Programação e Gestão de Interfaces em Visual C++

Andreia Melo

Resumo – **O interface gráfico de uma aplicação de software tem hoje em dia cada vez mais importância. Uma aplicação é tanto mais atractiva quanto melhor for a sua interface, eficiência e facilidade de utilização. Neste artigo pretende-se fornecer os princípios básicos e os primeiros passos para construir uma aplicação de software numa linguagem orientada por objectos como é o Visual C++.**

Abstract – **The graphical interface of a software application is a factor of increasing importance nowadays. An application is much more attractive and accepted as good as it is its interface, efficiency and easy to use. This paper is used to provide the basic principles and the first steps for building a software application using an object oriented language such as Visual C++.**

I. INTRODUÇÃO

Nos dias de hoje, qualquer tipo de aplicação executável em computador possui um interface gráfico. A utilização do computador é um processo que já se tornou banal e essencial e que faz parte da vida quotidiana de pessoas de todas as faixas etárias que utilizam programas para os mais diversos fins. A nível profissional, onde acedem a bases de dados em computadores terminais e efectuam operações sobre elas, em actividades associadas ao lazer, como por exemplo para aceder aos mais variados tipos de informação disponibilizada na Internet e jogos de computador. Todos estes gestos assumem uma interacção entre a pessoa e o computador e devem parecer o mais intuitivos possível, ou seja, qualquer tipo de pessoa, independentemente das suas características, ao utilizar o computador para um determinado fim não deve dispender o seu tempo a tentar compreender como pode interagir com uma dada aplicação, como por exemplo à procura dos comandos de que necessita, mas concentrar-se apenas na operação que deseja efectuar. Neste artigo vamos considerar aquele grupo de pessoas que pretendem construir as aplicações ou ferramentas de trabalho e a forma como o fazem, isto é, como devem utilizar os recursos de interface já existentes para criar uma aplicação executável, fácil de utilizar, eficiente e por isso atractiva. A linguagem de programação que aqui será abordada é o Visual C++, embora a linguagem Visual Basic [1] seja a mais popular para este efeito.

Este artigo divide-se em sete secções A primeira consiste nesta introdução. A segunda secção mostra quais os primeiros passos na construção de uma aplicação utilizando o Visual C++. Esta linguagem dispõe de uma filosofia de construção de um programa, a que foi chamada arquitectura documento/vista que será detalhada na terceira secção. A quarta secção descreve os recursos de interface com o utilizador e forma como podem ser utilizados. Na quinta secção é mostrado como se podem criar objectos personalizados, visualizá-los no ecrã e interagir com eles. Os diferentes tipos de aplicações onde estes princípios de interface podem ser utilizados são apresentados na sexta secção. A secção sete finaliza este artigo com algumas conclusões.

II. CONSTRUÇÃO DE UMA APLICAÇÃO

A primeira operação a efectuar quando pretendemos construir uma aplicação em Visual C++ no ambiente de desenvolvimento Microsoft Visual Studio 6.0 é criar um novo projecto. Neste passo surgem-nos várias opções mas a que aqui será seguida é a aplicação executável que utiliza a biblioteca MFC (*Microsoft Foundation Classes*), chamada **MFC AppWizard(exe)** e é um processo guiado através de um assistente. A biblioteca de classes que é aqui utilizada dispõe de todas as funções necessárias à construção de um interface. Para uma melhor familiarização com esta biblioteca sugere-se a consulta de [2] ou versões mais actualizadas.

A. Tipos básicos de interfaces de aplicações

Existem três tipos de interfaces de aplicações que podem ser criadas: SDI – *Single Document Interface*, MDI – *Multiple Document Interface* e *Dialog based* (baseada apenas numa caixa de diálogo). A primeira utiliza a janela principal da aplicação para mostrar o documento e só apenas um documento pode ser visualizado de cada vez. No próximo passo do assistente é dado a escolher se pretendemos suporte para bases de dados e se tal for desejado, deveremos indicar a fonte dos dados que irão ser utilizados.

