

Faculdade de Engenharia da Universidade do Porto
Mestrado em Engenharia Electrotécnica e de Computadores

Interfaces Gráficas com o Utilizador

Contribuição para a Concepção em Ambientes de Fabrico

Cristina Russo

1995

Interfaces Gráficas com o Utilizador

Contribuição para a Concepção em Ambientes de Fabrico

Faculdade de Engenharia da Universidade do Porto
Mestrado em Engenharia Electrotécnica e Computadores

Interfaces Gráficas com o Utilizador

Contribuição para a Concepção em Ambientes de Fabrico

Dissertação de Mestrado submetida à
Faculdade de Engenharia da Universidade do Porto

UNIVERSIDADE DO PORTO
Faculdade de Engenharia
BIBLIOTECA

Cristina Russo

1995

681.3(043)/RUSC/INT
UNIVERSIDADE DO PORTO
Faculdade de Engenharia
BIBLIOTECA
N.º 53249
CDU
Data 12 07 2000

Agradecimentos

Desejo expressar o meu agradecimento ao orientador desta tese, Professor Adriano Carvalho, pela disponibilidade e apoio científico prestado.

Estou especialmente grata ao Raúl Oliveira por ter lido versões preliminares desta tese e fornecido comentários de elevado valor. Agradeço-lhe ainda o carinho demonstrado durante a escrita deste trabalho.

Quero também agradecer ao Pina Martins e ao Filipe Pacheco por terem sido incansáveis na ajuda prestada.

Agradeço a todos os meus colegas do Grupo de Automação Industrial do Instituto de Sistemas e Robótica.

Expresso o meu agradecimento ao Instituto de Sistemas e Robótica pela disponibilização de meios materiais.

Estou também reconhecida ao Programa PRAXIS XXI pelo apoio financeiro concedido durante a fase final da realização deste trabalho.

Finalmente, quero agradecer aos meus pais e à Menó por todo o apoio que me deram.

Resumo

As interfaces gráficas com o utilizador representam um grande avanço no sentido de facilitar a interacção homem-computador. No entanto, as interfaces gráficas com o utilizador são com frequência piores do que poderiam de facto ser: muitas vezes são pensadas *a posteriori* e podem ser difíceis de compreender, fazendo com que o utilizador cometa erros desnecessários.

Esta tese explora os aspectos fundamentais subjacentes à concepção de interfaces gráficas com o utilizador para aplicações genéricas de uma forma sistemática. O caso particular das interfaces gráficas para ambientes de fabrico também é contemplado.

A utilização de metáforas e a sua validade é discutida, as principais características que as interfaces gráficas devem possuir são descritas e os factores humanos são abordados do ponto de vista da caracterização dos utilizadores, para aplicações de âmbito genérico. Uma análise similar é realizada para as interfaces gráficas em ambientes de fabrico, equacionando os requisitos particulares que estas colocam. Linhas mestras que devem reger a concepção de interfaces gráficas são fornecidas e discute-se da sua aplicabilidade às interfaces gráficas em ambientes de fabrico.

Técnicas importantes no desenvolvimento de interfaces gráficas são descritas. As ferramentas de suporte ao desenvolvimento de interfaces gráficas são analisadas e são fornecidos exemplos para os sistemas baseados em janelas mais correntes. O sistema X-Windows é apresentado com algum detalhe. O teste de interfaces gráficas com e sem o envolvimento dos utilizadores também é abordado, e as potencialidades das ferramentas de teste automático são analisadas. Finalmente, referem-se formas de providenciar documentação ao utilizador e estratégias para assegurar da sua eficácia.

Índice

Resumo *vii*

Capítulo 1 Introdução *1*

1.1 Motivação da autora *4*

1.2 Estrutura da tese *5*

Capítulo 2 Conceitos fundamentais *7*

2.1 Modelos conceptuais e metáforas *9*

2.1.1 Utilização de metáforas *11*

2.1.2 Classificação de metáforas *14*

2.2 Características de uma interface gráfica com o utilizador *15*

2.2.1 Transparência *15*

2.2.2 Consistência *16*

2.2.3 Funcionalidade *16*

2.2.4 Flexibilidade *17*

2.2.5 Controlo pelo utilizador *18*

2.2.6 Fiabilidade *19*

2.2.7 Facilidade de aprendizagem *19*

2.3 Tipos de GUIs *20*

2.3.1 WYSIWYG *20*

2.3.2 Manipulação directa *21*

2.3.3 Icónica *22*

2.4 Caracterização dos utilizadores *23*

2.5 Conclusões *24*

Capítulo 3 GUIs dedicadas a ambientes de fabrico *27*

3.1 Aspectos relativos à interacção humana *29*

3.1.1 Utilizador-tipo *29*

3.1.2 Metáforas em ambientes de fabrico *31*

3.1.3 Linguagem do utilizador *33*

3.1.4 Características das GUIs em ambientes de fabrico *34*

3.2 Participação dos utilizadores no desenvolvimento *37*

3.2.1 Factores político-sociais *38*

3.3 Requisitos e visualização da informação *39*

3.3.1 Entidades físicas *40*

3.3.2 Processos *42*

3.3.3 Dados históricos *46*

3.4 Constrangimentos relativos ao ambiente *46*

3.4.1 Concorrência de processos *47*

3.4.2 Limitações a nível da interacção *48*

3.5 Conclusões *49*

Capítulo 4 Metodologias na concepção de GUIs *51*

4.1 Processo de desenvolvimento *53*

4.1.1 Definir quem vai usar o sistema e para fazer o quê *53*

4.1.2 Escolher tarefas representativas *54*

4.1.3 Copiar *54*

4.1.4 Criar uma descrição inicial da GUI *54*

4.1.5 Analisar detalhadamente a descrição inicial *55*

4.1.6 Construir um protótipo *55*

4.1.7 Testar a GUI com os utilizadores *55*

4.1.8 Iterar *56*

4.1.9 Construir a GUI *56*

4.1.10 Manter o contacto com os utilizadores *56*

4.1.11 Construir novas versões *57*

4.2 Linhas mestras para GUIs *57*

4.2.1 Regras da Xerox Star *58*

4.2.2 Princípios gerais *61*

4.2.3 Utilização de cor em GUIs *70*

4.2.4 Utilização de som em GUIs *71*

4.3 Linhas mestras na concepção de GUIs em ambientes de fabrico *74*

4.4 Conclusões *76*

Capítulo 5 Desenvolvimento de GUIs *79*

5.1 Arquitectura de uma GUI *80*

5.1.1 Modelo MVC *80*

5.2 Tecnologia cliente/servidor *85*

5.2.1 Teoria *85*

5.2.2 Prática *86*

5.2.3 Características *88*

5.2.4 A tecnologia cliente/servidor na concepção de GUIs *92*

5.3 Tecnologia orientada por objectos *93*

5.3.1 Desenvolvimento de *software* *95*

5.3.2 Análise *99*

5.3.3 Projecto *100*

5.3.4 Programação *100*

5.4 Conclusões *102*

Capítulo 6 Ferramentas de suporte ao desenvolvimento de GUIs *105*

6.1 Programação visual *106*

6.2 Classificação das ferramentas *108*

6.3 Sistemas integrados de desenvolvimento *110*

6.3.1 Arquitectura de um UIMS *110*

6.3.2 Técnicas genéricas de desenvolvimento *112*

6.3.3 O que procurar num UIMS *114*

6.4 Ferramentas específicas de suporte ao desenvolvimento de GUIs *116*

6.4.1 X-Windows, OSF/Motif e AIC *117*

6.4.2 Macintosh e HyperCard *127*

6.4.3 Windows e Visual Basic *130*

6.5 Conclusões *133*

Capítulo 7 Teste e documentação de GUIs *135*

7.1 Teste de GUIs *136*

7.1.1 Sem os utilizadores *136*

7.1.2 Com os utilizadores *138*

7.1.3 Teste automático de *software* *142*

7.2 Documentação *144*

7.2.1 Manuais *145*

7.2.2 Ajuda *on-line* *146*

7.2.3 Treino *147*

7.2.4 Apoio ao utilizador *147*

7.3 Conclusões *148*

Capítulo 8 Conclusões *149*

8.1 Conclusões *149*

8.2 Perspectivas de trabalho futuro *152*

Referências *155*

Acrónimos *161*

MARCAS REGISTADAS

AIX e IBM são marcas registadas da IBM Corporation.
Apple, Apple II e Apple Macintosh são marcas registadas da Apple Computer Inc.
DEC e DECnet são marcas registadas da Digital Equipment Corporation.
HP-UX é uma marca registada da Hewlett-Packard Company.
HyperCard e Macintosh são marcas registadas da Apple Computer Inc.
Intel é uma marca registada da Intel Corporation.
Microsoft, MS e Microsoft Windows são marcas registadas da Microsoft Corporation.
Open Look é marca registada da American Telephone & Telegraph Inc.
OSF e Motif são marcas registadas da Open Software Foundation.
SUN e SunOS são marcas registadas da Sun Microsystems Inc.
UNIX é uma marca registada da Unix System Laboratories.
Visual Basic é uma marca registada da Microsoft Corporation.
Windows é uma marca registada da Microsoft Corporation.
X-Windows é uma marca registada do Massachusetts Institute of Technology.
Xerox é uma marca registada da Xerox Corporation.

Capítulo 1

Introdução

Quando a escrita foi inventada, há cerca de 5000 anos, a história da humanidade modificou-se radicalmente, uma vez que permitiu ao homem fazer registos das suas leis, civilizações, histórias e cálculos. As ideias também passaram a poder ser registadas de um modo permanente e a possuir a autoridade que o seu registo lhes dava.

A invenção do papel pelos chineses há cerca de 1500 anos, forneceu um meio de escrita barato, o que fez com que se tornasse económico registar todo o processo de raciocínio, em vez de apenas o seu resultado final. A consequência de tal invento foi o rápido florescimento das artes, filosofia, matemática, leis, e tudo o que pudesse ser registado através da escrita. Mais tarde, com o aparecimento da impressora móvel, a disseminação em massa da informação tornou-se um facto.

Mais recentemente, quando há cerca de 40 anos surgiram os computadores electrónicos, o homem descobriu que podia fazer tudo isto - pensar fora da sua própria

cabeça - sem fazer qualquer espécie de registo. E as consequências deste facto só agora começam a ser avaliadas. Veja-se a discussão actual que rodeia as leis do direito de autor, ou a questão da invasão de privacidade, com a globalização do uso de computadores.

O homem inventou a escrita de modo a tornar a comunicação com ele próprio (mais tarde no tempo) e com os outros, mais fiável e menos sujeita a erros. O computador pode ser visto como um novo desenvolvimento em termos de comunicação. O computador pode servir como mediador à comunicação entre pessoas, através das redes internacionais ou grupos de interesse, por exemplo.

Mas o computador também pode ser visto como um novo desenvolvimento no controlo. Os computadores podem mediar o controlo. O computador pode controlar dispositivos físicos, mas também pode servir como um meio de controlar pessoas - guardando e manipulando dados pessoais.

Em última análise, a finalidade dos computadores é beneficiar as pessoas. Mas para o utilizador tirar partido dos benefícios que o computador disponibiliza é necessário fornecer-lhe um meio de interagir com o computador. E esta interacção, que se traduz na interface do computador com o utilizador, deve ser facultada de uma maneira simples, fiável, eficiente e, se possível, intuitiva para o utilizador. A necessidade de dotar a interacção homem-computador com estas características, levou ao aparecimento de interfaces gráficas com o utilizador.

A interface gráfica com o utilizador é reconhecidamente um meio de aceder às potencialidades oferecidas pelos sistemas computacionais de um modo facilitado em termos cognitivos. A utilização de metáforas conhecidas pelo utilizador neste tipo de interfaces, aliadas à facilidade de o homem apreender informação apresentada graficamente, ajudou a que as interfaces gráficas com o utilizador se tornassem um meio de eleição para providenciar a interacção homem-computador.

Embora o campo de estudo da interacção homem-computador esteja muito em foco actualmente, em particular no que concerne aos factores humanos envolvidos, pensa-se que pouca atenção tem sido (ainda) devotada à interface dos sistemas computacionais com o homem. De facto, e historicamente, as interfaces computador-

computador foram alvo de um projecto mais cuidado, do que as interfaces homem-computador. Uma razão comummente apontada para este facto, é que o utilizador humano tende a desculpar as idiossincrasias de uma "má" interface e a arranjar maneira de tornear as dificuldades.

Muitas das interfaces com o utilizador existentes actualmente, provavelmente não seriam "utilizáveis" de todo se fossem usadas por um computador. Quando a comunicação computador-computador falha, os computadores são incompatíveis; os computadores simplesmente não "falam um com o outro". Se um processador de texto e uma impressora são incompatíveis, não existe grande coisa a fazer; pura e simplesmente não se consegue imprimir. No entanto, nunca se ouviu falar de um utilizador e um computador serem incompatíveis...

A evolução no controlo de dispositivos de fabrico através do computador, bem como a necessária automação dos ambientes de fabrico com vista à obtenção de melhores resultados quer em termos de capacidade de produção, quer a nível da qualidade, conduziram nos últimos anos a uma rápida introdução e disseminação do uso de computadores em ambientes de fabrico.

Esta globalização no uso de computadores em ambientes de fabrico traduziu-se numa primeira fase, no uso de computadores para realizarem prioritariamente funções de controlo de dispositivos. A interacção com o homem não era aqui relevante, uma vez que não se previa que estes sistemas funcionassem de um modo interactivo.

Só quando a evolução dos sistemas computacionais e a evolução da própria concepção dos sistemas produtivos permitiu que os computadores fossem utilizados neste tipo de ambientes para realizar outras tarefas, como sejam as de planeamento, monitorização ou controlo da produção, é que a necessidade de interacção com o homem a este nível se tornou premente.

Naturalmente, que aqui também estão em jogo outros factores. As capacidades gráficas (e de cálculo) dos computadores evoluíram rapidamente nos últimos anos e, por outro lado, a atenção devotada aos aspectos ergonómicos dos postos de trabalho também conheceu um incremento substancial. A conjugação destes factores, com

outros aqui não referidos, conduziu e conduz à necessidade de interfaces gráficas com o utilizador em ambientes de fabrico.

Este trabalho aborda aspectos inerentes à concepção cuidada de interfaces gráficas com o utilizador para sistemas computacionais, numa perspectiva genérica. O caso mais particular das interfaces gráficas com o utilizador vocacionadas para ambientes de fabrico também está consagrado neste trabalho.

Espera-se que este trabalho forneça bases para apreciar e avaliar diversos assuntos relacionados com as interfaces gráficas com o utilizador - GUIs (Graphical User Interfaces) aqui contemplados; leve a pensar a concepção de GUIs de uma forma clara e sistemática; ajude à compreensão dos aspectos que concernem a utilização de GUIs em ambientes de fabrico; e, acima de tudo, forneça um conhecimento crítico para reconhecer e atacar interfaces gráficas pobremente desenvolvidas e implementadas.

A seguir apresentam-se os motivos que conduziram a autora à realização deste trabalho, e a sua organização por capítulos é descrita.

1.1 Motivação da autora

Um dos factores que constituíram motivação para a realização deste trabalho foi a participação da autora no projecto ESPRIT 5104 - CNMA (Communications Network for Manufacturing Applications), cuja tarefa principal consistiu em desenvolver uma interface gráfica com o utilizador para um controlador de célula.

Nessa altura, fez-se sentir a necessidade de aprofundar os conhecimentos ao nível da concepção de interfaces gráficas com o utilizador, em particular para ambientes de fabrico. Mais, constatou-se que virtualmente qualquer pessoa possuía uma opinião "definitiva" sobre o aspecto da interface, o que levou a pensar na necessidade de existirem regras válidas para o desenho de interfaces gráficas com o utilizador.

O estudo subsequente na área da concepção das interfaces gráficas com o utilizador, foi levado a cabo, numa fase inicial, tendo em vista a consideração do caso particular das interfaces gráficas vocacionadas para ambientes de fabrico. No entanto, como rapidamente se constatou, a generalidade dos conceitos subjacentes à concepção

de interfaces gráficas com o utilizador para sistemas computacionais são de âmbito global. Este facto, justifica de algum modo a inexistência de bibliografia dedicada especificamente a interfaces gráficas em ambientes de fabrico.

Este trabalho apresenta assim os conceitos subjacentes à concepção de interfaces gráficas com o utilizador para sistemas computacionais, independentemente do campo de aplicação. Adicionalmente, e sempre que tal se revelou necessário, particulariza-se a abordagem realizada no sentido de contemplar o caso mais específico da concepção de interfaces gráficas com o utilizador para ambientes de fabrico.

Pensa-se que este trabalho constitui uma base para a compreensão de diversos factores envolvidos na concepção de interfaces gráficas. A consideração das interfaces gráficas com o utilizador em ambientes de fabrico, é justificada primordialmente por o trabalho de investigação e desenvolvimento do grupo onde a autora se encontra inserida, visar essencialmente este tipo de ambientes.

A perspectiva genérica adoptada na maior parte deste trabalho permite considerá-lo de interesse na área global das interfaces gráficas com o utilizador para sistemas computacionais. A consideração particular das interfaces gráficas em ambientes de fabrico, constituirá um motivo adicional de leitura para os interessados nesta área.

1.2 Estrutura da tese

Depois de introduzido o assunto da tese, como é que aquele se desenvolve ao longo dos capítulos?

- *Capítulo 2*: O Capítulo 2 [*Conceitos fundamentais*] constitui uma introdução à área das interfaces gráficas e apresenta noções fundamentais concernentes à interacção com o utilizador.
- *Capítulo 3*: As interfaces gráficas com o utilizador em ambientes de fabrico são introduzidas no Capítulo 3 [*GUIs dedicadas a ambientes de fabrico*], que lhes é exclusivamente dedicado.
- *Capítulo 4*: Existem determinadas linhas mestras que devem ser observadas na concepção de interfaces gráficas. O Capítulo 4 [*Metodologias na*

concepção de GUIs] descreve estas linhas mestras para aplicações genéricas, e analisa a sua aplicação nas interfaces gráficas para ambientes de fabrico. Neste capítulo são ainda apresentados os passos típicos do processo de desenvolvimento de interfaces gráficas com o utilizador.

- *Capítulo 5*: O desenvolvimento de interfaces gráficas com o utilizador é uma tarefa complexa. O Capítulo 5 [*Desenvolvimento de GUIs*] descreve algumas técnicas que devem ser consideradas no desenvolvimento de interfaces gráficas com o utilizador.
- *Capítulo 6*: O Capítulo 6 [*Ferramentas de suporte ao desenvolvimento de GUIs*] explora os conceitos subjacentes às ferramentas de suporte ao desenvolvimento de interfaces gráficas com o utilizador, descrevendo em particular os sistemas integrados de desenvolvimento de interfaces gráficas. Exemplos de ferramentas de suporte ao desenvolvimento de GUIs para os sistemas baseados em janelas mais correntes são apresentadas, e o sistema X-Windows é descrito.
- *Capítulo 7*: A necessidade de testar as interfaces gráficas com o utilizador está patente no Capítulo 7 [*Teste e documentação de GUIs*], onde diversas técnicas de teste de interfaces gráficas são exploradas. Adicionalmente, o conceito da documentação fornecida ao utilizador como parte integrante da interface gráfica é apresentado, e são descritas estratégias típicas para providenciar informação e ajuda ao utilizador.
- *Capítulo 8*: No Capítulo 8 [*Conclusões*] são apresentadas as conclusões mais relevantes da realização deste trabalho, e é referida uma área passível de ser explorada num eventual trabalho futuro.

Capítulo 2

Conceitos fundamentais

A interface com o utilizador é a parte do sistema computacional que permite ao utilizador aceder às potencialidades oferecidas pelo sistema. Em última análise, sem uma interface com o utilizador, os computadores seriam inúteis.

Uma interface gráfica com o utilizador - GUI (Graphical User Interface) constitui um meio de comunicação visual entre a aplicação e o utilizador. Este meio de comunicação visual inclui uma metáfora (imagem mental compreensível), um modelo cognitivo (organização dos dados, funções e tarefas e os seus papeis respectivos), o esquema de navegação através da informação, e a aparência geral (*look*) e comportamento (*feel*).

A noção de comunicação visual não é nova; pode ser encontrada em quase todo o lado como revistas, livros, filmes, vídeos, sinais, mapas e em quase todas as formas de anúncios impressos. Estas formas de comunicação visual ajudam à compreensão de informação complexa através de uma forma fácil de apreensão. Até os antigos Egípcios,

os Sumérios e outras grandes civilizações utilizavam símbolos gráficos (hieróglifos) para comunicar informação sobre eventos históricos e épicos. Actualmente, as culturas orientais continuam a usar um alfabeto que consiste em pictogramas para comunicar informação de uma forma tão visual quanto possível.

Os princípios da comunicação visual usados nas áreas acima mencionadas podem ser aplicados aos computadores na forma de componentes das interfaces gráficas como ícones, janelas, menus, diálogos e cursores.

As GUIs possuem diversas vantagens sobre as interfaces baseadas em caracteres, nomeadamente:

- Os símbolos gráficos, se cuidadosamente desenhados, são muito mais facilmente reconhecidos e memorizados. O cérebro humano, pura e simplesmente, tem uma capacidade muito superior para absorver informação gráfica do que processar caracteres em texto.
- Uma interface gráfica com o utilizador permite a este definir parâmetros e executar comandos por interacção directa. Invocar funções através duma interacção física directa com resposta visual instantânea, reduz o tempo de aprendizagem e dá ao utilizador uma sensação de maior controlo sobre o computador.

A interface gráfica com o utilizador básica é geralmente entendida como englobando elementos do tipo menus, janelas, teclado, rato, os *beeps* e outros sons que o computador faz e, em geral, todos os canais de informação que permitem ao utilizador e computador comunicarem.

É claro que usar um sistema computacional nos dias de hoje também envolve ler manuais, recorrer a linhas de ajuda especializada, frequentar cursos, fazer perguntas a colegas e, em suma, tentar perceber como é que o programa funciona. Um sistema ou programa bem concebido suporta estas actividades, logo este suporte também faz parte da interface com o utilizador. À distância, a interface é vista como sendo o *hardware* (Figura 2.1).

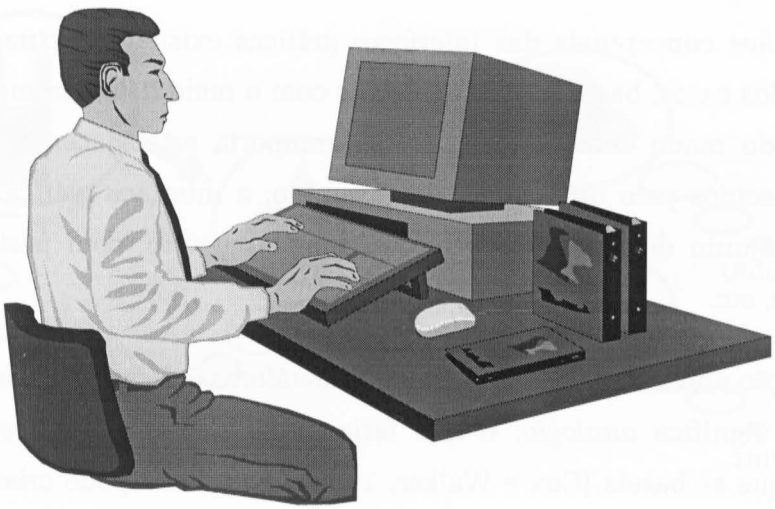


Figura 2.1 Interface com o utilizador ao nível de *hardware*.

No sentido de tornar a interacção humana com computadores mais fácil e compreensível, desenvolveram-se interfaces gráficas baseadas em metáforas e modelos cognitivos, das quais a interface gráfica do Macintosh foi pioneira na sua divulgação em massa, facilmente reconhecíveis pelos utilizadores. Este tipo de interfaces gráficas (com metáforas baseadas num ambiente de escritório) sofreram uma rápida disseminação, e actualmente encontram-se em virtualmente todos os computadores pessoais.

2.1 Modelos conceptuais e metáforas

Um modelo conceptual é a nossa percepção mental do que um dado sistema faz, e o modo como o faz. Parte do processo de aprendizagem envolve o desenvolvimento de um modelo conceptual.

O modelo conceptual é importante por duas razões: é uma ajuda para a *apreensão* do sistema e funciona como um guia para a *previsão* do comportamento do sistema. A previsão do comportamento é um factor relevante na aprendizagem do sistema, uma vez que frequentemente aprendemos através da experimentação. A aprendizagem de sistemas é difícil a menos que exista um bom ponto de partida para racionalizarmos o sistema, e um modo relativamente fácil de progredirmos nele.

Os modelos conceptuais das interfaces gráficas existentes actualmente são, na generalidade dos casos, baseados em analogias com o meio físico de modo a facilitar a compreensão do modo como a interface se comporta estabelecendo analogias com sistemas conhecidos pelo utilizador. Por exemplo, a interface gráfica do Macintosh, utiliza um conjunto de metáforas baseadas num escritório, com pastas de arquivo, caixote do lixo, etc.

A distinção entre modelos conceptuais e metáforas é defendida com base no facto que metáfora significa *analogia*, o que faria supor que o sistema seria análogo à metáfora em que se baseia [Cox e Walker, 1989]. Este facto pode criar problemas na percepção de como o sistema de facto se comporta.

Na verdade, se a alimentação falhar, uma folha de papel escrita em cima da secretária não fica em branco, mas é isto mesmo o que acontece se se estiver a trabalhar num processador de texto do Macintosh, baseado numa metáfora de secretária (*desktop*). Isto significa que, se a utilização de metáforas ajuda à compreensão de como o sistema se comporta através do estabelecimento de analogias com o meio físico conhecido do utilizador, torna-se necessário que este desenvolva conceitos adicionais, formando um modelo conceptual baseado em metáforas, para lidar com o sistema; neste caso, guardar o documento mesmo que este ainda não esteja finalizado...

Um problema que frequentemente aparece é que existem sempre inúmeros modelos conceptuais para um único sistema (Figura 2.2): o de quem desenvolveu a interface, o que está de facto consagrado na interface, e um para cada utilizador.

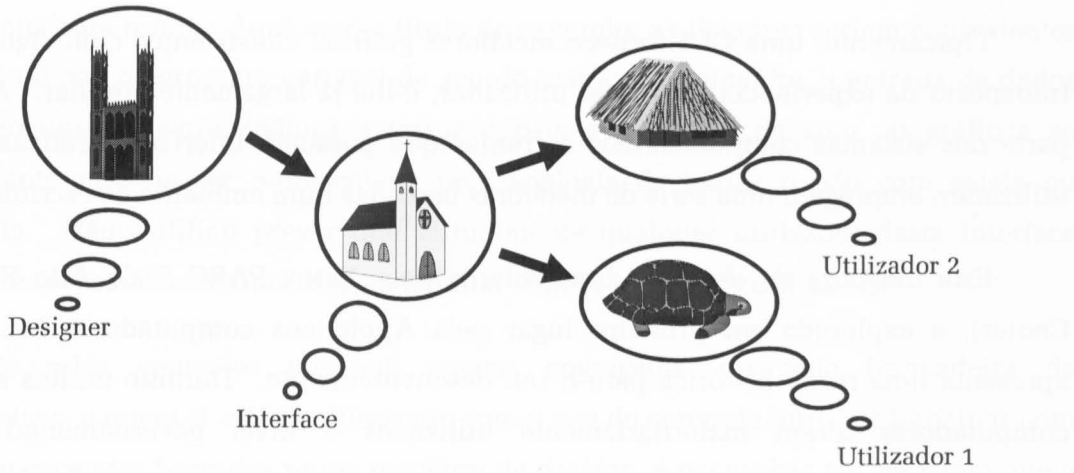


Figura 2.2 O problema de diferentes modelos conceptuais para o mesmo sistema.

A intensidade da divergência entre os vários modelos constitui uma boa medida da qualidade do modelo original do *designer*: um bom modelo em termos conceptuais, deve traduzir-se eficazmente na interface e ser facilmente compreensível pelos utilizadores, existindo portanto um elevado grau de convergência no modelo utilizado; pelo contrário, com um modelo pobre é natural que a confusão seja reinante!

2.1.1 Utilização de metáforas

A representação conceptual, bem como o raciocínio humano estão permeados pelo uso de metáforas: por exemplo, muitos conceitos importantes, como o tempo, são abstractos, e o ser humano tenta compreendê-los fazendo uso de outros termos mais concretos, como objectos e orientação espacial. É assim razoável supor que as pessoas provavelmente utilizam metáforas para entender os sistemas computacionais.

Para serem eficazes as metáforas devem ser congruentes e transparentes. Uma metáfora é *congruente* se possui uma relação adequada com aquilo que representa; é necessário existir harmonia entre as entidades e relações presentes no modelo, e as da fonte. Uma metáfora é *transparente* no sentido de ser uma boa representação da fonte, embora a metáfora não deva possuir uma ligação demasiado forte com a fonte que modeliza.

Tipicamente, uma GUI fornece metáforas gráficas consistentes cujo significado, transposto da experiência prévia do utilizador, é-lhe já largamente familiar. A maior parte dos sistemas computacionais correntes que possuem interfaces gráficas com o utilizador, empregam uma série de metáforas baseadas num ambiente de escritório.

Esta metáfora de *desktop*, desenvolvida pelo Xerox PARC (Palo Alto Research Center), e explorada em primeiro lugar pela Apple nos computadores Macintosh, apresenta uma razão histórica para o seu desenvolvimento. Durante muitos anos, os computadores foram maioritariamente utilizados a nível governamental e em ambientes de escritório, sendo natural que o seu desenvolvimento reflectisse estas necessidades, aquando da escolha de uma metáfora gráfica.

No entanto, os últimos anos, com o advento de computadores mais pequenos e económicos, trouxeram um mercado muito maior e heterogéneo, cujos utilizadores potenciais não possuirão na sua totalidade uma experiência de ambiente de escritório, no qual é baseada a metáfora de *desktop*.

Para que a utilização de metáforas em interfaces gráficas com o utilizador seja eficaz e proveitosa, é desejável que aquelas sejam familiares aos putativos utilizadores, de modo a que estes possam formar um modelo conceptual de como a interface se comporta, baseado em conhecimento previamente adquirido. Se as metáforas utilizadas numa dada GUI não possuírem significado para o utilizador, então a utilização de conhecimento prévio para construir um modelo conceptual do sistema não é possível.

Atendendo a que na actualidade, nem todos os potenciais utilizadores de computadores estarão familiarizados com o ambiente típico de um escritório, existe um sentimento crescente no sentido de se empregarem metáforas em GUIs baseadas em conhecimentos mais vastos do utilizador comum. [Langford e Jones, 1994] advogam que virtualmente qualquer utilizador de um computador já terá alguma vez na vida preparado comida, e sugerem a criação de uma GUI (KI - Kitchen Interface) baseada na analogia com uma cozinha, como o próprio nome sugere.

Na KI, a representação gráfica é similar à de uma cozinha comum, com um armário do pão, uma dispensa, um caixote do lixo, e outros elementos relacionados

com a analogia usada. Aqui, e só a título de exemplo, os ficheiros seriam equivalentes a fatias de pão de grossura variável de acordo com o seu tamanho, a entrada de dados num documento seria análoga a torrar o pão, e a adição de som ou gráficos ao documento poderia ser representada pela analogia de barrar o pão com geleia ou compota. Não é difícil prever que virtualmente qualquer utilizador desta interface conseguisse estabelecer uma relação familiar com as metáforas utilizadas!

Se estes conceitos parecem apenas engraçados ou uma brincadeira de académicos, a quem já está familiarizado com o uso de computadores e a trabalhar com as GUIs correntes baseadas numa metáfora de *desktop*, é necessário ter em conta que a audiência que [Langford e Jones, 1994] pretendem atingir com a KI não é a dos utilizadores usuais de computadores.

A preocupação em desenvolver sistemas computacionais com GUIs para atingir outro tipo de utilizadores que não os já existentes, está bem patente no desenvolvimento recente pela Microsoft de uma interface gráfica, BOB (data de lançamento no mercado prevista para o fim de Março de 1995) que emprega uma metáfora baseada no ambiente de uma casa com diversos quartos.

Nesta interface, o utilizador desloca-se "dentro" de uma "casa" com "quartos" - que podem ser decorados a gosto, incluindo a paisagem que se vê da janela. O utilizador pode ainda escolher os seus "guias pessoais" que o vão ajudar na utilização da interface. Existem mais de uma dezena de "amigos de BOB" (como a Microsoft lhes chama), cujo objectivo é dar ao utilizador informações sobre o que fazer para obter o que pretende. Não existem manuais escritos nem ajuda *on-line* - pelo menos da forma convencional.

Nesta interface, cuja aparência gráfica inicial é a de uma sala típica com uma mesa, calendário, relógio de pêndulo, etc., não são visíveis ícones convencionais. Assim, não há um símbolo convencional para o processador de texto - existe uma folha de papel sobre a secretária, com uma caneta esquecida por cima; não aparece um ícone para o calendário - há um calendário pendurado na parede que permite aceder ao programa; e por aí fora.

A intenção da Microsoft é atingir um nicho de mercado de potenciais utilizadores sem qualquer experiência prévia do uso de computadores, e que se relacionem de um modo intuitivo com este tipo de metáforas. A validade (ou não) desta abordagem será demonstrada pelo sucesso que a interface tiver...

O conceito importante a reter é que se é notório que a utilização de metáforas gráficas numa GUI facilita a aprendizagem da mesma e a percepção por parte do utilizador de como o sistema se comporta, isto só será verdadeiramente eficaz se o utilizador estiver familiarizado com as metáforas empregues. Só assim conseguirá relacionar-se com estas, e formar um modelo conceptual de como o sistema se comporta, baseado em conhecimentos prévios adquiridos. As metáforas usadas nas GUIs devem então fazer parte do universo de conhecimento do utilizador ao qual a interface se destina.

2.1.2 Classificação de metáforas

Já foi referida a necessidade de revestir a utilização de metáforas nas interfaces gráficas com o utilizador de cuidados especiais. [Erickson, 1990] sugere cinco dimensões segundo as quais uma metáfora pode ser avaliada:

- ▷ *Quão bem estruturada a metáfora é*: Quanto menos estrutura a metáfora possuir, menor será a sua utilidade.
- ▷ *Quão relevante é a metáfora para o problema*: Uma metáfora irrelevante pode conduzir os utilizadores numa direcção errada, ou criar-lhes falsas expectativas.
- ▷ *A facilidade com que uma metáfora pode ser representada*: A metáfora possui representações visuais e auditivas distintas, bem como palavras específicas associadas?
- ▷ *A adequação da metáfora à audiência pretendida*: Uma metáfora adequada é, logicamente, uma metáfora que os utilizadores acham fácil de compreender.
- ▷ *A facilidade com que a metáfora pode ser estendida* e adaptada para lidar com situações imprevistas.

A avaliação das metáforas segundo estas dimensões, permite inferir da eficácia e pertinência da sua utilização numa interface gráfica com o utilizador particular, possibilitando uma escolha judiciosa das metáforas a usar.

2.2 Características de uma interface gráfica com o utilizador

O objectivo de qualquer interface é permitir uma comunicação efectiva entre o computador e o homem, e neste sentido um dado número de características devem constar da interface. Estas características, pela sua importância, são descritas a seguir.

2.2.1 Transparência

Uma interface gráfica com o utilizador é uma ferramenta para aceder às potencialidades oferecidas pelo sistema computacional, do qual é a parte visível. Uma característica essencial na utilização de uma ferramenta qualquer é que esta "desaparece" quando é usada: o utilizador concentra-se no problema e não na utilização da ferramenta. Esta característica é designada por *transparência*.

O aspecto mais importante da transparência é óbvio: o utilizador deve poder ver a tarefa que está a realizar. Neste sentido, informação crucial não deve ser "escondida" do utilizador, por exemplo usando menus que cubram por completo o ecrã inquirindo o utilizador sobre o que fazer, antes de o deixar ver o que é que ele quer fazer, e a quê.

O utilizador deve poder ter uma imagem global da informação, pelo menos quando entra no sistema, de modo a poder orientar-se. Imagens mais particulares e informação com maior detalhe devem estar disponíveis, mas não devem ser a representação gráfica inicial do sistema.

