizado em 4 blocos principais (Figura 5).

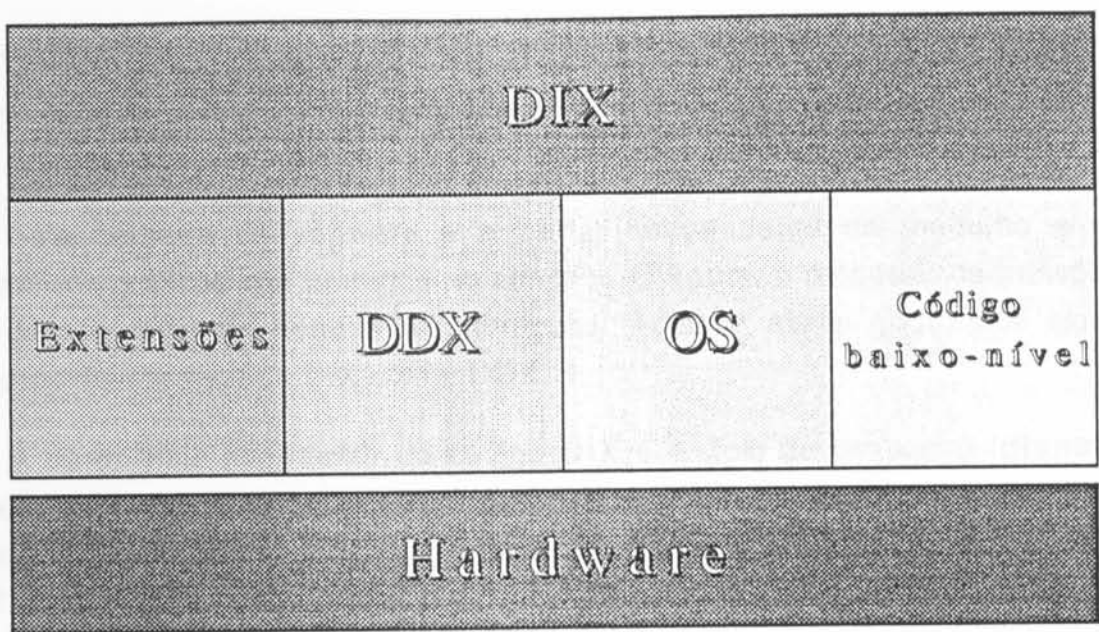


Figura 5
Estrutura do servidor X

DIX - código partilhado por todas as implementações baseadas neste servidor; depende do sistema operativo, mas é partilhado por todos os dispositivos gráficos sobre esse mesmo sistema operativo;

OS - código de sistema operativo, partilhado por todos os dispositivos gráficos;

DDX - código que é, em princípio, diferente para cada combinação de dispositivo gráfico e sistema operativo;

Interface de extensões - um mecanismo standard de adicionar funcionalidade ao servidor.

Os blocos fundamentalmente a modificar são o do sistema operativo e o bloco **DDX**; estes são, em termos funcionais, paralelos e nenhum está assente em cima do outro.

Seguidamente, descrevem-se as quatro camadas principais em que se organiza o código do servidor X.

2. A camada DIX

Esta camada de software é a parte independente da máquina e dos dispositivos gráficos que pertence ao servidor X. Assim, o processo de transporte não deverá provocar nenhuma alteração. Adiante, serão abordados alguns aspectos da interface do **DIX** com o **DDX**.

A parte mais importante do código **DIX** é o ciclo de despacho (**dispatch loop**). Cada vez que o processador percorre este ciclo, envia acontecimentos de entrada previamente acumulados e processa pedidos feitos pelos clientes. Este ciclo é a forma mais organizada e mais simples do servidor simular o multiprocessamento.

O ciclo de despacho do **DIX** cria a ilusão do multiprocessamento entre as diferentes janelas, embora o servidor não seja mais do que um único processo. Ele divide o trabalho em pequenas partes, facilmente assimiláveis (pedidos de clientes tais como comandos gráficos, acontecimentos fornecidos pelo servidor tais como acções de teclas, etc). O processamento de acontecimentos e de pedidos por diversos clientes pode ser imbricado facilmente, pelo que a necessidade de verdadeiro multiprocessamento não é realmente exigida ao servidor.

A funcionalidade da camada **DIX** é muito grande, e a maior parte das acções de baixo nível que desenvolve é executada por rotinas de **DDX** e de **OS**, adaptadas previamente.

3. A camada OS

Esta parte do código consiste em algumas rotinas que têm que ser desenvolvidas para cada tipo de sistema operativo. Basicamente, estas rotinas são responsáveis pela manutenção das conexões com os clientes e pela temporização das acções a serem executadas. Também devem fornecer uma interface de alto nível para os ficheiros de descrição de caracteres, tradução de nomes de ficheiros lógicos para nomes físicos e gestão dinâmica de memória (a um nível baixo).

Pedidos feitos pelos clientes são vistos como cadeias de bytes pela camada OS. Podem estar organizados em vários blocos de bytes entregues em sequência, ou então, um bloco poderá conter vários pedidos. Em teoria, o maior tamanho de um bloco pode ser 256 kbytes, embora seja dependente do servidor (a dimensão de buffer é negociada após o estabelecimento da conexão).

4. A camada DDX

Esta camada está dividida em várias partes (Figura 6). Assim, temos:

MI - este bloco faz a gestão do **frame-buffer**; as suas rotinas são portáteis e funcionam sobre um núcleo de rotinas mais elementares, eventualmente sobre um núcleo básico capaz apenas de manipular linhas de pixels em bloco

MFB - este bloco implementa a gestão de **frame-buffers** monocromáticos de profundidade 1 bit por pixel

CFB - este bloco é a versão para **frame-buffers** com mais de 1 bit por pixel, até 32 bits por pixel

Rotinas diversas - rotinas muito dependentes do hardware de que não existe uma implementação minimamente geral (podem ser estudadas as versões existentes para o DEC QVSS ou para a SUN monocromática).

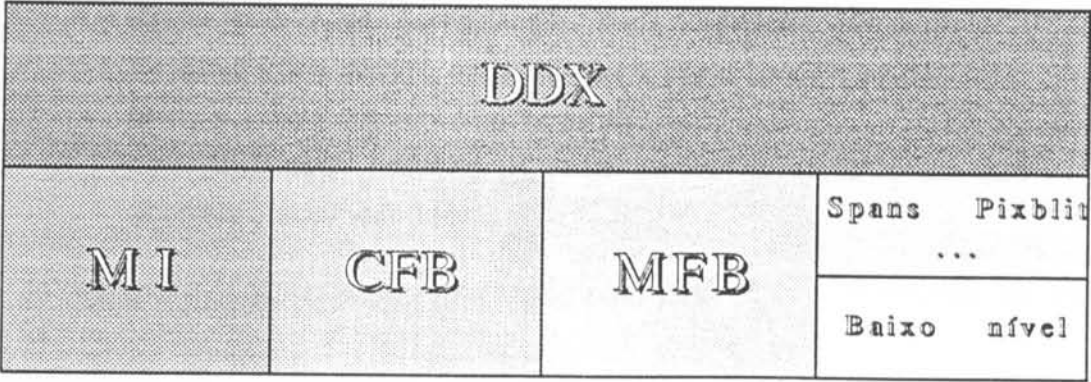


Figura 6
Estrutura do bloco DDX

Em seguida, serão abordados com detalhe alguns aspectos da interface entre o **DDX** e o **DIX**.

1. *Input*

Os dispositivos de entrada do X Windows podem ser de 3 tipos: teclado, apontador e "outros". "Outros" designa uma extensão; não existe suporte para além dos dois mencionados. Neste domínio, algumas rotinas **DDX** têm que ser implementadas; existem também rotinas **DIX** que têm que ser invocadas para executar as acções das rotinas **DDX**.

A. Processando Acontecimentos

O principal mecanismo de interface de entrada do **DDX** consiste numa rotina cuja função é entregar acontecimentos de entrada ao **DIX**, gerados pelo utilizador. O **DIX** chama esta rotina sempre que existe informação de entrada pendente, e também noutras ocasiões. Para cada dispositivo existe uma rotina **DIX** de entrega de acontecimentos, a ser invocada pela rotina **DDX** as vezes julgadas convenientes. Os acontecimentos devem ser fornecidos pela ordem em que foram gerados.

A estrutura que descreve a informação a fornecer deve conter:

tipo - se é botão pressionado, tecla libertada, etc

detalhe - um código especial para cada tecla, por exemplo

tempo - instante em que ocorreu

coordenadas - posição (X,Y) do cursor no instante em que ocorreu o acontecimento;

os restantes campos são preenchidos pelo **DIX**.

B. Notificando o DIX

No ciclo de despacho do servidor, o **DIX** verifica "periodicamente" se existe informação de entrada pendente. Este teste deve ser o mais rápido possível, por razões de desempenho. O servidor fornece mecanismos especiais para cumprir esta especificação.

C. Dispositivos de entrada

O controlo dos dispositivos de entrada é realizado através de 4 rotinas **DDX** relacionadas com o teclado e o apontador. No caso do dispositivo apontador, podem ser inclusivamente suportados acontecimentos intermédios, ou seja, que ocorreram durante a deslocação do dispositivo (o **DIX** só exige que se informe o acontecimento inicial e o final).

D. Inicialização

A inicialização dos dispositivos de entrada é complexa. É necessário definir os dispositivos, declarar as suas rotinas de interface, registá-los, etc. É durante a inicialização que se definem algumas características importantes do servidor, tais como poder armazenar posições intermédias do apontador, mapas de teclas, mapas de símbolos para as teclas, teclas de modificação, etc.

E. Teclado e códigos de teclas

Quando se envia um acontecimento de teclado, no fundo informa-se que uma determinada tecla foi activada (pressionada ou largada). Esta informação deve ser um código para cada símbolo associado à tecla; o mapeamento código-tecla pode ser qualquer um, porque é descrito numa tabela que é fornecida ao **DIX** na inicialização e que pode ser alterado.

2. Ecrãs

Os variados terminais gráficos existentes possuem capacidades muito diversificadas, desde o **frame-buffer** mais simples, com a sua memória totalmente controlada pelo processador **host**, até ao terminal que possui o seu processador gráfico de alto desempenho, e que realiza todo o trabalho de natureza gráfica, incluindo a gestão de tabelas de côr, matrizes de transformação 3D e outras estruturas de (muito) alto nível.

A. Requisitos fundamentais

O único requisito colocado aos ecrãs é que seja possível escrever e ler posições de memória no **frame-buffer** (o mecanismo não interessa). A profundidade de pixel não pode exceder 32 bits (a menos que se empreguem extensões). O servidor assume que os pixels no ecrã são quadrados. É da responsabilidade das aplicações compensar desvios no aspecto do pixel.

A inicialização dos ecrãs do servidor é efectuada a partir de uma rotina principal, que vai definir as estruturas de dados necessárias e invocar as rotinas **DDX** convenientes para colocar o hardware funcional.

O cursor é a representação visual associada ao dispositivo apontador. Sempre que é criada uma janela, é necessário especificar o formato do cursor quando ele viajar dentro das suas fronteiras, num dado ecrã. Para cada formato de cursor, existe uma estrutura de dados que descreve o seu comportamento gráfico:

- um bitmap para a sua imagem
- um bitmap para a imagem por trás (máscara)
- côr de fundo
- côr própria do cursor.

O mecanismo de visualização é o seguinte:

- 1ª aplicação da máscara, associando a côr de fundo aos bits "1";
- 2ª aplicação do cursor, associando a sua côr aos bits "1".

Deste modo, um cursor pode ter partes transparentes e partes opacas a 2 côres. Embora o X Windows permita qualquer formato para o cursor, o hardware de cursor, quando existe, apresenta limitações (geralmente, janelas de M por N

pixels, em que $M=N \leq 32$, etc). Porém, é permitida a redução do tamanho para dimensões apropriadas ao hardware.

No que respeita a cursores, é necessário implementar algumas rotinas que, embora não muito complexas, devem possuir um desempenho elevado.

B. Visuais, Profundidades e Classes

A profundidade de uma imagem é o número de bits que é utilizado, por pixel, para visualizá-la. Para cada ecrã é necessário especificar quais as profundidades suportadas (uma ou várias).

A classe de um terminal define se ele é monocromático, se possui uma tabela de cores e o modo como esta funciona.

O visual é a combinação de profundidade, classe e modo como os bits são mapeados em cores, para um dado ecrã. Cada visual possui informação para decompôr um pixel nas suas componentes primárias, total de definições na tabela de cores, etc. A classe de um terminal determina como a tradução pixel-componentes RGB é efectuada:

Pseudo - o valor do pixel é utilizado para pesquisar numa tabela a cor correspondente

"True" - o pixel é decomposto em componentes RGB, as quais são utilizadas para pesquisar em 3 tabelas distintas (RGB)

"Gray" - o valor do pixel é utilizado para pesquisar uma tabela afim de se determinar o nível de cinzento correspondente.

A tabela de cores pode ser, por outro lado, estática ou dinâmica (os valores podem ser modificados pelas aplicações). A conjugação destas 2 características define, finalmente, um total de 6 possibilidades.