Ao nível dos recursos de interface com o utilizador são disponibilizadas pelo assistente o menu, a barra de ferramentas, a barra de estado, o suporte para impressão e visualização prévia do documento a imprimir, o suporte de ajuda (*Context-sensitive Help*) e os controlos 3D. O aspecto das barras de ferramentas, *Normal* e *Internet Explorer ReBars* são também opções disponíveis. Aconselha-se a utilização da segunda pois actualmente todas as aplicações de trabalho utilizam este modelo de apresentação, isto é, parecem não existir os botões, que

são realçados apenas quando o utilizador coloca o apontador do rato sobre o objecto desenhado sobre a barra. Este tipo de barra de ferramentas permite além disto a colocação de outros tipos de controlos, tais como listas (*combo boxes*), *menus popup*, etc.

(a) (b) (c)

Figura 1. Diferentes tipos de interfaces básicas de uma aplicação: (a) SDI, (b)MDI e (c) caixa de diálogo.

B. As classes principais e a sua função

Terminando a função do assistente para a criação da aplicação executável, assumindo que foi escolhido o tipo de interface SDI ou MDI, verifica-se que foram construídas automaticamente cinco classes. *CAboutDlg*, que é utilizada para invocar e visualizar uma caixa de diálogo com informação específica acerca da aplicação, como por exemplo o seu nome, versão e autor(es). A classe *CExemploApp* é derivada da classe *CWinApp* que é utilizada como classe base sempre que se cria uma aplicação em Windows. Esta fornece as funções para inicialização do objecto que corresponde à própria aplicação, e cada instância desta, e para executá-la. Sempre que se utilize a biblioteca MFC só pode existir um objecto derivado da classe *CWinApp*. O objecto derivado desta classe, isto é, do tipo *CExemploApp*, deve ser declarado globalmente pois é construído ao mesmo tempo que outros objectos globais e já deverá existir quando o Windows invoca a função *WinMain*, fornecida pela MFC. Deve-se também redefinir a função *InitInstance* da classe *CWinApp* dentro da nossa classe *CExemploApp*. É aqui que se define a utilização dos controlos 3D (pela invocação da função *Enable3dControls*), que se define o template do documento que irá ser utilizado nesta aplicação e onde são invocadas as funções *ShowWindow* e *UpdateWindow* para visualização da janela principal.

III. CRIAÇÃO DE TEMPLATES DE DOCUMENTOS

Quando uma aplicação é executada em Windows o utilizador interage com documentos apresentados em *frames*. A *frame* de uma janela com um documento possui basicamente duas componentes: a *frame* propriamente dita, isto é, a delimitação da janela e o seu conteúdo. São estas *frames* que podem ser do tipo SDI ou MDI. No primeiro caso existe apenas uma enquanto no segundo existe um conjunto de janelas filhas que apresentam o conteúdo de diferentes documentos.

Os templates de documentos são utilizados para, durante a criação de um novo documento, através de um comando *New* ou *Open* do menu *File*, criar uma nova *frame* para visualizar esse documento. O construtor do template especifica o tipo do documento, a janela e a vista pois estes são os argumentos a ele passados. No caso de uma aplicação de interface SDI o código apresentado na fig. 2 ilustra a criação do template, invocando a função *AddDocTemplate* que neste caso é invocada apenas uma vez pois existe só um tipo de documento.

```
AddDocTemplate(new CSingleDocTemplate(
              IDR_MAINFRAME,
              RUNTIME_CLASS(CExemploDoc),
              RUNTIME_CLASS(CMainFrame),
              RUNTIME_CLASS(CExemploView));
```

Figura 2. Criação do template do documento para uma aplicação de interface SDI.

Os argumentos passados ao construtor consistem num identificador de recursos (*resource ID*) associado ao tipo de menu e aceleradores utilizados na aplicação, e a três invocações da macro RUNTIME_CLASS que devolvem um objecto *CRuntimeClass* do tipo especificado no seu argumento. Os três objectos *CRuntimeClass* passados ao construtor do template do documento fornecem a informação necessária à criação de novos objectos de classes que são especificadas durante o processo de criação do documento.