Outro aspecto relevante da transparência numa GUI é a utilização instintiva da interface; i.e., o utilizador não necessita pensar em que tecla vai carregar para realizar uma dada acção particular, por exemplo. Um bom modo de conseguir uma operação instintiva da interface é permitir ao utilizador que *aponte* para coisas. Esta ideia está subjacente a um elevado número de facetas das GUIs correntes, como sejam a selecção de objectos através de um toque com o rato num ícone ou a selecção de acções através de um menu de abertura vertical, etc.

2.2.2 Consistência

Um dos principais objectivos quando se desenvolve uma interface ideal com o utilizador é que esta seja intuitiva para os utilizadores independentemente da norma utilizada. No caso geral, a consistência dos elementos utilizados na interface e do seu comportamento, constitui o factor mais decisivo para que uma dada interface seja intuitiva para o utilizador.

Um sistema consistente é um sistema em que o modelo conceptual, a funcionalidade, a navegação[1] e as ligações ao *hardware* são uniformes e seguem um reduzido número de regras simples, não contendo portanto excepções ou condições especiais. O propósito básico da consistência é permitir ao utilizador a generalização do conhecimento de funcionalidades do sistema a partir de outras funcionalidades já conhecidas.

Um princípio a seguir no desenho de interfaces gráficas com o utilizador para alcançar propriedades de consistência é o *Princípio da Mínima Surpresa* (Law of Least Astonishment): os utilizadores não devem ser surpreendidos pela reacção do programa. Este princípio sugere que fazer o que o utilizador provavelmente considera normal ou razoável é a melhor solução.

A consistência permite evitar a frustração induzida quando um sistema não se comporta de um modo compreensível ou lógico. Uma interface consistente torna-se intuitiva para o utilizador.

2.2.3 Funcionalidade

Uma interface gráfica com o utilizador é funcional se permitir que o utilizador realize as tarefas que deseja, de um modo que este encontre apropriado. Para atingir esta característica, quem desenvolve a GUI deve determinar, sem margem para dúvidas, quais são as tarefas a realizar pela aplicação em causa, dividi-las em grupos

[1] Por navegação, nos sistemas baseados em janelas, entende-se a sequência de acções executada para aceder aos diversos componentes das interfaces gráficas com o utilizador (e.g., janelas, menus, botões).

relacionados, concebendo em seguida as janelas de diálogo para comunicar com o utilizador.

Este processo requer, no caso geral, uma interacção extensiva com os utilizadores aos quais a aplicação se dedica, procurando assegurar o conhecimento do que os utilizadores esperam da aplicação.

Frequentemente, as actuais aplicações gráficas por janelas providenciam uma janela, menu ou diálogo inicial para o utilizador começar a usar a aplicação. Isto é usualmente seguido por um conjunto de diálogos concebidos para obter mais parâmetros de operação da aplicação, conduzindo em última análise à invocação do resultado que o utilizador deseja.

2.2.4 Flexibilidade

As GUIs para aplicações genéricas, quando bem concebidas, são altamente flexíveis. Isto não significa que facultem ao utilizador a capacidade de fazer tudo o que deseja e da maneira que deseja; no entanto, interfaces gráficas flexíveis permitem uma certa "liberdade" ao utilizador no modo como realizam as tarefas pretendidas.

Um modo de se alcançar flexibilidade é utilizar um modelo conceptual que seja suficientemente genérico para abarcar uma classe de problemas, em vez de um único problema. É claro que esta característica é uma faca de dois gumes: existe o risco de a aplicação se tornar tão genérica, que não realize as funções para que foi originalmente concebida, da melhor forma. No entanto, é sempre possível chegar a um compromisso razoável.

Uma abordagem frequentemente usada para providenciar flexibilidade é facultar a utilização de uma *macro*, que é essencialmente uma linguagem de programação que pode ser usada para configurar uma aplicação de carácter genérico para um dado ambiente específico. Infelizmente, para a generalidade das pessoas esta programação é extremamente difícil, uma vez que exige a capacidade de escrever antecipadamente uma série de passos (muitos deles hipotéticos), e pensar em várias contingências improváveis, mas que em todo o caso podem ocorrer.

É de longe mais fácil trabalhar interactivamente com um sistema e lidar com os problemas à medida que eles ocorrem. Deste modo, a configuração do sistema deve ser feita permitindo ao utilizador que modifique as opções de visualização de um modo interactivo. Isto geralmente é conseguido utilizando duas facetas: um menu de *preferências* que permita ao utilizador mudar o modo como um objecto é representado no ecrã; e um menu de *visualização* que permita a selecção de vários objectos, e sua representação gráfica como uma lista ou individualmente.

2.2.5 Controlo pelo utilizador

O controlo pelo utilizador significa que é ao utilizador, e não ao computador, que compete decidir como uma tarefa em particular vai ser realizada. Um sistema bem concebido deixa o utilizador realizar quaisquer acções que este deseje, desde que aquelas sejam correctas. Providenciar controlo pelo utilizador envolve facultar ao utilizador a capacidade de organizar o seu trabalho à sua maneira.

Esta ideia de controlo do utilizador não tinha grande aceitação no passado, e ainda hoje existem contestatários. A causa subjacente a esta atitude é considerar os utilizadores como inerentemente "idiotas", devendo estes (e o sistema) ser protegidos das suas próprias acções. A ironia disto é que os sistemas baseados neste pressuposto são simplistas do ponto de vista do utilizador, uma vez que apenas lidam com casos simples e directos, não contemplando excepções, e complexos do ponto de vista de quem faz a manutenção do sistema, dado que possuem uma enorme quantidade de código construído para lidar com "tudo o que o utilizador possa fazer de errado".

Convém aqui referir, e sem prejuízo do que atrás foi escrito, que existem situações, como o caso exemplar de GUIs em ambientes de fabrico (Capítulo 3), nas quais não é possível facultar controlo total ao utilizador, sob pena de desse facto advirem consequências que podem ser perfeitamente desastrosas, e que podiam ser evitadas não permitindo certo tipo de acções ao utilizador.

2.2.6 Fiabilidade

A necessidade da característica de fiabilidade de uma interface gráfica com o utilizador é notória. O utilizador tem que poder confiar no desempenho da interface para realizar as tarefas pretendidas. Sem possuir fiabilidade, uma aplicação por mais atractiva que seja perderá a sua credibilidade até para o utilizador mais tolerante.

Uma das primeiras regras no desenvolvimento de *software* é GIGO - *Garbage In, Garbage Out*; ou seja, não se pode esperar desenvolver uma aplicação fiável e de boa qualidade com ferramentas e métodos de desenvolvimento não comprovados e não fiáveis.

Uma regra geral para quem concebe aplicações gráficas (aliás, para qualquer tipo de aplicações): se se "poupar" nas ferramentas de desenvolvimento, esse preço será mais do que pago em fases posteriores do desenvolvimento. Qualquer tecnologia que seja considerada parte da plataforma de desenvolvimento da aplicação deve estar perfeitamente comprovada, ser fiável e estável, de modo a produzir aplicações de qualidade.

É claro que a utilização de ferramentas fiáveis não constitui, por si só, garante da obtenção de uma interface gráfica com o utilizador fiável, mas constitui sem sombra de dúvida uma condição *sine qua non*.

2.2.7 Facilidade de aprendizagem

Na aprendizagem de um sistema novo, existem dois níveis distintos: aprender a semântica do problema, e aprender a sintaxe (gramática) da interacção. Uma vez que cada sistema provavelmente abordará um problema diferente, a semântica (i.e., o significado das coisas) será sempre diferente. É aqui que possuir um bom modelo conceptual se revela essencial.

Um dos problemas de sintaxe típicos é que diferentes sistemas possuem comandos com nomes diferentes para realizar as mesmas funções, comandos esses que têm que ser memorizados pelo utilizador. Os sistemas baseados em menus aliviam em parte esta tarefa, uma vez que o utilizador apenas precisa de saber o que cada opção significa, o que, convenha-se, é bem mais fácil.

Mesmo assim, os menus podem tornar-se confusos se nomes distintos forem usados para o mesmo conjunto de comandos. Por esta razão, é necessário identificar conjuntos de acções comummente usadas e atribuir-lhes nomes comuns. As diversas linhas mestras existentes para o desenvolvimento de GUIs (e.g., o Common User Access Standard da IBM) são uma tentativa de atingir aquele objectivo. O menu *File* existente nos sistemas do Apple Macintosh e do Windows, é um bom exemplo disto. O nome é normalizado em todas as aplicações destes ambientes, assim como o significado das várias opções *New, Open, Save, Save As.., Print.*

Qualquer que seja o tipo de sintaxe presente, o utilizador deve poder experimentar comandos que supõe estarem correctos, ou seleccionar opções para ver o que acontece, sem que os resultados sejam fatais. O utilizador deve ser prevenido de qualquer acção que possa provocar um erro fatal, ou impedido de a realizar, e idealmente deve poder voltar atrás no último comando efectuado. A experimentação constitui sem dúvida um factor primordial na aprendizagem de qualquer sistema.

2.3 Tipos de GUIs

Os estilos mais comuns que podem ser encontrados nas interfaces gráficas com o utilizador são interfaces do tipo WYSIWYG (What You See Is What You Get), de manipulação directa e icónica. Nenhum destes estilos é mutuamente exclusivo; as GUIs bem sucedidas misturam frequentemente elementos dos diversos estilos, para atingir objectivos que não são passíveis de alcançar através de um estilo único. Na análise seguinte tecem-se considerações sobre a aplicação, vantagens, desvantagens e inter-relação entre os diferentes estilos.

2.3.1 WYSIWYG

As interfaces do tipo WYSIWYG caracterizam-se por a informação visual constituir uma representação o mais realista possível da informação que se pretende veicular. Esta característica torna este tipo de interfaces muito popular.

Este tipo de interfaces é fundamental em aplicações gráficas interactivas. A representação com que o utilizador interage no ecrã é basicamente a mesma que a

imagem criada pela aplicação. Se a interface não for do tipo WYSIWYG, os utilizadores têm de traduzir entre a imagem mental do resultado desejado e os códigos de comando. Os resultados só se tornam visíveis depois de os códigos introduzidos serem processados.

No entanto existem algumas desvantagens associadas a interfaces deste género. Nomeadamente, quando a resolução espacial, de intensidade ou de cor do ecrã não coincide com a resolução "verdadeira" da imagem, torna-se difícil ou mesmo impossível criar uma correspondência exacta entre esta e a imagem que aparece no ecrã. Esta é a razão porque WYSIWYG é por vezes designado como *what you see is* all *you get*!

2.3.2 Manipulação directa

Uma interface gráfica de manipulação directa é uma interface em que os objectos, atributos ou relações que podem ser manipulados são representados de uma forma visual; as operações são invocadas por acções realizadas sobre as representações visuais, tipicamente através da utilização do rato. Isto é, os comandos não são invocados explicitamente pelos meios tradicionais como selecção de menus ou por teclado; antes, o comando está implícito na acção da representação visual. Um exemplo bastante popular de interfaces com o utilizador que utilizam manipulação directa é a interface do Macintosh: arrastar um ícone de um ficheiro de um disco para outro copia o ficheiro do primeiro para o segundo disco.

A manipulação directa é por vezes apresentada como sendo o melhor estilo para interfaces com o utilizador. As interfaces deste tipo são de facto bastante poderosas e fáceis de utilizar. Mas estas interfaces tendem a tornar-se lentas para utilizadores com experiência que são forçados a utilizar manipulação directa onde outro estilos seriam em geral bastante mais rápidos.

Uma interface gráfica com o utilizador combinando manipulação directa com as facilidades de uma linguagem de comando pode tornar-se mais rápida para o utilizador do que uma que dependa apenas da manipulação directa. As interfaces de manipulação directa envolvem, tipicamente, outros estilos de interfaces, usualmente comandos invocados a partir de menus ou através do teclado.

De facto, é de um modo geral difícil construir uma interface na qual todos os comandos possuem acções de manipulação directa. Isto só vem reforçar a convicção de que uma interface gráfica com o utilizador eficiente, não deve apoiar-se num estilo único de interacção: misturar diversos estilos é, de um modo geral, melhor do que aderir de uma forma esclavagista a um único estilo.

WYSIWYG e manipulação directa são conceitos separados e distintos. Por exemplo, a representação textual de uma imagem gráfica pode ser modificada via manipulação directa, e a imagem gráfica de um sistema WYSIWYG pode, se desejado, ser modificada recorrendo apenas a uma linguagem de comando da interface. Especialmente quando utilizados em conjunto, contudo, os dois conceitos são poderosos, fáceis de apreender, e razoavelmente rápidos de usar, como aliás o demonstram várias interfaces com o utilizador bem sucedidas.

2.3.3 Icónica

Um ícone é uma representação pictorial de um objecto, acção, propriedade, ou outro conceito. Na concepção de interfaces gráficas existe geralmente a possibilidade de escolher entre utilizar ícones ou palavras que representem esses conceitos. É de notar que a utilização de ícones não está directamente relacionada com a manipulação directa: texto pode ser directamente manipulado da mesma forma que ícones, e por vezes texto pode representar conceitos de uma forma mais eficiente que a utilização de ícones.

Qual é melhor então, utilizar texto ou ícones? Como para muitas questões no âmbito da concepção de interfaces com o utilizador a resposta é: depende! A utilização de ícones traduz-se em diversas vantagens. Ícones bem desenhados e conceptualizados podem ser reconhecidos mais facilmente do que palavras, e ocupar uma porção menor do espaço do ecrã. Se desenhados cuidadosamente, os ícones podem tornar-se independentes da linguagem.

Existem pelo menos três objectivos separados no desenho de ícones, cuja importância depende da aplicação específica em causa:

- ▷ *Reconhecimento*: Quão rápida e rigorosamente o significado do ícone pode ser apreendido.
- ▷ *Memorização*: Quão eficazmente o significado do ícone pode ser lembrado uma vez apreendido.
- ▷ *Discriminação*: Quão eficazmente um ícone pode ser distinguido de outro.

Acções sobre objectos (i.e., comandos) podem também ser representadas por ícones.

Alguns utilizadores não gostam de ícones como, por exemplo, a lata do lixo na interface do Macintosh, argumentando que tal ideia é juvenil ou caricata, estando "abaixo da sua dignidade". Quem concebe a interface pode ou não concordar com tal avaliação, mas a opinião do utilizador é, em geral, mais importante!

2.4 Caracterização dos utilizadores

Atendendo a que existe uma enorme variedade de pessoas que fazem uso de computadores, e particularmente interagem com as interfaces gráficas disponibilizadas para que esse uso seja o mais produtivo possível, é desejável isolar certas características que permitam distinguir os utilizadores. Esta distinção é relevante na medida em que pode influenciar o tipo de interfaces desenvolvidas.

Existe um elevado número de classificações possíveis, de entre as quais se destacam [Cox e Walker, 1993]:

- ▷ *Frequência de utilização*: Segundo esta perspectiva, os utilizadores podem ser classificados como principiantes, i.e., pessoas com pouca ou nenhuma experiência prévia; utilizadores casuais, i.e., pessoas que usam o sistema ocasionalmente; e utilizadores regulares, i.e., pessoas que utilizam o sistema com frequência suficiente para se tornarem experientes no seu uso.
- ▷ *Conhecimento da aplicação*: A distinção aqui faz-se entre peritos na área da aplicação, que estão familiarizados com os modelos subjacentes e assunções inerentes ao sistema, e operadores, que sabem como realizar tarefas de rotina

com o sistema, mas não percebem necessariamente como é que o sistema funciona.

- ▷ *Tarefas a serem realizadas*: Qualquer utilizador de um modo geral só utiliza parte do sistema. Por exemplo, alguns utilizadores estarão interessados em realizar transacções de rotina com o sistema ou em tomar decisões baseados nos dados provenientes dessas transacções, enquanto que outros estarão envolvidos na manutenção do sistema, instalando novo *software* ou modificando o existente.
- ▷ *Qualificações assumidas*: A realização de tarefas particulares num dado sistema requer frequentemente que o utilizador possua certas qualificações ou treino adquiridos independentemente da aplicação em particular.
- ▷ *Atitudes*: Diferentes utilizadores têm diferentes atitudes perante os sistemas computacionais, e isto frequentemente governa o modo como pretendem interagir com o sistema e o tipo de tarefas que desejam realizar. Algumas pessoas encaram uma aplicação como um desafio e tentam levar o seu uso até ao limite de aplicação, enquanto que outras são "mais conservadoras" e só tentam efectuar acções óbvias, e realizar tarefas que já lhes foram ensinadas. Por outro lado, algumas pessoas possuem modos de interacção preferidos, orgulhando-se, por exemplo, de "nunca terem utilizado o rato".

A maior parte dos sistemas, no entanto, será utilizada por pessoas provenientes de todas as gamas deste espectro. Uma classe particular de utilizadores à qual tem sido devotada muita atenção é o *utilizador poderoso* (*power user*). Este tipo de utilizadores é experiente e qualificado na utilização de uma aplicação particular e estende o seu âmbito de aplicação até ao limite, explorando situações para as quais o sistema não foi especificamente construído para tratar, mas às quais pode ser aplicado. Os utilizadores poderosos representam uma fonte de conhecimento de grande valor de uma dada aplicação, em termos do que é que pode ser feito com essa aplicação e como.

2.5 Conclusões

Os sistemas computacionais conheceram um enorme desenvolvimento nas últimas décadas, o que acarretou duas consequências primordiais. Por um lado, o nível de

complexidade das aplicações aumentou extraordinariamente. Por outro lado, e em grande medida devido ao aparecimento e divulgação em massa do computador pessoal, não só o número de utilizadores de computadores aumentou significativamente, como também o espectro de pessoas que usam computadores foi inegavelmente alargado.

A necessidade de utilizar aplicações cada vez mais complexas aliada ao facto de o uso de computadores ter deixado de ser um reduto confinado a peritos, levou ao desenvolvimento de mecanismos para facilitar a interacção homem-computador. As interfaces gráficas com o utilizador constituem um meio de eleição de facilitar essa interacção.

Na verdade, o facto de se veicular informação através do meio visual que tem reconhecidamente mais impacto no utilizador - uma vez que é uma forma de comunicação que lhe é familiar, e a capacidade de usar metáforas já conhecidas do utilizador - permitindo-lhe recorrer a conhecimentos prévios, tornam as interfaces gráficas fundamentais para facilitar a interacção homem-computador.

Pode assim dizer-se que uma GUI só será eficaz se cumprir o seu propósito: facilitar a interacção humana com o computador. Neste sentido, a utilização de metáforas e modelos conceptuais facilmente compreensíveis pelo utilizador desempenham um papel de suma importância.

Adicionalmente, as interfaces gráficas devem possuir determinadas características que possibilitem atingir com sucesso aquele objectivo, independentemente do estilo da GUI. Os factores humanos, particularmente no que diz respeito às características dos utilizadores, não devem ser descurados na construção de interfaces gráficas.

São estas as noções apresentadas neste capítulo e que devem ser entendidas como uma referência global no campo das interfaces gráficas com o utilizador.

Capítulo 3

GUIs dedicadas a ambientes de fabrico

A transição da "fábrica de hoje" para a "fábrica de amanhã" coloca novos desafios aos actuais sistemas de fabrico. O aumento global da competição em conjunto com o avanço da tecnologia computacional também forçaram os sistemas correntes de fabrico a mudarem-se continuamente. A introdução de novos produtos é mais frequente e a vida dos produtos é de menor duração. Muitas empresas não possuem a flexibilidade para se adaptarem às exigências do mercado primordialmente por causa da mudança de mercados relativamente estáveis para mercados dinâmicos.

O uso cada vez maior de computadores em sistemas integrados de fabrico resulta numa série de vantagens visíveis como, por exemplo, processamento mais rápido, tempos de *set-up* das máquinas reduzidos e aumento da qualidade.

Com a globalização do uso de computadores nos ambientes de fabrico, surgiu a necessidade de providenciar a interacção homem-computador - HCI (Human-Computer Interaction) de um modo simples e de fácil aprendizagem. As interfaces gráficas com o utilizador são reconhecidamente um modo de aceder às potencialidades oferecidas pelos sistemas computacionais de uma forma facilitada em termos cognitivos. A utilização de metáforas neste tipo de interfaces permite uma rápida apreensão das capacidades do sistema e das tarefas a realizar.

Para além de facilitar a interacção homem-computador, existe outra razão significativa para a utilização de interfaces gráficas com o utilizador em ambientes de fabrico. As capacidades gráficas das interfaces permitem visualizar aspectos importantes dos sistemas de fabrico que, de outra forma, utilizando interfaces baseadas em caracteres por exemplo, não seria possível. De facto, as GUIs permitem obter uma imagem global da fábrica através da representação da disposição espacial das células e dispositivos de fabrico, monitorizar a evolução de processos através de gráficos ou visualizar a movimentação de materiais, só para citar alguns exemplos.

A maior parte das interfaces com o utilizador actuais foram concebidas para utilizadores dedicados que, podendo não ser especialistas em programação, possuem, na generalidade dos casos, um conhecimento alargado dos sistemas computacionais. Contudo, operadores de máquinas e engenheiros da área de produção não são especialistas em computadores. São utilizadores que têm que prestar mais atenção às suas tarefas normais do que à operação com computadores.

Num ambiente de fabrico, para além de o operador não estar, de um modo geral, familiarizado com interfaces gráficas, existem outros factores a considerar. Os requisitos da informação a visualizar no ecrã são específicos deste tipo de ambiente e existem constrangimentos adicionais em termos de comunicações e mecanismos de interacção, por exemplo, que têm que ser equacionados no desenvolvimento de interfaces vocacionadas para ambientes de fabrico.

3.1 Aspectos relativos à interacção humana

Existem aspectos relevantes concernentes à interacção humana que não devem ser desprezados quando se desenvolvem interfaces gráficas dedicadas especificamente a ambientes de fabrico. De facto, o utilizador de GUIs nestes ambientes, de um modo geral, não possui familiaridade com interfaces gráficas para sistemas computacionais; por outro lado, a sua atenção necessita ser prioritariamente canalizada para a execução das tarefas que lhe estão atribuídas.

Na verdade, o facto de o operador fabril ter que interagir com o computador no âmbito das suas tarefas, constitui a maior parte das vezes um elemento de perturbação, dada a necessidade de aprender novos mecanismos e ferramentas de trabalho. Acresce a isto que muitas vezes a introdução e uso de computadores nestes ambientes é visto pelos trabalhadores com uma certa relutância, sendo entendido como um eventual concorrente e substituto do seu trabalho.

3.1.1 Utilizador-tipo

No sentido de se desenvolverem GUIs dedicadas a ambientes de fabrico, torna-se necessário caracterizar o utilizador-tipo de interfaces gráficas nestes ambientes. O utilizador final da interface gráfica condiciona significativamente a concepção de GUIs, primordialmente a nível das metáforas a utilizar, mas também em outros âmbitos, como, por exemplo, os mecanismos de interacção preferidos. É então importante possuir um conhecimento profundo das características mais relevantes do utilizador-tipo ao qual a interface se destina.

Naturalmente, um conhecimento profundo só se adquire através de um relacionamento directo com o utilizador. O utilizador deve ser considerado uma peça fundamental no desenvolvimento de GUIs (afinal, estas são-lhe dedicadas) e a interacção entre quem concebe a interface e o utilizador deve ser constante, ao longo das várias fases do processo de desenvolvimento.

No entanto, e correndo o risco de se cair numa generalização fácil e simplista, pode aqui caracterizar-se genericamente os utilizadores de GUIs em ambientes de fabrico. Note-se então que a caracterização de seguida apresentada deve ser entendida

como uma referência, até para serem abordados constrangimentos que se colocam relativos ao utilizador, e não como uma tipificação redutora de todos os utilizadores de interfaces gráficas em ambientes de fabrico.

É possível, então, caracterizar o utilizador-tipo de GUIs em ambientes fabris como sendo um operário relativamente especializado, com um conjunto de tarefas bem definidas a realizar, cujo à-vontade e familiaridade com interfaces gráficas será diminuto ou até inexistente. Este operário sempre terá realizado as tarefas ligadas ao processo fabril que lhe estão atribuídas à "sua maneira" (leia-se aqui sem usar computadores) e não estará de um modo geral vocacionado para a utilização de computadores e GUIs no âmbito do seu trabalho.

A utilização de novas tecnologias é muitas vezes vista pelos operários de um modo renitente, e a introdução de computadores nos ambientes fabris traz em muitos casos um medo, muitas vezes infundamentado, de que os computadores vão diminuir os postos de trabalho, causando assim insegurança nos operários.

Acresce a isto que o uso de sistemas computacionais permite exercer um controlo muito mais apertado sobre os próprios trabalhadores, dado que é possível registar de um modo efectivo quem realizou determinada operação, a que horas o fez, se ignorou um aviso fornecido pelo sistema, e outros detalhes relacionados. O facto de se poder guardar de um modo automático informação deste género, e disponibilizá-la para consulta sempre que necessário, também não é visto com agrado pelos utilizadores.

Por outro lado, e não menos importante, é a constatação que os operários em ambientes fabris necessitam de canalizar o grosso da sua atenção para a realização das tarefas sob a sua responsabilidade, não estando, de um modo geral, propensos a "perder" tempo a aprender a usar uma interface gráfica. Isto, mesmo que em última análise e como aliás é o objectivo, as interfaces lhe permitam a realização das tarefas, já conhecidas e outras que possam surgir derivadas da utilização de GUIs, de um modo mais rápido e eficaz.

Como já foi referido e é necessário salvaguardar, esta é uma perspectiva genérica dos utilizadores em ambientes fabris, e não uma tentativa de rotular os utilizadores de GUIs em ambientes fabris como pessoas renitentes e até adversas à interacção com

computadores. Casos há, fazendo mesmo parte da experiência da autora em desenvolvimento de interfaces gráficas para ambientes industriais, em que os utilizadores se encontram bastante motivados para o uso de computadores e para a interacção com estes via interfaces gráficas.

Quando aquela motivação não existir, será necessário criar condições para que se torne uma realidade, cabendo aqui a quem concebe a GUI um papel importante na obtenção deste objectivo. Isto pode (e deve) ser conseguido, desenvolvendo uma interface gráfica fácil de aprender e de usar, que implemente todas as funcionalidades requeridas e que vá ao encontro das expectativas do utilizador.

3.1.2 Metáforas em ambientes de fabrico

A utilização de metáforas nas interfaces gráficas com o utilizador reduz de um modo substancial o tempo necessário à aprendizagem de uma interface, permitindo ao utilizador formar um modelo conceptual de como o sistema se comporta muito mais rapidamente (Secção 2.1).

No entanto, nunca é demais referir que as metáforas só são eficazes se fizerem parte do universo de conhecimento do utilizador. O uso de metáforas em interfaces gráficas, cujo significado nada diga ao utilizador, pode tornar-se contraproducente, ou pelo menos não servirá de ajuda a uma melhor e mais rápida compreensão da operação da interface. Se o utilizador não conseguir estabelecer uma relação de conhecimento com as analogias utilizadas na GUI, não podendo portanto usar conhecimentos previamente adquiridos para inferir o modo como a interface se comporta, podem ocorrer duas situações, ambas envolvendo necessariamente a aquisição de novos conhecimentos.

Na primeira, o utilizador não fazendo uso das metáforas porque o significado destas lhe é desconhecido, terá que construir o modelo conceptual do sistema a partir do nada, i.e., sem fazer de uso da sua experiência.

Na outra, seja porque as analogias estão de tal modo embebidas na GUI que é necessário perceber o seu significado para apreender o comportamento da interface, seja por opção do próprio utilizador, pode ocorrer a situação de o utilizador, numa

primeira fase, adquirir conhecimento sobre as metáforas usadas, para, numa segunda fase, utilizar esse conhecimento para lidar com a interface.

Qualquer uma destas situações implica seguramente um maior tempo de aprendizagem da GUI, o que demonstra de facto a necessidade premente de assegurar que no desenvolvimento de GUIs se utilizem metáforas familiares ao utilizador.

A metáfora corrente para a interacção homem-máquina baseia-se acentuadamente no paradigma do trabalhador de escritório. A GUI típica representa a estrutura da interface como uma secretária de escritório, com ficheiros, documentos, e acções baseadas no comportamento típico existente num escritório (arrastar um ficheiro para o caixote do lixo para o destruir, ou pressionar duas vezes o ícone da impressora para imprimir um documento). Isto está adequado para as aplicações, como o processamento de texto, que são representativas do trabalho efectivo que tem lugar dentro de um ambiente de escritório do mundo real.

Contudo, o "escritório virtual" no ecrã do computador aparece como menos indicado quando o ambiente computacional é adaptado para ambientes menos similares, como seja o de fabrico. O ambiente de fabrico não é de todo semelhante a um ambiente de escritório; consequentemente, as mesmas analogias não são válidas. Neste caso é desejável providenciar uma metáfora da interface com o utilizador que seja mais representativa do paradigma físico ao qual o utilizador é familiar.

Interessa pois no desenvolvimento de GUIs para ambientes fabris utilizar metáforas baseadas nestes ambientes, para que se tire real partido da sua utilização. Não é possível caracterizar o tipo de metáforas a ser aqui utilizado dada a diversidade de processos fabris existentes. As interfaces desenvolvidas no campo dos ambientes fabris são dedicadas, pretendendo apenas atingir um segmento de mercado bem identificado e definido. O conjunto de utilizadores finais da interface é passível de ser identificado e caracterizado. Cada caso deve ser tratado como "o" caso, e as metáforas utilizadas na GUI devem fazer transparecer o carácter particular da aplicação.

Em conclusão, as metáforas utilizadas em GUIs para ambientes fabris devem reflectir primordialmente o ambiente específico onde as interfaces se inserem, uma vez

que este condiciona o conhecimento e a experiência dos utilizadores, e portanto a facilidade de aprender e consequentemente de usar a interface.

3.1.3 Linguagem do utilizador

Usar a linguagem do utilizador é uma regra básica que deve ser observada no desenvolvimento de qualquer interface gráfica com o utilizador. Em ambientes de fabrico a sua acuidade reveste-se de um carácter particular.

Por um lado, e dado que de um modo geral os utilizadores não terão conhecimentos no campo dos sistemas computacionais, não será de todo aconselhável cair na tentação de utilizar termos com significado evidente para qualquer pessoa habituada a lidar com computadores, mas que estes utilizadores em particular desconhecem na totalidade.

Por outro lado, os utilizadores inseridos em ambientes fabris possuem uma terminologia própria, muitas vezes consolidada ao longo de varias décadas, e usada diariamente no âmbito do seu trabalho. Devem ser também estes os termos a utilizar na interface. Mesmo que por vezes não sejam termos muito rigorosos do ponto de vista científico, convém notar que o importante é que o utilizador os conheça e os possa reconhecer na GUI.

Se a regra de usar a linguagem do utilizador é tanto mais fundamental quanto mais específico é o campo da aplicação, também é mais fácil de ser observada neste caso. Para aplicações de carácter mais genérico, o universo de putativos utilizadores é de dimensão tão elevada, que por vezes se torna difícil saber se os termos usados possuirão significado conhecido para todos os utilizadores.

Nas aplicações para fins mais específicos, como são as GUIs para ambientes de fabrico, esta regra torna-se mais fácil de ser cumprida porque o universo dos utilizadores é muito mais reduzido e de mais fácil caracterização.

O uso de linguagem própria dos utilizadores nas interfaces gráficas permite uma mais rápida compreensão das analogias ao ambiente de fabrico presentes na interface, ajudando assim à construção e consolidação, pelo utilizador, do modelo conceptual subjacente à aplicação.

3.1.4 Características das GUIs em ambientes de fabrico

Na Secção 2.2 foram analisadas, segundo uma perspectiva genérica, as principais características que uma interface gráfica com o utilizador deve possuir. Torna-se agora importante discutir da relevância e grau de aplicabilidade dessas características, no caso específico de interface gráficas com o utilizador vocacionadas para ambientes de fabrico. Esta análise é feita caso a caso.

Transparência

Esta característica é tanto mais relevante quando se trata de interfaces gráficas com o utilizador para ambientes de fabrico. De facto, o utilizador deve sempre poder ver a tarefa que está a realizar e concentrar-se na sua execução, e não na operação da interface. O seu trabalho consiste em realizar um dado número de tarefas, e não propriamente em tornar-se um perito na utilização da GUI.

Deve ser sempre fornecida realimentação visual ao utilizador para que este se possa assegurar se determinada tarefa foi de facto realizada, e com sucesso. Por outro lado, deve privilegiar-se a apresentação de uma imagem global do processo na interface, em detrimento de imagens ou dados particulares. Informação mais detalhada deve estar disponível e ser da opção do utilizador consultá-la ou não, como, por exemplo, dados históricos do processo, no caso de possuir permissão para o fazer. Existem casos em que tal informação só é acessível por alguns utilizadores, não estando disponível para consulta geral.

Consistência

A interface gráfica com o utilizador deve possuir um comportamento lógico. A consistência dos objectos e do seu comportamento permite que o utilizador possa generalizar os conhecimentos e aplicar conhecimentos adquiridos na realização de uma tarefa particular, para realizar outras tarefas.

Tarefas semelhantes, independentemente do campo de aplicação, devem ser realizadas sempre da mesma maneira. Por exemplo, quer se pretenda ver a capacidade de produção de uma célula de fabrico, quer da área de *shop-floor* inteira, o mecanismo para o fazer deve ser o mesmo. Isto permite uma operação da interface de um modo

intuitivo por parte de utilizador, o que é deveras importante quando se trata de utilizadores não familiarizados com sistemas computacionais. Se o utilizador puder generalizar conhecimentos e não for surpreendido a cada instante pelo comportamento da interface, a sua motivação para utilizar a GUI será crescente.

Funcionalidade

O utilizador em ambientes fabris tem como interesse primordial a realização das tarefas que lhe estão atribuídas, e utilizará a GUI como uma ferramenta para atingir esse objectivo. Neste sentido é necessário que a interface providencie meios que lhe permitam realizar todas as tarefas que deseja, e o encaminhe, através de diálogos por exemplo, na execução daquelas.

Se a interface, porque a nível do desenvolvimento existiu uma definição incompleta das tarefas a realizar, apenas lhe permitir a realização de algumas tarefas, tendo o utilizador que recorrer a outros meios que não a GUI para realizar as outras tarefas, rapidamente será levado a não utilizar de todo a interface. É assim necessário um constante envolvimento dos utilizadores que, volta a referir-se, neste caso podem ser relativamente bem identificados, para a definição da funcionalidade da interface.

Flexibilidade

As GUIs em ambientes de fabrico possuem, tipicamente, um fim específico, por exemplo controlo ou monitorização de um dado processo, e consequentemente a flexibilidade não é uma característica aplicável de um modo global. O modelo conceptual ao invés de genérico deve ser bastante particular e específico do ambiente em causa, e deve pretender-se abarcar um conjunto bem definido de problemas. Um modelo conceptual, particular de um dado ambiente de fabrico, permite que os utilizadores inseridos nesse ambiente estabeleçam uma relação de conhecimento muito mais fácil e rapidamente.

É de notar que a configuração do sistema não desempenha geralmente um papel importante nos ambientes de fabrico, dado o carácter específico da aplicação e dado que representa muitas vezes um elemento perturbador para um utilizador menos experiente em sistemas computacionais, ou pelo menos é ignorada.

Controlo pelo utilizador

Esta é uma característica que, nas GUIs para ambientes de fabrico, não é particularmente relevante, sendo até na maior parte dos casos contraproducente e a evitar. A interacção com frequência necessita ser modal[1], uma vez que deixar o utilizador experimentar comandos e permitir-lhe uma certa "liberdade", pode ter consequências desastrosas. Este é o caso de aplicações de controlo de processos onde os comandos sejam enviados via interface gráfica.

Permitir ao utilizador experimentar todos os comandos pode ter consequências indesejáveis significativas. Neste tipo de aplicações deve ser de facto previsto "tudo o que o utilizador possa fazer de errado" e prevenir essas acções, não as permitindo de todo. A facilidade de o utilizador organizar o trabalho à sua maneira também está, nas GUIs para ambientes de fabrico, bastante limitada, uma vez que na maioria dos casos as tarefas a realizar são sequenciais e possuem uma ordem predeterminada para serem executadas, que não é passível de ser alterada. Esta é outra das razões para a utilização frequente de interacção modal nas interfaces gráficas para ambientes de fabrico.

