Rumo à Portabilidade de Componentes de Diálogo

Marcelo Soares Pimenta
Depto. de Ciências Estatísticas e da Computação - UFSC
Caixa Postal 476 - Florianópolis - SC - Brasil
Cep 88040-900 - Fone: +55 482 319739
e-mail: cec1msp@brufsc.bitnet

Abstract

This paper describes the Canonicus model, an object oriented user interface representation model with two main goals: i) to allow dialog component portability among different user interface tools and ii) to provide abstractions to separate clearly user interface designer and application programmer roles. Canonicus' main caracteristics, including generic and concret classes, and our experience in its implementation are presented.

Keywords: user interface tools, object orientation, user interface representational model.

1. Introdução

Um programa interativo é formado por componente computacional (doravante denominado aplicação) e componente de diálogo (que implementa a interface com o usuário, abreviada IU). O componente de diálogo é, em geral, construído através do uso de uma Ferramenta para Desenvolvimento de IU (abreviada FIU). FIUs são conjuntos de rotinas ou *programas que auxiliam a definição e manipulação de IUs [1].. Exemplos de FIUs são Mac* Toolbox, MS-Windows Development Kit e OSF/Motif. Devido às idiossincrasias das FIUs, o programador de aplicação acaba criando diferentes versões da mesma aplicação: uma para cada FIU.

Além disto, a maioria das FIUs existentes são difíceis de usar pelo programador da aplicação, pois contêm literalmente centenas de rotinas [2]. Estas rotinas devem ser adequadamente escolhidas e combinadas não só para manipular todos os eventos de entrada como também para desenhar e manipular os elementos da IU (menus, janelas, botões, etc.) Este excesso de detalhes e tarefas resulta na indesejável unificação dos papéis do programador da aplicação e do projetista de interfaces, uma vez que, mesmo com o uso de editores e arquivos de recursos, estas FIUs não isolam o programador de informações relativas ao projeto de interfaces.

O objetivo deste artigo é apresentar o modelo Canonicus, um modelo representacional de IUs orientado a objetos que visa permitir:

- a portabilidade do componente de diálogo de um programa interativo entre diferentes FIUs; e
- a definição dos detalhes e tarefas da IU em diferentes níveis de abstração de forma a separar claramente os papéis do projetista de IUs e do programador de aplicação.

Na seção 2, apresenta-se o enfoque adotado para alcançar portabilidade. Na seção 3, destacam-se algumas características principais do Canonicus. A separação entre classes

genéricas e concretas é descrita na seção 4. A seção 5 apresenta alguns aspectos da implementação do Canonicus sobre FIUs comerciais. Finalmente, na seção 6 são tecidas algumas observações conclusivas.

2. Canonicus e portabilidade

Dois enfoques têm sido utilizados para prover portabilidade de programas interativos em relação ao componente de diálogo. Ambos são baseados na uniformização da comunicação entre aplicação e componente de diálogo, de forma que a aplicação use o mesmo protocolo de comunicação para chamar rotinas de diálogo em qualquer ambiente.
O primeiro enfoque é o uso de uma FIU transportável, uma FIU que possui uma implementação em vários ambientes [4] . O segundo é o estabelecimento de uma camada intermediária entre a aplicação e cada FIU nativa em um ambiente. Em verdade, esta camada intermediária provê uma interface comum com a aplicação para as diferentes FIUs e, por isto, é também denominada camada canônica [2]. As chamadas da aplicação à camada canônica são mapeadas para chamadas à FIU nativa subjacente. Deste modo, o "look and feel" do programa é o "look and feel" da FIU subjacente, característica adequada aos usuários habituados à consistência de um determinado ambiente. Um exemplo típico de uso deste enfoque é o sistema comercial XVT (eXtended Virtual Toolkit) [5], uma biblioteca de funções implementada inicialmente sobre a Mac Toolbox e o MS-Windows.

O Canonicus baseia-se no segundo enfoque para suportar aplicações em diferentes ambientes com diferentes FIUs e diferencia-se do XVT por usar o paradigma de orientação a objetos [6] para definição e uso da camada canônica e para a implementação do seu mapeamento para as FIUs nativas.

3. Características do Canonicus

O modelo Canonicus é um modelo de representação de IUs (ver [7] para definição de modelos de representação de IUs) orientado a objetos. Dessa forma, pretende-se uma descrição da comunicação IU-aplicação de maneira mais adequada ao projetista de interface e ao programador da aplicação, diminuindo a dificuldade de definição/manipulação de IUs.

A Mac Toolbox, por exemplo, pode ser considerada orientada a objetos na visao do usuário , enquanto sua comunicação com a aplicação é feita segundo o paradigma convencional de rotinas e funções. Com o Canonicus, toda comunicação entre IU e aplicação é estabelecida via mensagens direcionadas para os (e/ou a partir dos) objetos pertencentes a IU.