Na fig. 3 é apresentado o código de criação de um template de documento no caso das aplicações MDI. Neste tipo de interface a função *AddDocTemplate* tem de ser invocada para cada tipo de documento criado, como por exemplo, documentos de texto, de gráficos, etc.

```
AddDocTemplate ( new CMultiDocTemplate(
           IDR_SCRIBTYPE,
           RUNTIME_CLASS(CExemploDoc),
           RUNTIME_CLASS(CChildFrame),
           RUNTIME_CLASS(CExemploView)));
```

Figura 3. Criação do template do documento para uma aplicação de interface MDI.

O sistema operativo Windows é responsável pela maior parte da interacção do utilizador com a *frame*, incluindo as funções de deslocamento, redimensionamento, fecho, minimização e maximização. O trabalho do projectista reduz-se à gestão do conteúdo da *frame*.

IV. ARQUITECTURA DOCUMENTO/VISTA

Por defeito as aplicações que se constroem baseadas na biblioteca MFC utilizam um modelo de programação que separa os dados da maneira como estes são apresentados ao utilizador, separando também a forma como este interage com eles. Neste modelo, o objecto que corresponde ao **documento** MFC lê e escreve dados de forma persistente. No entanto, existe um novo objecto a que foi chamada **vista**, que manipula a visualização dos dados, como por exemplo o desenho dos dados numa janela para que o utilizador os possa seleccionar e editar. Além desta função a vista também é responsável por comunicar ao documento quaisquer alterações a este

efectuadas. As classes mais importantes que servem de suporte a este modelo são a:

- *CDocument*, cujos objectos são usados para guardar e controlar todos os dados do programa;
- *CView*, cujos objectos servem para apresentar os dados do documento e gerir a interacção entre os dados e o utilizador;
- *CFrameWnd*, cujo objecto fornece a *frame* existente numa ou mais vistas de um documento;
- *CDocTemplate* (ou *CSingleDocTemplate* ou *CMultiDocTemplate*), cujos objectos coordenam a existência de um ou mais tipos de documentos e gerem a criação correcta do documento, vista e *frame* associadas ao tipo de documento indicado.

Este modelo de programação, que se baseia no conceito de separação do documento da vista, revela-se bastante eficiente nas aplicações em que se pretendem várias vistas do mesmo documento, como por exemplo as folhas de cálculo ou a visualização de gráficos. Suponhamos que o utilizador da aplicação pretende visualizar os dados ou na forma de uma folha de cálculo ou na forma de um gráfico. E suponhamos agora que os pretende ver simultaneamente de forma a comparar os resultados. A forma mais fácil de implementar esta funcionalidade é dividir estes dois tipos de vistas em diferentes *frames*, ou dentro da mesma janela, dividida por um separador (*splitter*), cuja função será explicada na secção seguinte. Por outro lado, o utilizador pode desejar alterar os seus dados e ver essas alterações imediatamente reflectidas no gráfico. Podemos assim concluir que o modelo de programação que estamos aqui a considerar é o ideal para implementar este tipo de operações. Assim, o código comum a todas as vistas, como os métodos de cálculo sobre os dados, fica contido no documento que, por sua vez, necessita de actualizar todas as vistas sempre que ocorram alterações nos dados. Utilizando a biblioteca MFC as vistas criadas devem ser classes derivadas da *CView* e, neste exemplo que estamos a considerar, ambas devem estar associadas ao mesmo documento (que contém os dados ou obtem-nos a partir de uma base de dados). Na fig. 4 estão apresentadas as várias formas de associar diferentes vistas ao mesmo documento.

Para resumir seguem-se alguns tópicos relacionados com a função da classe que implementa o documento e a vista.

As tarefas principais de um documento MFC são:

- O armazenamento dos dados,
- A inicialização dos documentos e das vistas,
- A inicialização daquilo que adicionamos às classes do documento e das várias vistas,
- Limpar os documentos e as vistas e
- Criar múltiplos tipos de documentos.