Fiabilidade

O utilizador precisa de ter confiança no desempenho de qualquer aplicação, e essa confiança só se consegue com aplicações fiáveis. Dito isto, deve ser tido em conta que o utilizador de GUIs em ambientes fabris mostra-se muitas vezes renitente à utilização de novas tecnologias, "desconfiando" mesmo da eficácia destas. É por isso que esta faceta se revela aqui da maior importância.

Se a aplicação não possuir fiabilidade, a sua credibilidade é diminuta ou nula, e constitui um pretexto (mais do que razoável, convenha-se!) para não ser utilizada. Acresce a isto, que o uso de GUIs pouco fiáveis em ambientes de fabrico, dada a importância dos processos em questão, pode ter como consequência prejuízos incalculáveis, prejuízos estes directamente imputáveis a quem concebeu a interface.

[1] Um modo, num sistema computacional interactivo, é um estado da interface gráfica com o utilizador que se mantém por um dado período de tempo, não está associado com qualquer objecto particular, e só tem como objectivo interpretar as entradas do utilizador [Tesler, 1981].

Os erros tem que ser evitados na totalidade e a interface deve ser testada exaustivamente, antes de ser utilizada de facto.

Facilidade de aprendizagem

Uma das componentes porventura mais importantes na aprendizagem de qualquer sistema é a experimentação. No entanto, quando se trata de GUIs em ambientes de fabrico esta componente não pode muitas vezes ser explorada. Por razões já invocadas nos pontos anteriores, não pode ser dada permissão ao utilizador para executar todo o tipo de comandos, até porque muitos são irreversíveis. Por esta razão, a facilidade de aprendizagem desempenha nestes tipo de interfaces gráficas um papel fundamental.

A simplicidade na apresentação da informação deve ser uma constante, e devem ser sempre fornecidas ao utilizador indicações e mensagens claras do que "deve" fazer a seguir. No caso de comandos irreversíveis, e mesmo que se exija confirmação do operador para a sua execução, isto não constitui garantia que o operador saiba o que está a fazer. Nestes casos o utilizador deve ser posto a par das consequências da emissão de determinado comando.

A sintaxe usada na interface constitui um factor importante da facilidade (ou não) de aprendizagem da interface, e mais uma vez se chama a atenção para a necessidade de usar a linguagem própria do utilizador. A utilização de comandos universais só faz sentido se o operador conhecer o seu significado. É preferível usar termos específicos que só possuam significado dentro do âmbito particular da aplicação, mas que o operador está familiarizado com, do que utilizar termos universais que são compreensíveis apenas por utilizadores com experiência em interfaces gráficas.

3.2 Participação dos utilizadores no desenvolvimento

A participação dos utilizadores em várias fases do desenvolvimento de interfaces gráficas, como por exemplo na definição das tarefas que a interface deve suportar, é mais do que desejável. Os utilizadores não têm sempre razão e não conseguem antecipar com exactidão como vão usar novas tecnologias. Quem desenvolve GUIs para ambientes industriais deve ter esta perspectiva de futuro, que falta aos utilizadores. Mas, e este é geralmente o caso em ambientes de fabrico, se quem

concebe a interface não possui conhecimentos profundos no domínio de aplicação que a GUI deve suportar, então há que reconhecer que os utilizadores sabem muito mais acerca das tarefas a suportar pela interface gráfica.

Até aqui foi assumida uma separação entre os papeis de quem concebe a interface e dos utilizadores. Os *designers* não são utilizadores; recolhem informação dos utilizadores e reflectem-na nos sistemas que constroem. Existe uma abordagem alternativa, cujo pioneirismo se deve a trabalhadores na Escandinávia, que rejeita esta estrutura.

No *Design Participado* (Participatory Design) os sistemas são desenvolvidos pelo trabalho conjunto de quem concebe a interface e dos utilizadores: a ideia é *desenvolver com* em vez de *desenvolver para*. O poder de decidir se o sistema é de uma dada maneira em vez de outra, não está reservado aos *designers*, mas sim compartilhado por estes e pelos utilizadores trabalhando em colaboração.

Existem três atributos fundamentais que definem o conceito de *Design Participado* [Blomberg e Henderson, 1990]:

- O objectivo da actividade é melhorar a qualidade de vida dos utilizadores no que diz respeito ao trabalho.
- A actividade é de colaboração, com todos os objectivos e decisões activamente negociados e não impostos.
- A actividade é iterativa, no sentido que as ideias são testadas e ajustadas verificando como funcionam na prática.

3.2.1 Factores político-sociais

Não é de todo estranho o facto de o conceito de Design Participado ser oriundo da Europa e não dos Estados Unidos. Existem diferenças substanciais a nível de gestão empresarial entre muitos países europeus e os Estados Unidos que fundamentam este facto.

Em diversos países europeus os trabalhadores possuem uma influência significativa nas condições de trabalho, que são regulamentadas muito mais

estritamente que nos Estados Unidos. Em muitos países os trabalhadores têm assento nos conselhos de direcção das empresas, e os trabalhadores devem ser consultados sobre a introdução de novas tecnologias no local de trabalho. Isto contrasta com a visão prevalecente nos Estados Unidos segundo a qual a gestão deve decidir quais os sistemas que são melhores para a produtividade e impor esses sistemas aos trabalhadores, cujas opiniões sobre o assunto basicamente não têm importância e não são consideradas.

Muitas pessoas argumentam que os trabalhadores europeus possuem demasiado poder e detêm demasiada influência nas condições de trabalho, e que se devia adoptar o sistema dos Estados Unidos. A verdade no entanto, é que diversos países europeus têm uma produtividade *per capita* mais elevada que os Estados Unidos, e os Estados Unidos têm vindo a perder terreno nas exportações, especialmente nas indústrias avançadas, como Michael Porter demonstra no seu livro [Porter, 1990].

Desenvolver novas tecnologias em estreita colaboração com utilizadores potenciais, é uma boa ideia em muitas indústrias por várias razões que não se restringem às peculiaridades das interfaces gráficas com o utilizador. A impressão moderna utilizada actualmente na imprensa foi desenvolvida por uma empresa de novas tecnologias financiada por alguns dos seus potenciais utilizadores, jornais ingleses de grande tiragem [Porter, 1990]. Uma colaboração deste tipo fornece a informação necessária a desenvolver um sistema que tenha utilidade de facto, logo com sucesso, bem como permite o desenvolvimento de um mercado inicial.

3.3 Requisitos e visualização da informação

O campo dos ambientes de fabrico é tão vasto, existe um número tão alargado de indústrias, de processos de fabrico diferentes e de estruturas de organização distintas, que qualquer particularização pecará sempre por uma ligação excessiva a um segmento de indústria específico. De facto, não é possível fazer uma caracterização específica do tipo de informação a ser tratada nas interfaces gráficas em ambientes de fabrico.

Por outro lado, as aplicações também são inúmeras; por exemplo, pode tratar-se de GUIs para aplicações de controlo ou monitorização de processos produtivos, para citar os casos mais típicos.

No entanto, será possível referir de um modo genérico as necessidades em termos de informação e visualização desta para interfaces gráficas com o utilizador em ambientes de fabrico. Isto permitirá dar uma perspectiva global dos requisitos a nível de informação neste tipo de interfaces e a sua visualização, mesmo que este ou aquele caso em particular não seja coberto.

3.3.1 Entidades físicas

A estrutura hierárquica de uma fábrica engloba tipicamente três níveis. O nível inferior é constituído pelos dispositivos de fabrico, i.e., equipamento que intervém directamente no processo de fabrico. O nível seguinte consiste no agrupamento ordenado de um dado número de dispositivos de fabrico. Por exemplo, este nível pode incluir um conjunto de dispositivos de fabrico dedicados a funções similares (e.g., células de fabrico) ou agrupados sequencialmente de acordo com o processo produtivo (e.g., linha de montagem). Por último, existe o nível da oficina (*shop-floor*) que abarca toda a zona efectiva de fabrico. Nos casos de fábricas de grandes dimensões, usualmente existe ainda um quarto nível, logo abaixo da oficina, e comummente designado por área de fabrico.

Do mesmo modo que os diversos níveis reflectem a organização hierárquica, também as interfaces gráficas, dependendo do nível onde estão inseridas, devem reflectir as necessidades em termos de informação dos diversos níveis. Adicionalmente, a informação a ser apresentada na GUI, está obviamente dependente da aplicação subjacente à interface.

Por exemplo, numa GUI para uma aplicação de monitorização ao nível da oficina interessará fornecer informação sobre a localização do equipamento de transporte, as existências no armazém, a capacidade das áreas/células/linhas, ou relativa aos recursos humanos da fábrica, entre outra. No entanto, se se considerar, por exemplo, uma interface gráfica para um controlador de célula, os requisitos em termos de informação alteram-se. Neste caso, interessará apresentar, por exemplo, informação sobre as ordens de fabrico actualmente a serem processadas, como seja o estado destas (e.g., completas/incompletas), e sobre o estado dos dispositivos de fabrico existentes na célula.

Para além das considerações sobre a informação a veicular nos diversos tipos de aplicações, e em níveis hierárquicos distintos, é necessário ter em conta requisitos adicionais em termos do modo como a informação é tratada pela interface gráfica.

Um problema que se coloca com frequência na concepção de GUIs para ambientes de fabrico é o modo de apresentar grandes quantidades de informação no espaço limitado do ecrã. Isto ocorre tipicamente, nas GUIs ao nível da oficina ou da área, e para aplicações de monitorização.

No sentido de obviar este problema, deve considerar-se: a divisão do ecrã em áreas mais pequenas representativas das áreas físicas e que permitem o acesso a mais informação sobre uma dada área; a utilização de ícones (estes ocupam menos espaço do que texto); e como último recurso a utilização de *scroll-windows* (dado que o seu manuseio por vezes é confuso para o utilizador). Estas técnicas permitem apresentar grandes quantidades de informação, de uma forma ordenada, ao utilizador.

Ainda relacionado com este problema, uma funcionalidade deveras importante a considerar nas interfaces gráficas com o utilizador para ambientes de fabrico, é a capacidade de *zoom*. A divisão do ecrã em áreas mais pequenas, ou a utilização de ícones, pode ser bastante eficaz se aliada à capacidade de *zoom*, na apresentação de grandes quantidades de informação. Assim, o utilizador pode seleccionar uma dada entidade de interesse no ecrã principal, e obter informação particular concernente ao item seleccionado.

Por exemplo, se a estrutura da fábrica contemplar um modelo baseado em linhas de montagem, estas podem aparecer sob a forma de ícones, numa interface gráfica ao nível da oficina; a selecção de uma dada linha de montagem (através do seu ícone representativo) permitiria obter informação sobre os dispositivos constituintes da linha seleccionada. Estes dispositivos devem ser apresentados com a mesma disposição espacial que possuem na linha de montagem.

Deve notar-se que as entidades físicas devem ser visualizadas na interface gráfica de um modo tão realista quanto possível, com símbolos gráficos cuidadosamente desenhados para que a sua identificação se torne imediata, e mantendo as relações hierárquicas e de conteúdo e as ligações físicas existentes na realidade da fábrica.

De facto, e sempre que possível, deve tirar-se o máximo partido das capacidades gráficas do sistema. Por exemplo, a animação constitui um meio muito poderoso de veicular informação sobre entidades físicas cuja localização espacial se altera com frequência. Se tomarmos em consideração o antigo provérbio chinês de que "uma imagem vale por mil palavras", aliás aplicável a todos os conceitos subjacentes às interfaces gráficas, pode dizer-se relativamente à animação que "uma imagem em *movimento* vale por mil imagens *paradas*"!

A animação pode ser utilizada com bastante sucesso em interfaces gráficas com o utilizador para ambientes de fabrico para descrever, por exemplo, o fluxo das ordens de fabrico através da zona fabril, ou o percurso efectuado pelo equipamento de movimentação de materiais.

3.3.2 Processos

Os processos de fabrico devem ser visualizados na GUI através da apresentação gráfica, da evolução no tempo até ao instante actual, dos valores das variáveis características do processo. A evolução temporal do processo a apresentar no ecrã deve incluir apenas o intervalo de tempo relevante para a fase actual do processo.

Por exemplo, numa GUI dedicada ao controlo de um processo com n fases, a informação por defeito a apresentar no ecrã, deve ser apenas a informação concernente à fase em que o processo actualmente se encontra. Informação adicional sobre fases anteriores do processo pode ser consultada por escolha deliberada do operador, mas não deve ser a apresentação por defeito da interface.

O conceito fundamental subjacente à visualização do estado de um dado processo numa GUI, é fornecer por defeito apenas a informação relevante num dado instante de tempo. Esta informação implica frequentemente, para além do instante actual, a consideração de um determinado intervalo de tempo anterior.

No entanto, deve garantir-se sempre que este intervalo de tempo é de facto relevante, quer para a apreensão do estado actual do processo, quer para a antecipação de possíveis condições de avaria e/ou mau funcionamento, por parte do operador. Apresentar informação adicional que não está directamente relacionada com o instante

de tempo actual, em conjunto com o estado do processo, é contraproducente, atendendo a que pode prejudicar a percepção do operador do estado actual do processo.

Acresce a isto que o modo de apresentação das variáveis deve ser tão realista quanto possível, devendo tirar-se partido das possibilidades gráficas do sistema; por exemplo, se uma dada variável é contínua, deve ser apresentada no ecrã mantendo as suas propriedades de continuidade. Aqui deve ser salientada a importância que os gráficos têm na visualização deste tipo de informação.

Alarmes e avisos

A ocorrência de um aviso significa que um dado processo se encontra na vizinhança das condições limite de operação. Um alarme ocorre numa situação de maior gravidade, significando que as condições limite de operação do processo foram ultrapassadas, resultando na interrupção do processo.

A ocorrência de alarmes e/ou avisos, pela sua natureza de tempo crítico de actuação, não pode passar despercebida ao operador, escondida por outra informação. Assim, informação relativa à ocorrência de anomalias deve ser apresentada no ecrã sobreposta a qualquer outro tipo de informação existente no momento.

A necessidade de focar a atenção do operador para a ocorrência de avisos e alarmes obriga a equacionar com acuidade as diferentes alternativas passíveis de garantir a passagem prioritária desta informação. Em particular é de enfatizar o uso apropriado da cor e do som, não esquecendo as ligações de automatismos que lhe estão atribuídas em outras áreas da actividade humana.

A cor devido às suas componentes de fácil percepção e de automatismos normalizados ao nível do subconsciente, desempenha um papel fundamental neste tipo de interfaces, em particular na notificação da ocorrência de alarmes e/ou avisos. É completamente distinto utilizar a cor amarela para apresentar a ocorrência de um aviso num processo, apresentando um alarme a vermelho (as pessoas possuem a percepção que a cor vermelha indica maior gravidade que o amarelo), face à simples notificação no ecrã da ocorrência de um aviso ou alarme através de uma frase.

O som é outra faceta importante das interfaces com o utilizador nesta área. A afixação de alarmes ou avisos no ecrã pode (e com frequência deve) ser acompanhada de um sinal sonoro.

Os diferentes sentidos possuem diferentes prioridades a nível de atenção (embora as prioridades dependam das circunstâncias e das expectativas do utilizador); é geralmente mais fácil chamar a atenção de uma pessoa utilizando um estímulo auditivo do que um visual. Uma constatação deste facto provém de estudos realizados com automobilistas que revela que, na generalidade dos casos, estes prestam muito mais atenção a sinais sonoros das buzinas do que a sinais de luzes com máximos. Os bons sistemas interactivos tiram partido dos diversos sentidos do utilizador, particularmente usando estímulos auditivos quando a atenção rápida é requerida.

Para além destas considerações, a notificação na interface gráfica com o utilizador da ocorrência de alarmes e avisos, deve ser acompanhada de informação adicional fornecendo indicação da proveniência do alarme ou aviso (máquina onde ocorreu e a localização física desta) e da causa que provocou a sua ocorrência (se conhecida). Isto permite que o utilizador possa efectuar uma rápida identificação do problema e fornece-lhe meios para tomar uma medida judiciosa.

Operações locais

Numa situação ideal e em regime normal de funcionamento, o controlo do processo de fabrico é efectuado pelos sistemas computacionais (e.g., controlador de célula) e os dados necessários, por exemplo relativos às ordens de fabrico ou programas para *robots*, são trazidos de uma base de dados de um modo automático via rede.

No entanto, existe frequentemente a necessidade de a interface possibilitar a realização de operações locais, não integradas no sistema de informação global, nomeadamente para lidar com situações anómalas. Tipicamente, estas operações locais consistem na emissão de comandos e na entrada de dados, via interface gráfica. Por exemplo, a ocorrência de um alarme, ou mesmo de um aviso, pode implicar o envio de comandos através da interface gráfica com o utilizador, nos casos em que o sistema informático não possua uma base de conhecimento que lhe permita decidir qual a acção a tomar.

No caso de envio de comandos através da interface e da entrada de dados pelo operador, é preciso garantir que tais operações locais são realizadas correctamente, dadas as consequências desastrosas que frequentemente podem advir de uma incorrecta operação. A apresentação de comandos na interface gráfica deve ser feita de um modo cuidado. O operador deve visualizar de alguma maneira qual a acção no processo de fabrico que a sua actuação desencadeia. A confirmação de comandos neste tipo de interfaces é muitas vezes imperiosa, sendo um modo de se ter uma certa garantia (que nunca é absoluta), de que o operador sabe o que está a fazer.

Adicionalmente, cada botão de comando deve ter uma correspondência inequívoca com um só comando e uma representação única, de modo a não se criarem situações menos claras em que um dado botão com uma mesma representação, tem várias funções consoante o contexto onde se insere.

Dadas as consequências nocivas que podem advir de uma entrada de dados incorrecta, é imprescindível que esta seja realizada de um modo perfeitamente robusto, eliminando qualquer possibilidade de o operador cometer erros. É assim necessário uma validação constante destes dados, verificando possíveis inconsistências ou incoerências.

Uma das regras de ouro a seguir no desenvolvimento de GUIs em ambientes de fabrico é fazer a entrada de dados por parte do operador, sempre que possível, através de um mecanismo de escolha múltipla. Quando o operador necessita de introduzir os dados completos, está mais atreito a cometer erros do que se apenas tiver de escolher entre algumas alternativas.

Os mecanismos de escolha múltipla devem ser configurados automaticamente dependendo das escolhas anteriores, de acordo com as opções disponíveis. Informação adicional, como opções não válidas (mesmo que a sua visualização indique que a sua escolha não é permitida) pode criar confusão ao operador, e não acrescenta nada à compreensão da interface. Casos há em que a escolha anterior reduz uma ou mais opções posteriores a um único valor; nestes casos a entrada dos dados válidos deve ser feita automaticamente.

3.3.3 Dados históricos

Informação adicional como dados históricos do processo de fabrico, recolhidos em fases anteriores e guardados para uso futuro, como a evolução do processo no tempo ou a ocorrência passada de alarmes e avisos, e outra informação estatística, devem ser visualizados só por escolha deliberada do operador. Este tipo de informação não deve ser justaposto no ecrã junto a informação mais relevante e concreta em termos de funcionamento num dado instante de tempo.

Embora a informação histórica seja bastante importante, até para criar uma base de conhecimento baseada em ocorrências passadas, a sua apresentação não deve ser feita por defeito, para não interferir na percepção do utilizador do estado actual do processo.

Adicionalmente, é necessário ter em conta que informação deste tipo muitas vezes não deve estar disponível para consulta de todos os utilizadores, como pode ser o caso de informação estatística relativa à assiduidade dos trabalhadores ou informação histórica dos comandos enviados via interface gráfica e por quem.

Nestes casos, será necessário implementar mecanismos de protecção para garantir que a informação só é acedida por quem de direito. Isto pode ser conseguido utilizando códigos únicos de identificação do utilizador, por exemplo, tornando a disponibilização da informação dependente do utilizador que a solicita. Neste sentido, devem configurar-se os menus ou os itens disponíveis dos menus que permitem aceder a este tipo de informação de acordo com o nível de permissão do operador. Para além de constituir uma boa política geral a interface só apresentar acções permitidas ao utilizador, nestes casos, esta técnica possibilita ainda evitar hipotéticos constrangimentos que o utilizador pode experimentar, se se aperceber que existe um determinado tipo de informação que está acessível na interface, mas que ele não tem permissão de consultar.

3.4 Constrangimentos relativos ao ambiente

Existem determinados constrangimentos relativos aos ambientes de fabrico que podem impor restrições a nível da concepção de GUIs vocacionadas para estes ambientes.

Aliás, este é o caso no desenvolvimento de qualquer aplicação: as soluções encontradas e implementadas estão dependentes do tipo de ambiente onde a aplicação se insere.

Nas secções seguintes são descritos alguns destes constrangimentos, nomeadamente os decorrentes da necessidade de concorrência de processos e das limitações a nível dos mecanismos de interacção, para interfaces gráficas com o utilizador em ambientes de fabrico.

3.4.1 Concorrência de processos

Em ambientes de fabrico, os processos das próprias interfaces gráficas com o utilizador devem ser considerados e desenvolvidos como processos independentes dos restantes processos existentes, por exemplo, de monitorização ou controlo.

Neste quadro de independência dos processos, devem ser criados mecanismos que permitam às interfaces gráficas seguir o estado dos processos efectivos de monitorização e controlo e dos elementos controlados e monitorizados. A solução passa por janelas projectadas para controlo e monitorização de processos de fabrico terem associadas máquinas de estados que evoluem em consequência de eventos recebidos do subsistema de controlo ou monitorização.

As aplicações são consideradas como partes distintas, com código próprio, requerendo um processo de comunicações entre a aplicação da interface e o processo; isto passa pela definição da interface como um processo autónomo comunicando com os restantes processos de acordo com o modelo cliente/servidor (Secção 5.2), sendo a interface gráfica o cliente.

Adicionalmente, atendendo a que as áreas de fabrico são na generalidade dos casos de dimensão considerável, devem ser suportadas múltiplas instâncias das aplicações gráficas, para permitir o acesso ao estado do mesmo processo em mais do que um ponto da área de fabrico.

Para que seja possível existir mais do que uma interface gráfica, a aplicação subjacente à interface deverá suportar concorrência. Acresce a isto, a necessidade de se implementarem formas de estabelecimento de múltiplas conexões entre as várias

instâncias, e entre a aplicação gráfica e o processo que lhe está subjacente, garantindo sempre que as mensagens são trocadas entre as entidades correctas.

Não se pode permitir que vários utilizadores, em diferentes locais da área de fabrico, emitam comandos em simultaneidade para controlar o mesmo processo, através de diferentes instâncias da aplicação. Isto pode ser evitado utilizando por exemplo serviços de alto nível de gestão de concorrência, como `tomar controlo` e `libertar controlo` (eventualmente suportados por semáforos), garantindo assim a integridade dos dados.

3.4.2 Limitações a nível da interacção

Pela sua natureza, os ambientes da fabrico podem impor condições nos mecanismos de interacção a serem utilizados na interface gráfica, que devem ser levadas em conta aquando da concepção da aplicação.

O rato constitui o mecanismo mais comum de o utilizador aceder às potencialidades das interfaces gráficas correntes, pelas suas características de interactividade, nomeadamente a facilidade de rapidamente seleccionar e mover itens da aplicação. No entanto, podem existir casos em ambientes de fabrico, em que devido às condições de trabalho existentes, não seja possível fazer uma utilização efectiva do rato. Neste caso, a interface deve ser implementada com esta limitação presente; mover ícones através de teclas, por exemplo, é no caso geral bastante limitativo e difícil. Deve ainda considerar-se a viabilidade de utilizar outros mecanismos de interacção (e.g., *trackball*, ecrã táctil).

Por outro lado, a utilização dos teclados convencionais para permitir ao utilizador interagir com a interface gráfica, pode também não se revelar a melhor opção. Nos casos em que apenas exista um conjunto limitado de acções a tomar, e não seja necessário a introdução de dados na GUI, provavelmente utilizar um teclado dedicado, com apenas algumas teclas de funções, será mais aconselhável dada a facilidade de operação que permite. A existência de inúmeras teclas pode ser consideravelmente confusa para o utilizador, e não traz vantagens acrescidas dado que não são utilizadas de facto.

Outro aspecto a ser levado em consideração é o uso do som, por exemplo para chamar a atenção do operador para a ocorrência de uma falha. Os ambientes de fabrico são frequentemente ruidosos, e se a utilização de som constitui o único meio de obter a atenção do utilizador, há que garantir que ele de facto consegue ouvir o som. Caso contrário, outras hipóteses terão que ser consideradas, como por exemplo a utilização de cor. A conjunção dos dois mecanismos é, mesmo no caso de ambientes silenciosos, aconselhável; acaba por constituir mais uma tentativa de garantir que o utilizador se apercebe de determinado acontecimento.

Deve contudo aqui ser notado que quando se fala na utilização de som, em GUIs para ambientes de fabrico, como um meio de chamar a atenção do utilizador para a ocorrência de uma falha, não se está a referir os *beeps* que os computadores emitem (e.g., quando se comete um erro!). Este tipo de som provavelmente será com bastante frequência ignorado pelo operador, até porque deve ser tido em conta que este pode (e frequentemente é este o caso) não estar sentado defronte do computador, ou encontrar-se a uma distância suficientemente curta que lhe permita aperceber-se destes sons.

A utilização de som nos ambientes de fabrico deve revestir-se de medidas mais eficazes, como, por exemplo, utilizando uma buzina para o efeito, que para além da intensidade superior do som está inerentemente associada à ocorrência de problemas.

3.5 Conclusões

A interface gráfica com o utilizador do Apple Macintosh foi pioneira na criação de uma nova geração de utilizadores, que se sentiram mais à-vontade em interagir com um computador através de um mecanismo de '*point and click*' do que através de uma interface baseada em linhas de comandos. A evolução subsequente nesta área levou a que hoje em dia, virtualmente qualquer computador, desde o PC ao *mainframe*, possua uma interface gráfica de modo a facilitar e tornar mais produtiva a interacção com o utilizador.

A automação industrial introduziu definitivamente o computador nos ambientes de produção com funções de supervisão e controlo das tarefas a serem realizadas aos diferentes níveis da organização da área de fabrico. Neste ambiente, acentua-se a necessidade de introdução de automatismos de execução, a ergonomia dos postos de

trabalho, e a consequente necessidade de uma interface gráfica com o operador humano para efeito de interacção homem-computador.

Neste capítulo enquadraram-se as interfaces gráficas com o utilizador num ambiente de fabrico, dentro da área mais vasta das interfaces gráficas com o utilizador de aplicação genérica.

Pela sua diversidade, é impossível englobar numa única caracterização particular, a totalidade dos ambientes de fabrico no que diz respeito a factores concernentes às interfaces gráficas com o utilizador. No entanto, abordaram-se alguns aspectos de carácter relativamente geral que dizem respeito às GUIs vocacionadas para ambientes de fabrico.

O requisito mais importante a reter no domínio das interfaces gráficas com o utilizador para ambientes de fabrico, resulta da necessidade de que em cada instante o operador deve ser colocado unicamente face à informação relevante disponível no momento, e em cada caso particular, em diferentes condições de funcionamento, deve ser disponibilizado um modo inequívoco de procedimento. Isto deve ser feito sem possibilidade de interpretações menos óbvias por parte do operador das acções que dele se esperam.

Neste sentido deve fazer-se notar a necessidade de usar a linguagem própria do utilizador, providenciar interfaces gráficas fáceis de compreender e usar, fornecer mensagens concisas e indicações claras ao utilizador do que fazer, prevenir os erros, e colaborar de perto com os utilizadores em diversas fases (ou até na totalidade) do desenvolvimento.

Capítulo 4

Metodologias na concepção de GUIs

Porquê focar a atenção no processo de desenvolvimento de GUIs? Porque não descrever apenas o conjunto de características que definem uma boa interface gráfica com o utilizador (Secção 2.2)? O facto de se focar o processo em vez de apenas o resultado final merece alguma explicação.

O objectivo de uma interface gráfica com o utilizador é facilitar a sinergia entre o ser humano e o computador: as interfaces gráficas permitem às pessoas aprenderem os sistemas computacionais rapidamente e utilizá-los de um modo efectivo. Logo, uma interface deve ir ao encontro das tarefas às quais serve de suporte, bem como dos utilizadores que vão usar a interface. Atendendo a que existe uma variedade quase infinita de tarefas e de utilizadores, não existe uma definição simples do que uma "boa" interface de facto é.

Existem várias tentativas de fornecer linhas mestras[1] extensas e gerais para a construção de uma boa interface gráfica com o utilizador, mas estas linhas mestras são geralmente demasiado vagas para serem utilizadas de um modo efectivo. Por exemplo, uma regra geral pode ser "providenciar respostas adequadas". Mas como é que se determina o que é adequado?

Linhas mestras mais específicas para os elementos da interface gráfica final também foram desenvolvidas, fornecendo sugestões sobre, por exemplo, como deve ser feito o desenho dos menus, como denominar os ícones, e outras directivas. Mas este género de linhas mestras também não está isento de problemas. É impossível cobrir todas as combinações de tarefas, utilizadores e plataformas tecnológicas das interfaces; logo, nenhum conjunto de linhas mestras estará completo e abordará todos os casos possíveis.

Adicionalmente, estas listas de sugestões são, de um modo geral, tão extensas que se tornam difíceis de aplicar. Mais, por vezes para uma mesma situação existem frequentemente directivas contraditórias, e torna-se necessário recorrer à intuição para decidir quais são as mais importantes.

Pode fazer-se aqui uma analogia entre construir uma interface que seja bem sucedida e cortar um bocado de fio de um tamanho "certo". Linhas mestras gerais para o tamanho do bocado de fio, do tipo "suficientemente comprido para a função", não são de grande ajuda; e uma lista de definições específicas do tamanho correcto para todas as utilizações possíveis, não teria fim: 10 cm para atar um ramo de flores, 70 cm para um embrulho pequeno, etc. Mas é fácil descrever um processo que produza o tamanho certo: começar com um fio muito longo, atar as flores, embrulho, ou o que quer que seja, e cortar o fio que sobra.

Similarmente, não chega especificar todas as características que as GUIs na sua versão final devem possuir. É necessário apresentar métodos e procedimentos que permitam construir boas interfaces gráficas com o utilizador. Isto não quer dizer que o

[1] Linhas mestras (*guidelines*) na concepção de interfaces gráficas com o utilizador são um conjunto de directivas e sugestões que têm por objectivo primordial ajudar quem concebe interfaces gráficas a fazer escolhas fundamentadas, evitar erros (que outras pessoas já notaram) ou usar certas técnicas que já se revelaram eficazes.

mero facto de se observarem as metodologias descritas a seguir produzirá, como que por magia, uma boa interface. Para além da utilização destas metodologias na concepção de GUIs, devem ser tomadas decisões ao longo de todo o processo de desenvolvimento, baseadas no conhecimento dos utilizadores, nas suas capacidades cognitivas e limitações, e nas tarefas que a interface deve suportar. Acresce a isto que a interface só será bem sucedida se estiver integrada no processo de produção do *software* como um todo.

Este capítulo descreve metodologias a seguir no processo de desenvolvimento de GUIs que ajudam a produzir boas interfaces gráficas com o utilizador.

4.1 Processo de desenvolvimento

Uma interface gráfica com o utilizador só pode ser considerada bem sucedida se permitir fazer o que o utilizador necessita, e de um modo que este encontre apropriado.

Com este objectivo em mente, enunciam-se aqui os diversos passos que tipicamente [Lewis e Rieman, 1993] constituem o processo de desenvolvimento de interfaces gráficas com o utilizador, e que se pretende serem representativos das várias fases envolvidas na concepção de GUIs.

4.1.1 Definir quem vai usar o sistema e para fazer o quê

O termo consagrado para este passo é "análise das tarefas e dos utilizadores". A necessidade da análise das tarefas é óbvia: um sistema não é bom a menos que permita ao utilizador realizar as tarefas que necessita.

Mas, para além de fazer o que é necessário, um dado sistema tem que se fundir no mundo do utilizador e no seu trabalho. A interface deve requisitar informação ao utilizador na ordem que é provável que ele a receba, deve facilitar a correcção de dados incorrectamente introduzidos, o *hardware* deve adequar-se ao espaço que o utilizador tem disponível, entre outras considerações relevantes já abordadas (Capítulo 2).

A análise das tarefas e dos utilizadores requer uma colaboração estreita e pessoal entre quem desenvolve a GUI e quem a vai usar de facto.

4.1.2 Escolher tarefas representativas

Neste passo, é necessário identificar as tarefas representativas do que a interface gráfica deve permitir realizar. Estas tarefas devem ser tarefas que os utilizadores descreveram como pretendendo que fossem implementadas na interface, e podem ser inicialmente descritas por algumas frases. Mais tarde podem ser expandidas até ao nível de detalhe necessário à sua implementação.

As tarefas seleccionadas devem cobrir de uma forma razoavelmente completa a funcionalidade da interface. Produzir um conjunto efectivo de tarefas poderá ser considerado um teste ao conhecimento que foi adquirido relativamente aos utilizadores e ao seu trabalho.

4.1.3 Copiar

Copiar não deve ser entendido aqui no sentido de plágio. Devem ser procuradas soluções já testadas em interfaces populares existentes e aferir do seu grau de aplicação à interface em questão.

Soluções eficazes e que já passaram no teste com os utilizadores têm uma maior probabilidade de sucesso. Isto não invalida que a sua utilização deva ser revestida de cuidado especial no sentido de assegurar que as soluções encontradas são de facto as adequadas para um dado caso particular.

4.1.4 Criar uma descrição inicial da GUI

Esta descrição inicial deve ser feita no papel, o que possibilita que se faça uma primeira análise sem, no entanto, ser dispendido todo o esforço necessário à sua programação. Mesmo com sistemas integrados de desenvolvimento de GUIs, o esforço necessário à criação de protótipos simples leva ao compromisso com determinadas decisões, demasiado cedo no processo de desenvolvimento.

As tarefas representativas encontradas devem ser usadas como um teste para garantir que a interface possui toda a funcionalidade requerida. Se não se conseguem realizar todas as tarefas, então a definição actual da interface necessita ser melhorada.

4.1.5 Analisar detalhadamente a descrição inicial

Os custos de construir uma interface gráfica com o utilizador na totalidade, sem efectuar uma análise que permita prever o seu desempenho, e testá-la com um número suficiente de utilizadores para que todos os problemas de fundo sejam revelados, são inaceitavelmente elevados. Existem problemas que podem ser desde logo identificados através de uma análise detalhada da interface, realizada ainda numa fase inicial do desenvolvimento.

Aspectos a serem aqui considerados são, por exemplo, o número de teclas e operações mentais (decisões) para realizar as tarefas que a interface deve suportar. Isto permite identificar tarefas que necessitem um número demasiado elevado de passos para serem realizadas.

4.1.6 Construir um protótipo

Após uma análise da descrição no papel da GUI, é chegada a altura de construir algo mais concreto que possa ser apresentado aos utilizadores e que possa servir de uma descrição mais detalhada da interface para futuros desenvolvimentos.

Toda a funcionalidade da interface gráfica não necessita ser implementada nesta fase. Os esforços devem ser concentrados primordialmente nas partes da interface necessárias às tarefas representativas. Funcionalidades subjacentes da interface ainda não implementadas, podem ser emuladas recorrendo à técnica do feiticeiro de Oz (descrita mais adiante na Secção 7.1.2).

4.1.7 Testar a GUI com os utilizadores

Independentemente da esforço que tenha sido posto na análise da GUI, a experiência mostra que existem sempre problemas que só aparecerão quando a interface for testada com os utilizadores.

Aos utilizadores deve ser pedido a realização de uma ou mais tarefas representativas que a interface deve suportar.

4.1.8 Iterar

O teste com os utilizadores irá sempre pôr a descoberto alguns problemas da interface. Esse é o objectivo do teste: melhorar a interface. É assim necessário avaliar os resultados do teste, pesar o custo das correcções contra a severidade dos problemas, rever a interface e depois testar de novo.

Um dos aspectos a ter em conta aquando da revisão da interface é que os elementos que a constituem não funcionam de modo isolado. Rever um menu para resolver um problema que ocorre com uma dada tarefa, pode criar problemas com outras tarefas.