C. Mapas de cores

Um mapa de cores define o modo como os valores dos pixels são transformados em cores, independentemente dos dispositivos. Diferentes janelas no mesmo ecrã podem possuir diferentes mapas de cores simultaneamente. O mais recente mapa de cores instalado será sempre o mapa operacional, de modo que a janela associada será visualizada com as cores correctas. Para as outras janelas, as cores poderão parecer aleatórias!

O mapa de cores de um dado ecrã é inicializado quando esse ecrã é também inicializado. É natural que a maior parte das aplicações utilize um mapa de cores pré-estabelecido.

D. Caracteres

Uma fonte de caracteres é um conjunto de bitmaps que representam símbolos, para um determinado conjunto de códigos. Cada fonte representa unicamente um tipo de caracteres com um aspecto particular (altura, largura e outros atributos). O X Windows suporta fontes para conjuntos de caracteres com códigos de 8 e 16 bits (256 e 65536 caracteres possíveis). É fornecido um compilador de fontes e algumas fontes, num formato alto nível, afim de serem gerados ficheiros binários, utilizáveis pelo servidor.

E. Desenháveis

Um desenhável é um descritor de uma superfície na qual se podem desenhar gráficos, quer seja uma janela num ecrã ou um pixmap na memória. Um pixmap é um vector tridimensional de bits armazenado em memória de sistema, em vez de memória de visualização. Pode ser utilizado como origem ou destino de operações gráficas. Não existe nenhuma interpretação implícita dos pixels do pixmap, pois não existe nenhum visual ou mapa de cores que lhe esteja associado. A única informação existente é a profundidade do pixel; o pixmap só representa uma zona rectangular de pixels.

F. Janelas

Uma janela é um rectângulo visível, ou potencialmente visível, num qualquer ecrã, que serve para visualização de informação, quer seja gráfica, textual ou de outra natureza. A estrutura interna do **DIX** cria e controla as estruturas de dados do tipo árvore que representam as relações activas entre as janelas existentes. Janelas contidas noutra janela são designadas filhas e é feito o seu **clipping** segundo as fronteiras da janela ascendente. A janela raiz na referida árvore é a janela que representa todo o ecrã.

As fronteiras das janelas são desenhadas por rotinas **DDX**. O seu conteúdo é desenhado pelo cliente, através de pedidos feitos ao servidor. O mecanismo de "pintar" as janelas é comandado através de um sistema de acontecimentos de exposição. Quando uma região é exposta, o **DIX** gera um acontecimento do tipo exposição, explicando à aplicação que é necessário repintar a janela, e fornecendo-lhe as coordenadas da menor região que assegura a correcta regeneração.

Como um serviço extra, mas que é fundamental para um elevado desempenho, o servidor pode armazenar as alterações nas áreas escondidas em qualquer parte (por exemplo, memória de sistema); este mecanismo é designado **backing-store**. Quando uma parte de uma janela escondida é exposta, é então possível mover os pixels para a sua posição correcta, em vez de gerar um acontecimento de exposição e ter que esperar que o cliente responda. Mesmo que o tempo de resposta através da rede seja insignificante, o tempo de pintar uma área de uma janela é sempre maior do que copiar áreas já coloridas (ainda por cima, o hardware pode facilitar este tipo de operações). Mas, por outro lado, o backing-store pode ser complicado, porque os comandos gráficos que afectem áreas escondidas têm que ser interceptados e o seu efeito armazenado em qualquer parte do servidor, vulgarmente, memória de sistema. Em termos de programação, isto pode representar um esforço considerável, além de que também diminui ligeiramente o desempenho.

Quando uma operação sobre uma janela é requerida pelo cliente, um novo estado do servidor é determinado. Durante esta transição, o **DIX** informa o **DDX** de quais os rectângulos e em que janelas é que estão em vias de ser

obscurecidos, e que rectângulos e em que janelas é que vão ficar expostos. Este processo fornece pistas imediatas para a implementação do backing-store. Se o **DDX** não consegue regenerar as regiões expostas, então o **DIX** gera acontecimentos de exposição para os clientes. É da responsabilidade dos clientes regenerarem as áreas das janelas expostas e danificadas.

A parte não-dependente do servidor é que controla a árvore de janelas, embora hardware inteligente possa exercer algum controlo a partir de informação que o **DIX** coloca à disposição. A maior parte das rotinas associadas às janelas é endereçada a partir de apontadores, localizados em estruturas globais do **DIX**. Existem versões não-dependentes da maior parte destas rotinas, o que, à partida, permite utilizá-las sem grandes problemas. Servidores que suportem o mecanismo backing-store devem definir um conjunto de estruturas de dados especiais, declarar apontadores para rotinas específicas e controlar alguns campos das estruturas de dados associados às janelas, destinados a este fim. Basicamente, é preciso permitir que o hardware tenha sempre a possibilidade de interceptar as acções gráficas que envolvam áreas de pixels, de modo a que este se antecipe e execute as operações necessárias afim de se evitarem as exposições.

G. Contextos Gráficos

Um contexto gráfico contém variáveis de estado do servidor tais como côr de fundo do pixel activa, estilo de linha e largura activos, padrão activo, tipo de caracteres activa, e muitos outros atributos. Em muitos sistemas gráficos, o equivalente do contexto gráfico e do desenhável está combinado numa única entidade. A principal diferença entre ambos reside no facto do desenhável representar uma superfície de desenho e o que lá estiver desenhado, ao passo que o contexto gráfico descreve o processo de desenhar.

Ao contrário de muitos sistemas gráficos, não existe nada equivalente ao conceito de posição actual do cursor. Todas as operações gráficas são acompanhadas pelas coordenadas que lhes são relevantes.

O contexto gráfico também inclui os apontadores para as rotinas que desempenham as operações gráficas fundamentais (desenhar linhas, polígonos, arcos, etc). O **DDX** pode, se quiser ser eficiente, alterar dinamicamente estes apontadores, de modo a tirar partido do hardware/firmware subjacente ao servidor,

que em certas ocasiões realiza mais eficientemente as mencionadas operações. Assim, o **DDX** é avisado sempre que o cliente (ou **DIX**) vai usar um GC (contexto gráfico) modificado. Por exemplo, se o hardware possui características avançadas para o desenho de caracteres de largura fixa, o **DDX** pode interceptar alterações na selecção da fonte de caracteres activa, num dado GC, imediatamente antes de se realizar o desenho. Pode então modificar o GC para apontar para uma rotina especial de desenho de caracteres de largura fixa, utilizando o hardware para os desenhar, ou seleccionar uma rotina genérica de desenho de caracteres de largura variável, e que utiliza bitmaps através de software.

Durante a inicialização do ecrã, é necessário fornecer uma rotina de criação de GC. Na criação de um GC o ecrã deve definir alguns dados e determinar quais os mecanismos a serem invocados quando um GC for modificado. Também se indica para que alterações se deseja executar alguma acção. Quando um pedido de um cliente é processado e resulta na alteração do GC, a parte não-dependente do estado do GC é actualizada. Se o **DDX** requereu notificação para a modificação do GC, a rotina associada à modificação, previamente declarada, é invocada. Isto é obviamente útil para sistemas capazes de processar alterações de estado em paralelo com o processador do servidor. Por outro lado, o **DDX** pode não tomar nenhuma acção, se o quiser fazer:

A validação do GC ocorre na primeira operação gráfica que utilize esse GC, após ter sido alterado. Se existem alterações em campos do GC nos quais o **DDX** exprimiu interesse, a rotina associada à validação é invocada. Nesta altura, o **DDX** deverá actualizar o estado interno do GC. O estado interno de um GC pode ser armazenado como um ou mais blocos dos tipos:

- bloco privado de informação do GC
- estado do hardware
- alterações nos apontadores do GC.

3. Primitivas de Desenho

O protocolo do X (regras para a transferência de bytes entre cliente e servidor) define todas as acções gráficas através de operações elementares, denominadas primitivas de desenho. Qualquer implementação baseada neste servidor exemplo deve definir 16 primitivas para este fim.

Em termos mais concretos, cada GC deve possuir apontadores para 16 rotinas gráficas. Em qualquer momento, todos estes apontadores devem especificar uma rotina que execute a operação pretendida, embora possam ser alterados para apontar para outras rotinas que executem a "mesma" operação. É óbvio que uma implementação primária define somente 16 rotinas e deixa os apontadores dos GC's inalterados, enquanto que uma implementação mais "inteligente" permite comutar de apontadores (e rotinas), dependendo de qual fôr mais eficiente para o referido GC e para o desenhável activo.

O servidor base possui bastante código utilizável para implementar as primitivas de desenho (código **DDX - MI**). Dado que este código funciona para qualquer profundidade de pixel, é natural que seja imediatamente utilizável, na maioria das implementações. Não são feitas quaisquer suposições relativamente aos formatos dos **frame-buffers** e dos **pixmaps**, dado que toda a interacção é efectuada através de rotinas de baixo nível designadas **pixblit**. As rotinas **MI** assumem que todas as operações gráficas são realizáveis através da manipulação em bloco de linhas de pixels.

Com a excepção de uma primitiva de desenho, todas as restantes (15) são afectadas pelos campos dos GC's. Todas estas rotinas devem fazer **clipping** segundo as regiões activas do desenhável. Dado que podem existir muitas regiões para este efeito, a rotina de validação do GC deverá combinar todas estas regiões numa única, à qual as primitivas de desenho se deverão referir. As primitivas de desenho são:

CopyArea - copia um rectângulo de pixels entre desenháveis com a mesma profundidade, devendo ser capaz de copiar sobre o mesmo desenhável

CopyPlane - copia um plano de um rectângulo de um desenhável; dado que só copia um bit de cada pixel, esta rotina é capaz de copiar entre desenháveis de profundidade diferente (este é o único processo "legal" de realizar esta acção)

PolyPoint - desenha um conjunto de pontos de um pixel nas coordenadas fornecidas

PolyLines - desenha linhas entre as coordenadas fornecidas; linhas de largura "zero" são linhas que devem ser desenhadas o mais rapidamente possível, sem preocupações de "estética" ou qualidade

`PolySegment` - desenha linhas desligadas entre pares de coordenadas fornecidas

`PolyRectangle` - desenha contornos de rectângulos

`PolyArc` - desenha arcos de cónicas

`FillPolygon` - enche um polígono com o estilo de enchimento seleccionado (existem versões para polígonos convexos, não-convexos e "complexos")

`PolyFillRect` - enche vários rectângulos

`PolyFillArc` - enche uma área para cada arco fornecido

`PutImage` - copia um mapa de pixels para um desenhável

`GetImage` - copia os bits do desenhável origem para o endereço especificado pelo apontador de destino; os bits são escritos de acordo com as regras de alinhamento de pixmap que o servidor exige

`ImageText8` - desenha texto de modo *standard*

`PolyText8` - desenha texto usando o estilo de enchimento activo de modo a conseguir efeitos especiais tipo sombra, etc

`ImageText16`, `PolyText16` - semelhantes às anteriores, mas para caracteres com códigos de 16 bits.

As rotinas de desenho de primitivas devem ser obrigatoriamente fornecidas para cada servidor. Uma possibilidade de o fazer consiste em utilizar as rotinas **MI** do servidor base. Se estas forem utilizadas, no todo ou em parte, então devem ser implementadas as rotinas **pixblit**.

Estas rotinas leem e escrevem pixels e lidam directamente com a informação de imagem. Para o servidor base, estas estão implementadas no **MFB** (que é relativamente portátil). O cliente, o **DIX** e as rotinas não-dependentes deixam a responsabilidade de clipping para as rotinas **pixblit** (isto é, quem mexe no **frame-buffer** deve fazer clipping). Para facilitar as coisas, todas as regiões que têm que passar pelo clipping devem ser combinadas numa única região, ao nível da rotina

de validação de GC. É esta região que deverá ser processada pelas rotinas pixblit (o servidor base já faz isto).

Como com outras rotinas de desenho, existem campos nos GC's que informam as rotinas de mais alto nível de qual a rotina adequada, para uma dada funcionalidade (isto permite a optimização).

As rotinas pixblit estão divididas em três grupos. As rotinas `Spans` lidam simplesmente com linhas de pixels. As rotinas `Glyph` enchem células de caracteres. Por último, a rotina `PushPixels` é uma rotina de 3 operandos mais sofisticada que aplica um mapa de pixels através de uma máscara.

A. Rotinas Spans

Todas as operações gráficas baseiam-se em **spans**, que são blocos horizontais de pixels. Cada rotina necessita de um desenhável destino, um GC, o número de spans a processar e as coordenadas dos pontos. É nestas rotinas que inicialmente se deve investir em força. Aliás, no servidor base, existem muitas variantes destas rotinas, em que cada uma é ótima para um determinado conjunto de circunstâncias relacionado com o hardware.

`FillSpans` - enche linhas de pixels usando estilos, padrões, etc

`SetSpans` - para cada span, esta rotina copia bits para um desenhável destino, utilizando o **raster-op** activo do GC

`GetSpans` - recíproca da anterior.

B. Rotinas Glyph

Estas rotinas desenhavam células de caracteres (texto). É possível usar versões existentes no **MI** (muito lentas) ou adaptar as do **MFB**.

`PolyGlyphBlt`, `ImageGlyphBlt` - estas rotinas copiam os bitmaps para os desenháveis; a primeira é mais sofisticada, a segunda é mais rápida.