A vista MFC tem como funções:

- Mostrar e/ou desenhar os dados,
- Interpretar a interacção do utilizador,
- Efectuar as operações de *scrolling* e dimensionamento,
- Criar múltiplas vistas sobre um documento,
- Utilizar uma vista com diferentes tipos de controlos, tal como numa caixa de diálogo e
- Utilizar uma vista para mostrar os vários campos de informação de uma base de dados.

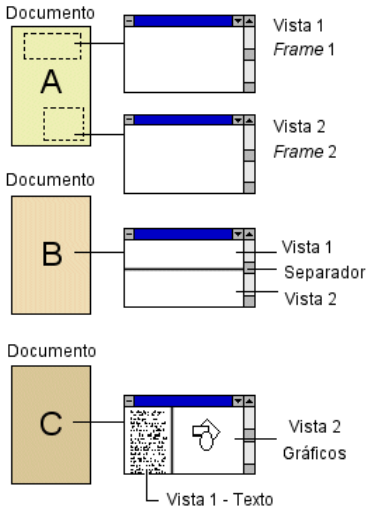

Figura 4. Diferentes vistas sobre o mesmo documento.

V. RECURSOS DE INTERFACE COM O UTILIZADOR

Existem vários tipos de interfaces gráficos, como por exemplo as janelas, barras de ferramentas, menus, aceleradores, separadores e controlos, a que chamamos recursos de interface.

C. Tipos de janelas

As janelas de uma aplicação podem ser dos mais variados tipos e cada uma delas possui uma *frame*, ou seja, uma delimitação cuja função pode variar dependendo da forma de derivação da classe *CFrameWnd*. Ao falar em janelas no sistema operativo Windows estamos a referirmo-nos a tudo que conseguimos visualizar, pois afinal tudo são janelas, desde a janela principal da aplicação a uma caixa de diálogo e até um simples controlo. Tudo isto depende apenas do programador quando lhe atribui uma forma, interface e funcionalidade.

D. Separadores

Quando falamos da janela que apresenta o conteúdo dos documentos que são utilizados, podemos incluir um separador para as diferentes vistas, como se pode ver na fig. 4 relativamente ao documento B (separador horizontal). A função do separador é dividir as janelas em vários painéis, cada um com a possibilidade de mostrar diferentes partes do documento e de diferentes maneiras. Os separadores, horizontais ou verticais, podem ser criados **estatica** ou **dinamicamente**.

Se for de forma estática a janela possui, definido à partida no programa, um número fixo de painéis, normalmente associados a classes diferentes. É necessário

um objecto do tipo *CSplitterWnd* declarado normalmente na classe que derivamos da *CMDIChildWnd*. Para criar um separador de forma estática é necessário redefinir a função *OnCreateClient* da respectiva classe e aqui invocar a função *CreateStatic* a partir do objecto a que corresponde o separador. Nesta função são passados como argumentos o número de linhas e de colunas em que queremos dividir a nossa janela de visualização. O próximo passo é criar todos os painéis invocando a função *CreateView* do separador. É nesta função que especificamos o painel a que nos referimos através do índice da linha e da coluna a que lhe corresponde, o seu tamanho mínimo e a classe, normalmente derivada da *CView*, a que vai ser associado. A escolha da vista inicialmente activa é efectuada invocando a função *SetActiveView*, à qual se fornece o painel da vista que pretendemos. Na fig. 5 é apresentada uma forma de divisão da janela de aplicação, em duas linhas e uma coluna. No entanto, a linha inferior é também dividida em duas vistas diferentes.