A notação do Canonicus é baseada na linguagem orientada a objetos Eiffel [8] e por isto usamos os termos "atributos" e "rotinas" ao invés dos tradicionais "variáveis de instância" e "métodos" da nomenclatura Smalltalk. Definir IUs usando o Canonicus é descrever classes (genéricas e concretas) de objetos de IU, especificando seus atributos, mensagens e rotinas. Consequentemente, para operar sobre a IU definida deve-se instanciar objetos destas classes e manipulá-los de acordo com o seu protocolo de mensagens.

Em [9] , encontra-se um exemplo que contém as definições e o uso de uma classe genérica Janela e duas subclasses concretas Jan-Mac e Jan-MSW, que implementam a classe Janela respectivamente sobre a Mac Toolbox [10] e o MS-Windows [11] . As descrições completas (com todos atributos, rotinas e mensagens) das classes genéricas, classes concretas para a Mac Toolbox e para o MS-Windows estão presentes em [12] .

Usando o Canonicus, o projetista de interfaces define as classes dos objetos que comporão a IU e suas apresentações. Os objetos são descritos por suas propriedades, seus componentes e seus operadores disponíveis. Sua apresentação define cores, formatos, estilos e tamanhos, etc. Usando o Canonicus, o programador da aplicação define quando usá-los e como associá-los, junto com seus operadores, às funções da aplicação.

4. Classes Genéricas e Concretas

Canonicus é um formalismo adequado para separar a definição (em níveis) da IU pelo projetista de interfaces de sua manipulação pelo programador da aplicação pois organiza os elementos de IU (menus, janelas, botões, etc.) em classes de objetos de IU . As classes podem ser genéricas (ou abstratas) ou concretas (ou de implementação). Por definição, uma classe *concreta é sempre subclasse de alguma classe genérica.*

Basicamente, uma classe genérica possui a descrição abstrata dos objetos de IU (suas características, componentes e comportamento) enquanto as suas subclasses concretas possuem a definição dos atributos de apresentação destes objetos (forma, cores, tamanho, etc.) e a implementação de seu comportamento. Para isto, na classe concreta são encapsuladas as estruturas de dados e o código desta implementação.

Na definição das classes genéricas, os atributos podem ser de tipos pré-definidos (integer, string, etc.) ou associados a classes definidas enquanto todas as rotinas são deferidas, i. é, não apresentam descrição de implementação.

Na definição das classes concretas, por sua vez, os atributos podem possuir tipos relacionados `a FIU especifica subjacente (chamados atributos FIU), adicionalmente aos outros tipos. Como uma classe concreta é subclasse de uma genérica, herda todos atributos desta, podendo também redefini-los ("override"). Além disto, todas rotinas possuem implementação associada. Os atributos FIU são necessários para a apresemtação dos objetos e para a parametrização desta implementação.

Rotinas deferidas são rotinas especificadas na classe genérica mas realmente implementadas nas suas subclasses concretas, podendo ser encaradas como a descrição genérica de um grupo de implementações específicas. Por isto, as classes genéricas são também denominadas classes deferidas e não podem ter instâncias, pois não há rotinas da classe para manipulá-las.

A definição da implementação das rotinas deferidas das classes genéricas é feita através do mapeamento de uma rotina para um conjunto (possivelmente unitário) de rotinas de uma FIU, o qual corresponde à implementação das tarefas da rotina. Isto é ilustrado na figura 1.

As rotinas deferidas são a chave para a portabilidade. O programador ***deve*** usar as rotinas deferidas, especificadas no nível abstrato. Assim, os detalhes de manipulação/exibição dos objetos, encapsulados no nível concreto, são escondidos do programador, que não precisa tomar conhecimento de qual implementação esta' sendo executada/utilizada pelo projetista de interface.

Esta definição tem por proposta:
a) encapsular no nível concreto as características de apresentação dos objetos de IU;

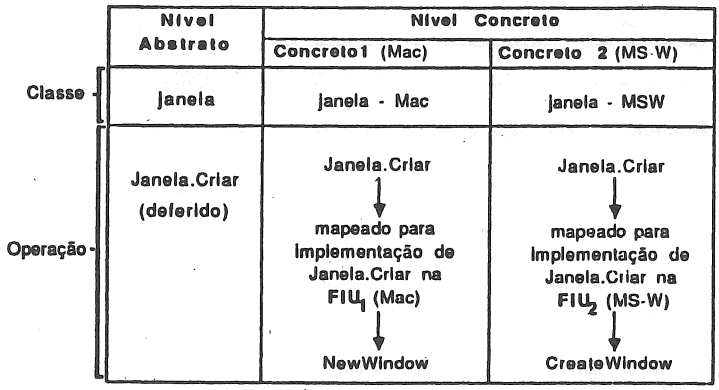

Figura 1 - Implementação de rotinas deferidas: um exemplo.

b) servir para a descrição da implementação das rotinas deferidas das classes genéricas;

c) servir de base para o processo de tradução do Canonicus para uma FIU específica, descrevendo o mapeamento entre eles.