C. Rotina PushPixels

Esta rotina escreve uma determinada forma descrita por um bitmap no desenhável destino com o estilo de enchimento activo, filtrando-a através de um bitmap que serve de stencil. É utilizada como base para muitas outras operações gráficas (com as implicações de desempenho inerentes). Existe uma implementação **MI** e outra **MFB**.

4. Rotinas diversas

Alguns tipos de hardware exigem que se executem certas acções antes do servidor poder ser "desligado". Existem duas rotinas que é necessário implementar para este efeito, mesmo que não façam nada. É também necessário implementar duas rotinas para processar argumentos de linha que tenham sido fornecidos aquando da activação do servidor (mais para sistemas UNIX).

5. Conclusões

O processo de adaptação do código do servidor X pode variar, em função do hardware e sistema operativo, desde o relativamente simples (por exemplo, para cartas monocromáticas em máquinas UNIX BSD - basta adaptar o núcleo MFB ou o núcleo SPAN) até ao mais complexo (por exemplo, para cartas gráficas "inteligentes" em máquinas não-UNIX BSD) [ANG1] [ROSE2]. Se se pretende obter uma implementação do servidor X com um bom desempenho, então é óbvio que não basta modificar um núcleo básico de rotinas; é necessário descobrir as rotinas gráficas críticas⁵⁶ e tentar optimizá-las em função dos recursos do hardware.

⁵⁶e não só as gráficas...

A arquitectura do servidor X foi concebida de modo a isolar as partes que não se relacionam directamente, e ao mesmo tempo, separar as referências ao sistema operativo e ao hardware, agrupando as rotinas envolvidas em módulos específicos. No fundo, o servidor é um conjunto de módulos, uns mais **machine-independent** (DIX, por exemplo), outros relativamente independentes (DDX) e ainda outros totalmente dependentes do hardware e do sistema operativo.

Os diferentes módulos podem ser assim caracterizados, em termos de utilização/aproveitamento dos recursos existentes:

- **DIX** - 100% utilizável em qualquer sistema
- **OS** - muito específico (baixo aproveitamento se diferir razoavelmente de UNIX BSD)
- **DDX MI** - 100% utilizável em qualquer sistema
- **DDX CFB** - ainda que seja código desenvolvido para **frame-buffers** sem qualquer "inteligência" gráfica, o aproveitamento é bastante elevado
- **DDX MFB** - semelhante
- **DDX SNF** - (suporte de fontes de caracteres sobre ficheiros) semelhante ao OS
- **DDX "hardware"** - a implementar para cada caso.

V. Experiências de Implementação de um servidor X para MS-DOS

1. Introdução

Afim de se obter experiência na implementação do servidor X Windows, foi ensaiada essa tarefa numa máquina MS-DOS com uma carta de rede Ethernet, que possuía TCP e com uma carta gráfica monocromática de muito alta resolução (1200x1600 pontos), desenvolvida localmente.

Este capítulo descreve as experiências realizadas neste ambiente, cujo objectivo era analisar a portabilidade do código do servidor X de base, como preparação para o trabalho de implementação na carta gráfica ESTIMULO.

2. Arquitectura

No sistema PC-AT com MS-DOS, os principais factores a ter em conta eram os seguintes:

- Software para desenvolver um servidor X, escrito em C e para funcionar em ambientes UNIX ou muito compatíveis.
- Uma máquina com processador 80386 e 4 megabytes de memória.
- Uma carta de rede Excelan com processador próprio a funcionar no barramento do PC, sobre código que executava localmente e no 80386[EXC1] [EXC2].

- Uma carta monocromática com processador gráfico dedicado (TMS 34010) a funcionar sobre o barramento do PC.

Dado que o software de servidor X disponível só suportava três tipos de protocolo de rede (**sockets**⁵⁷ para UNIX, TCP e DECnet), e pelo facto de não existir o conceito de socket em MS-DOS, só restaria a possibilidade de utilizar o protocolo TCP, o que obrigaria desde logo a executar parte do software do servidor no 80386.

Por outro lado, executar parcialmente sobre o 80386 (a parte de rede) e sobre o TMS 34010 (a parte gráfica) iria dificultar enormemente as coisas, pois implicaria dividir as tarefas e mexer no código fonte do servidor de um modo radical. Além disso, a memória disponível na carta gráfica (512 kbytes) era manifestamente insuficiente para as necessidades do servidor (pelo menos 2 a 3 megabytes), pelo que a parte gráfica nunca poderia ser totalmente executada por este.

A solução óbvia seria então colocar o servidor X a funcionar totalmente sobre o 80386, utilizando o software da carta de rede, e aceder à carta gráfica no nível mais baixo, isto é, para escrever e ler pixels directamente na memória bitmapped (Figura 7).

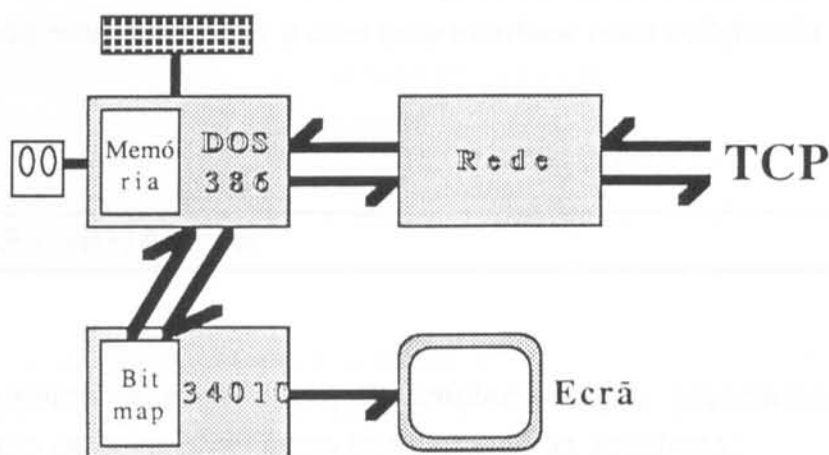


Figura 7

Acesso "directo" ao bitmap sem aproveitamento do TMS 34010

⁵⁷o conceito de socket é vagamente aparentado com o conceito de ficheiro, só que é possível, para além do escrever/ler, **listen** (escutar), **bind** (associar), **connect** (conectar), etc.

Uma outra possibilidade seria semelhante à anterior, mas em que o acesso à carta gráfica seria realizado através de primitivas, as quais executariam localmente sobre o TMS 34010, o que permitiria um maior desempenho do servidor. Estas primitivas estariam agrupadas num núcleo gráfico que aproveitaria as capacidades gráficas do TMS 34010 para executar as principais operações gráficas (linhas, texto, blocos, etc) (Figura 8).

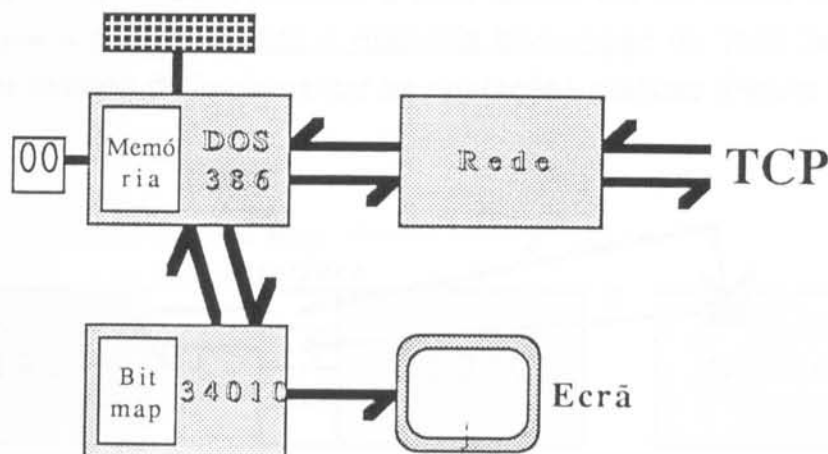


Figura 8

Acesso ao bitmap efectuado pelo TMS 34010 através de rotinas próprias e com uma interface mais sofisticada

3. Experiência

Relativamente à experiência de implementação propriamente dita, as dificuldades que se colocavam eram basicamente as seguintes:

- O sistema operativo não era UNIX nem compatível, pelo que, embora o compilador de C providenciasse alguma emulação de rotinas de sistema do UNIX, alguns aspectos importantes ficavam por resolver[BACH].
- Arquitectura 16 bits do sistema e do compilador de C, ao passo que o software do servidor assumia uma arquitectura de 32 bits (esta diferença é mais difícil de resolver do que parece à primeira vista).

- O código do servidor X foi escrito para cartas gráficas, monocromáticas ou não, que não possuíam quaisquer capacidades gráficas para além do acesso rudimentar ao pixel, o que neste caso implica desperdiçar as características gráficas do processador TMS 34010. Além disso, por exemplo, um acesso de leitura por parte do processador externo a uma posição de memória da carta gráfica exige, no caso mais favorável, uma operação de leitura no espaço de I/O sobre o barramento, e no mais desfavorável⁵⁸, duas escritas e uma leitura em I/O sobre o barramento, pelo que o acesso directo à memória bitmapped do TMS 34010 não é o melhor método de implementar as operações gráficas (Figura 9).

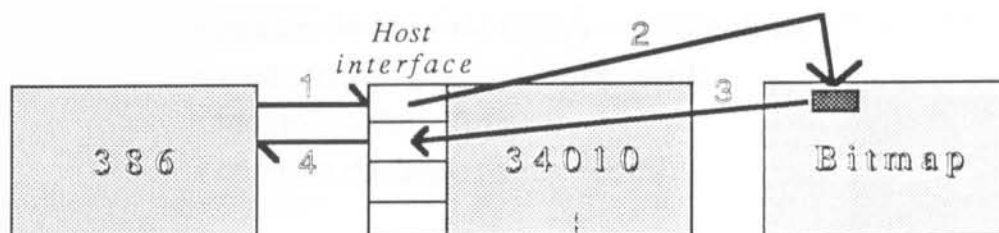


Figura 9

Exemplo de leitura do bitmap pelo host:

- 1 - escrita no registo de interface **HSTADDR**
- 2 - acesso do 34010 ao bitmap para ler valor
- 3 - valor depositado no registo de interface **HSTDATA**
- 4 - leitura do registo de interface **HSTDATA** para obter valor

Depois de analisadas cuidadosamente estas dificuldades, a experiência de implementação foi dividida em três fases:

⁵⁸que é muito mais provável...

1º Adaptação do código que envolvia quaisquer ligações com o UNIX, quer pelo facto de não haver funcionalidade correspondente, ou então, pelo facto de a funcionalidade pretendida ser diferente (coisas tão aparentemente triviais como ordem dos argumentos diferente, menos ou mais argumentos, etc). Esta tarefa obrigou ao estudo cuidadoso do código fonte do servidor, em termos das rotinas de sistema emuladas ou de novas rotinas de sistema fornecidas pelo compilador de C, pelas suas bibliotecas de rotinas e pela biblioteca de rotinas de carta Ethernet, relativamente ao protocolo TCP. Como resultado, a maior parte do código ficou intacto, embora algumas particularidades mais sofisticadas do UNIX não tenham sido imediatamente abordadas, tais como:

- gestão de base de dados - rotinas **dbm**;
- multiprocessamento - **fork, wait**;
- temporização - **timers**;
- controlo de dispositivos - **ioctl**.

Ao nível do protocolo TCP, a biblioteca de rotinas C da carta Ethernet apresentava uma compatibilidade bastante elevada, e embora algumas particularidades mais sofisticadas não existissem e quase todas as rotinas tivessem nomes diferentes e modos de invocação incompatíveis, a funcionalidade mínima foi mantida.

2ª A tarefa de adaptar código intencionalmente escrito para uma arquitectura de 32 bits para uma arquitectura de 16 bits revelou-se extremamente complexa, devido a várias razões. Os compiladores de C para arquitecturas de 32 bits possuem os seguintes tamanhos em bits para os tipos escalares **short int**, **int**⁵⁹ e **long int**: 16, 32 e 32. Porém, os compiladores de C disponíveis para MS-DOS possuem os tamanhos respectivos de 16, 16 e 32, ou seja, o comprimento do tipo **int** é diferente[KERN] (Figura 10).

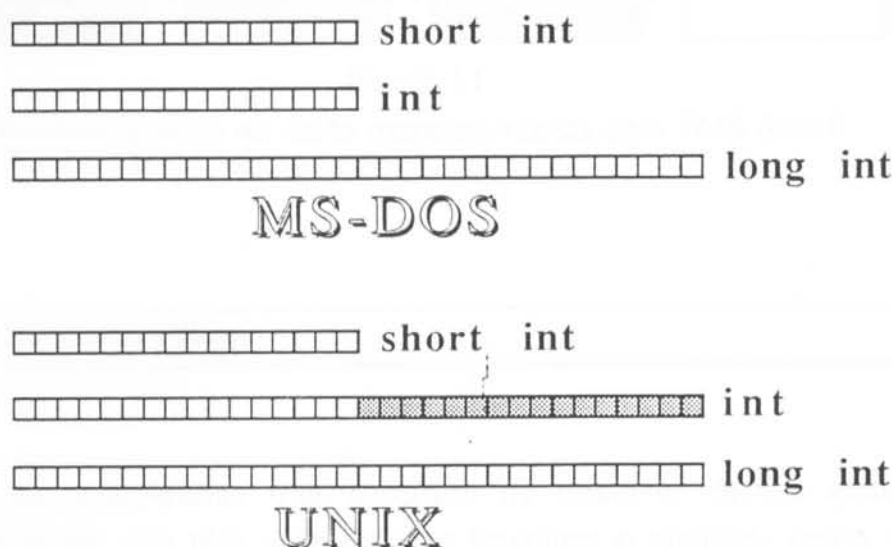


Figura 10

Comparação dos comprimentos em bits dos tipos inteiros de C em MS-DOS e UNIX

Infelizmente, muitas rotinas de sistema do UNIX que também existem em MS-DOS utilizam este tipo, o que cria logo inúmeros problemas de compilação e outros que não são detectáveis em tempo de compilação. Por outro lado, a arquitectura de 16 bits dos compiladores de C coloca graves problemas a nível da alocação de memória, isto é, os blocos de memória alocados não podem exceder 64 kbytes de comprimento e não existem mecanismos de gestão de memória virtual sobre disco; este problema, em alguns casos, não pode ser resolvido sem envolver uma modificação quase total de algumas partes do servidor.