As iterações devem parar quando os objectivos definidos inicialmente tiverem sido atingidos.

4.1.9 Construir a GUI

O ponto chave a considerar na construção de GUIs é fazê-lo tomando em conta que alterações futuras possam ser realizadas. Deve tentar-se antecipar modificações menores usando variáveis cuja alteração seja fácil; alterações mais substanciais devem ser previstas desenvolvendo código claramente modular.

Neste sentido, no capítulo seguinte são referidas diversas técnicas que desempenham um papel fundamental no desenvolvimento de GUIs, permitindo que a construção, manutenção e alterações que se revelem necessárias efectuar no *software*, sejam realizadas de um modo eficaz e facilitado.

4.1.10 Manter o contacto com os utilizadores

Para além de ajudar a responder à questão óbvia se o sistema está de facto a realizar as tarefas para as quais foi concebido, a interacção com os utilizadores depois da interface ter sido colocada no mercado, permite revelar outras funcionalidades que os utilizadores eventualmente pretendam ver contempladas em versões futuras da GUI.

Este contacto é muitas vezes protagonizado dentro do âmbito da prestação de serviços, nomeadamente de ajuda aos utilizadores e de manutenção do *software*.

4.1.11 Construir novas versões

Por muito cuidado que se tenha colocado na concepção da GUI, no sentido de providenciar a funcionalidade requerida e corresponder às expectativas dos utilizadores, a interface estará provavelmente inadequada no espaço de um par de anos.

Poucos produtos (se é que existem alguns) conseguem manter-se actuais sem que novas versões sejam construídas. Tanto as tarefas como os utilizadores sofrem modificações. Os padrões de trabalho alteram-se quer devido ao produto em si, quer devido a novos desenvolvimentos em termos de *hardware* e *software*. Por outro lado, os utilizadores ganham novas capacidades e expectativas.

Destes factores resulta a necessidade de se fazerem novas versões da interface gráfica de modo a corresponder aos novos padrões de trabalho e novas expectativas dos utilizadores.

4.2 Linhas mestras para GUIs

Linhas mestras são regras mais ou menos genéricas que devem ser aplicadas à concepção de interfaces gráficas com o utilizador. Contudo, é de salientar que não se consegue obter uma GUI pela simples aplicação de linhas mestras. Estas não são uma receita mágica que se aplica *et voilà*, o resultado é uma interface gráfica bem concebida. Devem antes ser consideradas linhas de referência a seguir para aferir da validade de um determinado *design*, e corrigi-lo se for caso disso.

Actualmente existem milhares de linhas mestras disponíveis [Smith e Mosier, 1984], que tentam cobrir todas as eventualidades. A utilização de determinadas linhas mestras fornece a um dado *design* a coerência que os investigadores daquelas tentaram arduamente conseguir. O objectivo é alargar a concepção de GUIs a pessoas mais inexperientes nesta área, de modo que estas possam realizar um melhor trabalho.

Das inúmeras linhas mestras para GUIs existentes nos dias de hoje, podem salientar-se dois trabalhos que foram e são preponderantes na concepção de interfaces gráficas com o utilizador. O trabalho pioneiro a nível de interfaces gráficas com o utilizador desenvolvido no Xerox PARC nos anos setenta (projecto Xerox Star), influenciou o estilo e forma das GUIs actuais e continua a ser usado como referência neste campo.

Outro trabalho citado em praticamente todos os livros sobre a concepção de GUIs são as linhas mestras presentes no IBM CUA (Common User Access) [IBM, 1989], que fornece recomendações para a sintaxe das aplicações gráficas que correm na gama de computadores da IBM, e constituiu um documento de referência no qual se basearam diversas linhas mestras para diferentes sistemas existentes nos dias de hoje.

É de notar que, desde que se siga uma estratégia para a interface de modo a que as aplicações sejam consistentes, não é importante que sintaxe se utiliza, se se garantir que esta é coerente com o sistema de suporte da aplicação. A sintaxe, consoante o sistema onde a aplicação deve correr, pode ser a referida no IBM CUA, a implementação da Microsoft destas recomendações, as directivas da Apple para as aplicações dos computadores Macintosh, ou da OSF (Open Software Foundation) para os sistemas UNIX (OSF/Motif), só para citar algumas linhas mestras existentes a este nível.

Atendendo a que a sintaxe pode variar de sistema para sistema, não parece relevante entrar em detalhes a este nível, sendo mais importante fornecer regras gerais que possam ser aplicadas independentemente da norma utilizada. Assim, de seguida, descrevem-se princípios gerais e enunciam-se regras provenientes do trabalho realizado no Xerox PARC a observar na concepção de interfaces, e aplicáveis a qualquer GUI independentemente do sistema onde esta se enquadra. Explora-se ainda a utilização de cor e som nas interfaces gráficas com o utilizador.

4.2.1 Regras da Xerox Star

O trabalho desenvolvido no Xerox PARC, há cerca de vinte anos, foi e continua a ser determinante para a aparência e comportamento das interfaces gráficas dos nossos dias.

As regras para a concepção de GUIs aí encontradas são simples e fáceis de aplicar, produzindo a sua observância interfaces gráficas com o utilizador fáceis de usar.

Muitas destas ideias aparecem nos livros actuais de linhas mestras para GUIs e podem ser encontradas na secção seguinte. No entanto, a importância prática e histórica destas linhas mestras, justifica a sua inclusão aqui. A Figura 4.1 mostra uma lista de princípios a seguir aquando da concepção de uma interface gráfica com o utilizador. Os conceitos apresentados à esquerda são simples e os da direita complexos.

Simples	Complexo
Concreto	Abstracto
Visível	Invisível
Copiar	Criar
Escolher	Preencher
Reconhecer	Gerar
Editar	Programar
Interactivo	Batch

Figura 4.1 Operações simples e complexas.

Concreto vs Abstracto

A sugestão aqui presente é privilegiar as interfaces concretas face às interfaces abstractas. As coisas concretas são mais fáceis de representar visualmente do que ideias abstractas. Pode ter-se uma ideia deste facto se se pensar em possíveis ícones para representar uma mesa, calor ou fiabilidade. Quanto mais concreta é uma coisa, mais fácil se torna representá-la, e portanto, mais fácil é para o utilizador manipulá-la visualmente.

Visível vs Invisível

Uma boa maneira de fazer com que uma interface gráfica com o utilizador seja difícil de compreender e usar é esconder informação do utilizador. Por exemplo, afixando mensagens numa parte do ecrã que não está visível ao utilizador. Esconder informação e forçar o utilizador a lembrar-se é meio caminho andado para uma péssima GUI.

Copiar vs Criar

Esta é uma das ideias mais poderosas na concepção de GUIs. Não se devem gerar novas soluções; tomam-se soluções já usadas e modificam-se. O utilizador não deve ter que inventar a partir do nada; deve sempre ser-lhe fornecido algo que possa modificar. Por exemplo, em vez de se criar um novo documento, utiliza-se um antigo e altera-se.

Escolher vs Preencher

Esta regra é uma variante da regra de copiar sempre que possível. Deve dar-se ao utilizador uma lista de itens passíveis de escolha, como alternativa a obrigar a que aquele se lembre do que precisa escrever. A regra deve ser: se um dado existe no sistema, então o utilizador não necessita de o introduzir nunca mais; deve-lhe ser facultada a possibilidade de escolher esse dado, sem no entanto lhe ser negada a hipótese de o escrever.

Em alguns casos é óbvio que esta regra é impraticável e constitui uma maneira perfeitamente insatisfatória de introduzir dados. Imagine-se o que seria produzir este trabalho escolhendo todas as palavras (ou até letras) de uma lista, em vez de as escrever!

Reconhecer vs Gerar e **Editar vs Programar** são outras variantes de uma ideia similar. Como foi dito no ponto anterior, deve partir-se sempre de alguma solução e tentar chegar a uma nova solução, modificando a antiga.

Interactivo vs *Batch*

Este conceito era muito mais relevante nos anos setenta, quando foi desenvolvido. Desde então as pessoas aceitaram que os sistemas interactivos são, regra geral, melhores do que ir guardando dados para depois os processar em conjunto (*batch*).

Por vezes os dados são preparados como resultado de qualquer operação e, quando conveniente, processados de uma vez só. No entanto, para a maior parte das aplicações, é, com frequência, melhor usar o computador como uma ferramenta interactiva que ajuda à entrada de dados, do que recolher todos os dados para depois realizar uma única operação.

4.2.2 Princípios gerais

É impossível estabelecer linhas mestras que consigam abarcar todas as eventualidades que podem ocorrer na concepção de interfaces gráficas com o utilizador. No entanto, podem enunciar-se alguns princípios genéricos que devem ser seguidos [Cox e Walker, 1993] como linhas de referência para a concepção de GUIs. Alguns destes princípios já foram referidos nas características que uma GUI deve possuir.

Deve ter-se presente que os princípios aqui apresentados são sugestões, não regras absolutas e por si só não constituem qualquer garantia de se conseguir uma boa GUI; é ainda necessário usar intuição e imaginação na concepção da interface gráfica e testar a interface com os utilizadores.

Conceber GUIs para se adaptarem ao modelo conceptual do utilizador

A ideia subjacente a este princípio é começar a construir a interface de fora para dentro; i.e., começar pelas tarefas que os utilizadores pretendem realizar e conceber objectos e manipulações para modelizar essas tarefas, e depois testar a interface com os utilizadores de modo a ver se é compreensível para estes. O reverso deste princípio é começar pelo *hardware*, depois construir *software*, e no fim acrescentar a interface gráfica, como quem barra manteiga sobre o pão...

Esta última metodologia costuma ter maus resultados. Acresce a isto que a interface gráfica deve ser entendida como uma parte fundamental da aplicação, e não como um acrescento de última hora.

Consistência

A consistência em interfaces gráficas com o utilizador parece uma virtude evidente *per se*. Numa primeira análise a consistência aparece como facilitando a passagem dos

utilizadores de umas aplicações e tarefas, para outras; significa que não se deve reinventar a roda, e facilita a aprendizagem dos utilizadores de uma aplicação nova ou da realização de novas tarefas. No caso geral, tudo isto é verdadeiro. No entanto, é necessário ter cuidado no modo como se é consistente. Se se decide que se deve ter sempre ecrãs verdes com texto castanho, então o facto de se ser consistente é mau, dado que é difícil ler texto castanho em fundo verde. É necessário assegurar que as regras aplicadas são eficazes.

O conceito de consistência pode, de facto, ser sujeito a interpretações erradas. A lição seguinte é ilustrativa.

> Professor, "... e todo o sistema deve ser mnemónico ..."
> Interrupção, "Quando se escreve 'A' o meu aborta"
> Professor, "... e consistente ..."
> Interrupção "Na verdade, quando se escreve qualquer coisa o meu sistema aborta."

Por vezes existem duas regras de consistência em conflito. Uma primeira regra poderia ser que as mesmas acções efectuam a mesma coisa. Podia dizer-se, para uma aplicação particular, que uma pressão dupla com o rato apagava o objecto no qual está posicionado. Contudo, uma outra regra para uma pressão dupla com o rato poderia ser: "executar a acção mais provável que o utilizador deseje realizar neste objecto, neste instante e neste contexto particular". A ideia é consistente, mas a acção em particular pode variar muito. De facto, a segunda regra aparece como capaz de produzir melhores diálogos.

Deve tratar-se a consistência, como todas as regras apresentadas nesta secção: com cuidado. Como princípio geral é melhor ser consistente, mas não ser pedante. Não obstante, podem existir boas razões para não se ser consistente.

Comandos Universais

Existem diversos comandos que podem ser encontrados na maior parte das aplicações e cujo significado é constante, independentemente do tipo de objectos a serem manipulados. Estes comandos, definidos inicialmente pelo projecto da Xerox, constituem um ponto de partida útil para os comandos genéricos necessários na maior parte dos sistemas (Figura 4.2).



Figura 4.2 Comandos universais.

Controlo pelo utilizador

Se uma GUI é prescritiva, com pouco controlo por parte do utilizador então, a menos que seja um produto muito simples será provavelmente uma má interface. Os sistemas mal concebidos requerem que o utilizador reaja ao computador, em vez de deixaram o utilizador pedir ao computador para realizar as tarefas que deseja. Se os utilizadores são instados a obter informação de apenas uma maneira, e são coagidos a reagir sempre do mesmo modo, é natural que a interface não seja muito bem sucedida.

Os utilizadores possuem diferentes modos de trabalhar e vão querer fazer as coisas à sua maneira. Meter os utilizadores numa camisa-de-forças de respostas, causa frustração e aborrecimento. Por exemplo, um programa que requeira sempre confirmação das acções, sem facultar ao utilizador a possibilidade de configurar esta faceta como opção, tornar-se-á de certeza fastidioso.

Interacção não modal

Um *modo* limita os comandos que o utilizador pode submeter, ignorando ou interpretando como erros outros comandos alternativos. Os diálogos não modais tendem a dar mais controlo ao utilizador. Existem, no entanto, situações em que não é possível ou desejável evitar que o sistema esteja num modo definido.

Quando se utilizam diálogos modais deve tornar-se perceptível ao utilizador que o sistema está num modo diferente modificando o ecrã ou restringindo as acções que o utilizador pode realizar. Por exemplo, se o utilizador deve tomar conhecimento de uma mensagem e responder antes de poder continuar, deve apresentar-se um diálogo modal que requer que o utilizador responda antes de lhe permitir efectuar outra operação.

Configuração

Independentemente do muito que se teste uma interface e da quantidade de informação obtida sobre os putativos utilizadores da GUI, os utilizadores efectivos irão com frequência desejar coisas distintas numa aplicação. Algumas pessoas gostam de ícones, outros preferem palavras. Umas gostam de cores fortes, enquanto outras preferem cores pálidas, quase monocromáticas.

Na verdade, não existe maneira de conciliar estas diferenças. Ou se força as pessoas a adoptar uma dada abordagem ou se rodeia o problema fazendo interfaces configuráveis pelo utilizador. Esta última hipótese é sem dúvida mais agradável.

Atalhos para utilizadores experientes

Quando se constrói uma interface leva-se em linha de conta quer novos utilizadores, quer utilizadores experientes. Infelizmente, utilizadores diferentes sentem-se mais à-vontade com diálogos diferentes. Um utilizador experiente não necessita de tantos passos para realizar uma tarefa como um utilizador principiante; um utilizador experiente pode tomar atalhos e saber o que está a acontecer. Pode resolver-se este problema facultando aos utilizadores várias maneiras de realizar a mesma acção. Assim, os diálogos funcionarão bem quer para os utilizadores experientes, quer para os mais inexperientes.

Um exemplo simples disto é a utilização de *hot-keys* nos menus de abertura vertical. Não se espera que os utilizadores principiantes comecem desde logo a utilizar estes atalhos, mas os utilizadores experientes rapidamente o farão. É, no entanto, necessário assegurar que os atalhos não interferem com os utilizadores principiantes e garantir que existe algum modo de os utilizadores descobrirem os atalhos.

Usar a linguagem do utilizador

Os utilizadores e as pessoas responsáveis pela concepção de interfaces gráficas possuem com frequência vocabulários diferentes, aliás de acordo com o meio ambiente onde se inserem. Sempre que possível deve então usar-se a linguagem dos utilizadores. Quando é imperativo usar um termo técnico, deve fornecer-se ao utilizador uma maneira rápida de descobrir o seu significado.

Motivação

Aplicações mal concebidas destroem a motivação do utilizador, e é necessário ter presente que, com frequência, os utilizadores têm hipótese de decidir se usam determinada aplicação ou não, dado que possuem maneiras alternativas de realizar as suas tarefas.

É com bastante facilidade que se consegue destruir a confiança do utilizador numa aplicação. Basta, por exemplo, "deixar" o utilizador trabalhar durante várias horas numa aplicação e depois perder todo este trabalho. Mesmo com mensagens de erro espirituosas, de certeza que o utilizador não ficará muito satisfeito, e evitará utilizar a mesma aplicação de novo. Numa palavra: os utilizadores têm que ter confiança nas aplicações que usam e esta confiança só se consegue com boas experiências.

Resposta

A resposta é um dos princípios fundamentais de qualquer sistema interactivo. Quando o utilizador realiza uma dada acção deve poder ver o resultado dessa acção. Numa interface gráfica se o utilizador selecciona um ícone deve ser fornecida qualquer indicação de que o ícone está seleccionado, mudando a sua cor ou forma por exemplo.

A ausência de resposta provoca ansiedade nos utilizadores, dado que existe incerteza sobre o resultado de determinada acção. O caso mais comum de ausência de resposta é a demora em mostrar o resultado de uma dada operação. Muitas vezes o utilizador realiza qualquer acção e decorrem alguns segundos até alguma coisa acontecer no ecrã. Por causa desta demora, o utilizador frequentemente assume que a acção não foi realizada e tenta repeti-la vezes sem conta, muitas vezes com consequências nefastas.

O ideal, mesmo para acções que, por exemplo devido a causas inerentes ao sistema, demorem algum tempo a realizar, é dar ao utilizador uma indicação de que o sistema tomou conhecimento da acção e está a processá-la. Isto pode ser feito mudando o ícone do rato (e.g., para uma ampulheta) ou através de uma mensagem para o utilizador, só a título de exemplos comummente utilizados.

Ver e apontar vs lembrar e escrever

Numa interface gráfica com o utilizador as aplicações devem ser visíveis. Tudo o que o utilizador necessita para perceber uma aplicação deve ser-lhe mostrado. Quando o utilizador quer fazer alguma coisa, olha e escolhe em vez de ter que se lembrar e depois introduzir um comando.

WYSIWYG

A técnica WYSIWYG faz mais do que simplesmente tornar os sistemas mais fáceis de usar. O que o utilizador vê no ecrã, torna-se para ele a nova realidade. As pessoas entendem uma aplicação pelo modo como a vêem e pelo modo como os objectos se modificam no ecrã. O aspecto dos objectos no ecrã torna-se a percepção do utilizador do que é que o objecto de facto é; o modo como o objecto se modifica no ecrã torna-se a percepção do utilizador das acções do sistema.

Simplicidade

A simplicidade é uma faceta importante das GUIs e deve ser uma preocupação constante quando se constrói uma interface gráfica. No entanto, é difícil definir o que é simples ou não. De facto, o que é simples para quem construiu a interface pode ser extremamente complexo para o utilizador. Sempre que possível deve "vestir-se a pele" do utilizador, para tentar antecipar as dificuldades que este possa sentir.

Contudo, a busca de simplicidade por vezes leva à implementação de soluções triviais, no pior sentido do termo; i.e., a aplicação pode tornar-se desinteressante ou com poucas potencialidades. Este factor deve ser seriamente considerado aquando da concepção da interface, e deve tentar-se alcançar simplicidade sem cair na trivialidade.

Fecho

Fecho num sistema interactivo significa que em dadas alturas do diálogo (interacção) o utilizador pode "fechar" o seu raciocínio, esquecendo os passos anteriores. Isto acarreta a implementação de acções independentes e compartimentadas. Uma sequência interactiva é dividida em fases limitadas e existem maneiras de mostrar ao utilizador que uma dada sequência terminou. Deste modo, o utilizador necessita de se lembrar apenas do estado final e pode esquecer os passos que realizou para o atingir.

Um exemplo interessante e demonstrativo de fecho é fornecido pelos menus de abertura vertical. Um menu deste tipo pode ser aberto (por exemplo, com o rato) na barra superior do menu. O sistema fornece indicação ao utilizador que uma acção foi iniciada ou abandonada fechando o menu, o que permite ao utilizador concentrar-se na fase seguinte do diálogo.

Reversibilidade

Deve ser permitido ao utilizador mudar de opinião e voltar atrás com facilidade. Se o utilizador realiza uma dada acção e depois decide que afinal não era isso que pretendia, deve poder voltar atrás. Operações reversíveis tornam a interface gráfica mais fácil de usar e aprender.

Se o utilizador souber que pode voltar atrás nas operações, tende a experimentar muito mais, uma vez que não será "punido" por isso. E a experimentação é um dado adquirido que constitui um modo rápido de aprendizagem.

Erros devem ser tratados com simplicidade

Quando um erro ocorre significa que o utilizador realizou qualquer operação que o programa não estava à espera. Isto pode acontecer porque o utilizador não percebeu o que era requerido pelo sistema ou porque o sistema foi usado de uma maneira inesperada, sendo necessário fornecer ao utilizador um modo simples de recuperar e mostrar-lhe o que era requerido.

As mensagens de erro devem ajudar o utilizador a perceber o que aconteceu e a correcção do erro deve ser uma operação mais simples do que a operação que o provocou.

Prevenir erros do utilizador

A prevenção dos erros numa aplicação significa antecipar determinados tipos de acções que o utilizador pode realizar e fornecer-lhe um meio de realizar aquelas sem que cometa erros. Isto por vezes implica facultar alternativas comuns à execução de certas tarefas.

Um exemplo da prevenção de erros nas aplicações gráficas correntes é a funcionalidade de *grid snap* nos programas de desenho. Quando esta funcionalidade está activa as linhas desenhadas devem começar e acabar nas intersecções da grelha de desenho. Atendendo a que quando se desenham várias linhas, estas geralmente estão ligadas entre si, esta funcionalidade permite evitar muitos erros de desenho. No entanto, deve ser opção do utilizador usar esta funcionalidade ou não, dado que existirão casos em que de facto se pretenda desenhar linhas que não estejam conectadas.

Fornecer indicações e mensagens claras

Uma das características mais frustrantes em muitos sistemas é não saber o que fazer a seguir. É particularmente desencorajante não saber como se sai de uma aplicação ou função. Deve existir sempre um modo de abandonar e o utilizador deve conhecê-lo. Se uma interface gráfica começa a executar uma operação bastante longa, o utilizador deve ser posto ao corrente do tempo que tal operação vai demorar a ser completada, e ter possibilidade de a interromper se assim o desejar.

Quando se fornecem mensagens estas devem ser claras e concisas e, se aplicável, deve ser indicado ao utilizador o que fazer a seguir. Uma mensagem do tipo "CARÁCTER ILEGAL INTRODUZIDO" é bastante críptica. De facto, o carácter não é ilegal, é apenas imprevisto. Será mais compreensível, e mesmo agradável, indicar ao utilizador que carácter ou tipo de carácter foi introduzido e sugerir o que deve fazer. Por exemplo, utilizando uma mensagem com o seguinte conteúdo "ERA ESPERADO UM NÚMERO; FOI INTRODUZIDO UM 'A'". Pode sempre contestar-se que a última mensagem é demasiado longa e desnecessária, mas isto pode ser torneado permitindo ao utilizador escolher o estilo - curto ou longo - das mensagens.

Facilidade de aprendizagem

Um produto fácil de aprender é também um produto fácil de usar, como demonstrado em [Whiteside *et al*, 1985]. Daqui decorre que um modo de construir uma GUI fácil de usar é construir uma GUI cuja aprendizagem seja fácil.

Com isto em mente, deve garantir-se que o utilizador pode inferir facilmente como usar a aplicação para executar as tarefas pretendidas, definir tarefas cuja execução seja facilmente compartimentada, deixar o utilizador construir o seu próprio conhecimento da aplicação sem ter que se preocupar em conhecer todas as suas potencialidades, e tornar simples a exploração de novas áreas.

Reduzir a carga na memória de curta duração

Uma vez seleccionada, a informação é processada e copiada para uma memória de curta duração - STM (Short Term Memory). Esta memória é onde se realiza o raciocínio. A STM tem uma capacidade de cerca de sete itens, e informação não usada na STM decai ao fim de cerca de 20 segundos; no entanto, se se utilizar a mente tem como consequência a perda de informação residente na STM muito mais rapidamente [Thimbleby, 1990].

Daqui se infere que boas interfaces gráficas colocam poucas exigências à STM. Neste sentido, a interface deve exigir que o utilizador se lembre de uma reduzida quantidade de factos, números ou palavras. As pessoas podem com relativa facilidade recordar-se de estratégias e técnicas para encontrar itens, mas têm dificuldades em lembrar-se de itens particulares.

Por exemplo, um processador de texto que exija ao utilizador que se lembre do nome dos ficheiros e os escreva, será difícil de usar. Um modo de ultrapassar esta dificuldade é, quando a interface arranca, mostrar os últimos ficheiros usados, permitir a selecção dentro deste conjunto, e permitir também a visualização e selecção de outros ficheiros. Isto reduz a carga na memória do utilizador e assemelha-se ao modo como as pessoas frequentemente trabalham.

Outra maneira de uma GUI reduzir a carga na memória humana é deixar o utilizador modificar o ambiente de trabalho, de modo a poder utilizar um ambiente

adequado por defeito. Deve assim fornecer-se ao utilizador um ambiente judicioso por defeito, e permitir que esse ambiente seja alterado com facilidade.

Bibliografia sobre linhas mestras

Existem inúmeros livros com linhas mestras para a concepção de interfaces gráficas com o utilizador oferecendo conselhos razoáveis a seguir na construção de GUIs. Existem ainda livros com linhas mestras para GUIs construídas em ambientes integrados de desenvolvimento, que acompanham o próprio *software* de desenvolvimento. Estes livros constituem uma excelente fonte de aconselhamento e sempre que possível as linhas mestras aí enunciadas devem ser seguidas. Isto tende a fazer com que todas as GUIs desenvolvidas para um determinado ambiente possuam o mesmo aspecto e comportamento.

4.2.3 Utilização de cor em GUIs

A utilização de cor numa interface gráfica com o utilizador pode melhorá-la mas, para a maior parte dos sistemas, o impacto efectivo e mensurável no desempenho dos utilizadores é diminuto. O seu valor principal reside em tornar a apresentação mais interessante para os utilizadores. Na verdade, o efeito é similar ao que acontece na televisão. O conteúdo informativo da maior parte dos programas televisivos é o mesmo a cores ou a preto e branco; no entanto, a maioria das pessoas prefere ver a cores...

De qualquer modo, existem ocasiões em que o uso de cores pode ser bastante útil. A cor ajuda a agrupar itens semelhantes, a distinguir grupos, chama a atenção para coisas importantes e pode ser usada como um código visual. Por outro lado, a cor é um estímulo visual forte, o que significa que combinações infelizes de cores serão de certeza notadas. Deve então adoptar-se uma atitude "conservadora" no uso de cores.

É de fazer notar que a utilização de cores não deve ser decisiva para o sucesso de uma interface gráfica com o utilizador. As cores devem ser utilizadas como uma ajuda para providenciar uma interface gráfica mais fácil de usar e mais atractiva. Contudo a interface deve funcionar independentemente das cores até porque, atendendo a que muitas pessoas só têm monitores monocromáticos, alarga-se o campo de aplicação da interface.

Embora seja impossível cobrir aqui todos os aspectos subjacentes à utilização de cores em GUIs, podem enunciar-se algumas regras gerais:

- A janela activa deve ter uma cor diferente de uma janela inactiva.
- Nas mensagens de aviso deve ser usada a cor vermelha.
- Quando o estado de um item muda, a sua cor também deve mudar.
- Não se deve ultrapassar quatro cores por ecrã.
- As acções devem ser indicadas com cores mais pálidas e o brilho deve ser usado como ênfase.
- Os fundos a preto, azul ou branco são melhores.
- Evitar combinações incompatíveis de cores. Exemplos disto são os pares azul/amarelo, vermelho/verde, verde/azul e vermelho/azul.
- Deve ser usado grande contraste de cor para os pares caracteres/fundo. Exemplos são branco no preto, branco em azul, ou preto no branco.
- O azul claro deve ser confinado às áreas de fundo do ecrã.
- A cor vermelha e a cor verde não devem ser usadas fora da zona central de atenção do utilizador, uma vez que são difíceis de ver na periferia do campo visual.
- As cores devem ser usadas de acordo com as expectativas do utilizador: azul para frio, vermelho para quente e perigo, verde para avançar e amarelo para esperar.

4.2.4 Utilização de som em GUIs

O som é um meio de comunicação tão importante para o ser humano que, embora ainda não seja muito utilizado na maior parte dos sistemas computacionais, não será de todo estranho presumir que cada vez se tornará mais usado como um meio de as pessoas interagirem com os computadores. Referem-se a seguir alguns usos possíveis do som em GUIs como um modo de obter resultados diversos.

Som como confirmação

Quase todos os computadores fazem ruídos e as pessoas utilizam esses ruídos subconscientemente como uma ajuda para lidar com o computador. Os teclados, por exemplo, são construídos para fazerem um barulho de '*click*' de cada vez que uma tecla é pressionada. As pessoas não gostam de teclados silenciosos porque o barulho das teclas ajuda-as a escrever. Não é que este factor seja tão importante como o movimento das teclas, mas os utilizadores não se sentem à-vontade com um teclado silencioso. O som das teclas actua como uma confirmação de que um carácter foi introduzido.

Os computadores fazem outros sons que actuam como uma confirmação de acções ou que indicam que o computador está a funcionar normalmente. Depois de trabalhar com um dado computador por algum tempo as pessoas conseguem identificar os sons que o computador produz e ter uma noção do que este está a fazer. Por exemplo, o barulho do movimento dos *drives* do disco fornece um padrão associado a executar determinados programas.

As características de diferentes sons podem ser usadas para confirmar acções em interfaces gráficas com o utilizador. A maior parte das GUIs possuem botões nos quais o utilizador pode carregar. Depois de o utilizador carregar no botão, a sua representação visual altera-se. Mas em vez de se alterar a aparência do botão, pode fazer-se com que o computador emita um som para indicar que o botão foi pressionado. Devem ser usados sons diferentes para botões com significados diferentes (Figura 4.3).

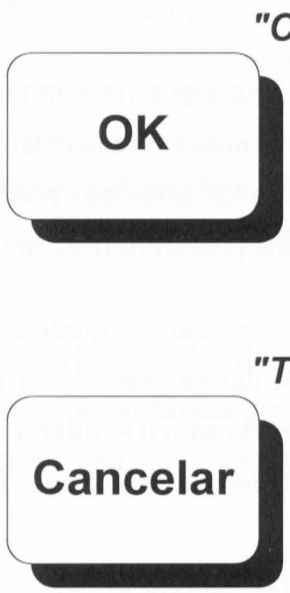


Figura 4.3 Botões sonoros.

Som como informação de estado

O som nas interfaces gráficas com o utilizador pode ser usado como uma indicação do estado de diferentes operações. Este tipo de indicação é usado pelas pessoas diariamente. Um exemplo disto é quando se enche um copo com água. Mesmo que não se olhe, à medida que o copo vai enchendo, o som modifica-se e fornece indicação de quanta água o copo tem. Aqui, o som representa o estado de enchimento do copo.

Nas GUIs é pois perfeitamente exequível utilizar o som como um indicador do estado de uma tarefa. Por exemplo, a maior parte dos sistemas correntes realizam tarefas em segundo plano como imprimir. Neste caso poder-se-ia usar um som de baixo volume (para não perturbar o utilizador) de uma impressora, que se alterava à medida que a impressão chegava ao fim. Deste modo, o utilizador tinha uma indicação auditiva do estado da tarefa. Este som também actua como uma confirmação da realização da tarefa mas, para além disto, a sua variação indica o estado daquela. Este tipo de sons depressa se "perdem" no ruído ambiente, e as pessoas só notam quando eles se alteram ou desaparecem por completo.

Som como ajuda espacial

Esta faceta é já bastante utilizada nos jogos de computador, onde o som indica a localização do jogador. Por exemplo, numa caverna existe um eco, à beira rio ouve-se o barulho da água a correr, etc. O mesmo princípio pode ser utilizado nas GUIs para indicar o local nas aplicações. Os sons podem indicar um local ou proximidade.

Na verdade os sons em si podem não ter qualquer significado particular (não-icónicos), mas a sua associação com determinado local na aplicação fornece-lhes um significado específico. Esta utilização do som é similar a usar o som como indicador de mudanças de estado; neste caso o estado que se altera é a localização.

Outros usos correntes

Uma utilização possível e bastante frequente do som nas interfaces gráficas com o utilizador já referida (Secção 3.3.2), é como tentativa de obter a atenção do utilizador para a ocorrência de uma anomalia.

Relacionada com esta utilização, está o uso do som como um indicador que o utilizador cometeu um erro. Esta faceta é frequentemente explorada em muitos sistemas computacionais actuais. Virtualmente qualquer utilizador já passou pela experiência de tentar invocar alguma acção e o computador "queixar-se" imediatamente através de um sinal sonoro, muitas vezes acompanhado por uma mensagem informando que foi cometido um erro (ou, na era do politicamente correcto: o utilizador realizou uma acção inesperada...).

4.3 Linhas mestras na concepção de GUIs em ambientes de fabrico

Algumas das linhas mestras referidas na secção anterior já foram abordadas na Secção 2.2 e a sua aplicação no caso das interfaces gráficas com o utilizador em ambientes de fabrico foi analisada na Secção 3.1.4. No entanto, e sem prejuízo do que já foi referido anteriormente, de seguida tecem-se comentários sobre a aplicação de algumas linhas mestras apresentadas e que possuem mais relevância na concepção de GUIs para ambientes de fabrico.

- *Escolher vs Preencher*: Se o operador necessitar de introduzir dados ou enviar comandos deve, sempre que possível, ser-lhe fornecida uma lista de escolhas possíveis.
- *GUI adaptada ao modelo conceptual do operador*: A GUI deve reflectir a especificidade das tarefas e dos putativos utilizadores de um ambiente de fabrico particular.
- *Consistência*: A previsibilidade do comportamento de uma interface consistente é de suma importância para utilizadores pouco familiarizados com aplicações para sistemas computacionais, como é frequente em ambientes de fabrico.
- *Linguagem do utilizador*: No mundo industrial a disparidade de vocabulário, entre os *designers* e os utilizadores, pode atingir níveis muito elevados.
- *Motivação*: No caso frequente de operadores já de si pouco motivados a usar novas tecnologias, se se destruir a confiança para usar a interface gráfica, a motivação desaparecerá de todo.
- *Resposta*: A impaciência de quem quer desligar ou comandar algo é muito maior do que a de quem quer simplesmente abrir um ficheiro. As consequências de acções repetidas num dado comando podem revestir-se de maior gravidade.
- *Simplicidade*: Dada a diferença óbvia de conhecimentos no âmbito das interfaces gráficas, é notoriamente importante tentar antecipar as dificuldades do utilizador e buscar a simplicidade.
- *Prevenção de erros*: As consequências de um erro do operador podem ser bastante mais nefastas do que simplesmente apagar um documento necessário. Todos os hipotéticos erros devem ser previstos e sumariamente evitados.
- *Mensagens*: Deve garantir-se que o operador não fica bloqueado sem saber o que fazer a seguir; devem ser-lhe fornecidas mensagens claras do caminho a seguir na realização de uma dada tarefa; se um erro foi cometido deve

indicar-se claramente o que o sistema esperava e fornecer-lhe ajuda para a correcção do erro.

- *Facilidade de aprendizagem*: Esta característica é tanto mais importante quanto menor é a familiaridade com interfaces gráficas; note-se que frequentemente é o primeiro contacto do operador com uma interface gráfica para sistemas computacionais.
- *Som como confirmação*: Realimentação através de informação auditiva das acções tomadas é aconselhável para utilizadores com menos à-vontade no uso de interfaces gráficas.

Consideraram-se aqui as linhas mestras, anteriormente apresentadas, que possuem maior importância nas interfaces gráficas com o utilizador em ambientes de fabrico. A exclusão desta análise de outras linhas mestras referidas justifica-se, no caso de GUIs para ambientes de fabrico, pela sua pouca relevância - caso da configuração; pelo facto de a sua aplicação poder ser contraproducente - caso dos comandos universais que podem não ter significado para o operador; ou porque muitas vezes não podem ser aplicadas de todo - caso da interacção não modal que, com frequência, não é exequível nomeadamente para a prevenção de erros, ou da reversibilidade dado que, muitas vezes, não é possível voltar atrás nos comando emitidos em processos de fabrico.

4.4 Conclusões

Embora importante, a enunciação das características que a aplicação final de uma boa interface gráfica com o utilizador deve possuir, não esgota o assunto da sua efectiva obtenção. Antes do produto final, muita água corre sob a ponte...