De fato, as classes genéricas formam uma camada canônica abstrata (a FIU Canônica) acima da camada concreta das FIUs, formada pelas classes concretas.

5. Aspectos de Implementação

O mapeamento da camada canônica formada pelas classes genéricas do Canonicus para as FIUs subjacentes (cujas características estão encapsuladas nas classes concretas) é implementado através da tradução de todos seus objetos e mensagens para elementos e rotinas de alguma FIU subjacente (por enquanto a MacToolbox, o MS-Windows Development Kit e o X-View respectivamente em ambiente Macintosh, PC e workstations Sun).

Esta opção por mecanismos de tradução foi tomada por três motivos principais:
- a flexibilidade decorrente do possível uso de vários "look and feel" desde que a tradução do Canonicus para as FIUs que os suportam seja provida;
- a preocupação prioritária com a uniformização da comunicação IU-aplicação como base para a portabilidade de componentes de diálogo;
- a facilidade de experimentação, pois é mais direto construir tradutores do que FIUs.

A tradução pode ser feita segundo dois enfoques: o compilativo e o interpretativo, descritos preliminarmente em [4] . Neste trabalho, por limitações de espaço, falaremos sucintamente da implementação de um tradutor segundo o enfoque compilativo.

Um tradutor que use o enfoque compilativo é tipicamente um pré-processador. Sua função é "varrer"o código fonte do programa interativo e realizar o mapeamento mencionado acima. A partir da especificação das classes genéricas e das concretas para uma FIU específica, o pré-processador deve realizar as seguintes tarefas no programa:

1) Inserir as declarações de variáveis e tipos específicos necessários à execução das rotinas da FIU específica;

2) Substituir as chamadas/invocações das operações/rotinas do Canonicus pelos trechos de código relativos à implementação destas rotinas na FIU particular.

Após a execução da tarefa 1, o conjunto das declarações do programa é a união das declarações que já existiam no programa com as variáveis e estruturas de dados para uso das FIUs específicas.

Após a execução da tarefa 2, o programa pode ser submetido a um compilador.

A figura 2 apresenta um esquema do processo de tradução:

i) O Programa Interativo contém, nos pontos de programa em que a IU deve ser acionada, mensagens aos objetos do Canonicus (chamada à FIU Canônica).

ii) O protocolo genérico de mensagens do Canonicus (camada canônica) deve ser traduzido pelo pré-processador para rotinas de interação de uma FIU específica, substituindo-se qualquer referência a elementos do Canonicus por referências a elementos presentes nas FIUs específicas.

iii) O pré-processador gera um programa interativo para uma FIU determinada (programa para FIU) com as devidas definições, chamadas ou corpos de procedimentos necessários para que a FIU execute o diálogo especificado.

iv) Como as chamadas do componente de diálogo (tanto ao Canonicus quanto à FIU específica) estão embutidas num programa, o passo seguinte é submeter este programa para FIU ao compilador disponível para que se obtenha o código executável do programa (Código Executável FIU). Através da execução deste código, o usuário final interage com o programa interativo.

O enfoque compilativo realiza todo mapeamento ***a tempo de compilação***. A cada alteração das interações ou dos pontos de interação do programa este deve passar pelos passos i a iv acima.

Atualmente está em estágio final de desenvolvimento um pré-processador que mapeira o componente de diálogo do Canonicus para a Mac Toolbox e espera-se que, em breve, os pré-processadores para o MS-Windows e o X-View também estejam funcionando.

6. Conclusões

Este trabalho descreveu as principais características do modelo Canonicus, que visa propiciar a portabilidade de componentes de diálogo de programas interativos desenvolvidos sobre algumas FIUs comerciais e diminuir a dificuldade de definição/manipulação de IUs.