⁵⁹geralmente, dado que é dependente do compilador.

3º A implementação do núcleo elementar de rotinas para acesso ao pixel e à linha foi abordada; esta tarefa não é complexa, na medida em que as mencionadas rotinas são bastante simples e exigem pouca sofisticação. É óbvio que o preço a pagar por esta simplicidade é ter-se um servidor X lento e pouco eficiente (Figura 11).

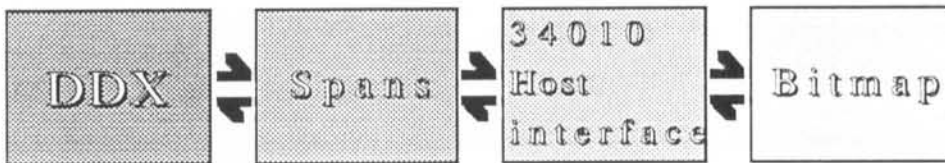


Figura 11

Pipeline gráfico na carta monocromática com TMS 34010

4. Conclusões

Devido às dificuldades que surgiram no decorrer destas experiências, algumas das quais não têm uma solução imediata e simples, estes testes de implementação foram interrompidos sem que se conseguisse obter um servidor pronto a funcionar. Aliás, não era esse o objectivo principal, mas antes avaliar a tarefa de implementação de um servidor X, enquanto o hardware necessário não estivesse disponível.

Algumas partes duma possível implementação estão relativamente funcionais, nomeadamente a parte de rede TCP e a parte gráfica de baixo nível, mas outras partes ainda se encontram longe de poder funcionar o mínimo que é necessário.

Como resultado destas experiências conclui-se que o código de servidor X Windows disponível não é adequado a máquinas com arquitecturas que não sejam de 32 bits[ROSE2] e cujo sistema operativo seja muito diferente do UNIX BSD, quer pelas próprias rotinas de sistema serem muito diferentes (como é o caso do MS-DOS, onde a maioria das rotinas de sistema UNIX não têm correspondência), quer pelas características intrínsecas do próprio sistema operativo (note-se que o MS-DOS é claramente orientado para funcionar em 16

bits). Em termos positivos, assinala-se a portabilidade da maior parte do código do servidor, especialmente da camada DIX, onde quase não foi necessário fazer alterações (a alteração mais frequente estava associada à instrução switch, a qual em UNIX permite utilizar cases com quantidades de 32 bits, e em MS-DOS tal só é permitido com quantidades de 16 bits; a solução mais fácil foi a utilização conjunta do pré-processador de C e da substituição do switch por if else's...[KERN]).

VI. Servidor X para carta gráfica do Projecto ESTIMULO

1. Introdução

O projecto ESTIMULO⁶⁰ pretende criar uma estação gráfica multimédia integrando voz, imagens, gráficos, etc. O hardware básico é idêntico ao oferecido por estações gráficas actualmente existentes, ao qual serão adicionados os dispositivos necessários para o tratamento de imagem, vídeo, voz e outros. O sistema operativo será um sistema operativo distribuído orientado para objectos e com elevado grau de compatibilidade UNIX.

A estação ESTIMULO é baseada numa estação INTEL, com sistema operativo UNIX System V (versão 3) e com barramento de interface MULTIBUS II.

O hardware de visualização é constituído por uma carta gráfica dotada com um processador gráfico evoluído (GSP TMS 34020), coprocessador numérico (TMS 34082), memória gráfica de visualização 1280x1024 pontos, 256 cores à escolha entre 16 milhões, memória de sistema (4 megabytes), controlador de interface MULTIBUS II (MPC), controlador de comunicações série (SCC 82530), interface com PC XT/AT e restante hardware (Figura 12).

⁶⁰ESTIMULO quer dizer ESTação Integrada MULTimédia

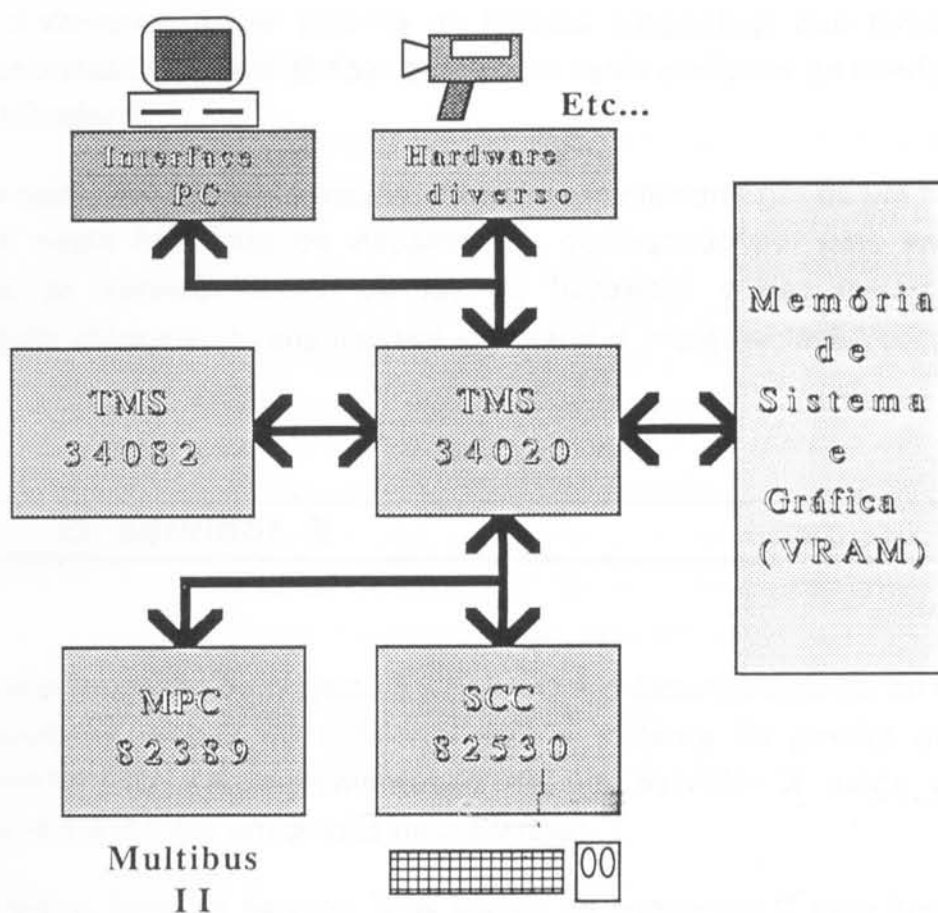


Figura 12

Diagrama de blocos da carta gráfica ESTIMULO

Esta estação possuirá uma interface gráfica sofisticada e configurável, baseada num sistema de gestão de janelas avançado. Dos sistemas de gestão de janelas existentes, de entre os quais alguns dos mais significativos foram referidos anteriormente, o sistema X Windows é aquele que funciona em maior número de plataformas diferentes, embora em quase todas com sistemas operativos UNIX ou compatíveis.

A combinação mais frequente em estações gráficas é actualmente o UNIX/X Windows:

- O sistema operativo UNIX é poderoso e versátil.

- O X Windows é um sistema de janelas extensível que fornece muita funcionalidade, sem obrigar a qualquer estilo particular de interface com o utilizador.

Este capítulo descreve o estudo, projecto e implementação de um servidor de janelas X neste hardware de visualização, começando por uma análise dos requisitos do servidor X em termos do hardware e das ferramentas de programação disponíveis, arquitectura adoptada e implementação/adaptação do código.

2. O servidor X

Um dos requisitos do projecto ESTIMULO é o desenvolvimento de uma carta gráfica evoluída, capaz de funcionar sob o sistema de gestão de janelas X Windows[ROSE1]. A implementação de um servidor X exige a análise cuidadosa dos aspectos envolvidos nessa tarefa:

- O código fonte do servidor X foi escrito na linguagem C para funcionar em sistemas UNIX BSD, sobre cartas gráficas tipo **frame-buffer** sem capacidade local de processamento e suportando o mecanismo socket para UNIX, TCP e DECnet.
- O processador TMS 34020 é um processador de arquitectura 32 bits, com diversas capacidades gráficas sofisticadas, mas capaz de funcionar como um processador normal[TEX2], interligado com o coprocessador numérico TMS 34082 microprogramado, o qual poderá apresentar características gráficas avançadas.
- Hardware diverso capaz de aumentar o desempenho gráfico, nomeadamente, uma paleta gráfica para gestão de cores, controlador de cursor gráfico (capaz de suportar autonomamente cursores **bitmapped** com tamanhos até 64x64 pontos), etc.
- A ligação entre a carta gráfica e a estação de base efectua-se através do barramento MULTIBUS II, capaz de atingir débitos de transmissão elevados[INTE1] [INTE2].

- Ferramentas de desenvolvimento para o TMS 34020, incluindo compilador de C, linker, assembler, bibliotecas de suporte, etc[TEX1].

Desde logo, alguns problemas se colocavam na implementação de um servidor X sobre este ambiente:

- Inexistência de suporte UNIX na carta gráfica; as bibliotecas de suporte do sistema de desenvolvimento fornecem diversas rotinas que emulam rotinas de sistema UNIX, mas muitos aspectos importantes não estão contemplados, tais como gestão de memória virtual e multiprocessamento.
- Inexistência de qualquer protocolo de comunicação entre a carta gráfica e a estação base; embora exista um controlador de comunicações para o MULTIBUS II, capaz de substituir eficientemente o processador central, nenhum dos protocolos suportados pelo servidor X existe inicialmente.
- Não utilização dos recursos sofisticados de processamento da carta gráfica pelo servidor X, o que envolveria a modificação do código do servidor.

O servidor X⁶¹ de base, tendo sido desenvolvido com a preocupação de fornecer um meio rápido de adaptar hardware gráfico para funcionar no sistema X Windows, está dividido em diversos blocos funcionais, os quais pretendem isolar entre si diversas particularidades de hardware, sistema operativo, etc. Basicamente, este servidor manipula janelas e operações gráficas simples, substituindo o utilizador, em nome das diversas aplicações que estão em execução. As suas tarefas principais são[JONE]:

- 1º Responder aos pedidos do protocolo vindos das aplicações clientes (geralmente operações gráficas e de desenho de texto).
- 2º Enviar informação de dispositivos de entrada (teclas activadas e acções do rato) e outros acontecimentos para os clientes.
- 3º Manter as conexões com os clientes funcionais.

O código do servidor está dividido nos seguintes blocos principais:

⁶¹a versão do servidor utilizada foi a versão 11 (*Release 3*).

- **DIX**; código independente de quaisquer dispositivos gráficos e comum entre todas as implementações.
- **OS**; código que difere em função do sistema operativo, mas que é partilhado por todas as diferentes implementações realizadas sobre o mesmo sistema operativo. É formado por um grupo reduzido de rotinas, muito específicas para cada sistema operativo. As suas tarefas são:
 - Gestão das conexões com os clientes.
 - Divisão do trabalho a efectuar pelos clientes.
 - Interface lógica com os ficheiros de descrição de caracteres.
 - Gestão de memória ao nível mais elementar.
- **DDX**; código que é eventualmente diferente para cada combinação de dispositivos gráficos e sistema operativo. Este bloco esconde todas as complexidades do hardware de visualização e dos periféricos e realiza a interacção através de dois dispositivos lógicos (teclado e ponteiro). As suas tarefas são diversas:
 - Gestão da entrada de informação através dos dispositivos virtuais teclado e ponteiro.
 - Gestão dos ecrãs a um nível intermédio⁶², incluindo mapas de cores, cursores, bitmaps, pixmaps, janelas, caracteres, etc.
- **Interface de Extensões**; código que permite a adição de nova funcionalidade ao servidor.

Os blocos de que devem ser "transportados" para novos ambientes de hardware/software são os blocos OS e DDX; estes blocos funcionam ao mesmo nível e nenhum se baseia no outro. O bloco DIX é (supostamente) totalmente operacional em todos os sistemas e não precisará de ser modificado. O bloco de extensões não é relevante para esta implementação, para já.

3. Arquitectura

⁶²o acesso directo ao hardware é escondido através de um núcleo de rotinas.

