

Capítulo 7

Programação orientada a eventos

A programação tradicional

No passado, um sistema computacional era composto por uma CPU responsável por todo o processamento (mestre) e uma série de periféricos (escravos), responsáveis pela entrada e saída de dados.

Neste paradigma, o usuário assumia uma posição de periférico, já que ele era somente um mecanismo de entrada de dados.

O custo/benefício de uma CPU era extremamente superior ao de qualquer equipamento ou (custo de trabalho) usuário a ela conectados. Consequentemente, todos os sistemas davam prioridade máxima ao processamento, em relação à entrada e saída de dados.

Com isso, era comum encontrarmos sistemas em que a fatia de tempo dedicada aos periféricos era sensivelmente menor do que a dedicada à CPU.

Esta diferença de custos implicava uma menor dedicação, por parte dos programadores, aos periféricos e usuários em geral.

Ao colocar o usuário como escravo, este paradigma prejudicava a interação direta com o sistema.

O usuário não tinha como realizar de forma ágil suas operações, já que o sistema não priorizava a entrada e a saída de informações.

O diálogo dos usuários com o sistema foi evoluindo, ao longo dos tempos, porém continuava se apresentando como uma relação mestre-escravo, em que o primeiro ditava a sequência da interação.

Mesmo os sistemas ditos avançados apresentavam uma hierarquia rígida de menus e/ou formulários, que eram preenchidos pelo usuário à medida em que este navegava por essa estrutura.

Interfaces gráfico-interativas: o usuário como mestre

Com o avanço tecnológico, o custo/hora da CPU foi diminuindo, a ponto de se tornar cada vez mais comum encontrar sistemas onde o usuário determinava como se dava a interação e a navegação por suas estruturas.

Com o barateamento da tecnologia, os sistemas gráficos foram se mostrando uma forma mais razoável de representar estruturas complexas, onde a interação ocorre de forma **assíncrona**.

O conceito de eventos como paradigma de programação

O funcionamento de um sistema gráfico se dá através de ações sobre os diversos elementos da interface (objetos visuais). Cada uma destas ações corresponde um ou mais eventos “enviados” para esses elementos. Estes eventos devem ser identificados e tratados de maneira adequada pelo programa aplicativo.

Para manipular uma interface interativa, a programação tradicional se utiliza do recurso de um loop de controle principal (*case/switch*), que gerencia as opções de navegação oferecidas pelo sistema.

A necessidade de cuidar de eventos assíncronos com um paradigma síncrono acarreta uma relativa complexidade deste loop principal.

O conceito de objeto (versus procedimento) como mecanismo de programação

O conceito de objeto consiste basicamente em agrupar dados e código num mesmo elemento de programação. Ao invés de procedimentos atuando sobre estruturas de dados, passamos a ter estes dados (propriedades) associados a métodos (procedimentos). Estes métodos são os responsáveis pelo controle de fluxo e de informação entre os objetos do sistema.

Propriedades correspondem a campos de um registro (*record/struct/class*).

O conceito de **objeto visual** consiste na associação de uma representação gráfica (elemento de interface) a um trecho de código responsável pelo comportamento do objeto. A comunicação entre um objeto de interface e a aplicação se dá através da chamada de métodos de outros objetos (de interface ou não).

O pacote gráfico orientado a eventos DOSGRAPH

DOSGRAPH é uma biblioteca simples com apenas dez funções que possibilita uma programação orientada a eventos. A função principal é a de captura de eventos, e só retorna quando ocorre um evento. O eventos do DOSGRAPH são apenas os de mouse: click e unclick com os dois botões e mouse motion.

Uma aplicação típica que utiliza o DOSGRAPH (e qualquer outro pacote orientado a eventos) tem como núcleo um loop que captura os eventos e faz o processamento adequado. Todo o processamento é feito em resposta aos eventos.

Como aplicações que utilizam o mouse são tipicamente gráficas, o DOSGRAPH fornece várias funções de desenho que podem ser utilizadas, por exemplo dgLine, que desenha uma linha com a cor corrente. As funções do DOSGRAPH podem ser classificadas em:

funções de desenho: dgLine, dgRectangle, dgFill, dgSetColor e dgSetMode;

funções de consulta: dgWidth e dgHeight;

funções de inicialização: dgOpen e dgClose;

funções de controle: dgGetEvent.

Os tipos e funções definidos pelo DOSGRAPH estão definidos no apêndice A desta apostila.

Utilizando o DOSGRAPH em uma aplicação

Para fazer uma aplicação que use o DOSGRAPH, é preciso incluir a biblioteca no projeto. O nome da biblioteca é dg.lib, e deve ser incluída junto com os arquivos fonte que implementam a aplicação.

É importante lembrar que a função dgOpen deve ser chamada antes de qualquer outra função do DOSGRAPH. Os resultados das funções antes de dgOpen é imprevisível.

Para executar a aplicação DOSGRAPH, é preciso que o arquivo egavga.lib esteja presente no diretório do executável. Caso este arquivo não esteja presente, a função dgOpen falha e gera uma mensagem de erro.

Exercício 11 - Uma aplicação orientada a eventos em C.

Implementar uma aplicação C que utilize o DOSGRAPH. Esta aplicação deve responder a um click com o primeiro botão desenhando um retângulo na posição do cursor do mouse. Um click com o segundo botão deve terminar a aplicação.

Exercício 12 - Classe Application

Implementar uma classe que represente uma aplicação DOSGRAPH. Sempre que for necessário fazer uma nova aplicação, basta criar uma nova classe derivada que responda aos eventos tratados, sem necessidade de usar as funções dgOpen, dgClose e dgGetEvent. Novas aplicações só precisam se preocupar com o processamento dos eventos.

