UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE MATEMÁTICA, ESTATÍSTICA E CIÊNCIA DA COMPUTAÇÃO
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

UMA FERRAMENTA GRÁFICA PARA NAVEGAÇÃO E CONSULTA EM BANCOS DE DADOS ORIENTADOS A OBJETOS

por

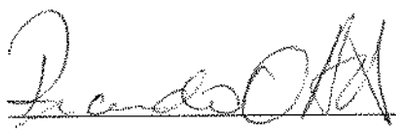
Juliano Lopes de Oliveira

24 de março de 1993

Uma Ferramenta Gráfica para Navegação e Consulta em Bancos de Dados Orientados a Objetos

Este exemplar corresponde a redação final da tese devidamente corrigida e defendida pelo Sr. Juliano Lopes de Oliveira e aprovada pela Comissão Julgadora.

Campinas, 24 de março de 1993.

Prof. Dr. 

Ricardo de Oliveira Anido

Dissertação apresentada ao Instituto de Matemática, Estatística e Ciência da Computação, UNICAMP, como requisito parcial para a obtenção do Título de MESTRE em Ciência da Computação.

Juliano Lopes de Oliveira

Uma Ferramenta Gráfica para Navegação e Consulta em Bancos de Dados Orientados a Objetos

Dissertação apresentada em 24 de março de 1993.

Banca Examinadora:

Dr. Ricardo de Oliveira Anido — DCC–UNICAMP
Orientador

Dra. Claudia Maria Bauzer Medeiros — DCC–UNICAMP
Coorientadora

Dr. Marcos Roberto da Silva Borges — NCE–UFRJ

Dr. Hans Kurt Edmund Liesenberg — DCC–UNICAMP
Suplente

Sumário

Esta dissertação analisa os problemas relacionados ao projeto e implementação de interfaces gráficas para sistemas gerenciadores de bancos de dados (SGBDs) orientados a objetos. Como resultado desta análise, são apresentadas diretivas para o desenvolvimento de interfaces para sistemas de bancos de dados. Estas diretivas foram empregadas na especificação e desenvolvimento de um novo sistema de interface — GOODIES — que permite navegação e consulta em SGBDs que possuem as características básicas do modelo OO.

O projeto e a implementação deste novo sistema são descritos, servindo como um estudo de caso do emprego das diretivas propostas. O sistema foi desenvolvido de forma a torná-lo independente da implementação de um SGBDOO específico. Isto permite que ele seja acoplado a diferentes sistemas de bancos de dados OO.

"O rio chega até o mar
porque aprende a contornar os obstáculos."

Dedico este trabalho ao meu pai, Sr. Urbano, e à minha mãe, D. Nadir, que sempre se superaram para oferecer todas as condições necessárias para a minha formação.

Agradecimentos

"No final, tudo acaba bem. Se alguma coisa não está bem, é porque ainda não chegamos ao final."

Diversas pessoas contribuiram para a realização deste trabalho. Gostaria de expressar aqui os meus sinceros agradecimentos a todas essas pessoas, e em especial a alguns amigos, com os quais eu compartilho a alegria que sinto ao ver este projeto realizado:

- Os meus irmãos: Roseli, Edir, Eni, Vânia, Vani, Eliane e Ronaldo. Graças ao apoio de todos eles, sem excessão, consegui superar as dificuldades do caminho.
- A Eliane, minha "procuradora", merece um agradecimento especial, pela dedicação e pelo zelo com que trata os meus interesses em Goiânia.
- O Ronaldo e a Zezinha, minha família em Campinas, tiveram muita paciência com minhas instabilidades.
- O Leonardo ("Leo"), meu sobrinho, que, como eu, está longe de casa, em busca de um ideal.
- O Anido e a Claudia, meus orientadores, estiveram sempre dispostos a discutir minhas propostas e a esclarecer minhas dúvidas. O mérito desta tese é, em grande parte, deles.
- Os meus companheiros de "república": Fileto, Herbert, Leonardo e Tutumi. Estivemos pouco tempo juntos, mas agradeço pelo incentivo na reta final deste trabalho.
- A Ivonne e a Márcia, que ajudaram a tornar mais agradáveis as poucas horas de lazer que desfrutamos, nos últimos dois anos.
- O pessoal do volei, "atletas" integrantes de um grupo onde conseguíamos esquecer os problemas e aliviar as tensões da semana.

Agradeço ainda ao Conselho Nacional de Desenvolvimento Científico e Tecnológico – CNPq – que deu o suporte financeiro à execução do Programa de Mestrado.

Abstract

This dissertation analyses the problems involved in the design and implementation of graphical interfaces for object-oriented database management systems (DBMSs). As a result of this analysis, the dissertation presents directives for the development of database system interfaces. The practical application of these directives was illustrated through the specification and implementation of GOODIES – a new interface system, which allows browsing and querying DBMSs that support the basic features of the OO model.

The design and implementation of this new system is described as a case study of the use of the proposed directives. The system development process was purposely conducted independent from any specific DBMS. Thus, it can be used on top of several OO database systems.

Conteúdo

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

1.1 Apresentação

O problema abordado por esta dissertação é o projeto e implementação de interfaces gráficas para Sistemas Gerenciadores de Bancos de Dados Orientados a Objetos. Desta forma, o assunto envolve duas grandes áreas da Ciência da Computação: sistemas de interface com o usuário e sistemas de bancos de dados.

Cada uma destas áreas oferece campo para elaboração de muitos trabalhos. Esta dissertação não pretende cobrir totalmente nenhuma destas áreas de pesquisa. De fato, o interesse principal desta tese é identificar e estudar os pontos comuns e os relacionamentos existentes entre as duas áreas, no sentido de aproveitar os esforços de pesquisa desenvolvidos na área de interfaces gráficas para melhorar os sistemas de bancos de dados existentes, e vice-versa.

Para alcançar o objetivo proposto, esta tese introduz e discute uma nova ferramenta gráfica para navegação e consulta a bancos de dados orientados a objetos. Desenvolvida a partir de conceitos consagrados e largamente utilizados em interfaces existentes para diversos tipos de sistemas, a ferramenta aplica estes conceitos às necessidades apresentadas pelos sistemas de bancos de dados orientados a objetos, com relação ao modo de representação de suas informações, e aos mecanismos de interação com seus usuários.

O restante deste capítulo está organizado da seguinte maneira. A próxima seção resume e comenta o conteúdo desta dissertação. A seção 1.3 apresenta uma visão geral do sistema de interface desenvolvido durante as atividades de tese. A seguir são apresentados conceitos básicos, oferecendo uma visão geral das áreas de pesquisa envolvidas neste trabalho. A seção 1.4 introduz noções básicas sobre *bancos de dados*. Na seção 1.5

é apresentado um modelo para interação entre computador e usuário. A seção seguinte encerra o capítulo, discutindo os principais estilos de interação conhecidos.

1.2 Conteúdo da Dissertação

Esta dissertação está organizada em sete capítulos: um capítulo introdutório; dois capítulos que revisam as áreas de conhecimento abordadas por esta tese; dois capítulos que descrevem o projeto da ferramenta desenvolvida; um capítulo abordando aspectos de implementação da ferramenta; e finalmente um capítulo que apresenta as conclusões obtidas das atividades desenvolvidas.

O primeiro capítulo desta dissertação introduz o problema abordado e oferece subsídios básicos, na forma de definições e conceitos, sobre os assuntos discutidos pela dissertação.

O capítulo 2 trata do modelo de dados orientado a objetos. São apresentadas e estudadas as características básicas deste modelo. Alguns sistemas que implementam o modelo OO são analisados. O capítulo é encerrado com uma discussão crítica sobre o modelo OO.

No capítulo 3 é apresentado um estudo sobre diversos sistemas de interface existentes para sistemas de bancos de dados que seguem os modelos relacional, semântico e orientado a objetos. É feita uma análise comparativa dos sistemas de interface estudados, visando a identificação de seus aspectos positivos e de suas deficiências. Finalmente são introduzidas as características básicas que devem estar presentes em um sistema de interface para bancos de dados.

Os capítulos 4 e 5 apresentam o projeto de um novo sistema de interface para sistemas de bancos de dados orientados a objetos, desenvolvido durante as atividades de pesquisa que resultaram nesta dissertação. O projeto deste sistema se baseou nas características apresentadas no capítulo 3. O modelo externo do sistema trata do relacionamento entre o usuário e o sistema de interface, enquanto o modelo interno gerencia a interação entre o banco de dados e o sistema de interface. Os capítulos 4 e 5 desta dissertação discutem, respectivamente, o modelo externo e o modelo interno do sistema desenvolvido.

A implementação do projeto apresentado nos capítulos 4 e 5 é o tema tratado no capítulo 6, onde são apresentados os recursos computacionais utilizados, os padrões e diretivas seguidas, e os módulos que compõem o sistema. O capítulo 6 mostra, ainda, as técnicas de projeto orientado a objetos empregadas no desenvolvimento do sistema.