Afim de resolver de uma forma satisfatória os problemas anteriormente mencionados, e tentando evitar, na medida do possível, modificar o código fonte do servidor X[ROSE2], foi decidido implementar emulações sobre a carta gráfica de mecanismos essenciais para a utilização do servidor X. Para conseguir tais objectivos, foi necessário estudar o servidor X em termos de dependência com rotinas de sistema UNIX.

Dado que o servidor está funcionalmente dividido em blocos, é mais conveniente analisar cada bloco, na medida do possível, independentemente dos restantes:

Bloco **DIX** - Este bloco é descrito como sendo portátil para qualquer sistema operativo. Nenhuma alteração foi efectuada.

Bloco **DDX** - Este bloco **device-dependent** realiza a interface gráfica de baixo nível com o hardware de visualização. Na primeira versão, esta interface é feita através de rotinas básicas, tais como:

- `GetSpans()` - esta rotina lê do frame-buffer uma linha de pixels, sendo capaz realizar algumas operações booleanas e outras;
- `FillSpans()` - escreve no frame-buffer uma linha de pixels, sendo igualmente capaz de efectuar operações sobre esses pixels;
- `PushPixels()` - esta rotina aplica sobre o frame-buffer uma determinada cor, padrão, etc, através de um bitmap que serve de máscara durante a operação (existe uma versão portátil desta rotina implementada sobre `FillSpans()`).

Bloco **OS** - Na versão existente do servidor, este bloco serve de suporte a algumas funções importantes, o que introduz uma dependência indesejável do servidor em relação ao sistema operativo. Basicamente, colocam-se duas hipóteses: alterar o código do servidor, adaptando-o às características do hardware, ou então emular, sobre o hardware, o sistema operativo, de tal modo que não seja necessário alterar profundamente o código do servidor. A segunda hipótese foi escolhida, porque envolve menos riscos e exige menos modificações⁶³[ANG1].

4. Implementação

A implementação deste servidor foi dividida em várias fases, abordando-se inicialmente os aspectos mais relacionados com o hardware:

1ª Implementação de mecanismos básicos.

2ª Comunicação - *Byte stream* (protocolo de transmissão através do MULTIBUS II) e emulação de *sockets*.

3ª Emulação do sistema operativo: ficheiros, alocação de memória, etc.

4ª Implementação das rotinas gráficas elementares DDX.

5ª Integração de janela vídeo.

1. Mecanismos básicos

Como já foi anteriormente referido, afim de se poder utilizar no máximo possível o código do servidor X disponível, foi necessário implementar diversos

⁶³Assim evita-se alterar código que já está completamente operacional.

mecanismos básicos e rotinas correspondentes para se conseguir ter uma emulação minimamente suficiente de sistema operativo UNIX BSD[BACH]. Em muitos casos, a emulação não é mais do que definir rotinas que não fazem nada, pelo menos na fase inicial do trabalho, sendo posteriormente modificadas de modo a executarem com a funcionalidade exigida. É evidente que não é possível implementar todo o **kernel** do UNIX sobre o hardware da carta gráfica, mas também tal não é necessário. As dependências em relação ao UNIX situam-se apenas em certas áreas específicas e a funcionalidade requerida pode ser obtida (simulada) de vários modos.

Os seguintes módulos foram implementados/modificados:

- Módulo **gsp** - rotinas de muito baixo nível para o GSP TMS 34020. Estas rotinas lidam directamente com todo o hardware, sendo a maior parte delas executada na inicialização do sistema. Todas as rotinas deste módulo são extremamente dependentes do hardware, nomeadamente do processador gráfico[TEX2].

Exemplo: `InitializeGSP()` - rotina de inicialização principal, que atribui valores aos registos do TMS 34020 (vídeo, de controlo, máscaras de interrupção, etc).

Este módulo é responsável pela instalação de **interrupt drivers**, especialmente o mecanismo de interrupção vertical do TMS 34020 (DPYINT), o qual serve para diversos fins (**timers**, relógio do sistema, etc). Este mecanismo é activado programando-se um registo do TMS 34020 com um valor correspondente à linha de varrimento vídeo em que se pretende interromper o processador, o qual executará uma rotina de interrupção associada. Dado que a frequência de varrimento vertical é de 50 Hz, então existirão 50 interrupções por segundo, o que corresponde a uma resolução temporal de 20 ms; embora seja um valor elevado, serve razoavelmente para os propósitos pretendidos (relógio de tempo real, **polling** periódico, etc).

- Módulo **scc** - este módulo contém as rotinas e as estruturas de dados de suporte do controlador de comunicações série (SCC) da carta, sendo por isso igualmente muito dependente do hardware. Inicializa devidamente a interface série (1 ou 2 canais até 19200 baud), o **interrupt driver** associado ao SCC e define rotinas para o processamento de combinações especiais de teclas, que vão simular outros mecanismos (referidos mais adiante).
- Módulo **stdio** básico - o objectivo é a emulação de **streams** tais como `stdin`, `stdout`, `stderr` mais rotinas de sistema `printf()`, `fprintf()` e `perror()`. Estas rotinas utilizam a interface AT (ou um canal do controlador de comunicações série da carta) para enviar os seus resultados, que são então recebidos no PC por um programa de monitorização da interface, que depois imprime as mensagens no ecrã.

No servidor, estas rotinas são utilizadas somente para escrita de mensagens de erro e de diagnóstico, basicamente, mas nesta implementação são extensivamente utilizadas para diagnosticar a evolução do sistema (note-se que este é o único processo de se saber o que o TMS 34020 está a fazer!)[TEX2].

- Módulo **time** - módulo com rotinas que devolvem informação acerca de tempo (tempo em milisegundos desde a activação, etc). Estas rotinas foram quase todas implementadas em muito baixo nível e são importantes, por exemplo, na medida em que a camada DDX têm que "carimbar" todos os acontecimentos gerados com o "tempo" (relativo ou absoluto) em que ocorreram, e é essencial que seja o mais possível exacto afim de que posteriormente se possam processar esses acontecimentos pela ordem correcta.

- Módulo **signal** - realiza a emulação de **signals** UNIX (o **signal** UNIX pode ser definido como uma excepção - interrupção provocada pelo utilizador, violação de memória, divisão por zero, etc)[BACH].

No sistema UNIX é possível associar dinamicamente rotinas com quase todas as excepções, o que permite uma grande funcionalidade. Em virtude das diferenças existentes entre o hardware necessário para um sistema UNIX e o que existe na carta gráfica, nem todos os **signal handlers** foram suportados, mas somente os essenciais (**SIGTERM**, **SIGKILL**, etc).

Alguns **signals** foram implementados directamente sobre o módulo **scc**, isto é, podem ser simulados através de combinações especiais de diversas teclas (Figura 13).

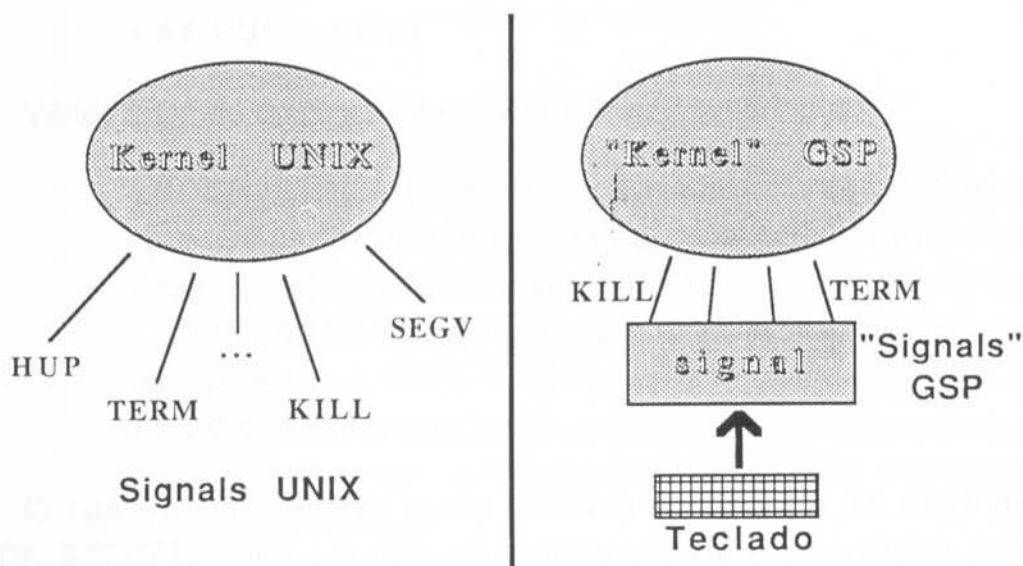


Figura 13
Emulação de **signals**

2. Comunicação - *Byte stream*

A comunicação entre a carta gráfica e a estação far-se-á através de uma emulação mínima de sockets UNIX sobre o MULTIBUS II, para a qual diversas

rotinas UNIX BSD foram implementadas (`connect()`, `read()`, `write()`, `writev()`, `select()`, etc). Dos três tipos de socket que o servidor suporta de base, foi escolhido o socket UNIX, na medida em que o seu código é mais simples e mais fácil de modificar[BACH].

Basicamente, o socket é uma entidade que permite a comunicação entre processos com um nível de abstracção muito elevado, quase como se fosse através de ficheiros, por exemplo. Existem diversos formatos para a comunicação com sockets:

- **AF-UNIX**, que utiliza nomes de ficheiros à maneira UNIX, isto é, o socket é representado por "ficheiros" de UNIX
- **AF-INET**, que utiliza endereços ARPA Internet para estabelecer conexões
- **AF-PUP** e outros.

Vários tipos de comunicação são igualmente suportados:

- **SOCK_STREAM**, canais bidireccionais fiáveis orientados ao byte que garantem a recepção sequencial correcta da informação (semelhante a **pipes** UNIX)
- **SOCK_DGRAM**, canais orientados ao byte em modo datagrama
- **SOCK_RAW** e outros.

O tipo de comunicação a ser utilizado no servidor da carta gráfica é o **SOCK_STREAM**, dado que este era o único suportado pelo código original (além disso, é mais simples a recepção de informação neste caso do que nos outros).

A conexão entre sockets é realizada através da rotina de sistema `connect()`; após o estabelecimento da ligação, podem ser usadas as rotinas `read()` e `write()`⁶⁴, ou então as rotinas mais específicas `send()` e `recv()` (que o servidor não utiliza). Para terminar a conexão, deve ser invocada a rotina `close()`. Existem também mecanismos especiais para a transmissão assíncrona de informação, mas aos quais o servidor base não recorre.

⁶⁴as rotinas `read()` e `write()`, num sistema com sockets, suportam simultaneamente leitura/escrita de ficheiros e leitura/escrita de sockets de uma maneira transparente para os utilizadores...

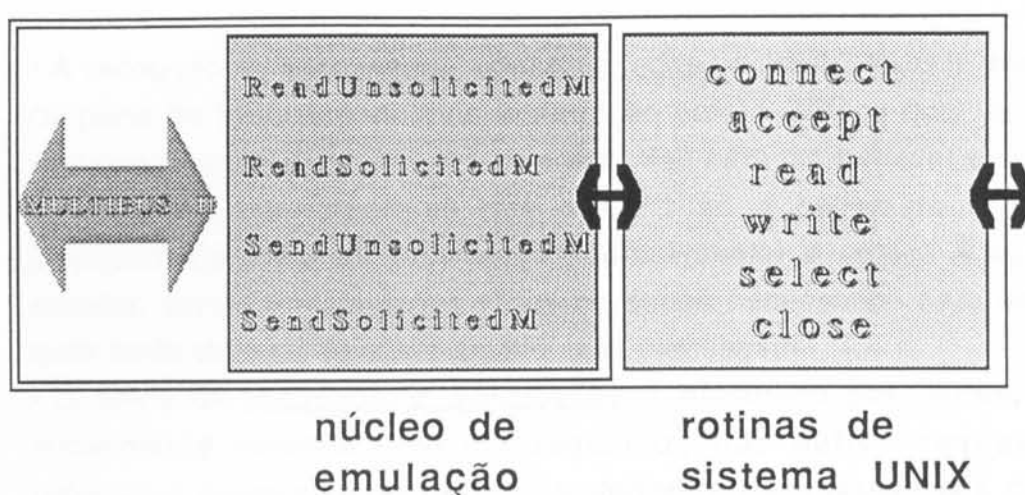


Figura 14
Comunicação com **sockets**

Os seguintes módulos foram criados/adaptados:

- Módulo **mpc** - este módulo efectua a gestão do controlador de MULTIBUS II (MPC)[INTE2]. Implementou-se o acesso ao MPC em baixo nível (inicialização de registos, etc), um **interrupt driver** para recepção de mensagens MULTIBUS II não-solicitadas mais estruturas de dados apropriadas (**buffers** de entrada e saída, filas de mensagens, etc) e as rotinas básicas de envio e recepção de mensagens (solicitadas e não-solicitadas)[INTE1] (Figura 14).

Estas rotinas já não lidam directamente com o MPC, embora ainda sejam demasiado dependentes do hardware para poderem ser utilizadas ao nível das camadas DDX ou OS.

Foram também implementados o interrupt driver para controlo dos acessos ao espaço de interconexão do MULTIBUS II e as rotinas de emulação propriamente ditas (`read()`, `write()`, `writew()`, `select()`, etc)[BACH].