Em particular, é necessário usar certas metodologias na concepção de interfaces gráficas com o utilizador. Foram aqui apresentados os passos que tipicamente caracterizam o processo de desenvolvimento de GUIs. A utilização de linhas mestras na concepção de GUIs traduz-se em diversas vantagens; nomeadamente, permite fazer uso de soluções já testadas por diversos peritos na área, evitando erros já cometidos no passado, e manter a consistência entre aplicações e sistemas.

Referiram-se diversas linhas mestras gerais que devem ser observadas na concepção de interfaces gráficas com o utilizador. Adicionalmente, discutiu-se da relevância da aplicação de linhas mestras gerais para o caso particular das interfaces gráficas com o utilizador em ambientes de fabrico.

Capítulo 5

Desenvolvimento de GUIs

O desenvolvimento de interfaces gráficas com o utilizador é complexo. Existem diversas razões importantes para este facto e que justificam de algum modo que o *software* das GUIs seja considerado como estando por inerência entre os tipos de *software* mais difíceis de criar [Myers, 1993]. Por exemplo, se se fizer uma lista das propriedades gerais que tornam qualquer sistema difícil de ser implementado, os requisitos de multiprocessamento, robustez e de funcionamento em tempo real, estarão no topo da lista. E todos estes requisitos estão presentes no *software* das interfaces gráficas com o utilizador.

Existem, no entanto, diversas técnicas que podem ser utilizadas no desenvolvimento de GUIs, e que embora não retirem a complexidade inerente ao desenvolvimento deste tipo de *software*, permitem atenuá-la.

5.1 Arquitectura de uma GUI

Muitas aplicações fornecem múltiplas representações gráficas da mesma informação. Por exemplo, uma aplicação de estatística pode apresentar os mesmos dados sob a forma de gráficos de Gantt ou *pie charts*. Uma folha de cálculo pode apresentar simultaneamente inúmeras células cujos conteúdos estão relacionados por uma fórmula. Um processador de texto pode apresentar um documento quer sob a forma de páginas formatadas, quer em *outline view*.

Em cada um destes casos é necessário assegurar que todas as representações permanecem sincronizadas e coerentes em todos os instantes de tempo. Se o utilizador cria um novo subtítulo no documento em *outline view* do processador de texto, a nova entrada deve também aparecer nas outras apresentações do documento. O utilizador deve poder interagir com qualquer representação do documento, e todas as outras representações devem imediatamente reflectir qualquer mudança que tenha ocorrido.

Existem vários modelos que tentam descrever as fronteiras entre as partes puramente funcionais do programa e a sua apresentação externa através da interface com o utilizador. Uma das abordagens mais conhecidas para representar a separação entre funcionalidade e interface é o modelo MVC (Model-View-Controller), usado no sistema Smalltalk desenvolvido no Xerox PARC.

Os conceitos fundamentais deste modelo, bem como o papel que representa na arquitectura de uma interface gráfica com o utilizador, são descritos na secção seguinte. Um exemplo da sua aplicabilidade às GUIs para ambientes de fabrico também é fornecido.

5.1.1 Modelo MVC

A arquitectura MVC fornece um meio poderoso de organizar sistemas que suportam múltiplas apresentações da mesma informação. Esta técnica é particularmente útil nos sistemas interactivos, como sejam as interfaces gráficas com o utilizador. O modelo MVC é baseado em três tipos de objectos: Modelo, Visualização e Controlador.

O objecto Modelo não possui características visuais, representando simplesmente dados "em bruto". Por exemplo, considere-se uma aplicação para ambientes de fabrico que permita ver o estado actual das várias ordens de fabrico. Neste sistema, o objecto Modelo pode manter informação sobre as ordens de fabrico actualmente a serem processadas, em lista de espera e talvez o tempo médio de execução de uma ordem de fabrico. O Modelo pode também suportar várias operações que alteram o estado dos dados do Modelo. A título de exemplo, o objecto Modelo pode suportar os métodos `interromper()` e `cancelar()`, bem como métodos que permitam ver as ordens de fabrico correntemente atribuídas a uma dada máquina e fornecer uma lista das ordens de fabrico mais recentemente processadas.

Os objectos do tipo Visualização apresentam o estado actual do Modelo ao utilizador. As Visualizações devem ser capazes de recolher dados de um Modelo, mas não se devem preocupar com a implementação interna do objecto Modelo. As Visualizações respondem a uma mensagem de `update()`, que tem por efeito que cada objecto do tipo Visualização faça a apresentação no ecrã do estado actual do Modelo, de acordo com as suas capacidades. Cada tipo de Visualização apresenta algum aspecto dos dados mantidos pelo Modelo. Por exemplo, o sistema que foi referido atrás pode providenciar uma Visualização que forneça sempre o estado actual de uma dada ordem de fabrico. Outra Visualização pode fornecer uma lista das ordens de fabrico interrompidas, enquanto outra ainda pode apresentar as ordens de fabrico recentemente terminadas. Algumas Visualizações podem ser baseadas em texto, enquanto outras podem representar a informação graficamente.

O objecto Modelo é responsável pela notificação de todos os objectos Visualização associados quando algum aspecto dos dados do Modelo se modifica. As Visualizações apenas formatam e fazem a apresentação dos valores actuais mantidos pelo Modelo. Por exemplo, o utilizador pode escolher ver uma lista das ordens de fabrico em fila de espera, ordenada por data de entrega ou pelo tempo de processamento necessário para o seu término; o utilizador pode escolher filtrar os dados para ver apenas ordens de fabrico a serem entregues entre certas datas; as ordens de fabrico em processamento podem ser apresentadas por máquina ou pelo numero de operações que ainda é necessário realizar. Nenhuma destas modificações altera o modelo subjacente. São

meramente opções de apresentação que determinam como cada Visualização apresenta no ecrã os dados mantidos pelo Modelo.

O terceiro tipo de objecto da abordagem MVC é o Controlador. Os objectos do tipo Controlador permitem ao utilizador manipular os dados no Modelo ou alterar o modo como uma Visualização apresenta os dados. Por exemplo, na aplicação mencionada anteriormente pode existir um objecto do tipo Controlador que permita ao utilizador introduzir os dados de uma nova ordem de fabrico, interromper ou cancelar uma ordem de fabrico, mudar a atribuição de uma determinada ordem de fabrico a uma dada máquina, entre outras operações possíveis.

Visualizações e Controladores funcionam com frequência de um modo fortemente ligado. Em alguns casos, um único objecto pode funcionar como Controlador e Visualização. Por exemplo, na GUI citada atrás o utilizador poderia atribuir uma ordem de fabrico a outra máquina, introduzindo os dados de identificação da nova máquina na mesma janela (Visualização) que faz a apresentação da atribuição corrente.

A Figura 5.1 mostra as relações entre os diversos objectos baseados na arquitectura MVC. O Controlador manipula o Modelo e as várias Visualizações através do envio de mensagens apropriadas, representadas na figura por mensagens do tipo `change()`. O objecto Modelo reage aos pedidos do Controlador e depois notifica todos os objectos do tipo Visualização dependentes que o Modelo se alterou. Cada um dos objectos do tipo Visualização reage às mensagens de `update()` através da obtenção da informação que necessita do objecto Modelo e fazendo a actualização da apresentação da informação.

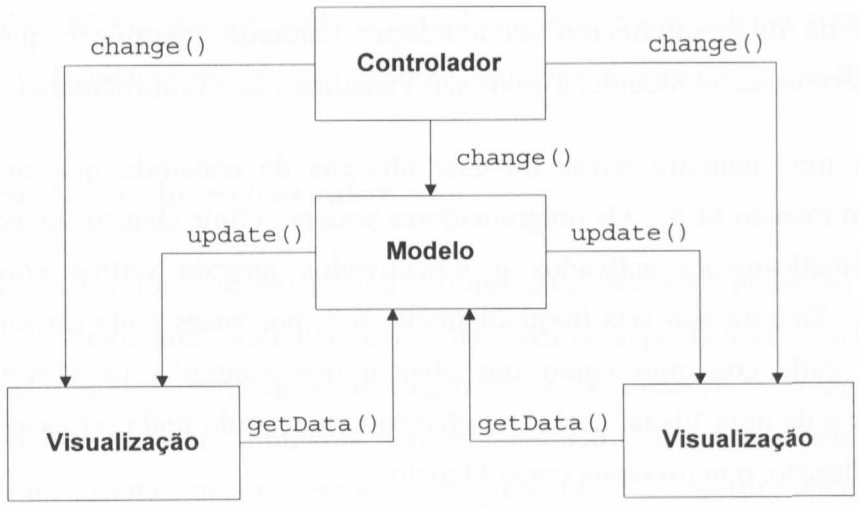


Figura 5.1 Relações entre os objectos do modelo MVC.

Em teoria, é possível criar conjuntos reutilizáveis de Visualizações e Controladores que funcionem como componentes da interface gráfica com o utilizador, e que podem ser conectados a modelos diferentes para implementar rapidamente aplicações complexas.

Na prática, Modelos, Visualizações e Controladores tendem a estar fortemente associados. Embora estes três tipos de componentes devam ser concebidos de modo a serem tão independentes quanto possível, os objectos do tipo Visualização têm que saber como obter os dados de um Modelo, e os Controladores necessitam saber o protocolo através do qual um tipo particular de Modelos e Visualizações podem ser manipulados.

A abordagem MVC tem muitos usos potenciais em sistemas interactivos. Usar uma arquitectura básica como o MVC pode ajudar aquando do desenvolvimento de sistemas orientados por objectos, pelo fornecimento de um modelo que já se provou ser eficaz em situações similares. Em vez de se partir do zero a tentar identificar os objectos, é muitas vezes mais fácil procurar objectos específicos que correspondam a uma arquitectura do tipo MVC.

Nem todas as GUIs se adequarão ao modelo MVC. No entanto, se o modelo MVC parecer a abordagem correcta para a arquitectura da interface gráfica, pode começar-se

o processo de desenvolvimento da aplicação tentando identificar que partes do programa pertencem ao Modelo, e quais são Visualizações e Controladores.

Existe uma maneira eficaz de usar objectos de comando que se baseia nos conceitos do modelo MVC. Os programadores podem definir objectos de comando que sejam automaticamente activados e desactivados quando outros comandos são executados. Embora isto seja frequentemente útil, por vezes pode ser mais eficiente pensar em cada comando como um objecto que contenha os atributos de um Controlador e de uma Visualização. O objecto de comando pode ser associado a uma parte da aplicação, que funciona como Modelo.

Na aplicação que tem vindo a servir de exemplo, considere-se um objecto de comando do tipo MudarAtribOFMaqCmd, que possibilita atribuir uma ordem de fabrico (OF), já atribuída a uma determinada máquina para ser realizado um dado conjunto de operações, a uma outra máquina. A necessidade de um tal comando justifica-se, por exemplo, pela avaria da máquina à qual inicialmente a ordem de fabrico foi atribuída.

Este objecto de comando deve estar activo quando a OF está atribuída a uma máquina, existe outra máquina disponível, e esta máquina possui a capacidade de realizar o conjunto de operações necessárias. O comando deve estar inactivo quando a OF não esta atribuída a nenhuma máquina, quando não existe qualquer máquina disponível, ou quando as máquinas disponíveis não possuem a capacidade de realizar o conjunto de operações requerido.

Um mecanismo para lidar com todas estas situações pode ser facilmente implementado baseado nos conceitos do modelo MVC. Deste modo, representando a OF como um objecto do tipo Modelo, possuindo uma propriedade do tipo *atribuível* como parte do seu estado, pode tratar-se o objecto MudarAtribOFMaqCmd como uma Visualização daquele modelo. O comando deve receber mensagens de `update()` sempre que o estado do modelo da OF se altera. Este tipo de mecanismo é tanto mais útil quando nas interfaces gráficas com o utilizador para ambientes de fabrico, é absolutamente necessário assegurar que comandos errados não são emitidos pelo utilizador.

O modelo MVC pode ser usado como a base de todo um sistema, e também pode ser usado para implementar subsistemas mais pequenos dentro de uma aplicação.

5.2 Tecnologia cliente/servidor

O termo cliente/servidor tornou-se nos últimos anos bastante poderoso em termos mediáticos. Virtualmente, qualquer revista ou livro da especialidade faz referência a esta tecnologia; muitos vendedores de *hardware* e de *software* reclamam o uso da tecnologia cliente/servidor; inúmeras conferências são dedicadas a este tema e os integradores de sistemas mostram-se ansiosos para ajudar a construir sistemas do tipo cliente/servidor.

Existe até uma anedota, reveladora da dimensão do interesse que esta tecnologia suscita, onde se compara a tecnologia cliente/servidor ao sexo na adolescência: toda a gente fala sobre isso, todos acham que os outros já o fizeram, quem de facto o fez foram muito poucos, e destes, a maior parte fê-lo mal...

Nesta secção analisam-se os conceitos subjacentes à tecnologia cliente/servidor, bem como a importância da sua utilização nas interfaces gráficas com o utilizador.

5.2.1 Teoria

A chave para se perceber a tecnologia cliente/servidor é compreender que é um conceito lógico. As partes do cliente e do servidor podem ou não existir em máquinas fisicamente distintas. Mais precisamente, a tecnologia cliente/servidor é um paradigma, ou modelo, para a interacção entre processos de *software* executados concorrentemente.

Os processos cliente enviam pedidos para um processo servidor, que responde com os resultados para esses pedidos. Como o nome indica, os processos servidores fornecem serviços aos seus clientes, usualmente através de processamento específico que apenas eles podem efectuar. O processo cliente, liberto da complexidade e dos encargos implícitos ao processamento da transacção, está apto a realizar outro trabalho útil. A interacção entre os processos cliente e servidor é uma troca transaccional e de cooperação, na qual o cliente é proactivo e o servidor é reactivo [Renaud, 1993].

Num ambiente verdadeiramente cliente/servidor, ambos os processos cliente e servidor são indiferentes ao facto de serem executados na mesma máquina ou em máquinas diferentes. Alguns protocolos de rede ou de comunicação entre processos - IPC (Interprocess Communication) podem não suportar o cliente e o servidor no mesmo sistema (e.g., IBM LAN Server). No entanto, isto é uma restrição de um protocolo particular - não uma característica do paradigma cliente/servidor. De um ponto de vista teórico, o tamanho da máquina também não é relevante. Não existe qualquer razão para que o processo servidor (e.g., servidor X) não possa correr num PC, enquanto o processo cliente corre numa máquina maior (e.g., estação de trabalho UNIX). A Figura 5.2 ilustra vários tipos de sistemas cliente/servidor.

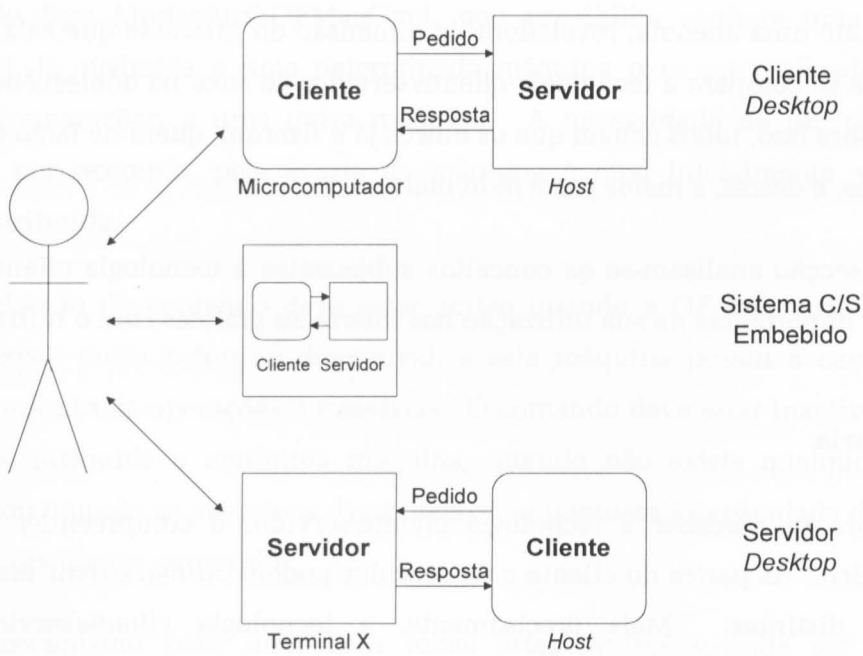


Figura 5.2 Diferentes tipos de sistemas cliente/servidor.

5.2.2 Prática

Os sistemas cliente/servidor são notoriamente mais interessantes quando os processos cliente e servidor são executados em máquinas separadas, conectadas via rede. Embora a rede possa ser local ou uma rede que abranja uma área de maior dimensão, as redes

locais - LANs (Local Area Networks) são o tipo mais comum de implementações cliente/servidor.

A Figura 5.3 ilustra o modelo que descreve os sistemas cliente/servidor. Na prática o termo *cliente* refere-se ao sistema onde o processo cliente é executado (representado na figura por um rectângulo arredondado). O termo *servidor* refere-se ao sistema no qual corre o processo servidor, representado por um rectângulo na figura. Note-se que isto não implica que outros clientes, ou outros servidores, não possam também correr nestes sistemas. Isto continua a ser uma visão lógica do sistema cliente/servidor. Deve ser lembrado que pode referir-se um "servidor de base de dados" ou um "servidor de impressora", sem necessariamente implicar máquinas distintas (embora este seja frequentemente o caso). É importante manter a distinção entre as visões lógicas e físicas de um dado sistema. Como a tecnologia cliente/servidor é um conceito lógico, manter uma perspectiva lógica facilita com frequência a sua compreensão.

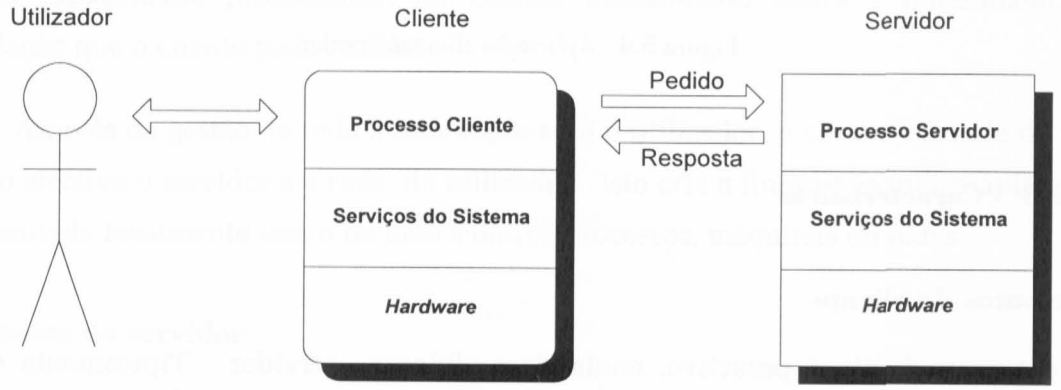


Figura 5.3 Sistemas cliente/servidor.

Existem três camadas básicas que fazem parte de ambos os sistemas cliente e servidor, como a Figura 5.3 ilustra. Estas camadas são relativamente arbitrárias, mas são úteis para se perceber os componentes dos sistemas cliente/servidor. A camada mais baixa e facilmente reconhecível, engloba os elementos de *hardware* do sistema. A camada seguinte, mais acima, é a camada de serviços do sistema, que compreende o sistema operativo, a rede, e os componentes do ambiente de janelas (se este existir). A

camada dos serviços do sistema inclui todo o *software* usado pela aplicação para controlar o *hardware*. A camada de topo é a camada de aplicação, consistindo nos processos cliente e servidor e outros processos da aplicação. Note-se que num sistema cliente/servidor a própria aplicação engloba ambos os processos cliente e servidor, como mostra a Figura 5.4.

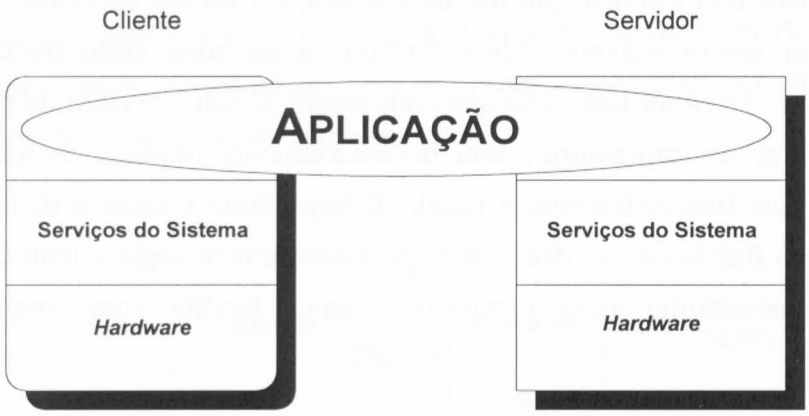


Figura 5.4 Aplicação cliente/servidor.

5.2.3 Características

Atributos do cliente

O processo cliente é proactivo, enviando pedidos ao servidor. Tipicamente é um processo dedicado à sessão com o utilizador, e começa e acaba com esta sessão. Um cliente pode interagir com um servidor único ou com múltiplos servidores, com a finalidade de realizar o seu trabalho. Contudo, pelo menos um processo servidor é sempre necessário.

Ao nível da aplicação, o cliente é responsável por manter e processar toda a interacção com o utilizador. Isto tipicamente inclui a realização das seguintes tarefas:

- Gestão do ecrã.
- Interpretação dos menus ou comandos.

- ▷ Entrada e validação de dados.
- ▷ Recuperação de erros.

No caso das GUIs, as tarefas do cliente incluem também:

- ▷ Gestão das janelas.
- ▷ Entradas do teclado e do rato (ou, embora menos comuns, *trackball*, ecrã táctil, etc.).
- ▷ Controlo das janelas de diálogo.
- ▷ Gestão de som e vídeo (em aplicações multimédia).

Na prática, o cliente não pode efectuar sempre toda a validação dos dados. De um modo geral, alguma da validação lógica da aplicação requer uma verificação cruzada com dados já processados. Não obstante, o cliente pode usualmente efectuar todas as actividades de preparação dos dados e a edição de dados de tabelas (como por exemplo, a validação dos códigos usados). Muitas vezes é possível guardar alguns dos dados previamente processados, no cliente, aumentando assim a quantidade de validação que o cliente pode realizar.

Através da gestão de toda a interacção com o utilizador, o cliente esconde de um modo efectivo o servidor e a rede, do utilizador. Isto cria a ilusão que toda a aplicação é executada localmente sem o recurso a outros processos, máquinas ou redes.

Atributos do servidor

O processo servidor é reactivo, sendo despoletado pela chegada de pedidos dos seus clientes. O processo servidor geralmente corre indefinidamente, fornecendo serviços a vários clientes. Estes serviços podem ser fornecidos directamente pelo próprio processo servidor ou indirectamente por processos escravos (*slaves*) distribuídos por cada pedido do cliente. A Figura 5.5 ilustra o uso de processos servidor do tipo *master/slave* para providenciar um servidor lógico para o cliente.

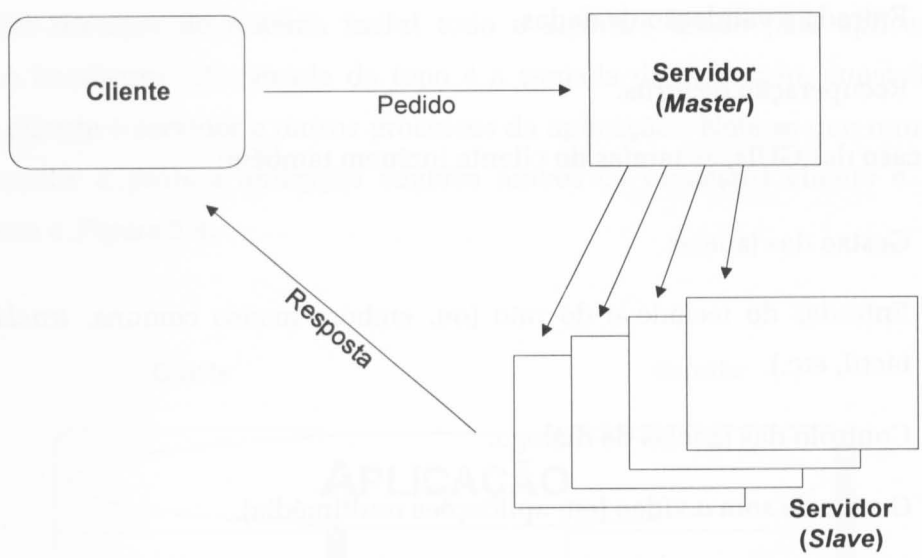


Figura 5.5 Processos servidor do tipo *master/slave*.

Um servidor é específico a nível funcional: realiza um conjunto de transacções predefinidas e funcionalmente relacionadas. Com frequência existem vários servidores para fornecer uma série de funcionalidades ao conjunto de clientes. Potencialmente, múltiplos clientes podem aceder a um servidor ao mesmo tempo. O servidor deve solucionar os aspectos de exclusão mútua (por exemplo, bloqueio), necessários à prevenção de uma eventual corrupção dos resultados.

Um servidor efectua toda a lógica transaccional necessária ao processamento de uma dada transacção e não interage com outros servidores. Se um cliente utiliza múltiplos servidores, é da responsabilidade do cliente invocar os servidores à medida que são requeridos. O processamento do servidor inclui tipicamente todo o processamento associado com o acesso, armazenamento e organização de dados partilhados; actualização de dados previamente guardados; e toda a gestão de outros recursos partilhados. Os recursos partilhados podem ser dados, capacidade da CPU (Central Processing Unit), armazenamento em disco, capacidades de impressão, comunicações, entre outros.

Atributos da comunicação

Como já foi referido, o estilo da comunicação entre o cliente e o servidor é transaccional e cooperante. Uma característica da comunicação transaccional é que o servidor envia apenas os resultados relevantes para o pedido do cliente. No caso ideal, a quantidade de dados enviada é a mínima necessária para que o cliente possa realizar o seu trabalho. Isto significa que, tipicamente, seja colocado pouco *overhead* na rede pela aplicação (para além dos *overheads* devidos aos protocolos).

Este facto não implica que as aplicações cliente/servidor não tenham impacto a nível da carga da rede - certamente que têm. Contudo, o impacto adicional é o mínimo de qualquer arquitectura de aplicações de rede. Outros paradigmas de processamento distribuído podem igualar o nível de carga da rede das implementações cliente/servidor, mas raramente resultam numa carga menor.

A natureza cooperante do paradigma cliente/servidor significa que uma quantidade significativa de processamento ocorre quer no lado do cliente, quer no lado do servidor. No caso geral, seria de esperar a utilização de mecanismos especiais para sincronizar este processamento concorrente. No entanto, a natureza da passagem de mensagens na comunicação cliente/servidor elimina a necessidade de sincronização explícita.

Usualmente, a comunicação cliente/servidor é implementada recorrendo a RPCs (Remote Procedure Calls). Numa invocação de um RPC, o cliente, após o envio do pedido ao servidor, pára o processamento e espera uma resposta do servidor. O servidor começa o processamento apenas aquando da chegada do pedido do cliente, e pára depois de ter concluído esse pedido (até que chegue um novo pedido). Quando o cliente recebe a resposta, retoma o processamento. Logo, o sincronismo entre o cliente e o servidor é implícito ao mecanismo de passagem das mensagens.

Existem, no entanto, alternativas ao uso de RPCs. Se o sistema operativo do cliente possui capacidades de multitarefa ou multiprocessamento, o sistema cliente pode executar outro trabalho enquanto o servidor está a processar o seu pedido. Este trabalho não está geralmente relacionado com a função que o servidor está a executar. Nos casos em que o sistema cliente está a correr múltiplas cópias do processo cliente, cada processo cliente aparece como independente na perspectiva do servidor.

Alguns tipos de comunicação entre processos (IPC) providenciam mecanismos para permitir ao cliente continuar o seu processamento após o envio da mensagem de pedido ao servidor. Este tipo de cliente é não-bloqueador, e deve "lembrar-se" de verificar o resultado mais tarde ou utilizar um mecanismo de IPC que o interrompa aquando da chegada do resultado. Na maior parte destes casos, o sincronismo entre cliente e servidor continua implícito ao mecanismo de passagem das mensagens. Uma excepção é o caso em que o processo cliente necessita impedir interrupções enquanto se encontra e executar segmentos críticos de código (como acontece frequentemente nos sistemas em tempo real).

É de ter em conta que pedidos de múltiplos clientes podem chegar ao servidor simultaneamente. Para além disto, um pedido pode chegar enquanto o servidor ainda está ocupado a processar um pedido anterior de um cliente diferente. É da responsabilidade do servidor fornecer meios de pôr os pedidos em lista de espera ou de os executar de uma maneira concorrente. A execução concorrente dos pedidos pode ser feita recorrendo a processos filhos que tomam conta de cada um dos pedidos. De qualquer forma, o servidor deve sempre reter a informação necessária que lhe permita enviar os resultados certos aos clientes correctos!

5.2.4 A tecnologia cliente/servidor na concepção de GUIs

A definição da interface gráfica com o utilizador como um processo cliente, responsável pela interacção com o utilizador, e utilizando a capacidade de processamento de um (ou mais) servidor para realizar tarefas específicas, possui vantagens inerentes, para além dos benefícios tributáveis a qualquer tipo de aplicação.

Por um lado, mais capacidade de processamento pode ser dedicada à interface gráfica, o que permite construir interfaces mais avançadas em termos da interacção com o utilizador. Na verdade, as interfaces gráficas, pela sua natureza, consomem com voracidade os recursos do sistema. Se se puder libertar capacidade de processamento e dedicá-la à interacção com o utilizador, tanto melhor.

Por outro lado, as interfaces gráficas com o utilizador nos sistemas computacionais acompanharam uma mudança no paradigma da interacção com o utilizador: a tendência evoluiu no sentido de cada vez mais o controlo passar a estar

no lado do utilizador (Secção 2.2.5). É, então, natural esperar que a parte da aplicação que interage com o utilizador esteja mais próxima deste. A tecnologia cliente/servidor adapta-se perfeitamente a esta tendência: a interacção é da responsabilidade do cliente sem envolver o servidor; qualquer adaptação da interface pelos utilizadores individuais é isolada dos outros clientes.

Relacionado com este ponto, está a necessidade frequente de permitir o acesso a um dado sistema computacional, aplicação ou recurso a diversos utilizadores através de uma interface gráfica. Este acesso torna-se facilmente exequível utilizando a tecnologia cliente/servidor.

O caso dos ambientes de fabrico é exemplar daquela necessidade, dado que frequentemente existe interesse em executar múltiplos clientes gráficos de um único processo servidor. Por exemplo, numa célula de fabrico de grandes dimensões, torna-se interessante a interface gráfica do controlador de célula poder correr em vários terminais, permitindo diversos pontos de acesso ao sistema de controlo. Cada uma das aplicações é um cliente do subsistema de controlo, que funciona como servidor e deve, no caso ideal, suportar concorrência de processos para atender aos pedidos solicitados pelos vários clientes que utilizam os seus recursos [Oliveira e Carvalho, 1994].

5.3 Tecnologia orientada por objectos

O termo "tecnologia orientada por objectos" engloba um conjunto de métodos, processos e ferramentas usados para construir sistemas de *software* a partir de objectos. Objectos são cápsulas de comportamento (funções) e estados (dados), cujo funcionamento interno está escondido dos outros objectos que utilizam os seus serviços. Os objectos podem ser tão pequenos e simples como cadeias de caracteres e ícones e tão grandes e complexos como subsistemas e servidores.

A tecnologia orientada por objectos não é nova. As suas raízes remontam ao início dos anos setenta, quando descrições da linguagem Simula se tornaram acessíveis a um largo número de pessoas. Os benefícios de se construírem sistemas recorrendo à orientação por objectos - estrutura do sistema melhorada, manutenção e reutilização dos componentes - são reconhecidos e largamente aceites.

Nos anos oitenta, o projecto orientado por objectos evoluiu como uma formalização da programação orientada por objectos. Desde essa data, muitos métodos de desenvolvimento de *software* apareceram, cada qual reivindicando ser baseado na tecnologia de orientação por objectos. Estes métodos possuem diferentes abordagens, provêm de diversas origens e são apoiados por diversas comunidades de peritos.

O interesse em métodos orientados por objectos tem aumentado rapidamente, à medida que responsáveis por projectos e pelo desenvolvimento de *software*, procuram diminuir os custos de desenvolvimento e de manutenção do *software*.

Existem diversas abordagens bem conhecidas para o desenvolvimento de aplicações baseadas em estilos de programação mais tradicionais. O desenvolvimento de aplicações recorrendo à tecnologia da orientação por objectos é relativamente recente, e as técnicas para desenvolver sistemas orientados por objectos não estão tão bem definidas como as técnicas mais tradicionais. Ao longo dos últimos anos muitas estratégias de desenvolvimento recorrendo à orientação por objectos foram apresentadas, mas poucas são largamente aceites ou utilizadas.

Algumas destas estratégias propostas põem ênfase em técnicas mais teóricas para desenvolver sistemas orientados por objectos, enquanto outras se baseiam na facilidade de obtenção de protótipos em ambientes interpretativos. Apesar das diferenças que naturalmente existem entre as diversas abordagens, todas estas técnicas têm por objectivo comum permitir a quem desenvolve sistemas de *software* partir de um dado problema e desenvolver um conjunto de classes que podem ser utilizadas para implementar a solução.

A utilização da tecnologia da orientação por objectos, se bem que ainda não amplamente divulgada e em uso, embora esteja em rápida fase de crescimento, adequa-se naturalmente às necessidades de desenvolvimento de interfaces gráficas com o utilizador.

De facto, os objectos são um modo natural para representar os elementos de uma GUI e para efectuar a sua manipulação directa. Para se tirar o máximo partido da tecnologia de orientação por objectos, para além da análise e do projecto orientados por

objectos, também uma linguagem de programação orientada por objectos deve ser usada para manipular os objectos.

A descrição relativamente detalhada, nas secções seguintes, do processo de desenvolvimento de *software* recorrendo ao paradigma da orientação por objectos fundamenta-se primordialmente numa razão chave: os benefícios mais importantes da utilização da tecnologia de orientação por objectos possuem uma relevância enorme quando se trata do desenvolvimento de GUIs.

Na verdade, se se atentar nos méritos do desenvolvimento de *software* orientado por objectos: produção de *software* menos complexo, de fácil reutilização, extensão e manutenção, pode verificar-se que todas estas facetas constituem vantagens de importância extraordinária no desenvolvimento de GUIs.

A produção de *software* menos complexo não necessita explicações adicionais, especialmente se se considerar o facto que vários estudos demonstram que a parte da interface gráfica constitui, em muitos casos, mais de metade do código e do tempo de desenvolvimento da aplicação total [Myers, 1993]. As vantagens inerentes à facilidade de reutilização, extensão e manutenção do *software* também serão bastante evidentes, se relacionadas com as fases do processo de desenvolvimento de GUIs descritas na Secção 4.1: iteração, construção da aplicação final, manutenção e construção de novas versões da interface gráfica.

Adicionalmente, com muita frequência as GUIs possuem elementos comuns, quer ao nível de elementos gráficos da própria interface (ícones, menus, botões, entre outros), quer ao nível de funcionalidades (como, por exemplo, procura, substituição ou ordenação segundo um dado critério). Por este motivo, a vantagem supracitada da reutilização de *software*, vê a sua importância acrescida, no caso específico do desenvolvimento de interfaces gráficas com o utilizador.

5.3.1 Desenvolvimento de *software*

O processo de desenvolvimento de *software* orientado por objectos - OOSD (Object-Oriented Software Development) pode ser considerado como sendo composto pelos processos de Análise Orientada por Objectos - OOA (Object-Oriented Analysis),

Projecto Orientado por Objectos - OOD (Object-Oriented Design) e Programação Orientada por Objectos - OOP (Object-Oriented Programming). Note-se que aqui OOP é usado para designar a fase de implementação do desenvolvimento de *software*. Contudo, historicamente, o termo refere-se ao processo global de desenvolvimento.

O OOSD em última análise produz um modelo de um sistema. Os componentes de um modelo orientado por objectos são objectos que estão agrupados em classes para efeitos de especificação. Para além das dependências cliente/servidor familiares (um objecto utiliza os serviços fornecidos por outro(s) objecto(s)), existem relações de herança entre as classes que servem para expressar a generalização e especialização dos conceitos representados pelas classes.