O sétimo e último capítulo desta dissertação apresenta as conclusões obtidas do trabalho realizado. É feita uma análise crítica do projeto e da implementação do sistema de

interface apresentado pela dissertação, comparando-o com outros sistemas já existentes. O capítulo termina com a discussão das contribuições alcançadas e com sugestões para extensão e melhoramento do trabalho aqui apresentado.

1.3 Visão Geral da Ferramenta

Apesar do grande número de sistemas existentes, poucas interfaces endereçam os problemas específicos de Sistemas Gerenciadores de Bancos de Dados Orientados a Objetos, como a representação de objetos complexos, a visualização das relações entre objetos e entre classes, e o suporte à navegação sobre esquemas e dados dos BDs.

Esta seção apresenta, em um nível de abstração bem elevado, a abordagem adotada para resolver estes problemas no sistema de interface desenvolvido. O objetivo é oferecer ao leitor uma visão superficial do sistema como um todo. É importante enfatizar que as idéias aqui apresentadas serão formalizadas ao longo desta dissertação, de modo que todo o conteúdo desta seção será detalhadamente discutido nos próximos capítulos.

1.3.1 Visualização e Navegação sobre o Esquema

Ao ativar a ferramenta, o usuário passa a ter acesso a uma janela contendo os bancos de dados disponíveis em um diretório do sistema. Ao selecionar um banco de dados, o usuário tem acesso à lista de classes que compõem o esquema daquele BD. A seleção de um item desta lista causa o aparecimento da janela que descreve a classe selecionada. A figura 1.1 apresenta no canto superior esquerdo uma janela de diretório; no canto inferior esquerdo uma janela de BD; e no lado direito uma janela de classe.

Uma janela de classe contém, basicamente, o nome da classe selecionada, seu tipo, isto é, a definição da composição de seus objetos, suas superclasses, suas subclasses, seus métodos, e uma lista de objetos que pertencem à classe descrita. O usuário pode navegar pelo esquema através das listas de sub/super classes, pode visualizar as definições dos métodos da classe, ou pode navegar pelos dados, selecionando um objeto da classe. A seleção de uma superclasse ou de uma subclasse causa o aparecimento da janela de descrição da classe selecionada.

As subclasses e as superclasses apresentadas na janela de descrição de classe correspondem à definição completa da hierarquia de herança para a classe descrita. Desse modo, são apresentadas não apenas as sub/super classes diretas, mas todos os ascendentes e descendentes da classe na hierarquia de herança. A descrição do tipo e dos métodos da classe também é completa: a descrição do tipo da classe contém os atributos herdados,

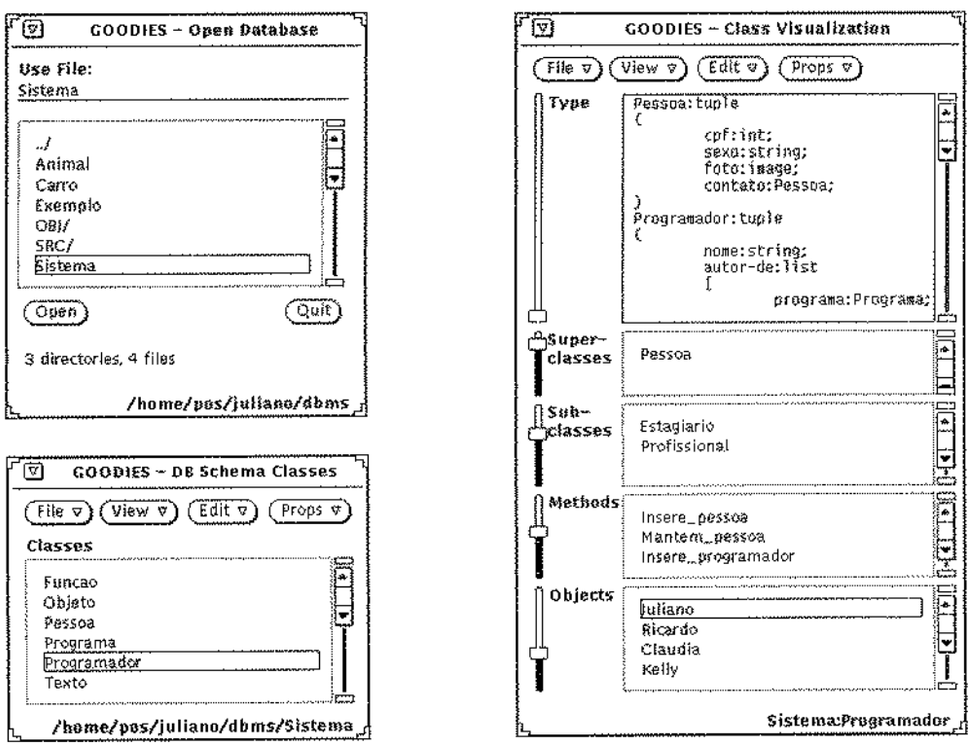


Figura 1.1: Janelas para Visualização de Esquemas

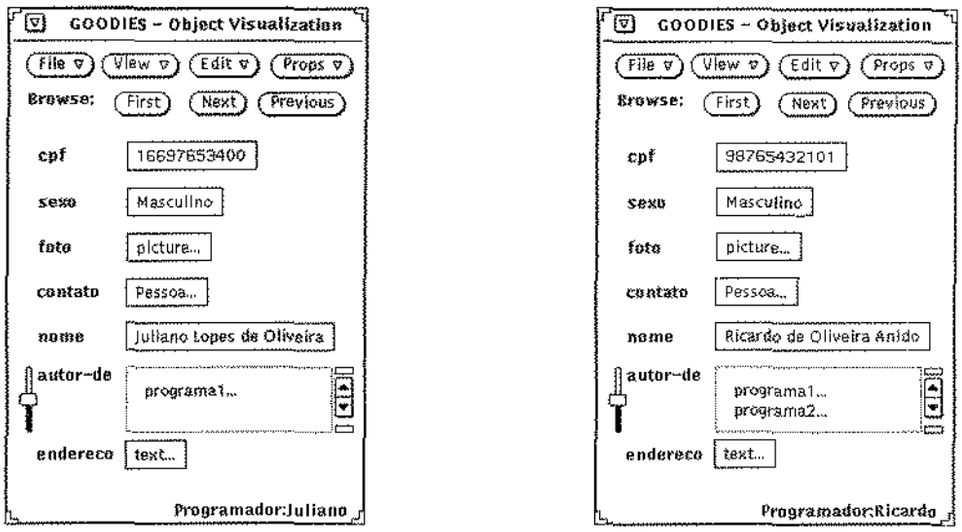


Figura 1.2: Janela de Objeto antes (esquerda) e depois (direita) da operação *Next*

e a lista de métodos da classe inclui os métodos herdados. Existem funções do sistema que permitem que o usuário selecione as superclasses cujos atributos e métodos herdados devam ser visualizados na janela de descrição da classe. Analogamente, os objetos pertencentes a uma subclasse fazem parte da extensão da classe, e o usuário pode selecionar as subclasses cujas instâncias entram na lista de objetos de uma classe.

1.3.2 Visualização e Navegação sobre os Dados

A passagem para navegação sobre dados é feita pela seleção de um objeto na lista de objetos membros da classe. Esta seleção abre uma janela de objeto contendo os atributos do objeto e seus respectivos valores. O valor de um atributo pode ser um objeto ou um conjunto de objetos, e a seleção de um valor deste tipo permite a navegação entre objetos relacionados (objetos complexos).

A janela de objeto permite a navegação sobre os objetos que pertencem a uma mesma classe, através de *operações de seqüenciamento*. Essas operações permitem visualizar o próximo objeto da classe, o objeto anterior, ou voltar ao início da lista de objetos da classe. A figura 1.2 mostra uma janela de objeto representando o objeto selecionado na figura 1.1, antes (esquerda) e depois (direita) da execução da operação de seqüenciamento que busca o próximo objeto da classe.

A janela de objeto possui ainda a capacidade de processar predicados definidos sobre o objeto descrito, aplicando estes predicados na determinação do objeto a ser apresentado, quando ocorre uma operação de seqüenciamento.

A *sincronização* de objetos torna o mecanismo de navegação mais simples. O usuário pode estabelecer ligações entre janelas de objetos, formando uma árvore de sincronização. Uma operação de seqüenciamento aplicada a uma janela de objeto (nó) da árvore de sincronização se reflete em toda a subárvore que tem como raiz o nó onde ocorreu a operação. Desse modo, o usuário pode navegar sobre um número arbitrário de objetos simultaneamente, respeitando-se os predicados definidos para cada janela de objeto pertencente à hierarquia de sincronização. Um objeto só pode ser *pai* de um segundo objeto em uma árvore de sincronização, se o primeiro objeto contém algum atributo que referencia o segundo.

O usuário pode selecionar os objetos que deseja visualizar através dos predicados definidos para as janelas de objeto. De modo semelhante, o usuário pode selecionar as informações que lhe interessam em uma janela de classe ou de objeto, escondendo as que não são importantes. O usuário indica os atributos que devem ser apresentados através de uma janela auxiliar que contém os campos definidos para a classe ou para o objeto.