O funcionamento deste módulo é descrito em seguida:

- O envio de mensagens não-solicitadas no MULTIBUS II é efectuado pelo GSP através do envio para o MPC dos parâmetros necessários e do comando de transmissão, sendo este repetido pelo MPC se ocorrerem



erros (na transmissão, por exemplo).

- A recepção de mensagens não-solicitadas no MULTIBUS II despoleta, da parte do MPC, um sinal de interrupção para o GSP, o qual vai o mais rapidamente possível instruir o referido MPC para lhe fornecer o conteúdo dessa mensagem (note-se que o MPC só é capaz de armazenar internamente uma só mensagem não-solicitada) e coloca-a numa fila circular, sendo armazenados todos os dados necessários para se poder mais tarde determinar quem enviou essa mensagem.

- O envio de mensagens solicitadas é efectuado por partes, sendo inicialmente enviada uma mensagem de tipo **buffer request** para notificar o receptor e só depois os dados (a transmissão dos dados é realizada recorrendo-se a uma instrução **bitblt** do GSP, a instrução **FILL L**, que permite a transferência rápida dos pacotes de 32 bytes para o buffer de entrada do MPC)[INTE2].

- A recepção de mensagens solicitadas é iniciada através da recepção prévia de uma mensagem tipo **buffer request**, à qual se segue o envio de uma mensagem **buffer grant** para confirmação, e só depois a recepção dos dados (novamente se emprega a instrução **bitblt FILL L** para se obter numa só operação os pacotes de 32 bytes do MPC).

Estas 4 operações estão encapsuladas basicamente nas rotinas mencionadas na Figura 14 (`ReadUnsolicitedMessage`, `ReadSolicitedMessage`, `SendUnsolicitedMessage` e `SendSolicitedMessage`), as quais servem de esqueleto às "falsas" rotinas de sistema.

- Módulo **connection** (adaptado do servidor) - este módulo foi modificado de modo a funcionar sobre a emulação de sockets UNIX acima mencionada. Algumas modificações foram feitas em virtude dos sockets não serem totalmente verdadeiros.
- Módulo **access** (adaptado do servidor) - implementa as rotinas que controlam o acesso de outras máquinas em rede ao servidor. Basicamente, muitas das rotinas deste módulo passaram a ser **stubs**, pois nesta fase inicial não existe rede e não existem verdadeiros sockets UNIX.

- Módulo **io** (adaptado do servidor) - este módulo define basicamente as duas rotinas utilizadas pelo **DIX dispatcher** para ler e escrever informação dos clientes (através dos sockets). Dado que foram implementados sockets mais as rotinas que lidam directamente com eles, poucas alterações foram feitas.
- Módulo **waitfor** (adaptado do servidor) - contém a rotina que o **DIX dispatcher** invoca e que suspende a execução do servidor, ficando este então à espera de:
 - dados vindos dos clientes
 - acontecimentos de entrada disponíveis
 - qualquer notificação do DDX
 - dados prontos a enviar para os clientes.

Como esta rotina depende unicamente, em termos de sistema operativo, da rotina de sistema `select()`, que já foi implementada no módulo **mpc**, este módulo não foi modificado.

A interacção entre os módulos acima mencionados é resumida na Figura 15, em que o módulo **mpc** é o módulo de mais baixo nível e os restantes aparecem englobados na camada **OS**.

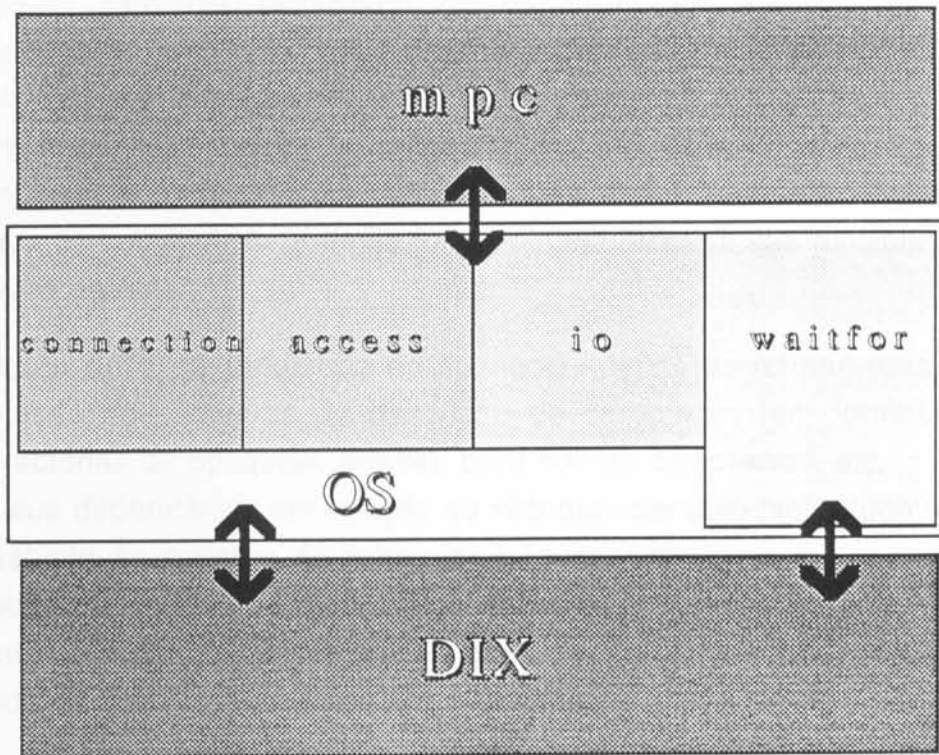


Figura 15
Estrutura dos módulos de comunicação

3. Sistema operativo

Embora a emulação de sistema operativo já tenha sido afluada anteriormente, este capítulo descreve outros módulos criados e adaptados do servidor que tiveram de ser implementados afim de se poder utilizar o código do servidor de base. Alguns destes módulos não exigem sequer um conhecimento muito aprofundado do hardware[ANG1].

- Módulo **osinit** (adaptado do servidor) - este módulo era responsável pela inicialização da camada OS, mas no caso presente, ficou bastante simplificado. Note-se que não existe um verdadeiro sistema operativo...

- Módulo **oscolor** (adaptado do servidor) - implementa um pequeno sistema de gestão de base de dados para utilização de nomes de côres em vez dos tripletos RGB de valores entre 0 e 255. Esta gestão era efectuada com recurso às rotinas de sistema **dbm** do UNIX BSD, mas foi necessário implementar uma emulação muito elementar destas rotinas para as substituir (o que não foi difícil pelo facto da sua funcionalidade ser muito básica).
- Módulo **osfonts** (adaptado do servidor) - define as rotinas para gestão e acesso dos ficheiros de descrição de caracteres (em formato SNF⁶⁵), directorias de pesquisa, **alias**es para nomes de ficheiros, etc.
A sua dependência em relação ao sistema operativo (sobretudo no que diz respeito ao sistema de ficheiros) é muito grande. A maior parte destas rotinas baseia-se num conjunto básico de rotinas de sistema UNIX, mas que obviamente não existem, pelo que, para se evitar reescrever este módulo, restam duas alternativas imediatas:

- emular as rotinas de sistema UNIX utilizadas e um sistema de ficheiros mínimo sobre a memória da carta
- substituir o núcleo básico por outro que não use ficheiros nem rotinas de sistema UNIX.

Foi escolhida a 1ª solução, porque implicava novamente fazer menos modificações no código base do servidor. Da memória de sistema da carta gráfica, aproximadamente 1 megabyte é reservado para simular um sistema de ficheiros semelhante ao do UNIX, implementando-se as rotinas necessárias para funcionar sobre este sistema virtual (nas fases iniciais de teste este módulo não vai ser considerado e só serão suportados alguns tipos de caracteres, sem recurso a ficheiros de fontes de caracteres).

⁶⁵o formato SNF (*server natural format*) é um formato binário de ficheiro orientado de acordo com as características do hardware ao **byte**, **word** ou à **longword** (alinhamento que provoque o melhor desempenho possível).

- Módulo **file** (criado) - implementa as rotinas de acesso a ficheiros, utilizando para esse efeito um bloco de memória de sistema (aproximadamente 1 megabyte) como **file system** virtual, sobre o qual é possível abrir, ler, escrever e remover "ficheiros"[BACH] (Figura 16). A funcionalidade deste módulo é, nesta fase, ainda muito elementar; o único interesse, para já, é poder utilizar as rotinas SNF e os ficheiros de descrição de caracteres em formato SNF (os quais são inicialmente despejados para a memória "file system"). No futuro, este módulo será alterado de modo a ser eventualmente possível obter mais ficheiros em formato SNF, vindos através da rede.

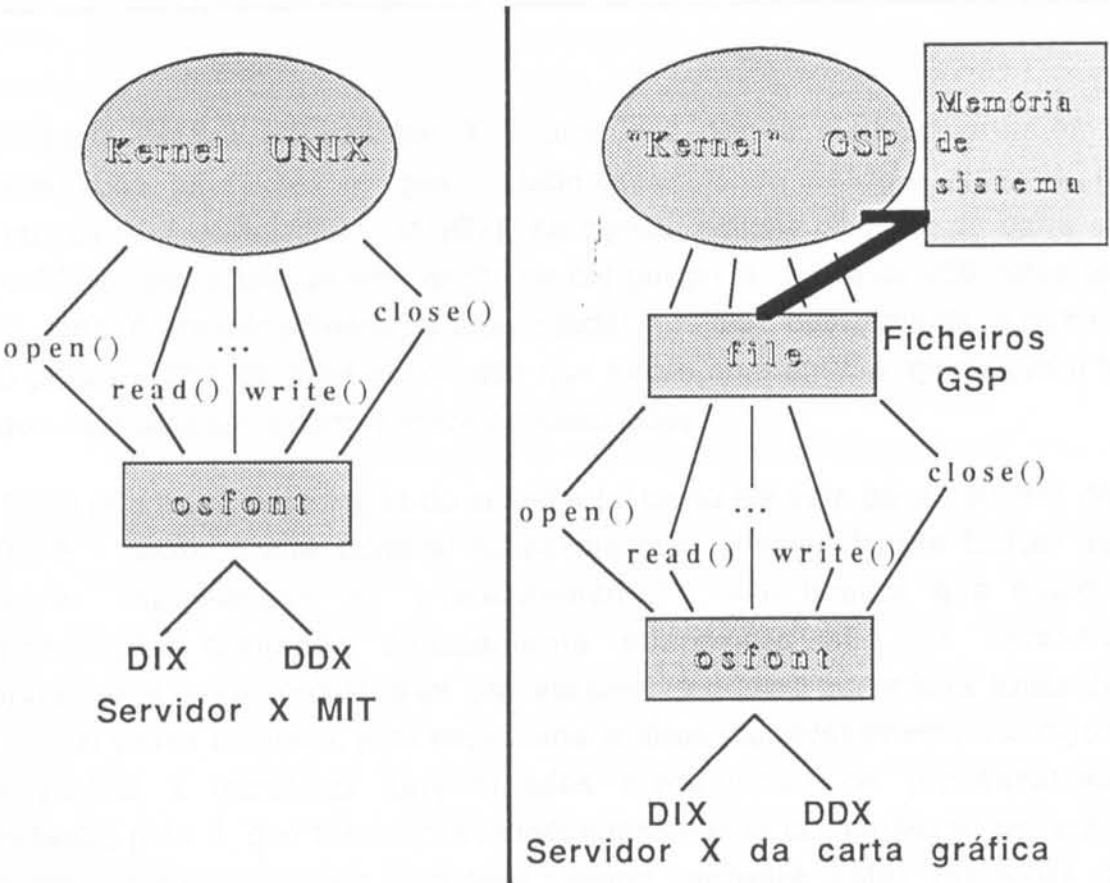


Figura 16

Organização dos módulos de gestão de caracteres

- Módulo **utils** (adaptado do servidor) - implementa diversas rotinas de utilidade geral tais como:

- impressão de mensagens de erro e outras
- interrupção completa do servidor
- obtenção do tempo do sistema à maneira UNIX
- alocação e dealocação de memória a um nível elevado (estas funcionam sobre as rotinas de gestão de memória de baixo nível do GSP já mencionadas).

4. Rotinas gráficas

A parte gráfica do servidor X é uma das partes mais importantes da implementação, na medida em que um bom desempenho gráfico é essencial, ou, pelo menos, desejável[ANG1] [ANG2]. As características gráficas da carta são 1280x1204 pixels, 8 bits de informação de cor por pixel, paleta de 256 cores com 24 bits cada (8 por cor primária) e processador "gráfico", basicamente, o que não sendo propriamente do mais sofisticado que existe hoje em dia, ainda assim faz com que seja uma carta com razoáveis capacidades.