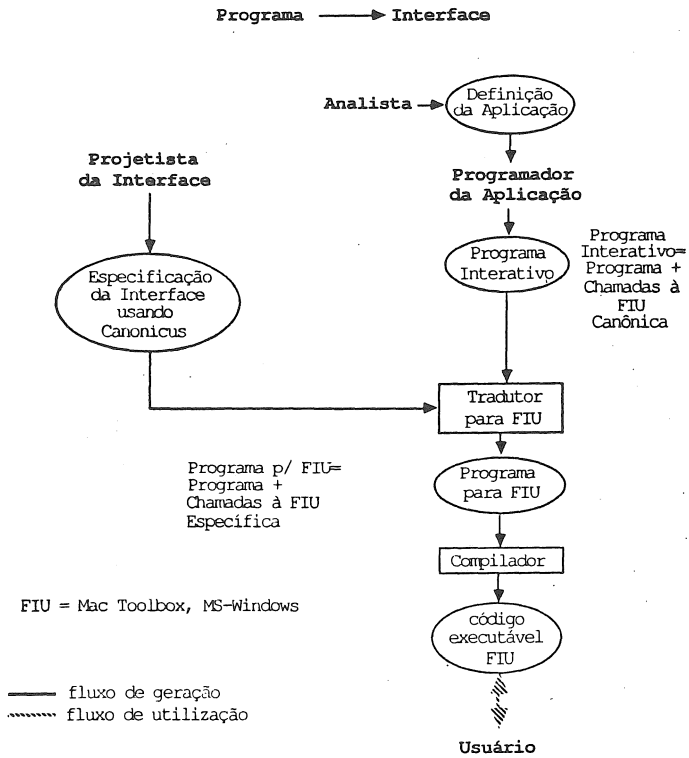

Figura 2 - Esquema do processo de tradução (enfoque compilativo)

O Canonicus é uma camada orientada a objetos padronizada entre as FIUs e a aplicação e é implementada através de mecanismos de tradução, com seus objetos e operações mapeados para elementos e operações das FIUs subjacentes. Desde que classes genéricas do Canonicus podem ser acessadas somente através das operações associadas a elas, os usuários das classes genéricas não tem acesso às especificidades das classes concretas. Isto provê um completo encapsulamento das peculiaridades das FIUs subjacentes e torna o uso de classes genéricas adequado para o programador.

Outra vantagem do Canonicus é a reusabilidade. O enfoque orientado a objetos adotado possibilita definir novas classes/componentes de IU a partir das classes primitivas existentes, que provêm um conjunto de objetos que podem ser usados como "blocos de construção", já devidamente testados e depurados.

O Canonicus pode servir como suporte a uma metodologia de desenvolvimento de IUs, com a separação dos papéis do projetista de interfaces e do programador de aplicações e com suas hierarquias de classes propiciando projeto de IUs com níveis diferenciados de detalhes. O uso de uma mesma notação para especificação (classes genéricas) e implementação (classes concretas) de IUs evita os erros decorrentes da tradução de um formalismo de projeto para um de implementação e beneficia as tarefas dos usuários projetistas, mais notadamente o programador, que pode usar a IU sem se preocupar em conhecer ou fornecer os detalhes para sua execução.

7. Bibliografia

[1] MYERS, B. "User Interface Tools: Introduction and Survey" IEEE Software, Jan 1989.

[2] MYERS, B. et alli "Garnet: Comprehensive Support for Graphical Highly Interactive User Interfaces". IEEE Computer Nov 1990.

[3] SZCZUR, M. "Transportable Applications Environment - An Integrated Design-to-Production UIMS", Ninth Annual Conference and Exposition to Computer Graphics Applications, Proc., Mar 1988.

[4] FRAINER, A.S. ; PIMENTA, M.S.; PRICE, R.T. "Como Obter Portabilidade de Programas Interativos" Anais X Congresso SBC, Vitória, Julho 90.

[5] VALDES, R. "A Virtual Toolkit for Windows and the Mac", BYTE Mar 1989.

[6] GIRARDI,M.R. ; PRICE, R.T. . "O Paradigma de Desenvolvimento por Objetos" Revista de Informática Teórica e Aplicada, V. 1. N. 2, 1990, pp 69-98.

[7] HARTSON, H.R.; HIX, D. "Human-Computer Interface Development: Concepts and Systems for its Management" ACM Computing Surveys V.21 n. 1 Mar 1989.

[8] MEYER, B. "Object Oriented Software Construction", Prentice-Hall, Series in Computer Science, 1988.

[9] PIMENTA, M.S.; HEUSER,C.A. "Canonicus: Um Modelo para Portabilidade de Programas Interativos". Anais do V Simpósio Brasileiro de Engenharia de Software, Ouro Preto, Out 1991.

[10] Apple Computer "Inside Macintosh V. I,II,III", Addison-Wesley, 1985.

[11] JAMSA, K. "Windows Programming Secrets", Osborne McGraw-Hill, Berkeley, USA. 1987.

[12] PIMENTA, M.S. "Um Modelo Canônico de Ferramenta para Desenvolvimento de Interface com o Usuário",
Dissertação de Mestrado. CPGCC-UFRGS. Jan 1991.