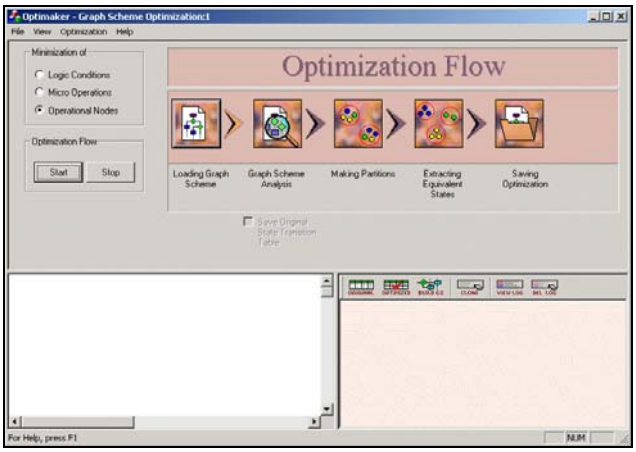

Figura 5. Exemplo de aplicação com três tipos de vistas diferentes separadas por 2 *splitters*.

Seguidamente é apresentado o código necessário para conseguir este tipo de interface. Existem dois separadores, instâncias da classe *CSplitterWnd*, que se chamam `m_split1` e `m_split2`. O primeiro divide a janela horizontalmente, pois podemos ver que são passados os argumentos 2 e 1 à função *CreateStatic* que correspondem ao número de linhas e de colunas, respectivamente. Tal como já foi explicado acima, deve-se invocar a função *CreateView* a partir da instância `m_split1` para criar a vista que podemos ver na parte superior da janela da fig. 5. A classe associada a esta vista foi chamada *CControlView* apenas porque contém vários tipos de controlos. No entanto, não podemos fazer este mesmo procedimento para as vistas inferiores, ou seja, não podemos invocar a função *CreateView* novamente utilizando a instância `m_split1`. Em vez disso passamos agora a usar a instância `m_split2` e invocamos a função *CreateStatic* mas com uma diferença. No primeiro caso, o ponteiro *this* foi passado como primeiro argumento à função *CreateStatic* porque este argumento corresponde a um ponteiro para a janela pai do separador, que no caso do `m_split1` é a *frame* da janela principal e daí a utilização do *this*. No caso do `m_split2` a janela pai é o próprio `m_split1`, que também é uma janela. Assim, na invocação da função *CreateStatic* é necessário indicar o ponteiro para o `m_split1` (`&m_split1`), o número de linhas desta nova janela (`1`) e o número de colunas (`2`). Para indicar que estamos a dividir a segunda linha de `m_split1` temos de indexá-la (de notar que os índices começam sempre por 0) utilizando a função *IdFromRowCol*, onde 1 corresponde ao índice da linha e 0 ao índice da coluna. A criação das vistas para `m_split2` é feita pelo método normal, já explicado anteriormente. Neste extracto de código terminamos por activar a vista relativa à parte superior da janela pela invocação da função *SetActiveView*.

```
m_split1.CreateStatic(this,2,1)

m_split1.CreateView(0, 0,
  RUNTIME_CLASS(CControlView),
  CSize(0,270), pContext))

m_split2.CreateStatic(
     &m_split1, 1, 2,
     WS_CHILD|WS_VISIBLE,
     m_split1.IdFromRowCol(1, 0)))

m_split2.CreateView(0, 0,
     pContext -> m_pNewViewClass,
        CSize(600, 0), pContext)

m_split2.CreateView(0, 1,
     RUNTIME_CLASS(CFrameTable),
        CSize(0, 100), pContext)

SetActiveView(
       (CView*)m_split1.GetPane(0, 0));
```

Figura 6. Extracto de código necessário à criação das três vistas da fig. 5 utilizando separadores estáticos.

Os separadores criados dinamicamente só podem ser utilizados sobre a mesma vista (a classe da vista tem de ser a mesma). Isto serve apenas para visualizar diferentes partes do documento mas com a mesma interface (tal como se passa na aplicação Microsoft Word).

O separador é um recurso de interface bastante útil porque sendo móvel, pode ser arrastado por intermédio do rato, permitindo ao utilizador aumentar a área do ecrã disponível para visualizar a sua informação. Por outro lado, se estivermos a falar de utilizadores inexperientes, o separador é um recurso que inicialmente pode esconder, em certos casos, os controlos colocados numa das vistas da aplicação.