1.4 Sistemas de Bancos de Dados

1.4.1 Definições

Os *Sistemas de Bancos de Dados* (ou SBDs) foram desenvolvidos para atender a necessidades de registro, manutenção e acesso a informações concernentes a diversas aplicações, como as atividades comerciais, administrativas, educacionais e de pesquisa, para destacar algumas. Um SBD é composto por diversos tipos de elementos distintos, conforme mostra a figura 1.3. Entre esses elementos se destacam os programas e os bancos de dados.

Os *Bancos de Dados* (ou BDs) são coleções de informações que possuem algum tipo de semântica intrínseca, representando aspectos do mundo real. Um BD é projetado e construído com dados de propósito específico, visando um determinado grupo de usuários e um conjunto de aplicações nas quais o grupo tenha interesse [EN89].

Os *programas* de um SBD realizam as tarefas de criação, controle e manutenção dos bancos de dados.O conjunto de programas responsáveis pela criação e controle dos bancos de dados é usualmente denominado *Sistema Gerenciador de Bancos de Dados*, ou simplesmente SGBD. O SGBD facilita: a definição dos BDs, permitindo a especificação de *esquemas*, isto é, dos tipos e estruturas das informações a serem guardadas; a construção

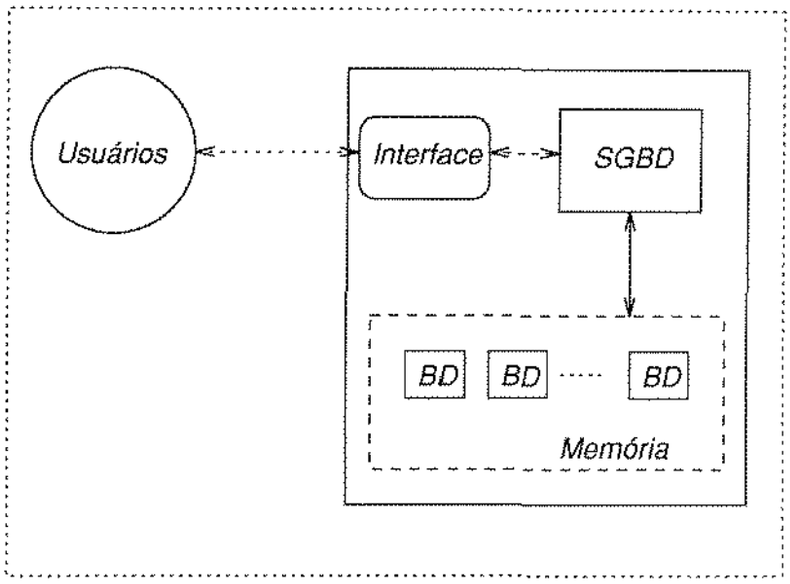


Figura 1.3: Componentes de um Sistema de Bancos de Dados

dos BDs, armazenando os dados na memória do computador; e a manipulação dos BDs, fornecendo funções para consulta e atualização de dados e esquemas.

1.4.2 Modelos de Dados

Um modelo de dados é um conjunto de conceitos usados para descrever a estrutura de um banco de dados. A "estrutura" do banco de dados compreende, entre outras propriedades, os tipos de dados e de relacionamentos suportados pelo SGBD, bem como as restrições que devem ser garantidas sobre estas informações. Dessa forma, o modelo de dados representa as informações contidas nos BDs, em um nível de abstração que elimina detalhes sobre o armazenamento destas informações.

É importante distinguir entre os conceitos de *dado*, que representa as informações armazenadas, e de *esquema*, que se refere à definição da estrutura dos dados armazenados, através de um modelo de dados. Os dados, no caso geral, apresentam um grande número de estados ao longo do tempo, enquanto um esquema, uma vez definido, apresenta certa estabilidade. Espera-se que o tempo médio decorrido entre modificações do esquema seja bastante elevado, em oposição às mudanças de estado dos dados, que, em geral, ocorrem com maior freqüência. É comum utilizar-se o termo *extensão* denotando os dados de um

banco de dados.

Os modelos hierárquico, rede e relacional formam a abordagem tradicional dos modelos de dados, e já foram exaustivamente discutidos na literatura sobre bancos de dados ([Dat86] e [EN89], por exemplo, fazem um estudo completo desses três modelos de dados). Muitos pesquisadores consideram os sistemas hierárquicos e CODASYL como representantes da primeira geração de SBDs, enquanto os sistemas relacionais formariam a segunda geração de sistemas de bancos de dados. Na verdade, o modelo relacional superou os modelos da primeira geração e se tornou o modelo de dados mais difundido em todo o mundo. O ponto em comum entre os modelos tradicionais é que todos eles são direcionados para aplicações convencionais, ou seja, aplicações tipicamente comerciais, que não envolvem *tipos não estruturados de dados*[1]. Além disso, os modelos tradicionais não possuem construtores que possibilitem a modelagem direta da semântica das aplicações.

Os modelos semânticos surgiram para oferecer ao projetista de BDs mecanismos mais poderosos para a representação da estrutura dos bancos de dados. A característica comum a todos os modelos semânticos propostos é a tentativa de prover mais conteúdo semântico que os modelos tradicionais. Entre os modelos semânticos se destacam o modelo ER [Che76], o modelo funcional [Shi81], e o modelo SDM [HM81]. As principais características introduzidas pelos modelos semânticos são a representação de tipos de dados não estruturados; a presença de mecanismos de abstração, como a generalização e a agregação; o suporte ao conceito de herança/derivação; e a possibilidade de construção de redes ou hierarquias de relacionamentos. Em [HK87] e [PM88] são apresentados e discutidos diversos modelos semânticos existentes.

O modelo de dados orientado a objetos (modelo OO) representa uma evolução no processo de modelagem de dados. O modelo OO apresenta um poder de expressão muito superior ao dos modelos tradicionais, tendo surgido da combinação de conceitos de modelos semânticos e de linguagens de programação orientadas a objetos. Dessa forma, enquanto os modelos semânticos só provêem mecanismos para abstração estrutural, o modelo OO oferece mecanismos para abstração estrutural e comportamental. Por suas capacidades adicionais, o modelo OO pode suportar uma vasta gama de aplicações onde a tecnologia tradicional de SGBDs provou ser inadequada [Cat91]. Estas novas aplicações, como os sistemas CAD (*Computer Aided Design*) e CASE (*Computer Aided Software Engineering*), exigem suporte efetivo à gerência de objetos complexos[2], e possivelmente *"multimeios"*[3]. O capítulo 2 desta dissertação analisa as características e propriedades do modelo de dados orientado a objetos.

[1]Tipos estruturados são os que aparecem na maioria das linguagens de programação, como *inteiro*, *caractere* e *real*. Em oposição, são exemplos de tipos não estruturados: *textos*, *imagens* e *sons*.

[2]Objetos compostos por outros objetos.

[3]Objetos que englobam tipos de dados estruturados e não estruturados, como seqüências de imagens e sons.

1.5 Sistemas de Interface com o Usuário

A importância dos fatores humanos nos sistemas de computadores tem aumentado à medida que os projetistas observam que a facilidade de uso contribui fortemente para o consumo de seus produtos. Por essa razão, nota-se uma clara preocupação em melhorar o relacionamento entre usuário e computador, na mesma proporção em que se busca aumentar a funcionalidade e otimizar o desempenho das máquinas e dos programas.

No sentido de conseguir um melhor relacionamento entre usuário e computador, avanços têm sido obtidos no entendimento do processo cognitivo do usuário [PH91], no desenvolvimento de padrões e diretivas para auxiliar os projetistas de interface [Shn87, Sun90], na construção de modelos que representam a interação entre homem e computador [MvD91, Nor91], e na implementação de sistemas gerenciadores de interface que permitem uma eficiente tradução de especificações e modelos em sistemas de interface prontos para utilização [HH89, Mra91].

É notório que a interação entre homem e computador deve abranger dois campos de estudo totalmente distintos, que são o comportamento humano e o processamento computacional. Os sistemas de interface lidam com esses dois domínios, e se baseiam, em sua maioria, em um ponto comum entre homem e computador, que é o fluxo de informações. O modelo de interface apresentado a seguir parte desse ponto em comum para representar a realidade existente na interação entre usuário e computador.

1.5.1 Modelo de Interface Homem–Computador

Para realizar uma tarefa qualquer, é necessário um fluxo de informação entre o usuário e o sistema de computador. No caso geral, o usuário precisa receber informações sobre o estado e o andamento da tarefa, enquanto o sistema necessita receber comandos e parâmetros do usuário para realizar o processamento desejado. A figura 1.4, adaptada de [Nor91], ilustra os principais aspectos envolvidos em um sistema de interface.

No modelo de interface representado pela figura 1.4, todos os componentes (homem e computador) estão envolvidos por um ambiente determinado pela tarefa a ser executada. Esta tarefa pode ser, por exemplo, a recuperação de informações de um banco de dados, ou a monitorização de um processo industrial. O fato é que a tarefa estabelece um contexto que influencia enormemente a interação entre o usuário e o computador, determinando fatores como, entre outros, a importância do tempo de resposta e o custo de recuperação de erros. Desse modo, o próprio ambiente produz um fluxo de informações que afeta o usuário, e em certos casos, o próprio computador (por exemplo, na tarefa de controle automático de temperatura de uma caldeira).