Desde o início que o paradigma da orientação por objectos foi olhado como capaz de revolucionar o processo de desenvolvimento de *software*, fazer com que este processo se tornasse mais natural e que os seus resultados fossem mais facilmente compreendidos. Os conceitos de objectos e classes eram considerados mais "naturais" do que dados e processos. Na verdade, a própria linguagem Smalltalk foi originalmente criada como uma linguagem de programação para as crianças usarem.

Nos dias de hoje, o objectivo do OOSD mantém-se o mesmo, i.e., ser um processo natural e coerente, e produzir *software* de fácil compreensão. O reconhecido mérito do OOSD [Sharble e Cohen, 1993] é a produção de *software* que é menos complexo, sendo portanto de mais fácil manutenção e extensão, e mais facilmente reutilizável.

Um sistema é *passível de manutenção* se pode ser modificado para se adaptar a modificações externas, como sejam mudanças de *hardware* ou sistemas operativos, ou para corrigir deficiências e melhorar o desempenho. No sentido de um nível elevado de manutenção ser atingido, um sistema deve possuir ligações fracas entre as partes que o constituem e possuir interacções que se baseiem apenas nas interfaces externas das partes que o constituem.

Se um sistema pode ser modificado de modo a suportar mudanças no seu objectivo inicial, então diz-se que é *extensível*. No sentido de facilitar que este objectivo seja atingido deve ser possível adicionar novos componentes e capturar e

isolar mudanças nos outros componentes. Para além disto é importante que referências aos componentes possam ser rapidamente actualizadas através do sistema.

Um sistema é *reutilizável* se as suas partes ou o seu todo podem ser utilizados para construir um novo sistema. A reutilização é melhorada se cada componente for desenvolvido à volta de um único objectivo. Um sistema reutilizável deve definir padrões genéricos para a interacção dos seus componentes.

De modo a alcançar estas qualidades, o OOSD apoia-se em quatro estratégias para reduzir a complexidade. Estas estratégias são dissecadas nos parágrafos seguintes e são: aumentar a coesão, reduzir o acoplamento, aumentar o polimorfismo e eliminar a redundância.

Unindo as partes de um sistema relacionadas entre si num único componente aumenta a coesão para esse componente e para o sistema como um todo. Um componente com uma elevada coesão é de mais fácil compreensão e mais susceptível de ser reutilizado. Um sistema de componentes coesos é também de mais fácil compreensão e tem uma manutenção facilitada.

A interacção entre partes dissimilares de um sistema aumenta o acoplamento. O acoplamento liga os componentes entre si. Diminuir o acoplamento liberta os componentes, aumentando assim a exequibilidade de extensão do sistema. Diminuir o acoplamento também simplifica a manutenção isolando os efeitos de mudanças ao nível dos componentes.

Polimorfismo é definido como a capacidade de tomar diferentes formas. Por exemplo, um procedimento com parâmetros formais polimorfos, pode ter diferentes parâmetros reais de cada vez que é invocado. Para além disto, uma referência polimorfa pode apontar para diferentes tipos de componentes de cada vez. Aumentando o polimorfismo, aumenta-se a capacidade de extensão do sistema.

Quando a mesma actividade é descrita ou o mesmo conceito definido em vários sítios diferentes num dado sistema, torna-se difícil modificar a actividade ou o conceito. Reduzir a redundância num sistema torna a sua extensão e manutenção mais fácil, bem como a sua compreensão.

Estas estratégias estão unidas e direccionadas por dois princípios básicos do paradigma da orientação por objectos, os princípios de encapsulamento e classificação.

Como foi referido anteriormente, o OOSD tem as suas raízes na programação orientada por objectos e evoluiu de cima para baixo: da programação para o projecto e deste para a análise. Fundamental para a OOP era o princípio do encapsulamento como definido em [Rentsch, 1982]:

> "O primeiro princípio da programação orientada por objectos pode ser designado por encapsulamento de inteligência: ver os objectos do exterior para fornecer uma metáfora natural do comportamento intrínseco"

Como é ilustrado na Figura 5.6, um objecto encapsula inteligência, que em última análise é representada em *software* por dados e processos. Um modelo orientado por objectos é composto por objectos que são construídos de modo a que cada objecto é visto do exterior e cada objecto é compreendido em termos do seu comportamento intrínseco, que por sua vez é expressado em termos dos serviços que fornece.

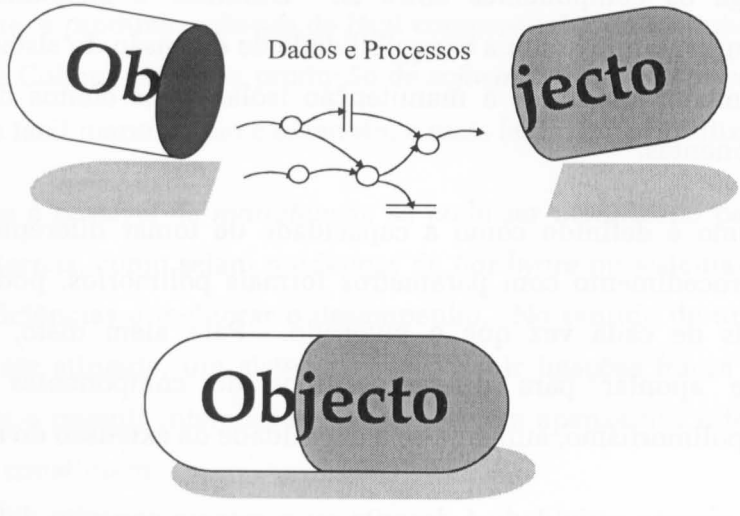


Figura 5.6 Um objecto encapsula inteligência.

O comportamento intrínseco de um objecto denota comportamento que provém da própria natureza do objecto, i.e., comportamento que é natural ao objecto. Como corolário, um objecto não deve ser chamado a realizar acções que não fazem parte da sua natureza.

O princípio do encapsulamento leva à criação de classes. Uma classe construída segundo este princípio representa um conceito único no domínio do problema e possui uma finalidade bem definida e coesa no domínio do modelo. Este princípio aumenta a coesão das classes e diminui o acoplamento entre elas.

Historicamente, o conceito de herança foi introduzido na programação com a linguagem Simula I onde o seu objectivo era modelizar o princípio da classificação, i.e., o arranjo ou distribuição das classes de acordo com a generalização e especialização do comportamento.

Este princípio rege a criação de estruturas de herança onde comportamentos que são partilhados por um grupo de classes são descritos apenas uma vez, numa classe que é um ascendente comum a cada uma das classes do grupo. A conformidade com este princípio diminui a redundância, ao mesmo tempo que evita a formação de relações indiscriminadas de herança entre as classes.

5.3.2 Análise

Análise e projecto são duas partes fundamentais do desenvolvimento de *software*. Posto de uma forma simples, a análise refere-se ao que o sistema é suposto fazer e o projecto refere-se a como é suposto fazê-lo.

O objectivo da análise é descrever um sistema existente no "mundo real", que será modelizado por *software*. A análise orientada por objectos abstrai-se dos conceitos do "mundo real" (domínio do problema) para formar objectos como partes do modelo orientado por objectos (domínio do modelo). As classes são formadas a partir dos objectos através da *generalização*:

> "Generalização é o acto ou resultado de remover certo tipo de distinções entre tipos de objectos de modo a que possamos ver as semelhanças." [Odell, 1991].

A análise pode envolver duas visões do sistema, uma visão interna e uma visão externa. A visão externa descreve os objectos em termos das suas actividades visíveis externamente (serviços), ao passo que a visão interna descreve as partes internas dos objectos que permitem essas actividades.

5.3.3 Projecto

O projecto orientado por objectos é fundamentalmente o processo de analisar o modelo produzido aquando da análise; optimizá-lo tendo em vista a redução do acoplamento, o aumento da coesão e o aumento do polimorfismo, e considerar o comportamento comum numa hierarquia de herança. Para além disto, o modelo é melhorado e formalizado.

As classes criadas aquando da análise passam pela fase de projecto sem grandes alterações essenciais, excepto modificações de modo a melhorar o desempenho ou compensar a falta de uma linguagem de suporte. Na fase de projecto existem ainda outros requisitos a ter em conta, como sejam: alocação de *hardware*, interfaces para dispositivos e recursos do sistema e controlo e apresentação no ecrã da interface com o utilizador.

Classes adicionais podem ser definidas de modo a prever aspectos que aparecem simplesmente como resultado da construção de um modelo de *software*, como inicializar o sistema, guardar o estado do sistema num ficheiro, utilizar bibliotecas externas, etc.

5.3.4 Programação

A programação orientada por objectos consiste na fase de implementação do desenvolvimento de *software*, e permite que os programas sejam mais fáceis de escrever, mais fáceis de manter e mais robustos.

Os objectos da OOP são blocos de código, não necessariamente objectos visualizados no ecrã, embora os objectos representados no ecrã sejam geralmente implementados com objectos do programa.

Os objectos do programa possuem algumas características importantes: são definidos hierarquicamente, e cada objecto é uma *instância* de uma classe de objectos similares; uma classe também é definida num único bloco de código. Por exemplo, os blocos de código que descrevem o menu *File* e o menu *Edit* (quase que omnipresentes em qualquer GUI corrente), seriam duas instâncias da classe dos menus.

Os objectos herdam o comportamento e as características definidas nos níveis anteriores (mais altos) da hierarquia, a menos que a herança seja explicitamente anulada pelo código do objecto, que tem precedência. A característica de herança permite ao programador mudar o tipo de letra de todos os objectos do tipo menu, fazendo apenas uma simples alteração na definição da classe.

Cada objecto também possui *dados privados* que definem as suas características particulares e mantêm informação acerca do seu estado actual. Os objectos comunicam entre si enviando *mensagens*. A resposta do objecto a uma mensagem faz parte do comportamento que o objecto herda, ou do comportamento particular do objecto que tem precedência sobre o comportamento herdado.

A programação orientada por objectos requer uma linguagem de programação orientada por objectos, como C++ ou Eiffel, que suporte as peculiaridades dos objectos, para além da funcionalidade básica oferecida pela maior parte das linguagens de programação modernas.

Existem diferenças entre as diversas linguagens de programação orientadas por objectos e nem todas suportam a totalidade dos aspectos descritos nos parágrafos anteriores. No entanto, todas estas linguagens fornecem uma estrutura de programação que se ajusta às mil maravilhas às necessidades da programação de GUIs.

A criação de um novo menu reduz-se à criação de mais uma instância da classe dos menus - uma linha de código! Depois resta preencher os detalhes únicos a esse menu: quais são os nomes dos itens do menu, onde é que ele vai ser apresentado no ecrã, e que mensagens cada item deve enviar quando seleccionado, são alguns exemplos.

Objectos adicionais de código implementam as funções do programa. Estes objectos de código, por vezes designados de *handlers*, são invocados por mensagens

enviadas pelos itens dos menus e outros tipos de objectos controlados pelo utilizador, bem como por outros objectos de código.

Se for necessário modificar o comportamento do programa, a hierarquia de classes permite que as alterações globais sejam efectuadas de um modo fácil e consistente, enquanto que os dados privados permitem que se modifique o comportamento de um objecto sem medo de provocar efeitos laterais inesperados.

5.4 Conclusões

A conveniência de se separar a representação gráfica da parte funcional da interface, a necessidade de modularidade e reutilização do *software* e o facto de as interfaces gráficas requererem bastantes recursos dos sistemas computacionais, entre outros factores, conduzem à necessidade de se utilizarem diversas técnicas no desenvolvimento de GUIs, no sentido de permitir satisfazer aqueles requisitos.

Assim, neste capítulo foram analisados o modelo MVC, a tecnologia cliente/servidor e o desenvolvimento de *software* orientado por objectos, cuja relevância no desenvolvimento de GUIs é expressa nos parágrafos seguintes, pela consideração das vantagens principais de cada uma destas técnicas.

A arquitectura MVC permite separar a funcionalidade da apresentação gráfica da GUI, o que se torna extremamente vantajoso especialmente nos casos em que existe necessidade de providenciar múltiplas representações gráficas da mesma informação. Adicionalmente, permite o desenvolvimento do código da interface gráfica de uma forma modular.

A definição da interface gráfica como um processo cliente de um ou vários processos servidores permite disponibilizar mais recursos para a construção de interfaces gráficas mais sofisticadas, bem como executar diversos clientes gráficos de uma dada aplicação.

A utilização da tecnologia orientada por objectos no desenvolvimento de interfaces gráficas com o utilizador é de suma importância por duas razões primordiais:

- ▷ Os objectos são um modo natural de representação dos elementos constituintes de uma interface gráfica.
- ▷ As vantagens do OOSD - facilidade de reutilização, extensão e manutenção do *software* - possuem extrema relevância no desenvolvimento de interfaces gráficas.

Capítulo 6

Ferramentas de suporte ao desenvolvimento de GUIs

A mesma interface gráfica que torna mais fácil o uso de uma dada aplicação ao utilizador, torna mais difícil o trabalho do programador. Claramente, a sofisticação das interfaces gráficas com o utilizador aumentou com o uso de estilos de manipulação directa e WYSIWYG, do rato, dos gestores de janelas, etc. Isto, por sua vez, tornou a tarefa de programação mais difícil. No entanto, as ferramentas de suporte ao desenvolvimento de interfaces gráficas também se tornaram mais sofisticadas e de maior utilidade, ajudando a diminuir a tarefa de programação.

As ferramentas de suporte ao desenvolvimento de *software* para GUIs possuem várias denominações e fornecem diversos níveis de funcionalidade. As mais simples, os *toolkits*, são bibliotecas de subrotinas que criam e gerem os variados componentes presentes no ecrã de uma interface gráfica, como janelas, menus e botões. Estas bibliotecas podem reduzir o tempo de programação e manter a consistência com

aplicações já existentes, mas ainda deixam a maior parte do trabalho à responsabilidade do programador, uma vez que apenas possuem uma interface programática.

Muito acima dos *toolkits* em termos de funcionalidade, estão os sistemas integrados de desenvolvimento de interface gráficas com o utilizador - UIMSs (User Interface Management Systems). Um UIMS fornece um *toolkit* e um ambiente de programação para a sua utilização no desenvolvimento de interfaces gráficas com o utilizador.

Nos melhores sistemas o ambiente de programação é maioritariamente visual; logo, em vez de se escreverem linhas de código a especificar, por exemplo, as coordenadas de cada botão numa janela de diálogo (e.g., 140, 30, 170, 80 para os cantos do botão), é possível seleccionar um botão da paleta de desenho e colocá-lo no ecrã no local desejado.

Alguns UIMSs são ambientes limitados que providenciam um desenvolvimento rápido de programas simples ou de protótipos iniciais. Outros podem requerer mais tempo e esforço de aprendizagem, mas permitem desenvolver desde um primeiro protótipo da GUI até à interface final com toda a funcionalidade incluída.

Um inquérito levado a cabo em 1991 [Myers e Rosson, 1992], mostrou que mais de 40 por cento dos programadores de GUIs utilizavam algum tipo de ferramenta de suporte ao desenvolvimento de interfaces gráficas, mais poderosa do que um *toolkit*. Os utilizadores de UIMSs gastavam aproximadamente 40 por cento do tempo de desenvolvimento da aplicação na parte da interface gráfica com o utilizador. Em comparação, os utilizadores de *toolkits* gastavam 60 por cento do tempo de desenvolvimento da aplicação na parte da interface gráfica. Em ambos os casos, a interface gráfica consistia em aproximadamente metade do código total da aplicação. Destes dados é possível aferir da validade da utilização de UIMSs no desenvolvimento de interfaces gráficas com o utilizador.

6.1 Programação visual

De um modo simples, a programação visual pode ser entendida como o desenvolvimento de aplicações através do desenho de imagens [Mikes, 1992]. A

abordagem visual é um objectivo antigo no desenvolvimento de *software*, e nos dias de hoje os sistemas integrados de desenvolvimento de GUIs constituem as ferramentas mais próximas desse objectivo (embora ainda não o atinjam na totalidade).

Na programação visual, os componentes gráficos da aplicação são desenhados no ecrã, de um modo interactivo. Uma vez a aplicação construída, o UIMS gera o código fonte para produzir a aplicação desenhada. Como estas ferramentas produzem código para construir os programas executáveis, o processo pode ser considerado uma forma de programação - daqui o termo *programação visual*.

Em teoria, não seria necessário possuir conhecimentos sobre uma norma particular de GUIs para se começar a desenvolver uma aplicação recorrendo a UIMSs. Na prática, no entanto, esse conhecimento é importante. Qualquer pessoa que desenvolva interfaces gráficas com o utilizador, independentemente da plataforma de janelas subjacente, deve possuir pelo menos um conhecimento fundamental sobre a tecnologia de janelas e o modelo particular de janelas utilizado (e as suas *nuances*), bem como um conhecimento completo da norma específica da GUI.

Sem este conhecimento, podem desenvolver-se interfaces gráficas com o utilizador que violem os livros de estilo[1] das respectivas plataformas das GUIs. De facto, como é que alguém pode desenvolver interfaces avançadas se não está familiarizado com a família de componentes gráficos (e a sua semântica) que uma dada plataforma de GUIs oferece?

Programadores experientes serão, de um modo geral, mais profícuos com ferramentas de programação visual, do que programadores sem experiência. No entanto, estes podem utilizar estas ferramentas como um meio bastante eficaz de adquirirem conhecimento sobre a plataforma de GUIs utilizada, constituindo também um meio útil para a construção rápida de protótipos.

[1] O livro de estilo (*style guide*) é o documento que especifica alguns elementos da interface com o utilizador e fornece linhas mestras na concepção de outros elementos, no sentido de tornar a aplicação consistente com outras aplicações que obedecem ao mesmo livro de estilo.

O maior benefício destas ferramentas é o facto de as aplicações poderem ser construídas muito mais rapidamente do que através de métodos convencionais de codificação "à mão".

6.2 Classificação das ferramentas

Não existe uma definição universalmente aceite para o que um UIMS de facto é, até porque a tecnologia é relativamente recente. Igualmente confusos são também os outros termos utilizados para denominar ferramentas similares de suporte ao desenvolvimento de interfaces gráficas: ambientes de desenvolvimento de interfaces com o utilizador - UIDEs (User Interface Development Environments), ferramentas de *design* interactivo - IDTs (Interactive Design Tools) e geradores de interfaces - IBs (Interface Builders), entre outros.

Algumas ferramentas permitem apenas a mera disposição no ecrã dos objectos da interface, fornecendo capacidades bastantes limitadas para a configuração da interface; outras ferramentas vão muito mais longe nas potencialidades oferecidas, incluindo até a capacidade de testar interactivamente a interface durante o período de desenvolvimento. Virtualmente todas as ferramentas geram algum tipo de código: algumas geram código fonte normalizado para uma plataforma particular da interface gráfica, outras geram código proprietário (mas portável), e outras ainda geram código fonte dentro de uma gama variada.

De facto, existe uma falta de normalização no que diz respeito às ferramentas de suporte ao desenvolvimento de interfaces gráficas com o utilizador. Por exemplo, não existem normas para a produção de código fonte e ficheiros da interface (usados para reconstruir a interface nas modificações subsequentes), e para a edição dos componentes da interface e das suas propriedades.

Basicamente, as ferramentas de suporte ao desenvolvimento de interfaces gráficas podem ser classificadas em três categorias. A primeira engloba os UIMSs (ou UIDEs), que facultam um ambiente integrado para realizar o desenho, protótipos, teste e a versão final de uma aplicação. Notoriamente, estas são as ferramentas mais interessantes (e dispendiosas)!

O aspecto mais significativo dos UIMSs é a capacidade interactiva de teste: a interface pode ser construída e testada até ao nível de detalhe desejado, antes de se fazerem compromissos com um determinado tipo de *design* e, na maior parte dos casos (obviamente dependendo do nível de detalhe atingido), sem um investimento substancial de tempo de desenvolvimento.

As ferramentas de *design* interactivo (IDTs) constituem a segunda categoria. Estas ferramentas fornecem apenas um subconjunto limitado das potencialidades oferecidas pelos UIMSs. Por exemplo, as IDTs usualmente não possuem capacidade para testar a funcionalidade da interface. A capacidade de teste que de facto possuem não permite testar o comportamento atribuído a um objecto da interface, sem primeiro abandonar a ferramenta e compilar um programa executável.

Outra limitação das IDTs é que, embora fornecendo a capacidade de construção de uma interface gráfica, oferecem pouca (ou até nenhuma) ajuda para a correcta combinação dos componentes da interface. Para além disto, não possuem geralmente qualquer capacidade de detecção e eliminação de erros, sendo portanto possível que programadores com pouca experiência tentem combinar componentes de uma maneira inapropriada, o que em última análise produz um erro de execução quando se corre a aplicação. Adicionalmente, muitas IDTs não permitem um acesso completo às propriedades dos componentes da interface através dos editores de propriedades. Por vezes, isto constitui uma tentativa de assegurar a adopção de um determinado livro de estilo da GUI.

A terceira categoria de ferramentas de concepção de GUIs pode ser designada de um modo genérico por ferramentas para a construção rápida de protótipos, uma vez que para pouco mais servirão. A mais valia deste tipo de ferramentas é fornecer uma ferramenta avançada para programadores pouco experientes efectuarem o desenho (e construírem protótipos) da interface gráfica. São ferramentas fáceis de usar, mas não possuem as funcionalidades requeridas para a construção da aplicação final da interface, nomeadamente em termos de comportamento da interface, para serem consideradas uma ajuda "séria" no desenvolvimento de interfaces gráficas com o utilizador.

O custo dos vários tipos de ferramentas constitui, na maior parte dos casos, um bom indicador da sua utilidade: as IDTs custam, na generalidade dos casos, na ordem de dez vezes mais que as ferramentas para a construção de protótipos; e o preço dos UIMSs é sensivelmente duas a cinco vezes superior ao preço das IDTs. Como em tudo na vida, o mais caro não é necessariamente o melhor... mas geralmente o preço tem a sua razão de ser.

6.3 Sistemas integrados de desenvolvimento

Os UIMSs permitem desenvolver interfaces gráficas com o utilizador muito mais rapidamente do que se a mesma aplicação tivesse que ser desenvolvida usando o método tradicional de escrever o código num editor de texto, e efectuar o processo iterativo de compilação, teste e eliminação dos erros detectados. Os UIMSs permitem o desenho interactivo dos componentes gráficos da interface no ecrã, e a configuração destes componentes recorrendo às facilidades oferecidas para a edição das suas propriedades.

O uso de programação visual nos UIMSs traduz-se na manipulação directa no ecrã dos objectos gráficos constituintes da interface. Os benefícios principais da programação visual são que se pode obter uma perspectiva WYSIWYG da interface à medida que esta vai sendo construída, e, como resultado, poupar uma substancial quantidade de tempo entre os intervalos codificação-execução da interface.

Este tipo de ferramentas possui um valor duplo: não só tornam a tarefa de desenvolvimento de GUIs mais fácil (e mesmo mais divertida), como também fornecem a capacidade inerente de rápida construção de protótipos. A necessidade de recorrer a protótipos numa fase intermédia do processo de desenvolvimento foi expressa na Secção 4.1.6.

6.3.1 Arquitectura de um UIMS

O diagrama seguinte representa a estrutura básica de um UIMS.

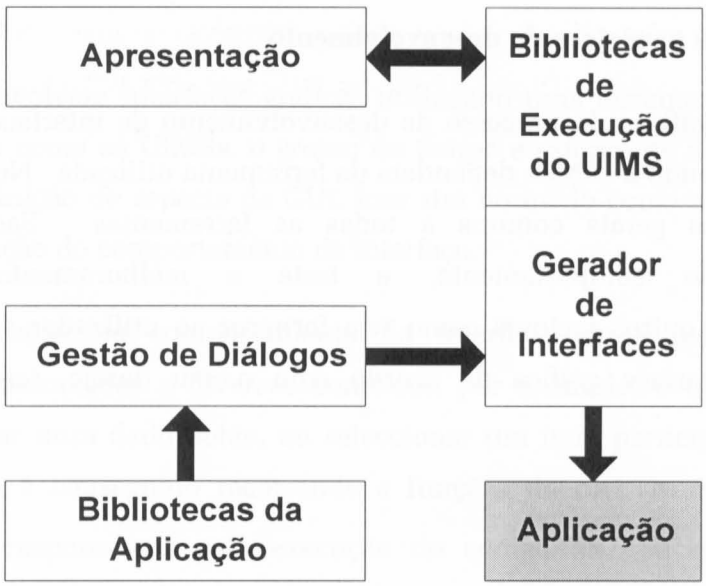


Figura 6.1 Estrutura de um UIMS.

Na generalidade dos casos, o modo como um UIMS funciona é o seguinte:

- Seleccionar e desenhar os componentes da interface.
- Configurar esses componentes através das facilidades oferecidas pelo UIMS para a edição das suas propriedades.
- Especificar o comportamento da interface, associando funcionalidade aos diversos componentes.
- Testar os componentes interactivamente durante o processo de desenvolvimento.
- Guardar o trabalho que foi realizado.
- Gerar o código fonte (e todos os ficheiros de suporte, *makefiles* incluídas) para a aplicação.
- Compilar o código fonte (no próprio UIMS ou num ambiente exterior) para produzir o executável da aplicação.

6.3.2 Técnicas genéricas de desenvolvimento

As técnicas específicas do processo de desenvolvimento de interfaces gráficas com o utilizador recorrendo a UIMSs dependem da ferramenta utilizada. No entanto, existem algumas técnicas gerais comuns a todas as ferramentas. São elas: desenho, especificação do comportamento, e teste e melhoramento da interface. Adicionalmente, outros factores como seja fornecer ao utilizador a possibilidade de configurar a interface gráfica de acordo com o seu desejo, também devem ser considerados.

Desenho

A fase de desenho consiste na criação dos componentes gráficos da interface com o utilizador e envolve factores relativos à representação externa da interface como: localização, tamanho e cor dos componentes, entre outros. O desenho da interface gráfica com o utilizador geralmente ocupa aproximadamente de 10 a 15 por cento do tempo total de desenvolvimento da GUI no UIMS [Mikes, 1992], e consiste em duas tarefas principais: criação dos elementos da interface regidos pelas especificações do livro de estilo da GUI, e criação dos elementos específicos da aplicação.

As directivas presentes no livro de estilo para a concepção dos elementos da interface gráfica devem ser obedecidas; só assim é possível construir uma interface consistente com outras aplicações da mesma plataforma de GUIs. A título de exemplo, o livro de estilo pode fornecer indicações sobre elementos dos menus (e.g., a ajuda deve ser colocada na parte situada mais à direita do menu), e definir onde colocar botões do tipo *OK* e *Cancelar*.

A criação dos elementos específicos da interface envolve factores como adicionar entradas aos menus, criar janelas e diálogos específicos da aplicação, e adicionar outros tipos de controlos[2]. Num UIMS o desenho dos componentes da interface é realizado de um modo WYSIWYG; este processo permite visualizar qual o aspecto da interface gráfica, quando a aplicação final estiver de facto a ser executada.

[2] Controlos, nas interfaces gráficas, servem para designar todos os elementos gráficos que controlam a interacção com o utilizador como por exemplo: menus, botões, janelas de diálogo, etc.

Comportamento

Quando se desenvolvem interfaces gráficas utilizando uma ferramenta de suporte ao desenvolvimento como os UIMSs, o grosso do tempo e esforço de desenvolvimento é gasto não na definição do aspecto da GUI, mas sim no modo como esta vai funcionar; i.e., na especificação do comportamento da interface.

A especificação do comportamento da interface passa pela definição das funcionalidades associadas a diversos controlos; por exemplo, o que acontece se o utilizador carregar num dado botão, ou seleccionar um item particular de um menu. Tipicamente isto é conseguido recorrendo a funções de `callback` definidas pelo programador, e responsáveis pela execução do código da aplicação necessário à realização da tarefa apropriada.

Na generalidade dos casos, o comportamento é adicionado depois do desenho estar concluído, embora por vezes algum comportamento simples seja definido em conjunto com o desenho da interface.

Teste e melhoramento

Uma vez definido o aspecto global da interface gráfica com o utilizador e especificado o seu comportamento, a GUI deve ser testada e os melhoramentos necessários efectuados.

Naturalmente, a interface deve ser testada com a aplicação subjacente conectada - e esta é uma vantagem fornecida pelos UIMSs quando comparados com outras ferramentas de suporte ao desenvolvimento de GUIs. A facilidade de utilização da interface, entre outras características, só pode ser determinada quando a aplicação é executada.

Configuração da interface pelo utilizador

O elemento final do desenvolvimento de GUIs recorrendo aos UIMSs consiste em facultar ao utilizador a capacidade de configurar a interface, permitindo-lhe modificar alguns aspectos da interface como, por exemplo, as cores. Deve ser definido o nível de configuração a ser fornecido, bem como o modo de a providenciar.

Vários UIMSs permitem definir interactivamente o nível de configuração da interface pelo utilizador. Deste modo, os atributos dos componentes da interface podem ser declarados como privados, públicos, ou por defeito. Atributos definidos como privados não estão acessíveis ao utilizador, e o UIMS é responsável por gerar o código necessário à especificação do valor escolhido para o atributo. Atributos públicos estão acessíveis ao utilizador, tipicamente através de ficheiros de recursos. Se um atributo é declarado por defeito, o UIMS utiliza o valor de defeito codificado no próprio objecto.

6.3.3 O que procurar num UIMS

Existem inúmeros sistemas integrados de suporte ao desenvolvimento de interfaces gráficas com o utilizador, e com a atenção devotada a esta área, naturalmente muitos mais surgirão no futuro.

Na avaliação de um UIMS existem diversos factores que devem ser judiciosamente equacionados - especialmente aqueles que se revelem críticos para o projecto em causa. São alguns destes factores que se referem nos pontos seguintes.

Registo de acções

A capacidade de capturar a sequência das acções do utilizador é útil para algumas fases de teste da facilidade de utilização da GUI, assim como para determinar a existência de erros intermitentes durante o teste de versões *beta* da interface. A descrição da sequência de acções registadas deve atingir um nível relativamente elevado (e.g., "o menu Editar foi aberto"), e não consistir apenas numa lista de teclas carregadas e de posições X,Y do rato.

Suporte para programas extensos

Devem ser procuradas linguagens de programação e aspectos do ambiente que permitam a aplicação de boas técnicas de programação: modularidade (ficheiros múltiplos, compilação por partes, suporte de versões), um bom programa de detecção e eliminação de erros, etc. Deve ser assegurado que a rapidez de execução dos programas gerados pelo UIMS está de acordo com as necessidades da aplicação.

Geração do código

O UIMS gera código executável ou produz ficheiros de código fonte para serem compilados com um compilador usual? A primeira abordagem é mais conveniente, enquanto a segunda fornece maior flexibilidade. É, no entanto, de ter em conta que código gerado automaticamente é geralmente impossível de ser mantido por programadores humanos.

Extensibilidade da interface

Podem adicionar-se novos controlos ao ambiente de programação, e estes terão a mesma facilidade de programação que o conjunto de controlos originais? Igualmente importante, que extensões se encontram disponíveis?

Extensibilidade do programa

Pode o programa gerado chamar subrotinas escritas noutra linguagem?

Técnicas específicas do sistema operativo

O UIMS deve suportar quaisquer técnicas especiais relativas à interface que os utilizadores de outros programas, no mesmo sistema operativo, se habituaram a tomar como um dado adquirido. Por exemplo, um UIMS para o Macintosh deve suportar o *clipboard* do Macintosh, e deve gerar aplicações que possam ser executadas através de uma dupla pressão com o rato nos ficheiros; um UIMS para o Windows deve suportar OLE (Object Linking and Embedding - Secção 6.4.3) e ajudar a construir um programa de instalação da aplicação.

Capacidade de desenvolvimento do protótipo até à aplicação final

É desejável poder começar e acabar o desenvolvimento utilizando um único UIMS. Isto permite rentabilizar o tempo gasto na construção do protótipo, uma vez que este pode fazer parte da aplicação final.

Construção de aplicações *stand-alone*

Naturalmente, é mais conveniente se o UIMS gerar aplicações que não obriguem o utilizador a adquirir uma aplicação de suporte para correr a interface gráfica.

Suporte de um livro de estilo particular

Alguns UIMSs verificam automaticamente as indicações fornecidas no livro de estilo da interface, como "todos os diálogos devem ter um botão de *OK*", ou "os menus não devem possuir mais do que 7 itens".

Suporte e longevidade do vendedor

O vendedor do UIMS tem reputação de fornecer manutenção ao sistema e responder a novas versões do sistema operativo? Será que a empresa ainda existe quando a GUI se tornar um sucesso?

Programação visual

É possível definir o aspecto e disposição no ecrã dos componentes da interface, visualmente? É possível especificar constrangimentos no desenho da interface, de preferência através de ferramentas de desenho ("alinhar à esquerda") ou através de constrangimentos programáveis ("todos os botões desta classe devem ter 50 pixeis de altura")?

6.4 Ferramentas específicas de suporte ao desenvolvimento de GUIs

Nas secções seguintes apresentam-se ferramentas de suporte ao desenvolvimento de GUIs específicas para os sistemas mais correntes actualmente, sendo dedicada especial atenção ao sistema X-Windows.

O ênfase dado ao sistema X-Windows justifica-se pelas características deste sistema, que o tornam num dos mais poderosos e sofisticados sistemas de janelas alguma vez desenvolvidos, representando o primeiro grande avanço numa interface gráfica com o utilizador, que pode suportar concorrentemente diversas aplicações baseadas em diferentes plataformas.

Uma das facetas mais importantes do sistema X-Windows é fornecer ao utilizador a capacidade de, simultaneamente, correr aplicações em diferentes sistemas, através de

uma rede. Estas aplicações podem ser provenientes de diversos fabricantes e correr em sistemas operativos diferentes.

A independência do X-Windows de um *hardware* ou de uma arquitectura de *software* específicos, oferece um meio de integrar sistemas computacionais heterogéneos e desenvolver GUIs consistentes para diversas aplicações. Esta faceta possui especial relevância no desenvolvimento de GUIs para ambientes de fabrico onde, tipicamente, existem sistemas computacionais provenientes de fabricantes distintos. Adicionalmente, o sistema X-Windows fornece uma excelente base para a programação orientada por objectos.

6.4.1 X-Windows, OSF/Motif e AIC

X-Windows

O sistema X-Windows, ou X, é um sistema de janelas, transparente em termos de rede. Com o X, múltiplas aplicações podem correr simultaneamente em janelas, gerando texto e gráficos, em monitores a cores ou monocromáticos. A transparência da rede significa que as aplicações podem correr em várias máquinas localizadas em diversos pontos da rede. Como o X permite que as aplicações sejam independentes dos dispositivos, as aplicações não precisam de ser reescritas ou mesmo recompiladas para funcionarem num *hardware* de visualização diferente.

O X fornece capacidades para a criação de texto e elementos gráficos bi-dimensionais (e.g., pontos, linhas, arcos, polígonos) numa hierarquia de janelas rectangulares. Cada janela pode ser pensada como um "ecrã virtual" e pode conter subjanelas dentro dela, até um nível de profundidade arbitrário. As janelas podem sobrepor-se umas às outras, como folhas de papel numa secretária e a sua localização, dimensão e ordem de sobreposição, podem ser alteradas dinamicamente.

As janelas constituem recursos pouco dispendiosos; aplicações que utilizam centenas de subjanelas são comuns. Por exemplo, as janelas são frequentemente utilizadas para implementar os componentes individuais da interface com o utilizador, como botões e menus.

Embora os utilizadores tipicamente se considerem como clientes do sistema, as aplicações X, na perspectiva da rede, são os clientes que utilizam os serviços de rede do sistema. Um programa a correr na máquina com o *hardware* de visualização fornece estes serviços e portanto é denominada servidor X. O servidor X actua como um intermediário entre as aplicações e o ecrã, recebendo as saídas dos clientes para o ecrã e enviando as entradas (efectuadas na maior parte dos casos, via teclado ou rato) aos clientes apropriados para processamento.