E. Menus e Aceleradores

Os menus de uma aplicação desempenham um papel importantíssimo pois é aqui que todas as tarefas efectuadas pela aplicação têm de ser catalogadas e separadas de forma a que o utilizador, por mais

inexperiente que seja, consiga encontrar facilmente todos os comandos que deseja. O menu principal de uma aplicação situa-se geralmente no topo da janela e deve estar organizado da forma mais *standard* possível. Por exemplo, quando pretendemos manipular ficheiros, é usual utilizar um menu do tipo apresentado na fig. 7.

Figura 7. Ordenação *standard* das operações num menu.

Aconselha-se este método *standard* de ordenar a apresentação da funcionalidade pois assim estamos a diminuir o risco de confusão de um utilizador quando este passa de aplicação para aplicação. Inconsientemente a pessoa habitua-se a encontrar os menus em determinados sítios e por isso o programador deve ter isto em conta quando pretende uma boa aceitação da aplicação que projectou.

A cada opção disponibilizada dentro do menu deverão ser atribuídas teclas de atalho para que utilizadores mais experientes de uma dada aplicação as possam usar depois de uma familiarização com o programa, aumentando assim a rapidez de utilização e, consequentemente, a sua produtividade. A estes atalhos chamam-se aceleradores pois destinam-se precisamente a este efeito.

Com o aparecimento do sistema operativo Windows 2000, podemos ver que a maioria das aplicações Microsoft dispõem de menus que nos primeiros instantes em que são activados se visualizam apenas as opções mais frequentemente utilizadas. Esta é uma evolução da interface que pretende eliminar do ecrã informação à partida desnecessária ao utilizador frequente mas que poderá causar alguma confusão, independentemente da experiência da pessoa, pois os comandos mudam de sítio e a maioria fica escondida, dificultando a procura de um comando que já há muito não é utilizado. Para substituir este método e facilitar realmente a procura dos comandos utilizados mais frequentemente deverão ser disponibilizadas as barras de ferramentas que serão descritas na secção seguinte.

Contudo, existe outro tipo de menus, denominados *popup*, que são activados com o botão direito do rato. Estes são normalmente personalizados dependendo do tipo de objecto apontado pelo rato. Seguidamente é apresentado um extracto de código escrito em Visual C++ (fig. 8) que ilustra a criação de um menu *popup*. Para que este seja visualizado ao apontar certo objecto, cuja classe seja derivada da *CObject*, o código apresentado deve ser escrito dentro da função *OnRButtonDown* que responde ao evento do clique do botão direito do rato. Esta função deverá ser redefinida na classe da vista onde o objecto se encontra. Vamos assumir que temos uma classe *CMyObj* drivada da *CObject* e duas classes *CMyObjX* e *CMyObjY* derivadas da *CMyObj*. O objectivo é mostrar um menu popup diferente para um objecto de cada classe. Os menus são criados graficamente no editor de recursos do Microsoft Visual Studio 6.0 como pertencentes a um único menu, como se este se comportasse como um array, com o identificador IDR_CURSOR_MENU. A invocação ou escolha de cada menu é efectuada pela função realçada no código, *GetSubMenu(0)*;

```
CMyObj *pObj;
CMenu menu;
menu.LoadMenu(IDR_CURSOR_MENU);

if(pObj->IsKindOf(RUNTIME_CLASS(CMyObjX))
  menu.GetSubMenu(0)->TrackPopupMenu(
      TPM_LEFTALIGN|TPM_RIGHTBUTTON,
      point.x, point.y, this);
else
if(pObj->IsKindOf(RUNTIME_CLASS(CMyObjY))
  menu.GetSubMenu(1)->TrackPopupMenu(
      TPM_LEFTALIGN|TPM_RIGHTBUTTON,
      point.x, point.y, this);
```

Figura 8. Criação de menus popup específicos de cada objecto.