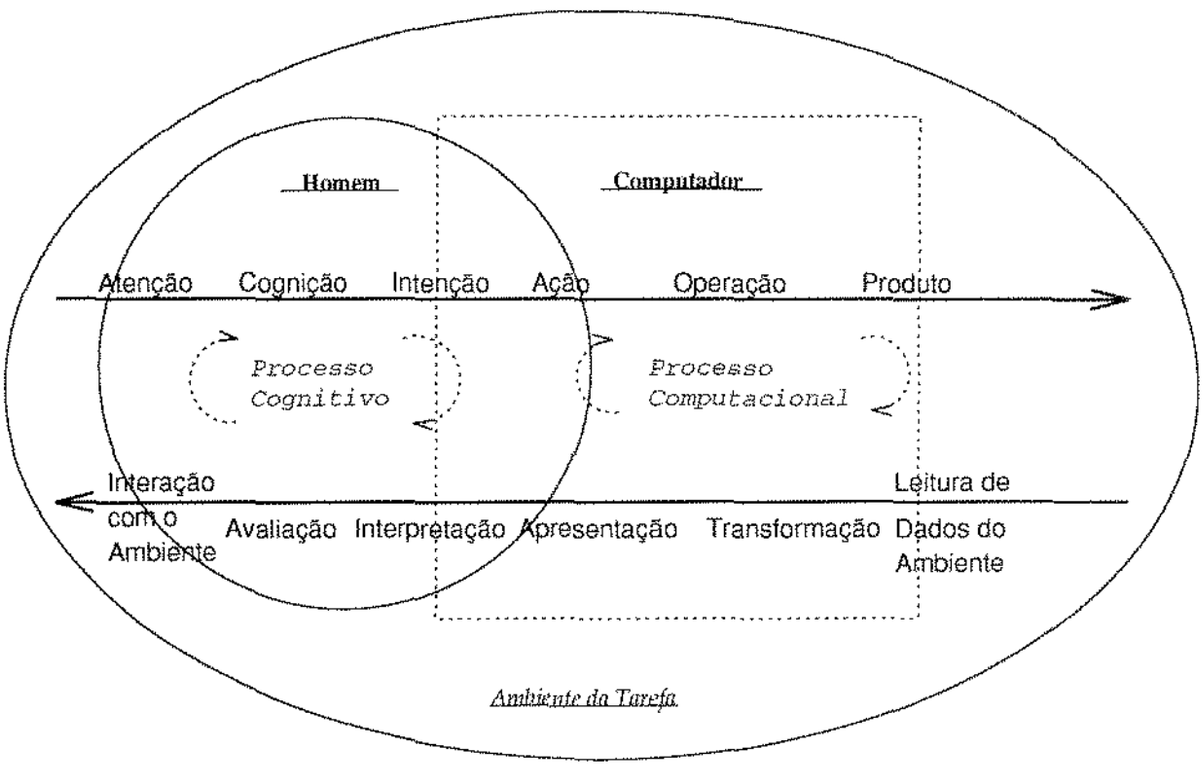


Figura 1.4: Interface Homem–Computador

Na figura 1.4, as atividades que se situam dentro do círculo (que representa o homem), mas fora do quadrado (que simboliza o computador), representam processos cognitivos que dizem respeito ao estudo do comportamento humano. De modo semelhante, as atividades que pertencem exclusivamente ao quadrado (computador) representam processos computacionais que não dizem respeito à área de interfaces.

As pesquisas em interface estudam os processos que se situam na intersecção entre o círculo e o quadrado. Entre estes processos, pode-se citar o mapeamento de intenções em atividades de entrada de dados, e a interpretação de informações apresentadas na tela, sob o ponto de vista do usuário. Relacionados ao computador estão, por exemplo, os processos de conversão de estruturas internas em apresentações compreensíveis para o usuário, e a tradução de entradas de dados em representações internas adequadas.

No modelo de interface apresentado, dois fluxos de informação se destacam. O primeiro fluxo se origina do ambiente da tarefa e caminha do usuário para o computador. O usuário analisa o contexto da tarefa e extrai informações que são processadas através de cognição, resultando em uma intenção que é passada à interface por meio de um comando ou ação do usuário. O fluxo continua, pois a ação do usuário causa uma operação do computador para interpretar o comando e produzir o resultado adequado.

O segundo fluxo de informação e controle parte do ambiente monitorado pelo computador (no caso de o computador possuir sensores que recebem dados do ambiente). O computador recebe as informações do ambiente e as transforma para uma representação interna. Em seguida as informações são processadas e apresentadas, em formato adequado, para o usuário. O fluxo segue com a interpretação da informação pelo usuário, que pode então tomar as atitudes convenientes, de acordo com a sua avaliação das informações apresentadas.

1.5.2 Caracterização de Sistemas de Interface

Sob o ponto de vista do usuário, os sistemas de interface podem ser caracterizados de diversas maneiras, entre as quais se destacam: a complexidade, a interatividade e o estilo de interação.

A *complexidade* de uma interface pode ser medida pela riqueza (características, variedade, volume, tipo) das informações fornecidas pelo computador para solicitar entradas do usuário, e pela forma com que as ações desejadas são transmitidas ao computador pelo usuário. O número de funções disponíveis e a sua adequação à tarefa a ser realizada também influenciam a complexidade da interface. Para ser útil, um sistema de interface deve prover um amplo conjunto de funções. É tarefa do projetista de interface prover um grande número de funções (que garantam a utilidade da interface) sem au-

mentar demasiadamente a complexidade do sistema. Interfaces baseadas em linguagens de comandos são poderosas, mas em geral complexas, enquanto os sistemas de menus apresentam capacidades mais limitadas, porém com uma complexidade muito reduzida.

Outro importante fator que caracteriza um sistema de interface é a sua *interatividade*. A interatividade de um sistema de interface é medida pela taxa de informações intercambiadas entre usuário e interface, levando-se em conta, ainda, a granularidade das informações comunicadas. Uma interface com baixa interatividade pode requerer conjuntos completos de comandos antes de qualquer processamento, enquanto uma com alta interatividade pode interpretar e processar comandos à medida que eles são informados. Interfaces gráficas possuem, em geral, alta interatividade, ao passo que interfaces textuais tendem a apresentar baixa interatividade.

Apesar da importância de aspectos como complexidade e interatividade, o principal fator que caracteriza um sistema de interface é, sem dúvida, o seu *estilo de interação* (ou *tipo de diálogo*) com o usuário. A próxima seção discute os principais tipos de diálogo que se aplicam nos sistemas de interface existentes na atualidade.

1.6 Estilos de Interação Homem–Computador

Os tipos de diálogo utilizados nos sistemas de interface podem ser agrupados, de um modo geral, em quatro grupos: linguagens, menus, telas formatadas e mecanismos de manipulação direta.

Embora existam protótipos que empreguem outros paradigmas de interação, como, por exemplo, interfaces dirigidas pelo olhar e sistemas de reconhecimento de voz, a grande maioria dos sistemas de interface em uso/produção se enquadra em um dos grupos citados acima.

Cabe observar ainda que alguns sistemas existentes não utilizam um único tipo de diálogo, mas sim uma combinação de propriedades dos quatro principais estilos de interação. Descrever-se-á, a seguir, os aspectos mais importantes de cada um desses estilos de interação.

1.6.1 Linguagens

As linguagens formam o mais antigo e mais utilizado estilo de interação entre homem e computador. Entretanto, com o desenvolvimento de novos tipos de diálogo, as linguagens tendem a perder esta supremacia, pelo menos em alguns tipos de aplicação, como, por

exemplo, navegação em bancos de dados. Para efeito de análise das características das linguagens como paradigma de interação com o usuário, elas foram divididas em três grupos distintos.

Linguagem de Comando

Diversas particularidades das linguagens de comando as tornam desejáveis, não tanto como instrumento único de interação, mas como componentes de modos mistos de diálogo. As estratégias desenvolvidas pelos estudiosos de linguagens para dar nomes adequados aos comandos influenciaram fortemente outros tipos de diálogo, como os sistemas de menus, por exemplo. Portanto, os princípios de projeto de uma linguagem de comando podem ser estendidos e utilizados em qualquer estilo de interação onde nomes sejam usados para representar opções ou ações.

As linguagens de comando parecem ser mais indicadas para usuários que possuem um bom conhecimento do sistema em que eles atuam. Neste caso, as linguagens de comando oferecem um modo rápido e eficiente de acesso à funcionalidade do sistema. Além disso, essas linguagens dão poder ao usuário, no sentido de permitir a realização de operações relativamente complicadas através de uma seqüência de comandos relativamente pequena. Por essas razões, é provável que as linguagens de comando permaneçam como o tipo de diálogo preferido de um grande número de usuários experientes.