Os clientes e servidores utilizam alguma forma de comunicação entre processos para trocarem informação. A sintaxe e a semântica desta "conversação" são definidas por um protocolo de comunicação. Este protocolo é a fundação do sistema X-Windows. Os clientes usam o protocolo para enviar pedidos ao servidor para criar e manipular janelas, para gerar texto e gráficos, para controlar as entradas efectuadas pelo utilizador, e para comunicar com outros clientes. O servidor utiliza o protocolo para enviar informação aos clientes como resposta a vários pedidos e para enviar as entradas do utilizador para os clientes apropriados.

Atendendo a que a circulação de dados perfazendo uma volta completa na rede é uma operação onerosa comparada com a execução básica de pedidos, o protocolo é essencialmente assíncrono, e os dados podem transitar em ambas as direcções (do cliente para o servidor e do servidor para o cliente) simultaneamente. Depois de gerar um pedido, o cliente tipicamente não espera que o servidor execute o pedido antes de gerar um novo pedido. Em vez disto, o cliente gera uma série de pedidos que são eventualmente recebidos pelo servidor e executados. O servidor não dá conhecimento da recepção de um pedido e, em muitos casos, não dá conhecimento da execução de um pedido. Isto é possível porque o protocolo subjacente é fiável.

O protocolo foi explicitamente concebido para minimizar a necessidade de inquirir o sistema de janelas para obter informação. Os clientes não devem depender do servidor para obter informação que os clientes inicialmente forneceram. Adicionalmente, os clientes não inquirem da existência de entradas, enviando pedidos para o servidor. Em vez disso, os clientes utilizam os pedidos para manifestar interesse em vários eventos, e o servidor envia as notificações de eventos assincronamente. A

operação assíncrona aparece como uma das diferenças mais significativas entre o sistema X-Windows e outros sistemas de janelas.

No sentido de obter um melhor desempenho, quando o cliente e o servidor residem na mesma máquina, a comunicação entre eles é frequentemente implementada usando memória partilhada. Quando o cliente e o servidor residem em máquinas diferentes, a comunicação pode ter lugar sobre qualquer camada de transporte da rede que faculte a transferência de dados em ambas as direcções, de um modo ordenado e fiável. Por exemplo, o TCP (Transport Control Protocol, da família de protocolos da Internet) e o DECnet (da Digital Equipment Corporation), são camadas de transporte comummente usadas.

Para suportar a computação distribuída num ambiente heterogéneo, o protocolo de comunicação é concebido para ser independente do sistema operativo, da linguagem de programação, e do *hardware* do processador. Deste modo, num único ecrã podem ser visualizadas aplicações escritas em múltiplas linguagens sob múltiplos sistemas operativos em múltiplas arquitecturas de *hardware*, simultaneamente.

Embora o X seja fundamentalmente definido por um protocolo de rede, muitos programadores de aplicações não querem pensar em *bits*, *bytes* e formatos das mensagens. Assim, o X possui uma biblioteca de interface. Esta biblioteca fornece uma interface de procedimentos familiar que mascara os detalhes de codificação do protocolo e as interacções de transporte, efectuando de um modo automático o envio de vários pedidos em conjunto, no sentido de minimizar o transporte para o servidor. Funciona de um modo análogo à biblioteca de I/O da linguagem C, que envia várias saídas em conjunto tendo em vista a minimização das chamadas ao sistema.

A biblioteca de interface também providencia outras funcionalidades úteis que não estão directamente relacionadas com o protocolo, mas que contudo são importantes na construção de aplicações. A biblioteca de interface efectiva difere para cada linguagem de programação - Xlib é a biblioteca para a linguagem de programação C.

Cada servidor X controla um ou mais ecrãs, um teclado (na generalidade dos casos), e um dispositivo de interacção - tipicamente, um rato. Podem existir vários

servidores X; frequentemente, existe um por cada estação de trabalho na rede. As aplicações podem correr em qualquer máquina, mesmo naquelas em que não existam servidores X. Uma aplicação pode comunicar com múltiplos servidores simultaneamente (por exemplo, para suportar conferências via computador entre pessoas em locais diferentes). Múltiplas aplicações podem estar activas ao mesmo tempo num único servidor.

No X, muitas das capacidades que são intrínsecas em outros sistemas de janelas são fornecidas por bibliotecas dos clientes. O protocolo X não especifica menus, botões, janelas de diálogo, ou como a aplicação deve reagir às entradas do utilizador. O protocolo e biblioteca do X evitam o mais possível ser preceptivos neste tipo de decisões e devem ser considerados como uma ferramenta de construção, que faculta um conjunto rico de mecanismos, possibilitando a implementação de vários tipos de interfaces com o utilizador. Os *toolkits* (que fornecem menus, botões, janelas de diálogo, entre outros controlos de uma GUI), bibliotecas gráficas de alto nível (que, por exemplo, podem transformar descrições abstractas de objectos em pedidos gráficos), e sistemas integrados de desenvolvimento de GUIs podem ser implementados sobre a biblioteca X. Embora a biblioteca X forneça os alicerces, espera-se que as aplicações sejam escritas usando estas ferramentas de alto nível em conjunto com as capacidades da biblioteca X, em vez de se recorrer unicamente ao esqueleto da biblioteca X.

Uma interface com o utilizador pode ser vista como possuindo duas componentes básicas: a interacção com o utilizador que é logicamente interna a uma aplicação (por exemplo, escrever texto num editor de texto ou mudar o conteúdo de uma célula numa folha de cálculo) e a interacção que é logicamente externa a uma aplicação (por exemplo, mover ou alterar o tamanho de uma janela da aplicação ou converter uma janela da aplicação num ícone). A interface externa com o utilizador está embebida em muitos sistemas baseados em janelas, mas não é este o caso do X. De facto, o protocolo X não define de modo algum a interface externa com o utilizador. Antes, fornece mecanismos com os quais uma variedade de interfaces externas com o utilizador podem ser construídas. Estes mecanismos são concebidos de modo que um único cliente, denominado gestor de janelas, pode fornecer a interface externa com o utilizador, independente de todos os outros clientes.

Um gestor de janelas permite adoptar uma política estrita na representação externa da interface (por exemplo, dividir o ecrã de modo a que as janelas das aplicações nunca se sobreponham), bem como fornecer de um modo automático o seguinte:

- ▷ Barras de título, bordos e outras decorações das janelas para cada aplicação.
- ▷ Ícones uniformes para as aplicações.
- ▷ Um meio uniforme de mover e alterar o tamanho das janelas.
- ▷ Uma interface uniforme para mudar o teclado entre aplicações.

Através de um conjunto adequado de convenções, que foram normalizadas e estão consagradas num manual designado por ICCCM (Inter-Client Communications Conventions Manual), as aplicações são insensíveis à interface externa fornecida pelo gestor de janelas e podem ser correctamente executadas, sem modificações, em múltiplos ambientes. Como o protocolo pode lidar com um espectro tão alargado de interfaces com o utilizador, nenhum único programa, *toolkit*, UIMS, ou gestor de janelas irá provavelmente usar todas as faculdades que o protocolo e a biblioteca X fornecem.

A Figura 6.2 representa a arquitectura por camadas do sistema X-Windows.

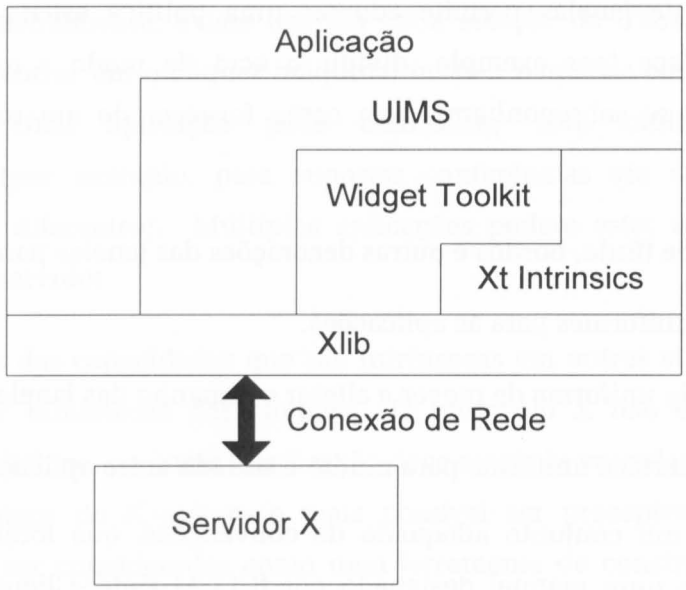


Figura 6.2 Arquitectura do sistema X-Windows.

Existem várias camadas primitivas. A camada X Protocol implementa os aspectos transparentes da rede, do sistema X-Windows. Esta camada implementa a funcionalidade que permite ao utilizador visualizar uma dada aplicação (na sua própria janela) num outro terminal X. Todos os terminais X correm uma cópia do servidor X. A camada Xlib contém todas as primitivas de baixo nível para construir aplicações no X-Windows. Este será o nível mais baixo a que, na generalidade dos casos, um programador recorrerá para desenvolver aplicações no X-Windows.

As duas camadas seguintes apresentam um nível mais alto de abstracção. A camada Xt Intrinsics utiliza as primitivas da camada Xlib e fornece um ambiente básico para construir e utilizar os componentes gráficos da interface - *widgets*[3]. A camada do *toolkit* tem estado sujeita a confrontações entre dois campos rivais tentando definir normas para o aspecto da interface com o utilizador. Os dois grupos desenvolveram *toolkits* (utilizando a camada Intrinsics) compostos por *widgets* baseadas nas suas normas. O primeiro grupo - UNIX International - produziu um *toolkit* chamado Open

[3] *Widget* é o termo consagrado para designar qualquer componente gráfico da interface com o utilizador (e.g. menus, botões, janelas de diálogo), no âmbito do X. Os atributos das *widgets* são designados por *propriedades*.

Look. O segundo grupo - Open Software Foundation (OSF) - desenvolveu outro *toolkit*, o Motif.

As organizações envolvidas no desenvolvimento de interfaces gráficas com o utilizador, têm alinhado por um ou outro dos dois grupos. Aparentemente, o Motif tem conseguido reunir mais consenso e, actualmente, é mesmo considerado um *standard de facto*.

Embora ambos os *toolkits* facilitem consideravelmente o desenvolvimento de GUIs, o processo ainda é considerado "pesado". A última camada, a camada do UIMS, tenta resolver este problema.

OSF/Motif

O Motif consiste numa biblioteca de subrotinas na linguagem C que definem um conjunto de objectos da interface gráfica e técnicas para a combinação desses objectos numa aplicação. O Motif permite criar programas controlados por eventos, orientados por objectos e adaptáveis através de ficheiros externos de recursos.

Todos os componentes gráficos da interface no Motif são designados por *widgets*, e estão definidos como uma hierarquia orientada por objectos. Uma classe típica do Motif são as *labels*, que possuem botões como uma subclasse, a qual por sua vez contém botões individuais como instâncias. As opções para cada classe e instâncias, como cor, texto, e comportamento, são especificadas como recursos numa base de dados do programa. A base de dados é criada no início do programa pela leitura de diversos ficheiros de texto - ficheiros de recursos - que descrevem o programa, o *hardware* e as preferências do utilizador.

As *widgets* numa aplicação também estão arranjadas hierarquicamente, mas num sentido diferente do *toolkit*: a janela principal do programa é a *widget* de topo da hierarquia, um menu dentro da janela pertence ao nível seguinte, e os botões dentro do menu pertencem ao nível a seguir. Muitos dos parâmetros dos níveis mais baixos são herdados dos níveis mais altos ou calculados com base nestes. Por exemplo, mover uma janela modifica os seus parâmetros de localização, e esta mudança propaga-se através da hierarquia até aos componentes pertencentes à janela.

O comportamento durante a execução de um programa em Motif é dirigido por eventos, controlados por um ciclo de eventos que faz parte do *toolkit*. O sistema espera pela ocorrência de eventos como pressões com o rato, movimento do cursor dentro ou fora das janelas, e teclas pressionadas. Quando ocorre um evento numa *widget*, o sistema verifica a tabela de translações, que associa uma dada acção a um evento particular e faz parte da lista de recursos da *widget*, e indica qual a acção a tomar, caso exista.

A tabela de translações pode ser modificada pelo programa durante a sua execução; isto permite que o comportamento de uma determinada *widget* possa reflectir a situação actual. Se a tabela de translações indicar que uma dada acção deve ser tomada, o sistema gera uma `callback` para o programa cliente, indicando-lhe o que é necessário fazer.

Escrever um programa em Motif é semelhante a escrever um programa típico em C, com alguns aumentos. Pode começar-se por escrever um módulo que define as funcionalidades básicas do sistema. Este programa pode ser compilado e testado independentemente do código da interface. O código da interface Motif pode ser escrito como um módulo separado.

Uma das partes do código Motif que é necessário criar são os ficheiros de recursos, que definem todos os parâmetros iniciais dos objectos no ecrã. Como o Motif não é um ambiente de programação visual, cada parâmetro tem de ser especificado textualmente.

Depois de definidos os recursos, é necessário escrever o programa que no fundo será o coração do sistema. Este código começará com informação inicial (*headers*) para referenciar os ficheiros do Motif, que incluem pelo menos 30 bibliotecas de classes de *widgets*. Neste código também é necessário definir os procedimentos (*callbacks*) que associam as acções da interface às funções definidas no módulo principal. A seguir é necessário especificar uma série de acções necessárias à execução do programa: inicializar o *toolkit* do Motif, abrir uma conexão de rede ao servidor, definir a *widget* de topo da hierarquia, ler as especificações da base de dados e definir todas as *widgets* da janela, e depois visualizar todas as *widgets*. Finalmente, o código terá que passar o controlo para um ciclo de eventos, que é fornecido como uma das rotinas do Motif.

Embora o Motif permita construir interfaces gráficas consistentes para o sistema X-Windows, a tarefa de programação recorrendo apenas ao *toolkit*, e sem fazer uso de uma ferramenta de suporte ao desenvolvimento de GUIs, ainda é bastante árdua: um simples programa que permita apenas visualizar uma única janela de diálogo, requererá aproximadamente 100 linhas de código C!

AIC (AIXwindows Interface Composer)

O AIC é um UIMS da IBM, para desenvolver interfaces gráficas baseadas no Motif e que corre no sistema operativo AIX. O AIX (da IBM) é uma das muitas variantes do UNIX que suporta o sistema X-Windows, como o SunOS (da Sun) ou o HP-UX (da Hewlett-Packard). O AIC possui diversas capacidades avançadas para a construção de aplicações em Motif, que incluem um interpretador de linguagem C, a capacidade de definir todas as propriedades acessíveis das *widgets*, uma biblioteca de subrotinas de programação para criar e manipular os componentes das interfaces gráficas do Motif, e diversos editores embebidos no UIMS.

A criação de menus complexos em X pode ser confusa e difícil para alguns programadores; o AIC simplifica grandemente esta tarefa através de um editor de menus que permite criar menus sofisticados do tipo *popup*, de abertura vertical e de opções, que podem conter outros menus em cascata. Os outros editores embebidos são: um editor de texto, um editor de tabelas de acções, um editor de tabelas de translações, um editor de eventos, e um editor de declarações suportado pela capacidade de edição da tabela de propriedades. Uma das facetas mais significativas do AIC é suportar o Motif na totalidade: todas as janelas, diálogos e recursos do Motif estão contemplados no AIC.

O desenho das interfaces no ecrã é efectuado de um modo visual, através de um editor WYSIWYG. No AIC os componentes da interface gráfica que são desenhados directamente na janela de raiz são tratados como componentes independentes da interface, como por exemplo janelas de diálogo. Os componentes que fazem parte de outros componentes - "pais" - são denominados "filhos" (e.g., os botões das janelas de diálogo).

Depois do desenho inicial da interface gráfica é necessário especificar o comportamento da aplicação. O desenho envolve a especificação do estado estático

inicial da interface; a especificação do comportamento envolve a definição de todos os elementos dinâmicos da interface. A especificação do comportamento no AIC é efectuada através do editor de propriedades das *widgets*. As propriedade dividem-se em nucleares, específicas e de comportamento. As propriedades nucleares incluem aquelas que são comuns a todas as *widgets*, como a cor de fundo, a localização X,Y, informação sobre a altura e largura, entre outras. As propriedades específicas são particulares de uma dada classe de *widgets*. Por fim, as propriedades de comportamento são usadas para especificar o código das `callbacks` aplicáveis a uma classe particular.

O AIC possui uma biblioteca própria de funções que permitem realizar tarefas complexas como converter recursos, alocar as entradas do mapa de cores, manusear automaticamente os filhos das janelas de diálogo, e controlar casos especiais da geometria dos componentes. Esta biblioteca também efectua a detecção, notificação e recuperação de erros, fornecendo uma ajuda substancial na fase de teste.

Adicionalmente, o AIC permite especificar e testar interfaces paramétricas; i.e., interfaces com um estado inicial dinâmico. Qualquer expressão válida na linguagem C é permitida para a definição do valor de qualquer recurso ou propriedade. A expressão em C é avaliada quando o componente da GUI é criado, quer no AIC aquando da construção da interface, quer no código gerado. O interpretador de C pode aceder e chamar código compilado da aplicação, permitindo desenvolver e testar GUIs paramétricas. O AIC também permite construir componentes paramétricos reutilizáveis.

A capacidade de testar a GUI em conjunto com a aplicação subjacente conectada está contemplada no AIC através do interpretador da linguagem C embebido no UIMS. Este interpretador permite misturar código compilado com código interpretado, facultando a modificação de variáveis globais, e até passar o endereço de uma rotina interpretada para uma rotina compilada, para referência e execução.

O interpretador de C pode ser usado para desenvolver e testar interfaces gráficas rapidamente, e uma vez completo o processo, compilar as interfaces individualmente para obter um melhor desempenho. É possível inserir código em C da interface nas `callbacks` e separar completamente o código da interface do código da aplicação

subjacente, possibilitando uma abordagem modular no desenvolvimento das GUIs. O facto do AIC permitir o acesso completo ao ciclo de eventos possibilita a entrada e processamento de eventos que podem não ser originários do *toolkit* do X. Esta capacidade é de suma importância, por exemplo, quando se pretende efectuar a entrada de dados provenientes de uma aplicação externa, em tempo real. Esta necessidade surge com frequência em GUIs para ambientes de fabrico.

O AIC permite gerar código portável na linguagem C ou C++, ficheiros de recursos do X, e um ficheiro especial usado para recriar a sessão de interface do UIMS, denominado ficheiro de interface. No ficheiro de recursos, pode escolher-se incluir ou excluir qualquer recurso de todas as *widgets* existentes na interface, possibilitando assim a definição do nível de configuração da interface acessível ao utilizador. Adicionalmente, o AIC permite gerar um ficheiro executável de compilação e, se desejado, correr esse ficheiro e compilar a aplicação dentro do próprio ambiente de desenvolvimento.

A consideração de todas as funcionalidades supracitadas permite catalogar o AIC como um UIMS extremamente valioso no desenvolvimento de interfaces gráficas com o utilizador, para o sistema X-Windows, em Motif. Mas este é apenas um exemplo das inúmeras ferramentas de suporte ao desenvolvimento de interfaces gráficas existentes para o sistema X-Windows. Existem diversas ferramentas para produzir aplicações baseadas no Motif, aplicações baseadas no Open Look (e.g., o DevGuide da Sun para a implementação XView do Open Look), e até aplicações que são independentes de uma dada norma de interfaces gráficas (e.g., o Garnet desenvolvido pela Carnegie-Mellon University, que não suporta qualquer estilo de interface particular).

6.4.2 Macintosh e HyperCard

O computador Apple II foi o primeiro computador da Apple a conhecer um grande sucesso. Cada computador Apple II incluía um interpretador da linguagem Basic para os utilizadores que desejavam escrever os seus próprios programas. Quando a Apple introduziu o Macintosh, muitos utilizadores potenciais sentiram-se desapontados pelo facto de este computador não fornecer uma linguagem de programação normalizada. O HyperCard, introduzido alguns anos depois, é um ambiente de programação de baixo

custo, requerendo pouco esforço de programação e que responde a algumas dessas preocupações dos utilizadores.

O HyperCard é uma aplicação que permite desenvolver interfaces gráficas para os computadores Macintosh, com relativa facilidade. A aplicação HyperCard em si, não só é necessária para desenvolver as interfaces gráficas, como também para as correr. Todos os computadores Macintosh actuais possuem uma aplicação denominada HyperCard Player, que consiste numa versão da aplicação HyperCard que permite correr os programas desenvolvidos nesta aplicação, mas que tem capacidades limitadas de programação. Os programas escritos no ambiente HyperCard são geralmente denominados "pilhas" (*stacks*) porque apresentam uma metáfora baseada numa pilha de cartas.

A ferramenta HyperCard fornece um ambiente de programação visual e orientado por objectos, no qual o utilizador pode criar e configurar três classes principais de objectos: botões e campos de texto, que são controlos do ecrã; e cartas, que são janelas individuais que contém os botões e os campos de texto. Por exemplo, uma lista telefónica pessoal pode ser implementada recorrendo ao HyperCard, como sendo uma pilha de 26 cartas, cada uma com campos de texto contendo nomes e números de telefone, e com um botão denominado de A a Z que permitem ao utilizador aceder à carta apropriada. Este exemplo constitui exactamente o tipo de programa para o qual o HyperCard foi concebido, e podia ser suportado por um número de funções já incluídas no próprio sistema, como "vá para a carta seguinte" ou "encontre este texto". Como todas as cartas são iguais exceptuando o conteúdo de texto, podiam até ser implementadas recorrendo a uma única classe, denominada "*background*".

A aplicação da lista telefónica pode ser programada interagindo apenas com o ambiente de programação visual: seleccionando itens dos menus, preenchendo as janelas de diálogo e colocando os botões e campos de texto no local desejado. Para aplicações mais sofisticadas, o HyperCard fornece uma linguagem semelhante ao Pascal chamada HyperTalk. Os procedimentos (*handlers*) do HyperTalk são activados por mensagens provenientes dos controlos disponíveis ao utilizador na interface (e.g., botões) e por outros procedimentos.

Dado que o HyperTalk é uma linguagem interpretada, os procedimentos podem ser testados assim que são escritos, sem necessidade de compilação prévia. Mas os procedimentos não correm a uma velocidade excepcionalmente grande e apenas fornecem um acesso limitado ao *toolkit* do sistema operativo do Macintosh, que é necessário para criar objectos normalizados das interfaces gráficas, como janelas de diálogo genéricas.

Estas limitações podem ser ultrapassadas escrevendo funções e comandos externos (XFCNs e XCMDs), que são procedimentos escritos numa linguagem de programação como C ou Pascal, podendo ser incorporados numa *stack* do HyperCard como recursos do código. Estes procedimentos podem ser invocados por um procedimento do HyperTalk como se fossem funções do próprio HyperTalk.

O HyperCard pode ser uma ferramenta muito útil para criar protótipos de interfaces simples, uma vez que uma ideia para o aspecto da interface pode ser apresentada para avaliação em menos de um dia, e a implementação de funcionalidade suficiente que permita realizar os primeiros testes com os utilizadores, também não demorará muito mais tempo.

No entanto, este sistema possui duas limitações substanciais. Primeiro, a menos que se recorra a comandos externos, os programas desenvolvidos parecem-se com *stacks* do HyperCard e não com programas do Macintosh. Mesmo recorrendo a comandos externos, os menus e as janelas de diálogo nunca se tornam verdadeiros objectos no ambiente de programação visual. Este facto faz do HyperCard uma ferramenta adequada para desenvolver protótipos de aplicações com botões e campos de texto, como as interfaces gráficas com o utilizador das ATMs (Automatic Teller Machines), mas não o ambiente ideal para desenvolver protótipos de aplicações do Macintosh.

A segunda limitação do HyperCard é que não foi concebido como um ambiente de programação para aplicações genéricas e com requisitos de funcionalidade elevados. Este ambiente é bastante bom para desenvolver protótipos simples ou aplicações pequenas que se enquadrem na metáfora de "pilha de cartas". Contudo, se se pretender ir além destes limites, rapidamente ser-se-á confrontado com um programa de enorme dimensão, num ambiente que fornece pouco suporte a nível de modularidade.

O HyperCard e programas similares são ferramentas simples para a concepção de GUIs, que devem estar disponíveis e usadas quando apropriado. Para testes elaborados e desenvolvimento de aplicações finais para o Macintosh, na generalidade dos casos, será de utilizar um UIMS mais poderoso. Este pode ser um ambiente de programação visual ou uma linguagem de programação com extensões orientadas por objectos que suportem todos os objectos da interface do sistema.

6.4.3 MS-Windows e Visual Basic

O Microsoft Windows é um produto ao nível do sistema operativo que suporta interfaces gráficas com o utilizador na plataforma Intel 80x86 (PCs, ATs, compatíveis, etc.). A versão 3 do Windows, introduzida em 1990, é um dos produtos mais populares alguma vez desenvolvidos. O tremendo mercado potencial para aplicações que corram sob o Windows atraiu naturalmente a atenção de inúmeras organizações de desenvolvimento de produtos, incluindo algumas que desenvolveram UIMSs.

A popularidade do Windows certamente que será, em parte, devida à disponibilidade de *hardware* compatível, poderoso e relativamente pouco dispendioso: as máquinas 80386 e 80486 de inúmeros fabricantes competitivos. Mas é de notar que a Microsoft tem mantido a sua posição de liderança neste mercado, através da continuada evolução na funcionalidade do *software*. Actualmente, o Windows fornece uma sofisticada interface com o utilizador dirigida por eventos, gestão da memória, a capacidade de executar vários programas em simultâneo e diversas maneiras de estes programas comunicarem entre si. As duas últimas facetas tornam possível o desenvolvimento de aplicações que efectivamente partilhem o código de outras aplicações que o utilizador já possui.

Alguns dos protocolos de comunicação interaplicações que o Windows suporta são:

- DLL (Dynamic Link Libraries). DLLs são rotinas do *toolkit* para desenhar e correr vários controlos, ou para executar acções baseadas em procedimentos como, por exemplo, a ordenação de dados. Muitas destas rotinas vêm com o Windows, e rotinas adicionais podem ser obtidas de organizações

independentes de desenvolvimento de *software*. Também é possível escrever DLLs próprias numa linguagem de programação, como o C, por exemplo.

- DDE (Dynamic Data Exchange). DDE é o protocolo fundamental para comunicação entre aplicações. Este protocolo permite a um dado programa (cliente) enviar dados a uma outra aplicação (servidor), pedindo à aplicação para executar um item de um menu ou uma *macro*. Também permite que o programa cliente receba dados enviados por uma outra aplicação.
- OLE (Object Linking and Embedding). OLE consiste em duas técnicas que permitem ao utilizador criar documentos contendo objectos criados e mantidos por aplicações diferentes. Por exemplo, estas técnicas permitem ao utilizador incluir um desenho "vivo" num documento do processador de texto. A selecção do desenho faz aparecer a aplicação gráfica usada para conceber o desenho, com o desenho já pronto a ser editado. Se o desenho está apenas ligado (*linked*), então a sua apresentação é sempre da responsabilidade do programa gráfico, mesmo quando o desenho é apresentado no processador de texto. Se o desenho está embebido (*embedded*), então o processador de texto mantém uma cópia estática do desenho, o que permite que o programa gráfico não necessite de estar a correr.

Quando se pretende utilizar comunicação entre aplicações no desenvolvimento de GUIs em Windows, é necessário saber que aplicações os utilizadores já possuem e conhecer as convenções que regem a comunicação interaplicações daquelas. Adicionalmente, é necessário construir o código e a interface que são únicos da GUI, bem como a "cola" que permite ligar tudo. Para simplificar esta parte do trabalho existem vários UIMSs vocacionados para a programação em Windows. Um destes é o Visual Basic da Microsoft.

O Visual Basic combina um editor visual de interfaces com uma linguagem de programação comandada por eventos. Criar uma interface no Visual Basic é assim uma questão de criar janelas ("*forms*"), menus, botões, e outros controlos interactivamente. Selecciona-se o controlo desejado na paleta de desenho, coloca-se na posição pretendida e especificam-se as suas características numa janela de diálogo.

O comportamento de cada controlo é definido em código Basic escrito na janela de código do controlo, e os controlos enviam mensagens entre si e para procedimentos que definem a funcionalidade do programa. Os programas, para serem executados, necessitam incluir um módulo de suporte do Basic, implementado como um ficheiro DLL. Extensões ao sistema podem ser realizadas com código DLL próprio, ou de outras fontes se se possuir a licença respectiva.

Conceptualmente o Visual Basic é bastante similar ao HyperCard. No entanto, o Visual Basic é um ambiente completo de desenvolvimento de interfaces gráficas genéricas, e fornece um suporte substancial para atingir esse objectivo. Menus, janelas, botões e todos os outros controlos do Windows podem ser criados via programação visual. Controlos adicionais licenciados como rotinas DLL provenientes de outras fontes - "*third-party* DLLs" - também são completamente integrados no ambiente visual.

A modularidade do código é suportada através da permissão de múltiplos ficheiros com diversos níveis de contexto das variáveis: locais a um procedimento (caso por defeito), partilhadas por todos os procedimentos numa *form* ou módulo, ou globais - acessíveis a todos os procedimentos. A técnica de suporte à execução através de um ficheiro DLL é comum no Windows e o Visual Basic permite criar um programa de instalação para a aplicação, para assegurar que todos os ficheiros necessários estão nos locais certos e actualizados, na máquina do utilizador.

O Visual Basic também suporta a comunicação entre aplicações; é possível chamar outras DLLs, enviar e receber mensagens de outros programas usando o protocolo DDE, e fazer com que outros programas se comportem como servidores OLE (embora a aplicação desenvolvida no Visual Basic apenas possa funcionar como cliente).

A resposta ao Visual Basic para o desenvolvimento de GUIs em Windows tem sido bastante positiva, sendo esta ferramenta bastante popular. Existe, no entanto, alguma preocupação com a rapidez de execução, mas frequentemente este problema pode ser ultrapassado codificando rotinas críticas como DLLs. Para programadores que não se sentem à-vontade com o paradigma da orientação por eventos, existem outros UIMSs que oferecem funcionalidades para a construção de interfaces gráficas com o

utilizador similares ao Visual Basic, mas que possuem uma linguagem mais baseada em procedimentos.

6.5 Conclusões

O desenvolvimento de interfaces gráficas com o utilizador é considerado de um modo unânime como sendo extremamente complexo. As ferramentas de suporte ao desenvolvimento de GUIs permitem aliviar de algum modo a tarefa de programação.

Os sistemas integrados de desenvolvimento de GUIs pelas suas facilidades de rápida construção, iteração e teste constituem ferramentas extremamente valiosas na criação de interfaces gráficas. O recurso dos UIMSs à programação visual permite o desenvolvimento de GUIs de um modo visual interactivo, o que para além de facilitar e tornar mais rápida a construção das interfaces, possibilita visualizar a todo o instante qual o aspecto real da interface e o modo como esta se comporta. No fundo, trata-se de fazer uso dos conceitos subjacentes às interfaces gráficas, no seu próprio desenvolvimento.

Todos estes temas foram abordados neste capítulo. Adicionalmente, o sistema X-Windows, dada a sua normalização *de facto* nos ambientes UNIX, foi descrito com algum detalhe. Outros sistemas correntes baseados em janelas foram referidos. Para cada um dos sistemas de janelas apresentado, foram descritas ferramentas específicas de suporte ao desenvolvimento de GUIs.

Capítulo 7

Teste e documentação de GUIs

A necessidade de testar as aplicações desenvolvidas é uma realidade para qualquer tipo de aplicação. As interfaces gráficas com o utilizador, pelas suas características de interacção, representação externa e complexidade do próprio código, colocam requisitos adicionais e requerem estratégias próprias com vista à realização de testes neste tipo de aplicações.

Por outro lado, virtualmente qualquer interface gráfica actual com o utilizador para sistemas computacionais, necessita de providenciar informação e ajuda ao utilizador, para que este possa utilizar o sistema de uma forma eficaz.

Deve ter-se presente que a interface com o utilizador não é apenas a interface *on-line*, i.e., janelas, menus, ícones e outros mecanismos de interacção representados no ecrã, mas que a documentação constitui uma parte importante e significativa da interface.

São estes os assuntos abordados neste capítulo.

7.1 Teste de GUIs

Por muito empenho que se coloque na construção de uma boa interface gráfica, definindo os seus requisitos sem ambiguidade, envolvendo o utilizador ao longo de todo o processo, utilizando ferramentas de apoio ao desenvolvimento, obedecendo às linhas mestras, será sempre necessário testar a interface.

Existem sempre detalhes que numa fase anterior podem ter escapado à atenção do *designer*, pormenores imprevistos cuja ocorrência foi olvidada e outros aspectos que só na fase de teste são clarificados. Só avaliando e testando a interface se poderá ter uma medida mais exacta da sua validade. O "verdadeiro" teste final só é efectuado no mercado com os utilizadores.

A seguir referem-se duas etapas de avaliação e teste de interfaces gráficas com o utilizador, com e sem o envolvimento dos utilizadores, bem como ferramentas de teste automático, em particular, as funcionalidades que estas devem suportar.

7.1.1 Sem os utilizadores

O envolvimento dos utilizadores ao longo de várias fases do processo de construção de uma interface gráfica é fundamental. No entanto, também é necessário avaliar uma GUI sem envolver os utilizadores na avaliação.

O tempo disponível dos utilizadores não é um recurso ilimitado, e portanto, ao testar uma interface com os utilizadores, esta deve estar o mais possível livre de problemas, para que os utilizadores não percam tempo encontrando erros triviais que podiam perfeitamente ter sido identificados antes. Acresce a isto, que é um sintoma de falta de profissionalismo apresentar uma aplicação com erros que nunca passariam numa fase de teste minimamente rigorosa, para os utilizadores testarem.

Uma segunda razão para se avaliar uma interface gráfica sem os utilizadores é que uma boa avaliação pode encontrar problemas que uma avaliação com apenas alguns utilizadores provavelmente deixaria passar em branco. Uma GUI para um computador

pessoal popular pode vir a ser usada por milhares de pessoas, mas ser testada com apenas algumas dezenas, antes que seja lançada uma versão *beta* do produto. Todo o utilizador terá um conjunto diferente de problemas e o teste não descobrirá problemas que os utilizadores do teste não experimentaram.

Por outro lado, o teste com os utilizadores, devido à sua duração limitada, também não descobrirá problemas que eventuais utilizadores venham a ter depois de adquirirem uma certa experiência com a interface.

Pode argumentar-se que a avaliação sem os utilizadores também não porá a descoberto todos os problemas que a interface possa ter. No entanto, utilizar os dois processos de certeza que aumenta as hipóteses de sucesso.

Podem referir-se três abordagens para a avaliação de GUIs sem os utilizadores. A primeira - *exercício cognitivo* - utiliza uma técnica centrada nas tarefas que o utilizador deseja realizar usando a interface. A segunda denomina-se *análise de acções* e permite prever o tempo que um utilizador experiente leva a realizar uma dada tarefa. Por último, a terceira abordagem consiste numa *avaliação heurística*, do tipo *check-list*, que permite endereçar uma vasta gama de problemas.

Exercício cognitivo

O exercício cognitivo (neste contexto) é um modo formal de imaginar os pensamentos e acções das pessoas, quando utilizam uma dada interface pela primeira vez.

Sucintamente, o exercício desenrola-se do seguinte modo: possuindo um protótipo ou descrição detalhada da interface, e conhecendo o utilizador-tipo da mesma, selecciona-se uma das tarefas que a interface deve suportar. Depois é necessário encontrar a motivação para cada uma das acções que o utilizador terá que executar de modo a realizar a tarefa.

Este exercício tem que ser realizado baseado no conhecimento geral do utilizador e nas solicitações e respostas fornecidas pela interface. Se não se conseguir encontrar a motivação que leve o utilizador a realizar uma dada acção, então ter-se-á localizado um problema na interface.

Análise de acções

A análise de acções é um processo de avaliação que força a olhar detalhadamente para a sequência de acções que o utilizador tem de realizar para completar uma tarefa na interface.

A análise de acções consiste em duas fases fundamentais. A primeira é decidir quais os passos físicos e mentais que o utilizador vai executar para completar uma ou mais tarefas na GUI. A segunda fase é analisar esses passos à procura de problemas.

Os problemas que este tipo de análise pode revelar são, por exemplo, que é necessário um número demasiado grande de passos para realizar uma tarefa simples, ou que demora muito tempo a realizar a tarefa ou ainda que existem demasiados aspectos a aprender na interface. Esta análise também pode revelar falhas na descrição da interface; i.e., coisas que a interface supostamente faria, mas que afinal não faz.