Dado que a implementação do servidor tal como ele vem para sistemas de 8 bits (**CFB** - *colour frame buffers*) se baseia num simples **frame-buffer** sem quaisquer capacidades de processamento⁶⁶, isso implica que qualquer implementação realizada directamente sobre ela não irá aproveitar vantajosamente outros recursos de processamento gráfico porventura existentes. Para utilizar esses recursos, será necessário analisar cuidadosamente o código da parte gráfica e modificar determinados mecanismos de processamento generalizado (isto é, que funcionam independentemente do hardware) por outros especificamente relacionados com esse mesmo hardware. Estas alterações não obrigariam necessariamente a "deitar fora" tudo o que existe; pelo contrário, dado que a parte gráfica do servidor está implementada de um modo modular e hierarquizado, em que rotinas de complexidade e funcionalidade crescente se baseiam noutras mais simples, então seria eventualmente possível substituir, a

⁶⁶para além do acesso ao pixel e à linha.

diversos níveis, certos mecanismos por outros mais específicos e mais rápidos, sem que fosse necessário alterar o restante (exemplo: o desenho de segmentos de recta é efectuado na versão normal do servidor invocando uma rotina de nível médio dentro do DDX que cumpre os seus objectivos através de várias chamadas a rotinas de baixo nível que lidam directamente com os pixels - porém, se existir a capacidade no hardware de desenhar esses segmentos rapidamente, nem que seja somente um tipo específico entre todos os tipos possíveis, então poder-se-á unicamente modificar a rotina de nível médio para explorar os casos especiais que o hardware suporta e nos outros, proceder como anteriormente).

Nesta fase inicial, o módulo CFB é integralmente aproveitado sem modificações, pelo que será de esperar que o desempenho gráfico não seja o melhor possível (o que não quer dizer que seja baixo)[ROSE2]. Para atingir este objectivo, foi unicamente necessário implementar três rotinas muito simples e dependentes do hardware pelas quais é feita a interface entre o referido hardware e o DDX:

- `GetSpans()`, que acede directamente ao bitmap e devolve um bloco horizontal de pixels
- `FillSpans()`, que escreve no bitmap um bloco horizontal de pixels
- `PushPixels()`, que aplica um bloco rectangular de pixels sobre o bitmap, eventualmente através de uma máscara, etc⁶⁷.

Devido ao facto de a carta gráfica possuir um processador (TMS 34020) capaz de realizar acções gráficas muito eficientemente através das suas instruções gráficas especiais e também um coprocessador avançado (TMS 34082) que suporta diversas operações 2D e 3D directamente, parece óbvio que uma implementação da parte gráfica do servidor X baseada numa interface tão elementar não aproveitará praticamente nenhum dos recursos sofisticados deste conjunto (porém, dado que a maior parte da implementação da camada DDX utiliza aritmética *floating-point* de precisão dupla, a existência de um coprocessador numérico é imediatamente uma vantagem, pelo que não se pode dizer que esteja totalmente desaproveitado).

⁶⁷ existe uma implementação de `PushPixels()` feita à custa de `GetSpans()` e `FillSpans()`, mas é provavelmente demasiado lenta.

Uma dificuldade importante numa adaptação inteligente do servidor ao hardware acima descrito reside no facto de as modificações a realizar se situarem a diversos níveis na estrutura hierarquizada do DDX, e isto explica-se pela diversidade de operações gráficas que o hardware é capaz de fazer.

As instruções de natureza eminentemente gráfica que o TMS 34020 suporta são[TEX2]:

Baixo nível

- **LINE, FLINE e LINIT** - implementam uma parte importante do algoritmo de Bresenham de desenho de segmentos de recta
- **DRAV** - instrução de desenho de pixel e avanço para o "seguinte"
- **PIXT** - transferência de pixel
- **PIXBLT, VBLT** - transferência de blocos rectangulares de pixels
- **FILL, VFILL, PFILL** - enchimento de áreas de pixels com um determinado valor
- **TFILL** - semelhante ao anterior, mas para áreas trapezoidais

Médio nível

- **CLIP** - filtra uma área em relação a uma janela rectangular
- **CPW** - compara um ponto em relação a uma janela

A seguinte tabela descreve mais detalhadamente as capacidades de cada instrução:

Instrução	Window check	Transparency	Plane mask	Pixel process.
PIXT	√	√	√	√
DRAV	√	√	√	√
LINE	√	√	√	√
FLINE	-	√	√	√
PIXBLT	√†	√†	√†	√†
VBLT	-	-	√††	-
FILL	√	√	√	√
PFILL†††	√	√	√	√
VFILL	-	-	√††	-
TFILL	√	√	√	√
CLIP	√			
CPW	√			

† - com algumas restrições em função do tipo específico de operandos

†† - a máscara de planos é a da VRAM

††† - é capaz de aplicar directamente um padrão de comprimento até 32 bits

Analisando esta tabela, conclui-se imediatamente que muitas rotinas gráficas de médio e alto nível podem beneficiar, em termos de desempenho, da utilização destas capacidades do TMS 34020. Algumas rotinas ou operações possíveis são:

- Desenho de linhas com qualquer padrão e largura zero (esta rotina é muito utilizada por qualquer servidor X).
- Transferências de blocos rectangulares de pixels (e não só) na memória gráfica e também na memória de sistema (e entre ambas), com possibilidade de processamento dos pixels de origem e de destino, etc (esta operação é também das mais executadas por qualquer servidor X).
- **Clipping** (a operação de clipping é executada em muitas situações e em diversos níveis pelo servidor; em muitos casos, a utilização desta capacidade do TMS 34020 é óbvia).

- Desenho de caracteres em modo emulação (neste modo, pretende-se que o servidor desenhe os caracteres o mais rapidamente possível sem preocupações de estilo, etc).

Outro aspecto importante é o da visualização do cursor. A solução genérica que o servidor utiliza não é obviamente a mais rápida possível, mas é aquela que funciona em todos os tipos de **frame-buffer** sem hardware inteligente; esta solução envolve processamento quer a nível do DDX quer quase a nível do DIX, pelo que é complexa e lenta. Dado que a carta gráfica dispõe de circuitos integrados para gestão de cursor (com tamanho até 64x64 pixels, sendo a informação do cursor sobreposta à informação que vem da memória bitmapped à entrada do canal vídeo), esta tarefa pode ser realizada facilmente quase ao nível do hardware (assim, o problema de retirar e depois colocar o cursor aquando da execução de qualquer operação gráfica praticamente desaparece do bloco DDX).

5. Janela vídeo

Uma das características inovadoras desta carta gráfica é a existência de uma janela vídeo (256x256x24 bits)[JPP]. O facto desta janela ser armazenada numa zona de memória própria, diferente da memória gráfica normal, e de ser misturada com o conteúdo da memória gráfica à entrada do canal vídeo simplifica consideravelmente alguns aspectos da sua integração com o servidor X (refira-se que se pretendem evitar quaisquer modificações a nível do DIX, as quais poderiam comprometer futuras versões).

Basicamente, existem dois modos de integrar esta janela especial na hierarquia de janelas do X Windows (Figuras 17 e 18).

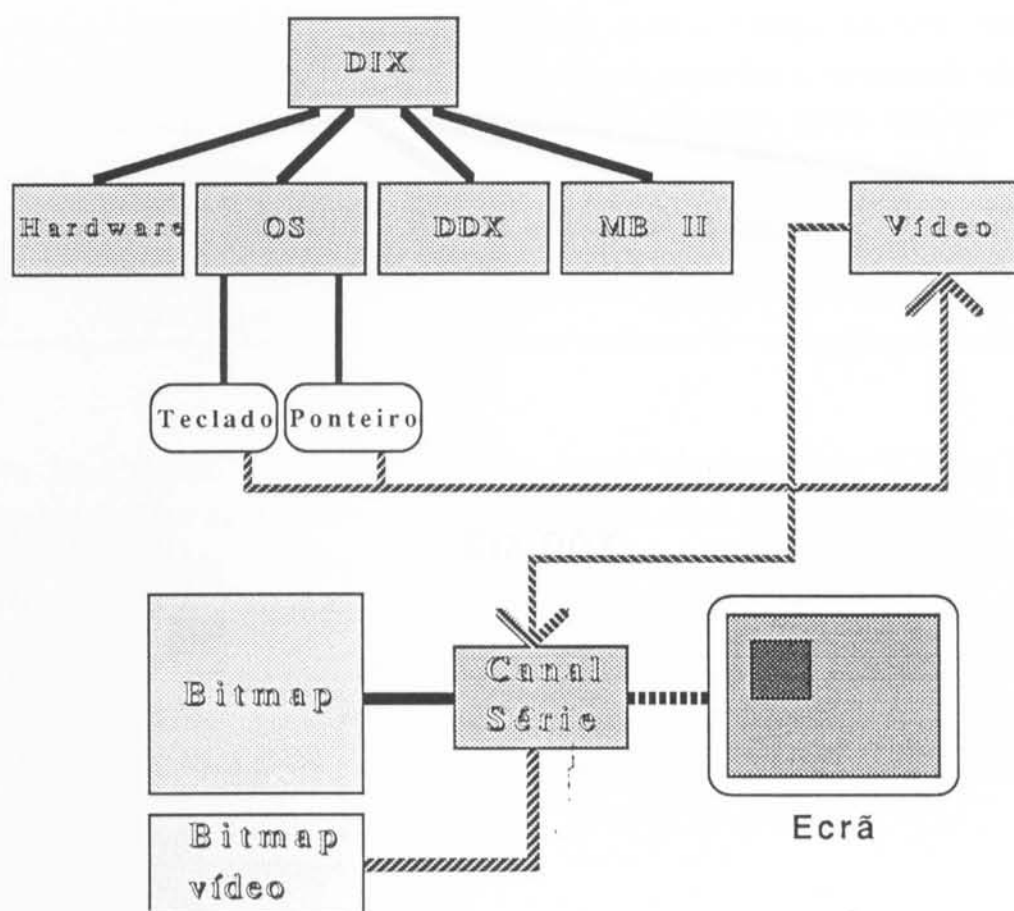


Figura 17

Integração de janela vídeo independente do sistema de gestão de janelas

Ao nível do módulo de gestão de teclado e ponteiro são reconhecidas certas combinações de teclas e/ou botões, as quais comandam a manipulação da janela vídeo (visibilidade, posição, tamanho, etc). Nesta situação, o servidor X desconhece totalmente esta janela, a qual só pode existir por cima das janelas normais ou não existir. Este é o meio mais simples de integrar esta janela vídeo no sistema de gestão de janelas, e além disso, permite visualizar o seu conteúdo em tempo real, sendo por isso a solução adoptada, para já (Figura 17).

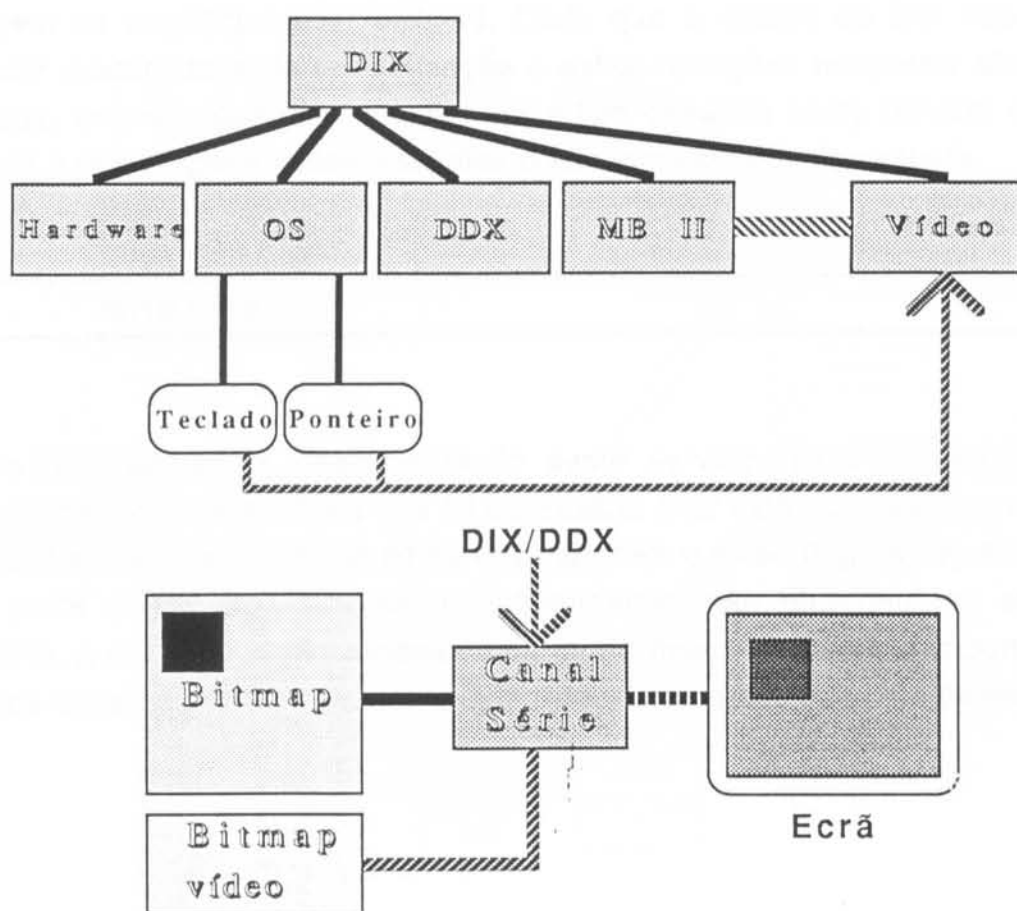


Figura 18

Integração de janela vídeo dentro do sistema de gestão de janelas

Em paralelo com o módulo de gestão do MULTIBUS II **mpc**, é implementado um módulo de gestão da janela vídeo (o qual é comandado a partir de combinações de teclas e botões processados pelo módulo **scc**), o qual injecta "falsos" comandos de protocolo X relacionados com a janela vídeo, que se comporta assim como mais um cliente que quer a sua janela visualizada algures na hierarquia de janelas existentes[SCHE3] (Figura 18).