Para usuários novatos ou casuais, no entanto, as linguagens de comando são difíceis de aprender, requerem um esforço razoável de prática e memorização, e levam a altas taxas de erros. Há que se levar em conta, porém, que estas desvantagens diminuem em quantidade e importância, à medida em que cresce a freqüência e a duração de uso do sistema.

O uso de abreviações e sinônimos contribui para o aumento da versatilidade das linguagens de comando. As abreviações apresentam ainda as vantagens de economia no tempo de digitação e no espaço de tela necessário para a entrada de dados. Muitos dos estudos iniciais sobre interação homem-computador abordaram as questões relacionadas à formação de nomes de comandos e de abreviações [PH91].

Um dos princípios comumente aceitos com relação ao projeto de linguagens de comando é o princípio de consistência na organização das regras léxicas. Levando-se em consideração que para um usuário comum é difícil ter em memória todos os nomes de um conjunto de comandos, fica evidente a importância do desenvolvimento de um conjunto de regras que permitam ao usuário deduzir o nome de um comando. Estas regras devem valer também para as abreviações. Portanto, os nomes usados em uma linguagem de comando devem ser gerados em conjunto, com base em um conjunto de regras consistentes, e não

individualmente, de maneira *ad hoc*. A tarefa principal de um projetista de linguagem de comando é produzir um conjunto de nomes cujas operações sejam entendidas de modo claro e simples pelo usuário, e ao mesmo tempo que sejam fáceis de serem lembrados e/ou deduzidos.

Linguagem de Consulta

A presente discussão considera como linguagens de consulta o subconjunto das linguagens de comando formado pelas linguagens de propósito especial usadas somente para recuperação de informações em sistemas de bancos de dados.

A discussão em separado das linguagens de consulta se justifica por três razões. Em primeiro lugar, essas linguagens são o tipo mais freqüentemente utilizado de interface para SGBDs, e o objetivo desta dissertação é estudar estas interfaces. A segunda razão provém do fato de as linguagens de consulta serem empregadas em um só tipo de aplicação, permitindo um estudo mais aprofundado do ambiente da tarefa, o que não é possível nas linguagens de comando genéricas. Finalmente, a comunidade de usuários a que se destina uma linguagem de consulta é formada tanto por usuários experientes quanto por iniciantes, ao passo que os usuários de linguagens de comando genéricas são, na maioria dos casos, profissionais ligados à área de computação.

As linguagens de consulta diferem entre si em três aspectos: paradigma de programação, modelo de dados e forma sintática. O aspecto *paradigma de programação* identifica se a linguagem de consulta é procedural ou não. Em uma linguagem procedural, o usuário deve especificar o procedimento de busca aos dados armazenados nos BDs. Em oposição, em uma linguagem de consulta não-procedural o usuário precisa informar apenas as características do conjunto de dados que deve ser recuperado, deixando que o sistema encontre um algoritmo eficiente para buscar os dados solicitados.

O modelo de dados do SGBD é outro aspecto que diferencia as linguagens de consulta. Como foi visto na seção 1.4, diferentes modelos de dados trabalham com diferentes estruturas de dados, de forma que a linguagem de consulta deve refletir as estruturas usadas no modelo de dados empregado pelo SGBD.

O terceiro fator de diferenciação entre as linguagens de consulta é a forma sintática. A sintaxe utilizada em diferentes linguagens de consulta varia em termos de *dimensão* e de *notação*. Duas dimensões são usadas em um grande número de linguagens de consulta: linear e tabular. Duas linguagens de consulta bastante populares, SQL e QBE, representam exemplos de utilização de sintaxe linear e tabular, respectivamente. O aspecto notação da forma sintática também possui duas variantes básicas: a notação posicional e a notação por palavra-chave.

O projeto de uma linguagem de consulta deve levar em consideração a possibilidade de o usuário da linguagem ser inexperiente no uso de computadores. Dessa forma, uma linguagem de consulta deve prover uma retro-alimentação de informações e um serviço de auxílio ao usuário bem mais amigável que aqueles oferecidos pelas linguagens de comando [PH91]. É importante informar ao usuário a atividade que está sendo desenvolvida pelo sistema, em especial para sistemas lentos ou para consultas complexas, que podem ter um tempo de execução prolongado. Este tipo de informação pode ajudar também na formação, por parte do usuário, de um modelo conceitual do comportamento do sistema.

Linguagem Natural

Uma interface em linguagem natural tenta estabelecer um tipo de diálogo que se assemelhe o mais possível à maneira habitual pela qual os seres humanos se comunicam. Todos os outros tipos de sistemas de interface impõem ao usuário a aprendizagem de novos conceito de diálogo, o que limita a utilização desses sistemas a usuários que tenham tempo e conhecimento disponíveis para aprender estes estilos artificiais de comunicação. Uma interface que implementasse, de maneira completa, uma linguagem natural possibilitaria, portanto, que pessoas leigas pudessem utilizar os serviços de um sistema de computador, sem a necessidade de qualquer tipo de treinamento prévio.

Os benefícios que uma interface em linguagem natural pode trazer atraíram pesquisadores no mundo inteiro. Apesar disso, nenhum sistema implementado até hoje conseguiu apresentar uma interface suficientemente poderosa para aceitar e compreender as inúmeras combinações possíveis de formas sintáticas e semânticas existentes em um idioma. A maioria das palavras e das frases de uma língua, como inglês ou português, possuem diversos significados, dependendo do contexto para a sua correta interpretação.

Devido a estas enormes dificuldades para a implementação de um sistema completo de interface em linguagem natural, os sistemas existentes atualmente limitam-se a um domínio específico de tarefa. O contexto envolvendo a tarefa é embutido no sistema, através de técnicas de representação de conhecimento.

Uma aplicação típica em que pode ser empregado este tipo de interface é a recuperação de dados em SGBDs. Neste domínio, as interfaces em linguagem natural se relacionam intimamente com as linguagens de consulta. Em muitos sistemas, a consulta efetuada em linguagem natural pelo usuário é transformada para a sintaxe própria da linguagem de consulta do SGBD, para posterior processamento. Pode-se considerar, neste caso, a linguagem natural como uma linguagem de consulta mais flexível. No entanto, os estudos que comparam a utilização de linguagem natural com linguagens de comando e outros tipos de interface para SGBDs (por exemplo, [BR92]), não apontam uma tendência clara

de obtenção de melhores desempenhos por parte dos usuários com o uso de linguagem natural.

1.6.2 Sistemas de Menus

Em termos de percentagem de utilização, os sistemas de menus perdem apenas para as linguagens de comando como método de interação com o usuário. Um menu pode ser definido como um conjunto de opções apresentadas para o usuário, onde a seleção de uma opção resulta na modificação do estado da interface. As opções de um menu podem ser representadas por números, palavras e ícones, entre outras formas.

A interação entre o usuário e um sistema de menu inicia-se com a intenção de realizar uma ação. Uma vez decidida a ação a ser tomada, o usuário irá observar uma opção do menu, reconhecendo a função que ela representa, e compará-la com a sua intenção inicial. O usuário então decide se a função apresentada no menu corresponde à ação desejada, e neste caso, seleciona a opção; caso contrário ele passa a considerar a próxima opção do menu, repetindo o processo anterior. O tempo gasto para cada opção envolve, desta forma, os tempos necessários para codificação, comparação e decisão.

Existem duas características no processo de pesquisa de um menu que influenciam o desempenho do usuário. A primeira é o processo de busca visual, fortemente influenciada pela classificação adotada para as opções do menu. A segunda característica é o tipo de comparação realizada entre a opção codificada e a intenção do usuário. Existem, basicamente, três tipos de comparação. Primeiro, o usuário pode estar buscando um alvo específico (uma dada palavra, ou um determinado ícone, por exemplo) entre as opções do menu. O segundo tipo de comparação ocorre quando o usuário busca enquadrar a sua intenção em uma classe de opções. Este tipo de busca ocorre, notadamente, nas hierarquias mais altas do sistema de menu. A terceira maneira pela qual o usuário compara opções e intenções é através de casamento ou equivalência de funções. Neste caso, o usuário sabe a função que deseja executar, mas não sabe como esta função é chamada no sistema. Ao contrário do tipo de comparação anterior, este tipo se aplica aos níveis mais baixos da hierarquia de menus.

Embora alguns estudos sobre desempenho de usuários de menus tenham apresentado resultados contraditórios, algumas vantagens e desvantagens parecem ser inerentes aos sistemas de menus. Como vantagens pode-se destacar a apresentação de todas as opções possíveis, livrando o usuário da tarefa de memorização de comandos. Para usuários inexperientes, os sistemas de menu servem como guias, sugerindo caminhos possíveis e reduzindo a possibilidade de erros. As desvantagens dos sistemas de menu são maiores para usuários experientes. A navegação por um longo caminho de menus é cansativa e con-

some um espaço de tela muito maior que o espaço requerido para a entrada de nomes de comandos, por exemplo. O uso de menus, portanto, parece ser indicado para sistemas que oferecem relativamente poucas funções, e que precisam oferecer fácil acesso para usuários novatos.