F. Barras de ferramentas

As barras de ferramentas, normalmente situadas por baixo do menu de uma aplicação, têm como objectivo disponibilizar atalhos ao utilizador para as suas opções mais frequentes. Em inúmeras aplicações de software comerciais existe também a possibilidade de personalizar estas barras mediante o perfil de cada utilizador.

Consistem num conjunto de pequenos botões, com um determinado desenho que deve sugerir a função do botão quando este for premido. Cabe ao programador criar estes desenhos, devendo contudo seguir determinadas regras *standard*, como no caso dos menus. As opções mais usuais, como as operações sobre ficheiros, possuem já ícones aos quais o utilizador associa a sua função devido a um certo hábito de utilização e familiarização com as figuras apresentadas. Assim, a criatividade do programador não deverá ter liberdade total devido aos factores acima mencionados. A criação dos desenhos deve seguir uma filosofia de forma a que todos os ícones sejam consistentes. Existem basicamente três esquemas/filosofias de representação:

1. desenhar algo que represente o antes e o depois do comando que se pretende,
2. desenhar a acção do comando (ex. abertura de um ficheiro) e
3. desenhar um objecto que represente a ferramenta necessária para efectuar a operação desejada (ex. impressão de um documento)

Os esquemas mais frequentes são o 2º e o 3º pois levam-nos a um desenho mais simples e a um resultado mais intuitivo. No entanto, não devem ser misturados na mesma barra de ferramentas [3] pois conduzem a uma mais lenta compreensão do significado por parte do utilizador. Contudo, verifica-se que em aplicações comerciais este erro é frequentemente cometido, como se pode ver na fig. 9 onde temos uma barra de ferramentas com ícones pertencentes ao 2º e 3º esquemas apresentados acima.

Figura 9. Barra de ferramentas - Esquema de representação misto (objectos, acções e ferramentas).

Por outro lado, existe também a possiblidade de desenhar os ícones de outra forma, normalmente em barras de ferramentas específicas para uma dada operação. Neste caso é usado o mesmo objecto em vários botões da barra, diferenciados por algo que representa as diferentes acções sobre esse objecto, como é o caso da fig. 10 [4, 5].

Figura 10. Barra de ferramentas – Esquema de representação através de um objecto comum.

VI. CRIAÇÃO, VISUALIZAÇÃO E INTERACÇÃO COM OS OBJECTOS

Nesta secção irão ser mencionadas as várias formas de desenho de objectos no ecrã ou mais concretamente dentro de uma *frame* criada pela aplicação de software que pretendemos projectar. Para desenhar directamente sobre essa janela podem ser utilizados vários dispositivos apontadores, como por exemplo o rato ou outros mais sofisticados e dedicados a criações gráficas. Para mais detalhes acerca de como desenhar directamente no ecrã sugere-se a leitura de [2].

Por outro lado, podemos ter objectos pré-definidos e que podem ser colocados, movidos e/ou editados pelo utilizador na janela da aplicação [4]. Neste caso devemos, como programadores, criar os nossos próprios objectos, utilizando classes derivadas da *CObject*, também pertencente à biblioteca MFC. Cada classe deve possuir uma função para desenhar um dos seus objectos (instâncias da classe) no interior da janela. A biblioteca fornece inúmeras funções para desenho, tais como *Line*, *Rectangle*, *Ellipse*, *Polygon*, etc para figuras geométricas e, por exemplo, *TextOut* para visualizar texto.

De acordo com o modelo documento/vista, devemos associar ao documento os objectos e a sua manipulação, nomeadamente a criação e operações efectuadas sobre eles. A vista deverá ser utilizada apenas para os desenhar e permitir a interacção entre o utilizador e o objecto. A classe da aplicação que implementa a vista, isto é, aquela que é derivada da *CView* (ou *CSrollView*), possui uma função *OnDraw* que deverá ser redefinida para que possamos especificar dentro dela a forma de desenho dos vários objectos. É aqui que serão invocadas as funções de desenho específicas de cada objecto criado e que pertence ao documento.