Avaliação heurística

A avaliação heurística consiste em avaliar a interface do ponto de vista da obediência às linhas mestras. Esta avaliação é do tipo *check-list*, na qual se vai analisando item a item a aplicação das linhas mestras. A avaliação necessita ser realizada por pessoas com um conhecimento considerável no âmbito da concepção de interfaces, para traduzir os princípios gerais para uma dada situação específica.

Atendendo a que um único analista não encontrará todos os problemas existentes na interface, e que cada analista porá a descoberto problemas diferentes, torna-se desejável que esta avaliação seja levada a cabo por várias pessoas. Cada pessoa deve fazer a análise individualmente e os resultados individuais daí decorrentes, devem ser combinados numa única lista.

7.1.2 Com os utilizadores

Não se pode afirmar se uma interface vai ser bem sucedida ou não, sem que esta seja utilizada de facto por pessoas. Deste modo, à medida que a construção da interface avança, mas antes de se apresentar o produto final, é necessário testar a GUI com os utilizadores. Isto significa ter pessoas a testar o sistema e observar o que acontece.

Para se realizar o teste, é necessário arranjar pessoas para testar a interface, tarefas para aquelas realizarem e uma versão do sistema para ser testada. São estas as necessidades aqui consideradas.

Escolha dos utilizadores para teste

O objectivo do teste é antecipar o que vai acontecer quando utilizadores "reais" começarem a usar o sistema. Logo, os melhores utilizadores para realizar o teste são pessoas que constituem uma amostragem representativa das pessoas que se espera ter como utilizadores futuros. Se os utilizadores de teste não forem representativos dos utilizadores futuros, aspectos cruciais da interface como, por exemplo, o vocabulário usado, podem evoluir numa direcção errada.

Por vezes existe dificuldade em testar a interface com utilizadores que representem de facto os utilizadores futuros, que se espera venham a usar a interface. Neste casos, deve testar-se a interface gráfica com utilizadores que de algum modo representem uma aproximação dos utilizadores pretendidos, tendo o cuidado de não deixar que reacções e comentários de pessoas que não efectivamente os utilizadores aos quais a interface é dirigida, conduzam a concepção da GUI.

Selecção das tarefas para teste

No teste é necessário dar alguma tarefa aos utilizadores para realizar, e tomar nota se o conseguem fazer ou não. Assim como os utilizadores de teste devem ser representativos dos utilizadores reais, também as tarefas devem reflectir as tarefas reais.

Por vezes pode ser necessário modificar as tarefas representativas do sistema para que sejam usadas numa fase de teste. As razões para este facto incluem a eventualidade de estas tarefas demorarem demasiado tempo (para teste) a realizar, ou o facto de que para a sua realização se assume um conhecimento prévio específico - que o utilizador de teste pode não possuir. Podem então simplificar-se as tarefas tendo, no entanto, o cuidado de evitar mudanças que possam descaracterizar a sua realização na GUI, podendo assim distorcer os resultados do teste.

Providenciar uma versão do sistema para teste

A máxima da construção iterativa de GUIs deve ser '*post factum nullum consilium*' - depois do facto consumado o conselho de nada vale. Assim, deve testar-se a interface nas fases iniciais do processo de implementação, quando ainda é possível fazer alterações sem incorrer em custos elevados, através da utilização de protótipos de teste. Estes são versões do sistema que podem não implementar a interface na totalidade, quer a nível de apresentação quer a nível de funcionalidade, mas que permitem mostrar as facetas chave aos utilizadores.

É de notar que protótipo é aqui utilizado no sentido lato do termo; i.e., um protótipo não necessita ser uma versão *beta* do sistema, podendo ser apenas uma simulação feita em papel. No entanto, casos há em que uma simulação em papel, embora de custos diminutos e perfeitamente passível de alterações, não será suficiente para testar a interface. Isto acontece essencialmente nos casos em que o sistema necessita interagir de muito perto com o utilizador, para poder ser aproximado por um simples simulacro. A ideia aqui é construir uma parte substancial do aspecto visual da interface, mas evitar a implementação de todas as funcionalidades, simulando. Isto pode ser feito recorrendo à técnica do *Feiticeiro de Oz* (Figura 7.1).

Esta técnica consiste em utilizar um intermediário humano como assistente, para executar as funções da interface ainda não implementadas e gerar as respostas que os utilizadores devem ver. Por outras palavras, um operador escondido, o Feiticeiro de Oz, pode interceptar os comandos enviados para a interface apenas parcialmente implementada, e efectuar uma interpretação adequada se os comandos ainda não estão implementados.

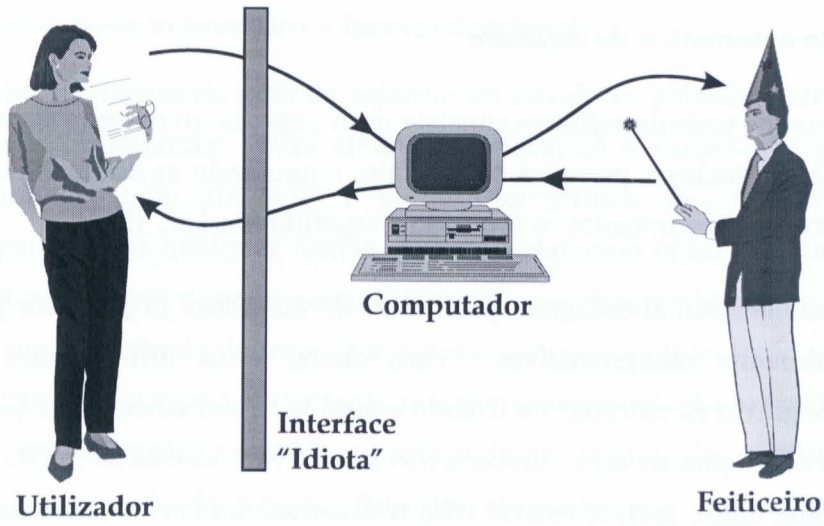


Figura 7.1 Onde vive o Feiticeiro.

Esta técnica foi utilizada com eficácia pela IBM para testar *designs* alternativos para um sistema de transcrição de fala, para o qual a componente de reconhecimento da fala ainda não estava implementada. A IBM construiu então um sistema protótipo, no qual a fala do utilizador de teste era direccionada para uma pessoa que a escrevia com rapidez num teclado, e a sua saída era redireccionada para o ecrã do utilizador de teste. Esta técnica pode ser adaptada para muitas situações nas quais a interface a ser testada necessita de responder a entradas imprevisíveis dos utilizadores, para as quais a funcionalidade subjacente ainda não está implementada.

O importante a realçar aqui é a necessidade de fornecer uma versão da interface para ser testada, sem incorrer em esforços substanciais de implementação e ainda a tempo de realizar alterações no sistema. Algum do esforço de implementação pode ser salvaguardado se o protótipo, no futuro, fizer parte da interface real. Os sistemas integrados de desenvolvimento de interfaces gráficas com o utilizador (Secção 6.3), permitem simular uma interface com funcionalidade mínima e depois fazer a integração à medida que os módulos funcionais se tornam disponíveis.

7.1.3 Teste automático de *software*

O método usual de teste de *software* consiste num engenheiro de teste sistematicamente tentando todas as teclas e pressões com o rato, e anotando as anomalias. Depois dos problemas terem sido corrigidos, o processo é repetido [Parker, 1995].

Infelizmente, esta abordagem, para além de fastidiosa (e portanto predisposta a erros), é raramente compreensível. Para obviar estas dificuldades, apareceu a abordagem de gravar as entradas do teclado e testá-las automaticamente para verificar a existência de condições de erro. Embora não seja novidade, esta forma de teste tornou-se recentemente ainda mais acessível com o desenvolvimento de ferramentas de teste cada vez mais sofisticadas.

Através da repetição das entradas do teclado inúmeras vezes, as falhas existentes numa aplicação podem ser capturadas e guardadas para um exame futuro. Isto é particularmente útil quando o *software* está a ser alvo de melhoramentos contínuos, onde uma modificação mínima numa parte do código pode afectar o comportamento em outro lado da aplicação (o que pode não ser perceptível para uma pessoa a testar fisicamente o sistema).

Para aplicações baseadas em caracteres o teste de *software* não é tremendamente difícil. Todas as entradas do teclado efectuadas pelo utilizador podem ser gravadas, e este registo pode ser usado sempre que seja necessário testar a aplicação.

No entanto, as interfaces gráficas representam um desafio uma vez que o *software* de captura necessita ser muito mais sofisticado: para além das entradas por teclado, é necessário capturar modificações gráficas, movimentos do rato, acções do tipo '*drag-and-drop*' e outras acções típicas realizadas nas GUIs.

As primeiras ferramentas de teste automático de *software* de GUIs prioritariamente gravavam e comparavam *bitmaps* dos ecrãs da interface, comparando-os antes e depois de uma sequência de teste. Desde então outras funcionalidades foram acrescentadas.

Ferramentas de teste automático - facetas desejáveis

As ferramentas modernas de teste de *software* de interfaces gráficas oferecem diversas potencialidades importantes. Para além de fornecerem a capacidade de registar as teclas pressionadas pelo utilizador e permitirem utilizar esse registo sempre que necessário (capturando qualquer desvio do comportamento esperado), um sistema de teste de *software* deve possuir uma linguagem de desenvolvimento que permita desenvolver uma sequência de teste, sem ser necessária uma interacção efectiva com a aplicação. Esta capacidade é importante quando a representação externa da interface pode sofrer modificações contínuas, tornando a rotina de captura de teclas pressionadas uma perda de tempo. Este tipo de linguagem também permite testar diversas condições que podem não ser directamente acessíveis através da representação externa da interface gráfica.

A capacidade de simular múltiplos utilizadores também é importante, dado que o comportamento da aplicação pode sofrer alterações à medida que mais utilizadores acedem a esta. É de realçar que embora esta faceta seja muito importante para algumas aplicações, como sejam programas cliente/servidor, para outras aplicações pode ser pouco relevante.

A capacidade de verificar imagens do ecrã com uma imagem de referência em qualquer instante, através da comparação de *bitmaps* ou de *strings*, também é importante porque o aspecto da aplicação não se deve alterar com a sua execução repetida. Contudo, como as janelas da GUI no ecrã podem mover-se umas relativamente às outras dependendo das condições do sistema, muitos sistemas de teste com capacidade de *bitmaps* permitem comparações de posições absolutas e relativas.

De suma importância para um sistema de teste de *software* é a possibilidade de obter análises precisas depois de um teste ter sido realizado diversas vezes. Quanto mais informativas e detalhadas forem as análises de teste, tanto mais úteis serão para a avaliação da interface.

Finalmente, o teste automático pode fornecer um benefício lateral importante: um conjunto de métricas mostrando os tempos de resposta do programa e o comportamento do sistema. Esta faceta pode indicar quais os aspectos que mais necessitam de optimização, ou pode ser usada como diagnóstico.

Como conclusão, as ferramentas de teste automático de *software* de interfaces gráficas com o utilizador são bastante dispendiosas. Por vezes, pode ser difícil justificar o custo deste tipo de ferramentas a menos que o tempo poupado relativamente ao teste manual seja cuidadosamente considerado. No entanto, o teste automático poupa tempo de uma forma substancial e conduz a aplicações mais sólidas porque sequências repetitivas de teste podem executar qualquer acção que uma dada aplicação pode realizar.

7.2 Documentação

A ideia central de uma interface gráfica com o utilizador é facilitar a interacção com o computador. Isto implica que o utilizador necessita compreender a interface e saber como utilizá-la. Para que isto aconteça, constitui frequentemente um requisito fornecer ao utilizador documentação que torne a interface inteligível. O ponto de vista aqui subjacente é que a documentação deve ser considerada uma parte integrante da interface global com o utilizador, uma vez que este usa a documentação como um recurso para realizar as tarefas pretendidas no sistema.

A *documentação* (do utilizador) é aqui entendida como englobando qualquer forma de explanação da interface. Pode incluir manuais do utilizador, ajuda *on-line*, programas de treino e apoio ao utilizador. Todas estas componentes ajudam o utilizador a compreender a aplicação e a estabelecer um modelo conceptual apropriado.

Mas até que ponto a documentação é importante? Não é verdadeira a teoria que "ninguém lê os manuais"? [Rieman, 1993] argumenta que muito pelo contrário: num estudo realizado ficou demonstrado que utilizadores de todo o espectro de aplicações, recorrem a fontes de informação externas quando não sabem como realizar uma dada tarefa num computador. A fonte à qual recorrem varia: pode ser um manual do utilizador, o administrador local do sistema, ou uma linha telefónica de apoio ao utilizador.

A realidade é que os utilizadores quase nunca sabem tudo o que necessitam acerca de uma aplicação, e quando precisam de realizar alguma tarefa nova, recorrem às fontes de informação disponíveis para obter a ajuda necessária.

Contudo, não se deve considerar o suporte externo como um meio de compensar uma má interface *on-line*. Se é verdade que os utilizadores irão ler o manual ou pedir ajuda quando não sabem como usar a interface, deve ser notado que não o desejarão fazer se o puderem evitar, até porque constitui um uso pouco produtivo do seu tempo.

Deve lutar-se o mais possível por construir uma interface "pronta a utilizar", pelo menos a nível das funcionalidades básicas. A ajuda externa deve existir prioritariamente como um apoio a ser usado pelos utilizadores que se desviam dos caminhos comuns, e como um recurso para os utilizadores que tentam descobrir facetas mais avançadas que podem ter sido intencionalmente escondidas dos utilizadores principiantes (como *macros*, por exemplo).

A seguir referem-se os tipos de documentação mais correntes, segundo uma divisão tradicional.

7.2.1 Manuais

Para muitos utilizadores, o manual é a fonte de informação, fora da interface *on-line*, mais acessível. Para outros utilizadores, o primeiro sítio onde pedir ajuda é ao administrador local do sistema, ou a outro utilizador mais experiente - mas estes utilizadores adquiriram a maior parte do seu conhecimento através dos manuais. Em geral, um bom manual constitui a extensão mais importante de uma interface propriamente dita.

Algumas das regras básicas que devem ser seguidas aquando da escrita de um manual do utilizador, para que este seja eficaz, são:

- *Ser sucinto*: A brevidade é um ponto fulcral da escrita de um manual do utilizador. Nos manuais demasiado longos, o utilizador frequentemente não consegue encontrar a informação que necessita, porque esta está disseminada numa grande quantidade de texto irrelevante.

- ▷ *Fazer uma apresentação geral e uma sequência de passos para cada tarefa*: A apresentação geral deve ser simples, e deve demonstrar-se a execução sequencial de uma tarefa simples mas real.
- ▷ *Usar a linguagem do utilizador*: Deve ser tido em conta que se está a escrever um manual *do utilizador* e não uma descrição técnica da interface para peritos na concepção de GUIs.
- ▷ *Usar figuras parcimoniosamente*: À excepção de imagens da própria interface no ecrã, as figuras usualmente não acrescentam muito à compreensão de um manual.
- ▷ *A apresentação deve ser atractiva e funcional.*

7.2.2 Ajuda *on-line*

Quando se tenta fornecer grandes quantidades de informação a ajuda *on-line* revela-se em geral tão (ou até menos) eficaz que os tradicionais livros e manuais, pese embora as funcionalidades de hipertexto fornecidas por estes sistemas, como a procura de palavras e a ligação a tópicos relacionados. Para sistemas de ajuda *on-line* básicos a regra a seguir é: se é necessário apresentar mais do que uma descrição breve de um aspecto da interface não se deve recorrer apenas à ajuda *on-line*.

A ineficácia de textos longos na ajuda *on-line* é provavelmente o resultado de uma conjugação de diversos factores: texto num ecrã de computador é usualmente menos legível que impresso em papel; é mais fácil as pessoas perderem-se a "navegar" nos ecrãs de texto do que nas páginas de um livro; os ecrãs não são actualizados tão rapidamente quanto o virar de páginas; a janela de ajuda muitas vezes sobrepõe-se à parte da interface acerca da qual o utilizador tem dúvidas; e as pessoas não praticaram ler texto no ecrã durante toda a sua vida como fizeram com texto impresso. O efeito combinado destes factores contrapõe todas as vantagens associadas com a ajuda no ecrã.

Mas se a ajuda *on-line* não será o sítio indicado para colocar um manual inteiro, possui outros usos potenciais. A ajuda *on-line* é o local por excelência para colocar factos breves que os utilizadores necessitam frequentemente ou que é provável que

esqueçam. Um exemplo demonstrativo disto é a equivalência de teclas para os itens dos menus (*hot-keys*). Em muitas interfaces estas equivalências são mostradas nos próprios menus, o que também constitui uma forma de ajuda *on-line*.

7.2.3 Treino

Aqui, o termo treino engloba quer cursos, quer tutoriais, *on-line* ou não, que o utilizador pode fazer sozinho. Os cursos e tutoriais devem abordar as tarefas representativas da interface gráfica, aliás como o manual do utilizador. Ao contrário do manual, contudo, os programas de treino devem ser estruturados de modo a forçarem o utilizador a envolver-se activamente no funcionamento do sistema. Uma versão minimal do manual básico pode servir como peça central de um programa de treino.

A ideia do *manual minimal*, sugerida e testada na IBM, vai mais longe no conceito de brevidade do manual básico; o manual minimal é intencionalmente incompleto. Neste manual a falta de detalhes é cuidadosamente associada a tarefas com objectivos claros, forçando o utilizador a investigar a interface *on-line* para descobrir como é que funciona. Diversos estudos, entre os quais [Carroll, 1990], demonstraram que esta abordagem minimalista resulta em tempos de aprendizagem consideravelmente mais reduzidos que o treino tradicional completamente orientado.

7.2.4 Apoio ao utilizador

O apoio ao utilizador traduz-se, na generalidade dos casos, em linhas telefónicas e/ou de correio electrónico, operadas por técnicos especializados que fornecem ajuda aos utilizadores. A comunicação entre um técnico especializado e um utilizador menos experiente é, no caso geral, problemática, e ainda se torna mais difícil quando é efectuada via telefone ou correio electrónico, em vez de pessoalmente.

No sentido de minimizar os problemas de comunicação entre os utilizadores e os técnicos algumas regras básicas devem ser observadas, de entre as quais a mais importante é, mais uma vez, *usar a linguagem do utilizador.*

7.3 Conclusões

O teste é parte integrante do processo de desenvolvimento de aplicações computacionais. Neste capítulo, foi justificada a necessidade de testar as GUIs com e sem o envolvimento dos utilizadores. Diversas estratégias de teste sem os utilizadores foram descritas, e foram apresentadas os requisitos colocados quando se realizam testes com os utilizadores. Adicionalmente, referiram-se os conceitos subjacentes às ferramentas de teste automático de *software*, bem como os requisitos adicionais que as interfaces gráficas com o utilizador colocam a este tipo de ferramentas. Neste contexto, descreveram-se as funcionalidades que as ferramentas de teste automático de GUIs devem possuir.

Por outro lado, uma interface gráfica com o utilizador só é um meio efectivamente útil de aceder às potencialidades do sistema computacional, se o utilizador conseguir de facto usar a interface de um modo eficaz. Para atingir este objectivo deve ser fornecida ajuda ao utilizador para a compreensão da interface. Esta ajuda pode revestir-se de diversas formas: documentação impressa, tutoriais (impressos ou *on-line*), cursos de aprendizagem da interface e ainda linhas de prestação de ajuda ao utilizador (tipicamente, via correio electrónico ou telefone).

Capítulo 8

Conclusões

8.1 Conclusões

Há alguns anos atrás, o estilo de interacção com os sistemas computacionais requeria que os utilizadores conhecessem a linguagem de interacção, soubessem o que deviam fazer, e depois escrevessem os comandos cuidadosamente escolhidos no computador. Os utilizadores tinham que se lembrar dos comandos e da sua sintaxe. Este estilo simbólico deu lugar a um estilo mais visual e imediato de ver e apontar, nas interfaces gráficas com o utilizador, onde as acções têm um efeito imediato visível.

As interfaces gráficas com o utilizador aumentaram substancialmente a rapidez de aprendizagem e do uso exploratório dos sistemas computacionais. O sucesso do estilo de interacção visual das interfaces gráficas com o utilizador deve-se primordialmente ao facto de a carga cognitiva requerida ao utilizador ser reduzida.

A concepção de interfaces gráficas com o utilizador requer, em primeiro lugar, um passo inventivo ou criativo, seguido de uma filtragem - de modo a que a invenção seja polida e transformada numa interface gráfica exequível e de sucesso. As interfaces gráficas beneficiam mais directamente os seus utilizadores imediatos, mas estes benefícios podem ser rapidamente esbanjados se as interfaces gráficas não forem concebidas de uma forma eficaz. É então necessário tentar ser sistemático na concepção de interfaces gráficas com o utilizador.

Quase todos os métodos existentes para o desenvolvimento de sistemas computacionais pressupõem que as pessoas sabem de facto o que querem, e frequentemente, que aquilo que querem é imutável. Se o programador souber exactamente aquilo que é suposto programar, existem diversos métodos que lhe permitem produzir um programa, ou saber que lhe é impossível progredir até à obtenção de um sistema que funcione - por causa da inconsistência ou complexidade do problema ou devido a limitações nos recursos (incluindo o seu próprio cérebro) disponíveis. No entanto, a concepção de interfaces gráficas, até pelo pendor criativo que possui, é frequentemente encarada de um modo pouco sistemático.

A parte principal deste trabalho tentou fazer com que a concepção de interfaces gráficas com o utilizador, quer para aplicações genéricas, quer para ambientes de fabrico, estivesse mais aberta a uma análise deliberada e avaliação científica. Nesta perspectiva, diversos conceitos e técnicas subjacentes à concepção de interfaces gráficas com o utilizador, foram apresentados e explorados.

O uso de metáforas nas interfaces gráficas foi amplamente discutido e demonstrou-se da necessidade de o utilizador estar familiarizado com o ambiente no qual as metáforas se baseiam, de modo a poder tirar partido de conhecimentos prévios para a compreensão da interface. As principais características de uma interface gráfica bem concebida foram apresentadas, bem como o porquê da sua importância. Atendendo a que o universo de putativos utilizadores de uma interface gráfica é variado, apresentaram-se algumas características segundo as quais as diferenças entre os utilizadores devem ser equacionadas na concepção das interfaces.

Nos ambientes de fabrico os requisitos que se colocam às interfaces gráficas podem diferir dos requisitos para um ambiente mais genérico. Neste sentido, as

interfaces gráficas com o utilizador em ambientes de fabrico foram analisadas sob diversos prismas: metáforas, utilizador-tipo, características da interface, requisitos e visualização da informação, e alguns constrangimentos adicionais.

A necessidade de sistematizar a concepção de interfaces gráficas conduziu à descrição das fases que tipicamente constituem o processo de desenvolvimento. A necessidade de manter a consistência entre aplicações e a conveniência de usar soluções já testadas, entre outras razões, conduziu à apresentação de linhas mestras a seguir na concepção de interfaces gráficas com o utilizador. Também se discutiu da sua relevância para o caso particular de interfaces gráficas em ambientes de fabrico.

Foi considerada a utilização de três técnicas no desenvolvimento de interfaces gráficas. Uma técnica particular ao desenvolvimento de sistemas interactivos, o modelo MVC (Model-View-Controller) - que permite separar a apresentação da parte funcional da interface, e duas técnicas genéricas no desenvolvimento de *software*: a tecnologia cliente/servidor e a tecnologia orientada por objectos. Para cada uma das técnicas supracitadas apresentaram-se os seus fundamentos e as vantagens decorrentes do seu uso no desenvolvimento de interfaces gráficas.

Explorou-se a utilização de ferramentas de suporte ao desenvolvimento de interfaces gráficas. Foram caracterizados diversos tipos de ferramentas de suporte, em particular os sistemas integrados de desenvolvimento. Adicionalmente, analisaram-se exemplos de ferramentas de suporte ao desenvolvimento de interfaces gráficas para os ambientes baseados em janelas mais divulgados; destes últimos, o sistema X-Windows mereceu uma descrição mais pormenorizada.

Dois pontos chave na concepção de interfaces gráficas com o utilizador são o teste - para aferir a validade da interface, e a documentação fornecida ao utilizador - para melhor lhe permitir apreender o comportamento da interface e tirar o máximo partido da sua utilização. Estes dois temas também foram abordados. No teste descreveram-se técnicas para testar a interface sem envolver os utilizadores, exploraram-se factores concernentes ao teste com os utilizadores e analisaram-se as potencialidades das ferramentas de teste automático. Relativamente à documentação, referiram-se as vias correntes de facultar informação e ajuda ao utilizador e foram fornecidas estratégias para garantir da sua eficácia.

8.2 Perspectivas de trabalho futuro

Uma área não abordada de uma forma particular nesta tese foi a formação baseada em computador - CBT (Computer Based Training), utilizando interfaces gráficas com o utilizador. Esta componente possui particular interesse nos ambientes de fabrico e pode ser explorada num trabalho futuro a realizar no âmbito de interfaces gráficas com o utilizador vocacionadas para ambientes de fabrico.

De facto, neste trabalho, e na perspectiva dos ambientes de fabrico, abordaram-se as interfaces gráficas como um modo de o utilizador interagir com determinadas aplicações características destes ambientes, nomeadamente aplicações de controlo e monitorização da produção. No entanto, as interfaces gráficas podem ainda providenciar formação relativa às próprias tarefas que os operadores devem realizar. Esta faceta pode ser utilizada com sucesso para auxiliar os operadores a realizar tarefas com uma componente manual elevada, e que se alterem com frequência.

A interface gráfica, para além de permitir ao operador dar ordens ou introduzir dados relativos a operações que realizou, pode facultar informação preciosa sobre, por exemplo, como efectuar uma operação nova, ou como melhorar a realização de uma operação na qual aquele possui um rendimento baixo. Deste modo, poder-se-ia aumentar o rendimento do operador, e evitava-se a necessidade de interromper o trabalho para receber cursos de formação. A concepção de uma interface gráfica com estas características teria que ser realizada em colaboração com equipas de tempos e métodos e controlo da produção.

Com esta componente acrescida de formação através da interface gráfica, o que está em causa é não só facultar ao operador um meio mais fácil de interagir com aplicações computacionais, como foi tratado neste trabalho, mas adicionalmente providenciar-lhe formação sobre as próprias operações que lhe estão atribuídas. Naturalmente, nesta perspectiva, outros requisitos se colocam relativamente ao modo como veicular esta informação adicional ao operador, de uma forma compreensível e eficaz. O operador passa a utilizar a interface gráfica como um meio de receber formação sobre o seu próprio trabalho, tendo em vista melhorar a capacidade de realização deste.

A área de formação baseada em computador está em rápida expansão, sendo já utilizada com sucesso em muitos domínios de aprendizagem. Os factores relativos à sua aplicação em ambientes de fabrico, poderiam ser estudados num possível trabalho futuro que desse continuidade ao trabalho já realizado na área das interfaces gráficas com o utilizador.

Referências

Asente P. J. e Swick R. R. (1990). *X Window System Toolkit. The Complete Programmer's Guide and Specification*. Burlington, Massachusetts, Digital Press.

Berlage T. (1991). *OSF/Motif. Concepts and Programming*. Reading, Massachusetts, Addison-Wesley.

Blomberg A. L. e Henderson A. (1990). Reflections on Participatory Design. *Conference on Human Factors in Computer Systems*. CHI '90, 353-359. New York, ACM.

Carroll J. M. (1990). *The Nurnberg Funnel: Designing Minimalist Instruction for Practical Computer Skill*. Cambridge, Massachusetts, MIT Press.

Coad P. e Yourdon E. (1991). *Object-Oriented Analysis*. Englewood Cliffs, New Jersey, Yourdon Press, Prentice Hall.

Coad P. e Yourdon E. (1991). *Object-Oriented Design*. Englewood Cliffs, New Jersey, Yourdon Press, Prentice Hall.

Cox C. e Walker D. (1993). *User Interface Design*. Englewood Cliffs, New Jersey, Prentice Hall.

Dichter C. (1993). One for All ... *Unix Review*, 65-73.

Erickson T. D. (1990). *Working with Interface Metaphores. The Art of Human-Computer Interface Design*. Reading, Massachusetts, Addison-Wesley.

Fisch E. A. (1993). Understanding and Improving the User Interface Design Process. *ACM SIGSOFT*, **18** (2), 27-29.

Foley J., van Dam A., Feiner S. e Hughes J. (1990*). Computer Graphics. Principles and Practice*. Reading, Massachusetts, Addison-Wesley.

Grudin J. (1993). INTERFACE. An Evolving Concept. *Communications of the ACM*, **36** (4), 110-119.

Hutt A. T. F. (Ed.) (1994). *Object Analysis and Design. Description of Methods*. New York, John Wiley & Sons.

IBM (1989). *SAA Common User Access. Advanced Interface Design Guide*. SC26-4582-0, IBM Corporation.

IBM (1989). *SAA Common User Access. Basic Interface Design Guide*. SC26-4583-0, IBM Corporation.

Johnson C. e Harrison M. (1994). Software Engineering for Human Computer Interaction. *SIGCHI Bulletin*, **26** (2), 46-47.

Johnson E. F. e Reichard K. (1991). *power programming ... MOTIF*. Portland, Oregon, MIS Press.

Johnson H. (1994). Relationship Between User Models in HCI and AI. *IEE Proc.-Comput. Digit. Tech.* **141** (2), 99-103.

Jones O. (1989). *Introduction to the X Window System*. Englewood Cliffs, New Jersey, Prentice Hall.

Kobara S. (1991). *Visual Design with OSF/Motif*. Reading, Massachusetts, Addison-Wesley.

Langford D. e Jones C. (1994). The Kitchen Interface - a Lateral Approach to GUI. *SIGCHI Bulletin*, **26** (2), 41-45.

Lee S. J. e Shunk D. L. (1992). The Development of an Intelligent Interface for Knowledge Workers in Computer Integrated Manufacturing. *IFAC-INCOM '92*, 221-229.

Lewis C. e Rieman J. (1993). *Task-Centered User Interface Design. A Practical Introduction*. Authors electronic edition.

Marcus A. (1993). Human Communications Issues in Advanced UIs. *Communications of the ACM*, **36** (4),101-109.

Martin C. D. (1993). The Myth of Awesome Thinking Machine. *Communications of the ACM*, **36** (4), 120-133.

Mikes S. (1992). *X Window System Program Design and Development*. Reading, Massachusetts, Addison-Wesley.

Morse A. e Reynolds G. (1993). Overcoming Current Growth Limits in UI Development. *Communications of the ACM*, **36** (4), 73-81.

Murray B. S. e Edmonds E. A. (1994). Flexibility in Interface Development. *IEE Proc.- Comput. Digit. Tech.*, **141** (2), 93-98.

Myers B. (1993). Why are Human-Computer Interfaces Difficult to Design and Implement? *Technical Report CS-93-183*. Carnegie Mellon University.

Myers B. A. e Rosson M. B. (1992). Survey on User Interface Programming. *Human Factors in Computing Systems*, Proceedings SIGCHI '92, 195-202.

Myers B. A., Giuse D. A., Mickish A. e Kosbie D. S. (1994). Making Structured Graphics and Constraints Practical for Large-Scale Applications. *Technical Report CS-94-150*. Carnegie Mellon University.

Nielsen J. (1993). Noncommand User Interfaces. *Communications of the ACM*, **36** (4), 83-99.

Odell J. (1991). Managing Object Complexity, Part I: Abstraction and Generalization. *JOOP*, **5** (5), 19-22.

Oliveira R. e Carvalho A. (1993). Arquitectura para um Controlador de Célula. *1º Encontro Nacional do Colégio de Engenharia Electrotécnica*, Lisboa.

Parker T. (1995). Automated Software Testing. *Unix Review*, 49-56.

Petre M. e Price B. A. (1992). Why Computer Interfaces Are Not Like Paintings: the User as a Deliberate Reader. *The St. Petersburgh International Conference on Human-Computer Interaction*, 217-223.

Petzold C. (1992). *Programming Windows 3.1*. Microsoft Press.

Porter M. (1990). *The Competitive Advantage of Nations*. New York, Free Press.

Reed P. (1994). ANSI/HFES Software User Interface Standardization: Critical Issues. *SIGCHI Bulletin*, **26** (2), 12-15.

Rembold U., Nnaji B. O. e Storr A. (1993). *Computer Integrated Manufacturing and Engineering*. Reading, Massachusetts, Addison-Wesley.

Renaud P. E. (1993). *Introduction to Client/Server Systems. A Practical Guide for Systems Professionals*. New York, John Wiley & Sons.

Rentsch T. (1982). Object Oriented Programming. *SIGPLAN Notices*, **17** (9), 51.

Rettig M. (1993). Cooperative Software. *Communications of the ACM*, **36** (4), 23-28.

Rieman J. (1993). The Diary Study: A Workplace-Oriented Tool to Guide Laboratory Studies. *Proc. InterCHI '93. Conference on Human Factors in Computer Systems*. New York, ACM Press.

Santos C. e Carvalho A. (1994). Interfaces Gráficas com o Utilizador para Ambientes de Produção. *1º Encontro Nacional do Colégio de Engenharia Electrotécnica*, Lisboa.

Scheifler R. W. e Gettys J. (1992). *X Window System. The Complete Reference to Xlib, X Protocol, ICCCM and XFLD*. Burlington, Massachusetts, Digital Press.

Sharble R. C. e Cohen S. S. (1993). The Object-Oriented Brewery: A Comparison of Two Object-Oriented Development Methods. *ACM SIGSOFT*, **18** (2), 60-73.

Shneiderman B. (1993). Beyond intelligent machines: Just do it! *IEEE Software*, 100-103.

Smith S. L. e Mosier J. N. (1984). Design Guidelines for User-System Interface Software. *Report ESD-TR-84-190*, Mitre Corporation, Bedford, Massachusetts.

Snyder A. (1993). The Essence of Objects: Concepts and Terms. *IEEE Software*, 31-42.

Spurr K. e Layzell P. (Ed.) (1992). *CASE. Current practice. Future prospects*. New York, John Wiley & Sons.

Tesler L. (1981). The Smalltalk Environment. *BYTE*, August 1981, 90-147.

Thimbleby H. (1990). *User Interface Design*. Reading, Massachusetts, Addison-Wesley.

Thimbleby H. (1994). Formulating Usability. *SIGCHI Bulletin*, **26** (2), 59-64.

Wadlow M. G. (1994). The Zen of Interface Design. *SIGCHI Bulletin*, **26** (2), 16-17.

Whiteside J., Jones S., Levy P. e Wixon D. (1985). User Performance with Command, Menu and Iconic Interfaces. *Proceedings of CHI '85. Conference on Human Factors in Computing Systems II*. New York, ACM Press.

Young D. (1992). *Object-Oriented Programming with C++ and OSF/Motif*. Englewood Cliffs, New Jersey, Prentice Hall.

Acrónimos

AIC	AIX Interface Composer
AT	Advanced Technology
ATM	Automatic Teller Machine
CBT	Computer Based Training
CNMA	Communications Network for Manufacturing Applications
CPU	Central Processing Unit
CUA	Common User Access
DDE	Dynamic Data Exchange
DLL	Dynamic Link Library
GIGO	Garbage In - Garbage Out
GUI	Graphical User Interface
HCI	Human-Computer Interaction
I/O	Input/Output
IB	Interface Builder
ICCCM	Inter-Client Communications Conventions Manual
IPC	InterProcess Communication
KI	Kitchen Interface

LAN	Local Area Network
MVC	Model-View-Controller
OF	Ordem de Fabrico
OLE	Object Linking and Embedding
OOA	Object-Oriented Analysis
OOD	Object-Oriented Design
OOP	Object-Oriented Programming
OOSD	Object-Oriented Software Design
OSF	Open Software Foundation
PC	Personal Computer
RPC	Remote Procedure Call
STM	Short-Term Memory
TCP	Transport Control Protocol
UIDE	User Interface Development Environment
UIMS	User Interface Management System
WYSIWYG	What You See Is What You Get
XCMD	eXternal CoMmanD
XFCN	eXternal FunCtioN

FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO
BIBLIOTECA
0000053249