1.6.3 Telas Formatadas

Este tipo de diálogo refere-se ao tipo de interação onde o usuário se restringe a completar os campos livres de telas pré-formatadas, num processo análogo ao de preenchimento de formulários de papel. O emprego de telas formatadas se dá tipicamente em aplicações de entrada de dados, e em aplicações de manipulação de informações comerciais. É um tipo de diálogo pouco indicado para outros tipos de aplicação, devido à sua natureza extremamente limitada.

Mesmo em aplicações de entrada de dados, já existem interfaces mais eficientes, como por exemplo as leitoras de códigos de barra. Entretanto, o uso de telas formatadas ainda é intenso, devido às vantagens apresentadas, notadamente no que se refere a usuários novatos ou casuais. Entre estas vantagens se destacam a simplicidade de entrada de dados, o reduzido conhecimento necessário para realizar o diálogo, e a facilidade de reconhecimento do contexto do sistema.

1.6.4 Manipulação Direta

O desenvolvimento das capacidades gráficas dos computadores, associado à evolução dos dispositivos de entrada, notadamente dos *"mouses"*, possibilitou o surgimento de um novo estilo de interação, que representou um considerável avanço em relação às técnicas tradicionais de diálogo. Trata-se do paradigma de manipulação direta de objetos, introduzido por [Shn83].

A manipulação direta é um estilo de interação que se baseia na movimentação de objetos gráficos. Esta movimentação é realizada pelo usuário, de acordo com seus conhecimentos de espaço, geometria, movimento, e de significado de objetos familiares. Objetos gráficos são representados por ícones, que servem a dois propósitos: economia de espaço em tela e auxílio à memória do usuário.

O mecanismo de interação pode ser assim sintetizado: o usuário analisa o estado corrente dos objetos representados graficamente, formulando uma ação. Esta ação é interpretada pelo sistema de interface, e se ela levar a um estado consistente, é executada. Caso contrário, os objetos são apresentados no seu estado original, e o usuário é avisado

sobre o problema ocorrido.

Muitos sistemas existentes utilizam o mecanismo de interação descrito acima, estendendo-o, porém, com facilidades de "manipulação indireta". Em geral, estas facilidades são providas sob a forma de ações (em oposição a objetos) selecionadas a partir de *pop-up menus* ou de botões de comando.

As principais características do paradigma de manipulação direta são:

- A representação contínua de objetos de interesse;
- A substituição da sintaxe complexa das linguagens por ações físicas (como seleção e "arrastamento") que afetam os objetos representados;
- A utilização de operações rápidas, incrementais e reversíveis, cujo impacto nos objetos é imediatamente visível para o usuário.
- A abordagem gradual para aprendizagem, permitindo o uso do sistema com um mínimo de conhecimentos, e a expansão da capacitação do usuário através do próprio uso do sistema.

O paradigma de manipulação direta representa uma evolução dos estilos de interação anteriores, tomando destes as principais vantagens, e procurando eliminar os pontos negativos. Como as linguagens de comando, a manipulação direta permite a realização de operações complexas com um reduzido conjunto de comandos (ações). No entanto o usuário não precisa memorizar uma sintaxe complexa, mas sim uma metáfora bastante utilizada pelos seres humanos, que é a movimentação de objetos. Dos sistemas de menus, o paradigma de manipulação direta utiliza o conceito de apresentação de opções na tela. As opções, neste caso, são representadas pelos objetos gráficos. A vantagem é que o fluxo de controle não se limita a uma única ação por vez; o usuário pode realizar operações complexas em paralelo.

A principal dificuldade para o projeto de um sistema de manipulação direta é a obtenção de um modelo gráfico que represente a realidade da tarefa de maneira adequada. Ainda que se consiga um modelo apropriado da realidade, existe o problema de aprendizagem dos componentes do modelo por parte do usuário. Um ícone, embora significativo para o projetista, pode ser tão difícil de assimilar quanto uma nova palavra. Há ainda o risco de o usuário entender a representação analógica do modelo, mas tirar conclusões errôneas sobre as operações possíveis.

Apesar destas dificuldades, os benefícios conseguidos com o uso dos princípios de manipulação direta são muitos. Entre eles pode-se destacar: a facilidade de aprendizagem

através de uma simples demonstração de uso do sistema; a facilidade de retenção de conceitos operacionais; a verificação imediata dos efeitos das ações; a diminuição da ansiedade e o aumento da confiança do usuário no sistema, já que o usuário inicia as ações, sente que as controla, e prevê as respostas possíveis do sistema.

O sistema de interface apresentado nesta dissertação se aplica, como já foi dito, a SGBDs orientados a objetos, e utiliza como principal meio de interação com o usuário o paradigma de manipulação direta.

Capítulo 2

Sistemas de Bancos de Dados Orientados a Objetos

2.1 Introdução

Os sistemas de bancos de dados relacionais se tornaram um dos métodos mais utilizados para armazenamento e manipulação de informações, principalmente em aplicações comerciais. O modelo relacional superou seus modelos predecessores por ser mais flexível e mais fácil de usar. A flexibilidade do modelo relacional está ligada ao fato de que relacionamentos entre registros não precisam ser definidos *a priori*, já que o operador junção (*join*) permite o relacionamento dinâmico entre registros, com base nos valores de seus atributos. A facilidade de uso é garantida pelo emprego de uma única estrutura para representação de informações (tabela ou relação), e pela disponibilidade de linguagens não-procedurais que operam sobre este tipo de estrutura [SZ87].

No entanto, o modelo de dados relacional, assim como os demais modelos tradicionais, apresenta sérias limitações com relação às necessidades de um grande conjunto de aplicações que vêm surgindo continuamente. Estas novas aplicações incluem projetos de engenharia (CAD/CAM), bancos de dados gráficos, de imagens, cartográficos e geológicos, e representação de conhecimentos relacionados a todos os tipos de atividades humanas (medicina, música, astronomia, etc).

Para atender aos requisitos destas novas aplicações, é necessário um modelo de dados muito mais expressivo e flexível que o modelo relacional, e é essa a proposta do modelo de dados orientado a objetos. Estendendo os modelos semânticos, o modelo OO provê conceitos de modelagem tanto da estrutura quanto do comportamento das informações

armazenadas. Dessa forma, ele permite a criação e a representação de estruturas de dados complexas, de maneira uniforme e coerente.

O suporte a estas estruturas complexas exige avanços não apenas do modelo de dados, mas também do seu sistema de gerenciamento. Por isso, os sistemas de bancos de dados que tentaram oferecer as capacidades do modelo OO através de mecanismos de gerenciamento de dados do modelo relacional não obtiveram sucesso. A principal razão do insucesso dos sistemas de bancos de dados construídos como camadas adicionais aplicadas a SGBDs relacionais foi o fraco desempenho apresentado por estes sistemas híbridos [BM91].

Sob o ponto de vista do modelo de dados, um SGBDOO deve suportar não só a definição estrutural de objetos complexos e de relacionamentos entre objetos, mas também a modelagem do comportamento dos objetos. Com relação ao gerenciamento de objetos, os sistemas OO devem levar em consideração problemas como controle de concorrência (sobre objetos e seus componentes) [CRR91], evolução de esquemas (incluindo herança) [Zic91, BCG+87] e controle de versões [Kat90, LF91], entre outros.

Este capítulo aborda os aspectos relacionados ao modelo de dados OO, dando pouca ênfase aos detalhes de implementação. Para o leitor interessado no estudo de problemas existentes para a implementação dos conceitos relacionados ao gerenciamento de objetos, os trabalhos referenciados no parágrafo anterior servem como ponto de partida.

Na próxima seção são discutidas as características fundamentais do modelo de dados orientado a objetos. A seção 2.3 apresenta um conjunto de funções que devem estar presentes para que um SGBDOO seja considerado funcionalmente completo. Na seção 2.4 são analisados os modelos de dados de alguns SGBDOOs existentes, e a seção 2.5 encerra este capítulo com uma análise crítica do modelo de dados orientado a objetos.

2.2 Características do Modelo OO

Apesar do grande número de SGBDOOs já implementados, não existe na comunidade científica de pesquisadores sobre bancos de dados uma definição formal unanimemente aceita para o modelo de dados orientado a objetos. Recentemente, diversos trabalhos propuseram características básicas que devem estar presentes para que um sistema seja considerado "orientado a objetos". Dentre estes trabalhos pode-se destacar as propostas contidas em [ABD+89], [Com90], [Cat91], [Jac91] e em [BM91]. São descritas, a seguir, as características que representam um consenso entre as diversas propostas, e que são fundamentais para um sistema de bancos de dados orientado a objetos:

1. Possuir as características básicas de um *SGBD completo.*

   Para ser um SGBD completo, é necessário que o sistema suporte os conceitos de *persistência, gerenciamento de memória secundária, recuperação de falhas, facilidade de consulta "ad hoc"*, e *concorrência.*

   A persistência é a habilidade de os dados serem mantidos após a execução de um processo, para poderem ser utilizados por processos subseqüentes. O usuário não deve ser obrigado a mover ou copiar um dado explicitamente para torná-lo *persistente.* De maneira semelhante, o usuário não deve se preocupar com os mecanismos utilizados pelo sistema para o controle da memória secundária. Além disso, no caso de falhas, o sistema de BD deve ser capaz de recuperar-se, retornando, após a correção da falha, a um estado consistente.