VII. DIFERENTES TIPOS DE APLICAÇÕES

A programação de uma aplicação de software utilizando o método documento/vista associado à familiarização com a técnica de desenho de objectos no interior de uma janela abre caminho para o projecto dos mais variados tipos de aplicações. De seguida são apresentados três tipos possíveis e os respectivos objectivos.

Dado que é possível dispor de diferentes interfaces para o mesmo documento, poder-se-à criar facilmente uma aplicação para estudo e avaliação de diferentes interfaces por vários perfis de utilizadores. Neste caso a aplicação projectada teria várias “caras” com diferentes características que o utilizador poderia escolher e mudar a qualquer instante. Assim se poderiam efectuar estudos onde se avaliassem diferentes colocações de controlos no ecrã, esquemas de cores, organização dos dados no ecrã pertencentes a uma base de dados, o acesso aos vários comandos e opções da aplicação, etc.

Por outro lado, temos a criação de ferramentas específicas com controlos personalizados e especializados para uma dada área de investigação. Ao nível da simulação de um dado sistema, qualquer que ele seja, podem ser criados graficamente objectos virtuais, ou seja, o mais parecidos possível com os objectos com que lidamos diariamente e interagir com eles. Em sistemas de hardware, por exemplo, poderia ser construída a sua interface através de controlos actuados pelo utilizador, como por exemplo, leds, interruptores, visores, etc.

Finalmente, a personalização de controlos e a utilização de imagens pode conduzir à criação de aplicações com interfaces parecidos com páginas HTML, cuja composição depende apenas da criatividade do programador, com um elevado grau de liberdade, pois os utilizadores da geração actual estão cada vez mais familiarizados com esta tecnologia.

VIII. CONCLUSÕES

As aplicações que utilizam a biblioteca MFC usam normalmente o modelo de programação documento/vista para gerir informação, diferentes formatos de ficheiros e a representação visual dos dados ao utilizador. Na generalidade esta é a forma mais eficiente e apropriada de construir uma aplicação porque separa de forma clara os dados da sua visualização, eliminando código redundante e simplificando assim a sua escrita. No entanto, este não é o método mais apropriado quando se pretende projectar uma aplicação muito simples onde este modelo não se justifique ou se se pretender portar código que à partida misture a gestão dos dados com a sua visualização. Neste caso a separação das duas tarefas pode ser extremamente difícil e por isso este modelo seria indesejável.

Relativamente ao interface da aplicação podemos concluir que embora existam regras e se deva seguir

directivas de standardização, essas regras nem sempre são cumpridas mesmo em aplicações comerciais, como já foi mencionado. Ao projectar uma aplicação o programador tem total liberdade na escolha do interface. No entanto, não deve esquecer o perfil dos utilizadores da sua aplicação e avaliar a sua rapidez de aprendizajem pois este factor é determinante na aceitação e consequente utilização frequente do software produzido [6].

REFERÊNCIAS

[1] Andreia Melo, Beatriz S. Santos, C. Ferreira, J. Sousa Pinto, "Software Application for Data Visualization and Interaction in a Location Routing Problem", Electrónica e Telecomunicações, Vol. 2, Nº 4, pp 471-476, Janeiro, 1999.

[2] Ivor Horton, "Beginning Visual C++ 4", Wrox Press, 1996.

[3] Deborah J. Mayhew, "Principles and Guidelines in Software User Interface Design", Prentice Hall, 1992.

[4] Andreia Melo, "Especificação, Optimização e Teste de Algoritmos de Controlo Hierárquicos", Dissertação de Mestrado em Engenharia Electrónica e de Telecomunicações, Universidade de Aveiro, Janeiro, 2000.

[5] Andreia Melo, Valery Sklyarov, "Ambiente Integrado para Especificação, Projecto e Verificação de Unidades de Controlo em FPGAs", Electrónica e Telecomunicações, Vol. 2, Nº 4, pp 477-485, Janeiro, 1999.

[6] Beatriz S. Santos, J. Sousa Pinto, "An Introductory Course on Human Computer Interaction", Electrónica e Telecomunicações, Vol. 3, Nº 3, pp 228-232, Janeiro, 2001.