O maior problema desta solução é a actualização (e consequente visualização) do conteúdo da janela em tempo real e com a taxa correcta, pois tal implica ciclicamente enviar o seu conteúdo através do protocolo para ser processado pelo DIX e afixado no ecrã. Como a visualização é feita pelo menos a 12 imagens por segundo, então é necessário enviar o seu conteúdo pelo protocolo, processá-lo pelo DIX/DDX/etc e afixá-lo em menos de 1/12 do segundo,

além de manter a sincronização entre duas imagens sucessivas (afim de evitar que a imagem se modifique aos "saltos"). Dado que o código do DIX não utiliza quaisquer mecanismos de temporização e estas restrições temporais são muito apertadas, antevejo que uma solução deste tipo baseada neste servidor e neste hardware e que respeite essas restrições dificilmente será implementada.

5. Análise

Na fase actual da implementação deste servidor para a carta gráfica, praticamente todos os módulos que foi necessário criar estão completos e aqueles que foram adaptados do código do servidor também o estão (Figura 19). Embora a maior parte tenha sido testada individualmente (não directamente sobre o hardware), a sua integração constitui uma tarefa muito complexa e importante, e que pode tornar visíveis problemas que individualmente não são detectáveis.

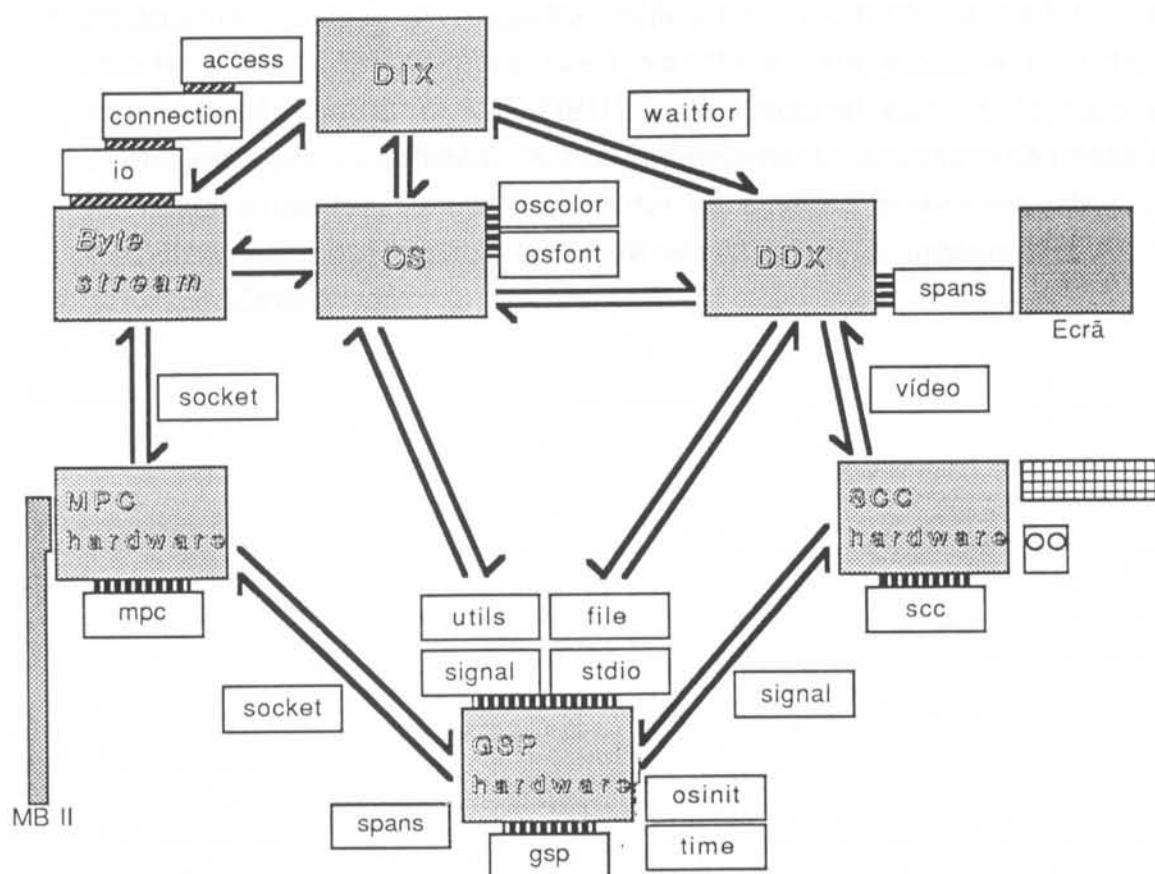


Figura 19
Arquitectura do servidor X (fase actual)

Pelo facto do hardware da carta gráfica não estar totalmente funcional, não é possível, nesta fase, dizer até que ponto esta implementação de um servidor X está operacional. Para realizar o teste do servidor, diversas estratégias são possíveis, verificando-se, sempre que possível, cada módulo independentemente:

- Testar o comportamento do servidor relativamente aos módulos de interacção do teclado e do rato, e algumas operações gráficas essenciais (criação de janelas, cursor, primitivas tipo linha, arco, etc).
- Simular o MULTIBUS II através de um programa que envie comandos do protocolo X, à escolha do utilizador, controlável a partir do PC de suporte, sendo estes comandos injectados ao nível do módulo **mpc** (que seria ligeiramente adaptado para esta estratégia) e analisar a resposta do servidor.

- Implementar do lado da estação gráfica uma extensão à biblioteca de interface Xlib (ou até uma nova biblioteca), de modo a suportar a comunicação sobre o MULTIBUS II, de acordo com o formato de mensagens que foi definido. Depois de implementada, seria então possível ter nesta altura ter clientes a executar na estação gráfica em UNIX e a visualizar na carta gráfica em X Windows sobre a implementação do servidor[SCHE2].

VII. Conclusão

O objectivo deste trabalho era a implementação de um servidor X Windows numa carta gráfica, desenvolvida localmente, que será posteriormente integrada numa estação UNIX com barramento MULTIBUS II.

Embora essa implementação pudesse ser totalmente realizada somente a partir das especificações do protocolo X, essa estratégia não foi adoptada em virtude de exigir que o trabalho fosse iniciado do nada[ANG1], ao passo que utilizando a versão do MIT só seria, em princípio, necessário adaptar determinadas partes mais dependentes e específicas para se obter um produto minimamente funcional, sem que se tivesse que passar por todas as dificuldades e problemas que este tipo de tarefa levanta.

Apesar desta estratégia, ainda assim a maior parte do trabalho foi empregue a contornar os problemas relacionados com o sistema operativo e a sua relação com o hardware e não a resolver os problemas de natureza gráfica que uma implementação optimizada levaria a supôr. Na realidade, a adaptação da parte gráfica do código do servidor, na sua versão mais básica, é extremamente simples e rápida; para versões optimizadas, a adaptação é obviamente mais complexa.

O código adaptado para esta implementação está praticamente desenvolvido nas suas linhas gerais (isto é, todos os módulos projectados já existem e já estão minimamente operacionais), mas tal não quer dizer, infelizmente, que a implementação esteja testada e isenta de erros. Por indisponibilidade da carta gráfica quase desde o início, sobre a qual só se fizeram alguns testes muito rudimentares de compilação e de benchmark, este trabalho não foi suficientemente testado afim de se poderem corrigir os erros e resolver os problemas, que forçosamente terão de existir. Aliás, a fase de integração e teste é essencial neste trabalho, porque só ela permite determinar as partes da implementação cujo desempenho é menos satisfatório, e consequentemente, fazer com que sejam analisadas e substituídas por outras mais eficientes.

Logo que o hardware esteja 100% funcional será então possível completar e melhorar esta implementação, desde que, evidentemente, nessa altura ainda se justifique essa tarefa.



VIII. Bibliografia

ADOB

Adobe Systems "POSTSCRIPT Language Reference Manual"
Addison-Wesley, 1985.

ANG1

Angebrannt, S., Drewry, R., Karlton, P., Newman, T. "**Strategies for Porting the X V11 Sample Server**" Massachusetts Institute of Technology, 1988.

ANG2

Angebrannt, S., Drewry, R., Karlton, P., Newman, T. "**Definition of the Porting Layer for the X V11 Sample Server**" Massachusetts Institute of Technology, 1988.

BACH

Bach, Maurice "An Introduction to the UNIX™ Operating System"
Prentice-Hall, 1987.

BRES

Bresenham, J. "Algorithm for Computer Control of a Digital Plotter"
IBM Systems Journal 4, nº 1, páginas 25 a 30, 1965.

ENGL

English, W., Englebart, D. Berman, M. "**Display Selection Techniques for Text Manipulation**" *IEEE Transactions on Human Factors in Electronics*, volume 8, nº 1, páginas 21 a 31, 1977.

EXC1

Excelan "EXOS 8052-01 TCP/IP Network Software For IBM PC running DOS - Reference Manual Rel. 3.2.3" Excelan, 1986.

EXC2

Excelan **"EXOS 205 Intelligent Ethernet Controller For IBM PC - Board Rev. F Reference Manual"** Excelan, 1986.

FOL1

Foley, J. e Van Dam, A. **"Fundamentals of Interactive Computer Graphics"** Addison-Wesley, Reading, Massachussets, 1982.

FOL2

Foley, J., Van Dam, A., Feiner, S., Hughes, J. **"Computer Graphics, Principles and Practice"** Addison-Wesley, Reading, 1990.

GETT

Gettys, J. **"Problems Implementing Windows Systems on UNIX"** *Usenix Proceedings*, Denver, Colorado, 1986.

HARR

Harris, D. **"Computer Graphics and Applications"** Chapman and Hall, Londres, 1984.

INTE1

INTEL Corporation **"82389 MPC MULTIBUS II Bus Interface Controller"** 1987.

INTE2

INTEL Corporation **"MPC User's Manual"** 1986.

JAMS

Jamsa, K. **"WINDOWS Programming Secrets"** McGraw-Hill, 1988.

JONE

Jones, O. **"Introduction to the X Window System"** Prentice Hall, 1989.

JPP

Pereira, J. P. **Tese de Mestrado** Porto, Fevereiro de 1991.

KAY1

Kay, A. **"Microelectronics and the Personal Computer"** *Scientific American*, Setembro, 1977.

KAY2

Kay, A., Goldberg, A. **"Personal Dynamic Media" Computer**, Março, 1977.

KERN

Kernigham, B., Ritchie, D. **"The C Programming Language"** Prentice-Hall, 1978.

LEMM

Lemmons, P. **"An Interview: the Macintosh Design Team"** *BYTE*, Fevereiro, 1984.

MAC1

"Inside Macintosh: Volume I" Addison-Wesley, 1988.

MAC2

"Inside Macintosh: Volume II" Addison-Wesley, 1988.

MAC3

"Inside Macintosh: Volume III" Addison-Wesley, 1988.

NEWM

Newman, W. e Sproull, R. **"Principles of Interactive Graphics"** McGraw-Hill, Nova Iorque, 1979.

ROSE1

Rosenthal, D. e Gosling, J. **"A Window Manager for Bitmapped Displays and UNIX"**, de *"Methodology of Window Managers"*, F. Hopgood, Holanda, 1986.

ROSE2

Rosenthal, D., de Boor, A. **"Godzilla's Guide to Porting the X V11 Sample Server"** Sun Microsystems, 1989.

SCHE1

Scheifler, R., Gettys, J. **"X Window System - The Complete Reference to Xlib, X Protocol, ICCCM, XFLD"** Digital Press, 1990.

SCHE2

Scheifler, R., Gettys, J **"Xlib - C Language X Interface"**, 1989.

SCHE3

Scheifler, R. **"X Window System Protocol, Version 11 (Alpha Update)"** Massachusetts Institute of Technology, 1987.

SMIT

Smith, D. et al **"Designing the Star User Interface"** *BYTE*, Abril, 1982.

SUN1

Sun Microsystems **"Sample NeWS 1.1 Manual"** 1988.

SUN2

Sun Microsystems **"Programmer's Reference Manual for Sun Windows"** 1985.

SUTH1

Sutherland, I., Sproull, R. **"A Clipping Divider"** *Fall Joint Computer Conference*, Washington, 1968.

SUTH2

Sutherland, I. **"A Man-Machine Communication System"** *Fall Joint Computer Conference*, Washington, 1968.

TESL

Tesler, L. **"The Smalltalk Environment"** *BYTE*, Agosto, 1981.

TEX1

Texas Instruments **"TMS 34010 C Compiler Reference Guide"** 1989.

TEX2

Texas Instruments **"TMS 34020 User's Guide"** Texas Instruments Graphics Products, 1990.

WADL

Wadlow, T. **"The Xerox Alto Computer"** *BYTE*, Setembro, 1981.

WARN

Warnock, J. e Wyatt, D. **"A Device Independent Graphics Imaging Model for Use with Raster Devices"** *Computer Graphics*, volume 16, nº 3, Julho, 1982.

WILL

Williams, G. "The Apple Macintosh Computer" *BYTE*, Fevereiro, 1984.