   Alguns autores consideram essencial a presença de uma linguagem de consulta não-procedural [Com90]. Para outros, a facilidade de consulta *ad hoc* não precisa ser oferecida através de uma linguagem de consulta, mas por qualquer mecanismo que apresente uma funcionalidade equivalente [ABD+89]. O sistema deve, ainda, suportar o conceito padrão de *atomicidade* de transação e de acesso compartilhado. Pode ser utilizado o conceito de *serialização*, embora alternativas menos rígidas possam ser oferecidas.

2. Suportar *objetos complexos* e *identidade de objetos.*

   Cada entidade do mundo real é modelada por um objeto. O sistema mantém um identificador único[1] para cada objeto, conhecido como *oid*[2]. Um oid é imutável e independe dos valores dos atributos do objeto que ele representa. O conceito de oid estabelece duas noções de equivalência entre objetos: dois objetos podem ser idênticos, se possuem o mesmo identificador, ou podem ser iguais, se possuem o mesmo valor. Através dos oids é possível o compartilhamento de subobjetos e a construção de redes genéricas de objetos.

   Objetos complexos são compostos por outros objetos, sendo que a regra de composição permite qualquer combinação de um conjunto pré-determinado de operadores (tuplas, listas, vetores, conjuntos, entre outros). O estado de um objeto complexo é composto por valores atômicos (por exemplo, inteiros e caracteres) e por estados de outros objetos. Assim, um objeto complexo pode fazer referência a um número arbitrário de objetos, inclusive de forma recursiva. No entanto, as referências não possuem, por definição do modelo, qualquer semântica especial. Sem embargo, relacionamentos (como agregação ou associação) podem ser inferidos a partir das referências existentes entre objetos complexos e seus subobjetos.

[1] A proposta apresentada em [Com90] traz uma definição diferente, onde o identificador de objeto só é gerado pelo sistema na ausência de uma chave primária para o objeto.

[2] Oid é uma abreviatura do termo inglês *object identifier* (identificador de objeto).

3. Prover *encapsulamento.*

   Um objeto possui dois componentes: interface e implementação. A interface é pública e contém a especificação do conjunto de operações que podem ser executadas sobre o objeto. A implementação de um objeto é privada, e possui dados que representam o objeto, além de procedimentos (métodos) que descrevem a implementação de cada operação da interface do objeto. Assim, o objeto encapsula tanto os dados quanto os programas que manipulam estes dados.

   O estado de um objeto é representado pelos valores de seus atributos. A única maneira de se consultar ou modificar este estado é pelo envio de mensagens ao objeto, indicando o método que se deseja aplicar. A proposta de [Com90] introduz um encapsulamento menos rígido, onde a linguagem de consulta do BD pode fazer acesso direto aos atributos de um objeto.

   A implementação de um objeto pode ser modificada sem afetar a sua interface. Desse modo, o encapsulamento provê independência lógica dos dados, à medida que permite modificação de operações sem modificar qualquer programa que utiliza tais operações.

4. Suportar o conceito de *classe*, e permitir *herança* e *hierarquia* de classes.

   A classe é um mecanismo de abstração que representa a definição de um conjunto de objetos. Objetos que compartilham a mesma estrutura (tipo) e comportamento (métodos) são agrupados em classes. Uma classe descreve, desse modo, os atributos e métodos definidos para os objetos que pertencem a ela. Cada objeto pertence a alguma classe, isto é, todo objeto é *instância* de uma classe[3]. Portanto, as instâncias de uma classe possuem a mesma estrutura e comportamento .

   Uma classe pode ser definida como uma especialização de uma ou mais classes, estabelecendo uma hierarquia de classes. A classe definida como especialização de outra é chamada subclasse desta, e herda atributos e métodos de sua superclasse (ou seja, da classe que ela especializa). O modelo OO suporta o conceito de herança múltipla, isto é, uma classe pode ser herdeira de diversas superclasses[4]. Um objeto é instância de sua classe, e é considerado membro de todas as suas superclasses. Um membro de uma classe, desse modo, pode possuir outras propriedades (atributos e métodos) além daquelas definidas pela classe.

   Com o conceito de herança, é possível modificar as definições de tipos e de comportamentos de maneira incremental, adicionando ou alterando definições que modificam a classe original. A herança e a instanciação são considerados os mecanismos de reutilização mais importantes do modelo OO.

[3]Há modelos que permitem que um objeto pertença a mais de uma classe [BM91].

[4]De acordo com a proposta de [ABD+89], apenas a herança simples é essencial. A herança múltipla seria uma característica opcional do modelo OO.

5. Permitir *sobreposição* e *acoplamento tardio*.

   O uso do mesmo nome de operação para denotar diferentes operações é denominado sobreposição de operações. O significado da operação só é determinado quando ela é aplicada sobre um determinado objeto. Em sistemas OO, a sobreposição de operações ocorre quando métodos de diferentes tipos de objetos possuem o mesmo nome. Um exemplo de sobreposição seria a definição de um método na classe e na subclasse que a especializa.

   Para que a sobreposição seja possível, é necessário que a ligação do nome da operação com seu respectivo código seja feita em tempo de execução, ou seja, o sistema deve permitir o acoplamento tardio (*late binding*). Dessa forma, uma mensagem deve primeiramente determinar o tipo do objeto ao qual ela foi aplicada, para em seguida fazer a chamada do método apropriado.

6. Prover *extensibilidade* e ser *computacionalmente completo*.

   Além de oferecer tipos pré-definidos como inteiro, caractere e real, um SGBD deve poder ser estendido pela definição de novos tipos, a partir daqueles já existentes. O sistema não deve fazer distinção na utilização de tipos pré-definidos e de tipos definidos pelo usuário. Programadores devem poder utilizar os novos tipos para escrever suas aplicações.

   O SGBDOO deve prover, ainda, uma linguagem de programação que tenha capacidade de expressar qualquer tipo de função ou cálculo, ou seja, uma linguagem computacionalmente completa. Tal característica pode ser obtida pelo acoplamento de uma linguagem de programação já existente ao BD.

As seis características definidas acima formam um núcleo comum à maioria das implementações de SGBDOOs. É importante observar que essas características não limitam o modelo e, em geral, os SGBDOOS apresentam muitas propriedades adicionais, como por exemplo: a capacidade de modelagem de versões; um conjunto mais rico de construtores de tipo, contendo, além de conjuntos, listas e tuplas, outros construtores (vetores, por exemplo); e facilidades para definição de visões, com capacidade de atualização de dados. Entretanto, ao contrário das características essenciais introduzidas, essas propriedades adicionais não são comuns a todas as propostas, e representam, em muitos casos, propriedades específicas de um único SGBDOO.

2.3 Categorias de Funções em SGBDOOs

Devido à diversidade de suas origens e ao fato de terem sido desenvolvidos para resolver problemas específicos, os SGBDOOs existentes não provêem um conjunto homogêneo de

funções. Se por um lado diversos trabalhos discutem as características fundamentais do modelo OO, por outro lado são poucos os autores que abordam a questão da funcionalidade dos sistemas de bancos de dados orientados a objetos. Esta seção introduz as categorias de funções identificadas por Mishra e Eich [ME92], e que constituem uma proposta inicial para o estabelecimento do conceito de *completeza funcional* em SGBDOOs.

A completeza funcional em SGBDOOs pode ser abordada pelo estudo dos sistemas existentes, visando a definição de um conjunto contendo as funções disponíveis nestes sistemas. Estas funções são divididas em categorias, onde as funções de cada categoria realizam operações relacionadas a uma área específica de atividades do SBD. De cada categoria são removidas as funções que podem ser geradas pela aplicação das funções remanescentes, resultando em um conjunto de funções essenciais para a categoria. O conjunto de categorias de funções assim obtido representa o conceito de completeza funcional para SGBDOOs. É importante observar que, devido à própria natureza evolutiva do modelo OO, esta definição de completeza deve ser extensível, de forma que novas funções e categorias possam ser adicionadas, à medida que surjam SGBDOOs mais poderosos.

Pela definição apresentada acima, um SGBDOO é considerado funcionalmente completo se ele provê todas as categorias de funções, e se cada categoria de função é, em si, completa. Portanto, a verificação da completeza funcional é efetuada em dois níveis: a completeza de categorias e a completeza de funções em cada categoria. O conjunto de categorias, como foi observado, pode ser determinado pela união de todas as categorias de funções oferecidas pelos SGBDOOs existentes. As funções em cada categoria devem ser escolhidas de modo que elas possam ser usadas para gerar toda e qualquer operação exigida por um SGBDOO na respectiva categoria de função.

Segundo Mishra e Eich [ME92], para ter completeza de categorias um SGBDOO deve suportar as seguintes categorias de funções:

- Evolução de esquema;
- Gerência de versões;
- Gerência de visões;
- Reorganização de dados;
- Linguagens de programação do BD;
- Processamento de transações; e
- Gerência do subsistema de armazenamento.

A completeza de funções em cada categoria é uma área de pesquisa ainda em aberto, embora já existam trabalhos que propõem conjuntos mínimos de funções em algumas das categoria citadas acima. Por exemplo, o trabalho de [BCG+87] apresenta as funções essenciais para que um sistema OO seja funcionalmente completo na categoria *Evolução de esquema*. Uma vez determinadas as funções básicas de cada categoria, o conceito de completeza funcional apresentado pode ser bastante útil, não apenas na avaliação de um determinado sistema, mas principalmente para o estudo comparativo entre diversos SGBDOOs.

2.4 SGBDs que Implementam o Modelo OO

Nesta seção são descritos, de maneira sucinta, os modelos de dados de alguns SGBDOOs existentes, a fim de possibilitar o estudo da aplicação das características fundamentais do modelo OO nestes sistemas. Embora muitos pesquisadores considerem os SGBDs relacionais estendidos como exemplos de implementações do modelo OO, a discussão a seguir abrange apenas os sistemas OO *"puros"*. Os trabalhos publicados sobre os sistemas POSTGRES ([SRH90, SK91]) e STARBURST ([LLPS91]) representam uma boa fonte de informação sobre sistemas que implementam extensões do modelo relacional.

2.4.1 GEMSTONE

O sistema GEMSTONE [CM84] foi projetado com os seguintes objetivos: prover suporte a níveis arbitrários de estruturação de dados; permitir a definição de operações sobre tipos; separar os conceitos de definição e instanciação de tipos; evitar a imposição de limitações sobre o tamanho dos esquemas e dos itens de dados; permitir variações em objetos estruturados; permitir itens de dados arbitrários como valores de atributos; suportar modificações nos esquemas sem reestruturação dos BDs; prover maior capacidade de modelagem, através da estruturação de dados flexível e da habilidade de modelar modificações ocorridas no mundo real; capturar a história dos estados do BD como parte do modelo de dados, suportando consultas a estados passados; e finalmente, possuir uma única linguagem para manipulação de dados, cálculos genéricos e comandos do sistema.

Para atingir estes objetivos, a abordagem adotada foi a aplicação de um modelo de dados flexível a uma linguagem de programação de propósito geral já existente. A tentativa inicial foi com a linguagem Pascal, mas verificou-se posteriormente que uma linguagem orientada a objetos seria mais adequada. A linguagem escolhida foi Smalltalk, porque os conceitos básicos desta linguagem são compatíveis com os do modelo de dados desejado para o SGBD. Smalltalk se baseia nos conceitos de objeto, mensagem e classe. Um objeto

é essencialmente uma área privada de memória com uma interface pública. A memória privada é estruturada como uma lista de variáveis de instância. Um objeto aceita mensagens que solicitam acesso a partes de sua memória privada. Uma classe é um grupo de objetos estruturalmente similares, e que respondem ao mesmo conjunto de mensagens. A definição da classe contém os métodos que seus objetos utilizam para responder às mensagens. As classes são organizadas em uma hierarquia, de modo que duas ou mais classes podem compartilhar estruturas e métodos comuns em uma superclasse.

O modelo de dados de GEMSTONE estende os conceitos de Smalltalk, e se baseia em conjuntos rotulados de valores heterogêneos. Cada elemento de um conjunto possui um nome único no conjunto e um valor, que pode ser um conjunto ou um tipo simples (números ou caracteres). Um conjunto de valores ou um valor estruturado pode aparecer em qualquer lugar onde ocorre um valor simples. Um único valor pode ter estrutura interna arbitrariamente detalhada, pois o aninhamento de conjuntos é ilimitado. São permitidos ainda elementos opcionais em conjuntos, e o valor associado a um nome de elemento não é restrito a um único tipo de domínio. O aspecto temporal é representado pela substituição do valor de um elemento por um conjunto de valores. O nome do elemento é associado a cada valor através de um *tempo de transação*, ou seja, a ligação entre o nome do elemento e seu valor associado é indexado pelo tempo.

A hierarquia de classes de Smalltalk foi modificada em GEMSTONE pela remoção das classes para acesso a arquivos, comunicação e manipulação de tela, e pela inclusão de classes para controle de transações, autorização, replicação e controle de índices. A definição das classes, a hierarquia de classes e o código que define o comportamento de cada classe são armazenados no próprio BD, e estas informações podem ser manipuladas em tempo de execução (dicionário de dados ativo). O mecanismo de versão de GEMSTONE suporta as políticas de versões lineares, paralelas e ainda hierarquias de versões [BOS91].

Apesar de ter sido desenvolvido a partir da linguagem Smalltalk, o sistema GEMSTONE provê interfaces para múltiplas linguagens e ferramentas. As linguagens aceitas por GEMSTONE são C, C++, e o próprio Smalltalk. Além disso, GEMSTONE suporta consulta a SGBDs relacionais, através da linguagem SQL. Neste caso, o sistema transforma o resultado da consulta (uma relação) em objetos do BD.

2.4.2 OBJECTSTORE

OBJECTSTORE [LLOW91] é um SGBDOO que também se baseia em uma forte integração entre uma linguagem de programação orientada a objetos (C++) com as características dos SGBDs tradicionais. O objetivo do sistema é prover uma interface de programação única, tanto para dados persistentes quanto para dados transientes (ou tem-

porários). Como a linguagem C é um subconjunto de C++, OBJECTSTORE suporta acesso através de ambas as linguagens C e C++. A idéia fundamental de OBJECTSTORE é que a persistência não faz parte da definição do tipo de um objeto. Objetos de qualquer tipo podem ser alocados em memória temporária ou no BD, e não há uma classe persistente especial. Diferentes objetos do mesmo tipo podem ser persistentes ou transientes no mesmo programa. Informações sobre o esquema são armazenadas em um BD separado (BD para meta-dados).

As vantagens da utilização de C++ para o desenvolvimento de um SGBDOO são: a facilidade de aprendizagem da linguagem do BD, que é um superconjunto de C++; a eliminação do trabalho de tradução entre a representação dos dados no disco e na memória principal; o poder de expressão de C++, e a capacidade de reutilização de código; e a possibilidade de conversão de aplicações já existentes para o sistema de bancos de dados, entre outras. Além das capacidades inerentes à linguagem C++, OBJECTSTORE possui facilidades para definição de coleções (conjuntos, listas, etc), suporta trabalho cooperativo (também conhecido como *"groupware"*) baseado num mecanismo de controle de versões, consegue expressar relacionamentos bidirecionais de maneira direta, e possui capacidade de otimização de consultas.

As coleções de OBJECTSTORE são implementadas através de uma biblioteca de classes, provendo uma grande variedade de comportamentos, que incluem controle de ordenação (listas) e de duplicação (*bags* e *sets*). Estruturas mais eficientes e complexas, como *b-trees* e tabelas *hash* também estão disponíveis nas classes de coleções. Mais ainda, o usuário pode descrever o uso da estrutura, através de estimativas de freqüências de inserção, iteração e remoção, deixando a cargo do sistema a escolha da representação mais adequada. O usuário pode ainda associar uma política à coleção, definindo como a representação deve mudar em resposta a alterações na cardinalidade da coleção. Percebe-se, portanto, que uma grande preocupação com o desempenho orientou o projeto de OBJECTSTORE. As facilidades oferecidas pelo SGBD reduzem a tarefa do projetista de aplicações, que passa a descrever padrões de acesso, ao invés de codificar estruturas para todas as suas classes de objetos.

Relacionamentos bidirecionais são representados em OBJECTSTORE como um par de ponteiros inversos, de forma que, se um objeto aponta para outro, o segundo objeto possui um ponteiro inverso que aponta para o primeiro. O sistema mantém a integridade destes ponteiros, e são suportados relacionamentos um-para-um, um-para-muitos e muitos-para-muitos. Sintaticamente, relacionamentos são como atributos membros em C++, mas a atualização do valor de um relacionamento causa a atualização do relacionamento inverso, de modo que ambos estejam sempre consistentes.

Em OBJECTSTORE, uma consulta é simplesmente uma expressão que opera sobre uma ou mais coleções, e produz uma coleção ou uma referência a um objeto como re-

sultado. Predicados de seleção que aparecem nas expressões de consulta também são expressões C++ ou expressões de consulta. Qualquer coleção pode ser consultada, e expressões de consulta podem ser aninhadas para formar consultas complexas.

OBJECTSTORE suporta trabalho cooperativo através da abordagem de *transações longas*. Um usuário copia os dados de seu interesse para um espaço privado, criando uma versão; após realizar as modificações desejadas, o usuário pode retornar os dados para a área comum de desenvolvimento, tornando as modificações visíveis para os demais membros do grupo de trabalho. Enquanto isso, outros usuários podem usar as versões anteriores, sem preocupações com controle de concorrência, a não ser durante as sessões de edição na área comum.

2.4.3 ODE

O sistema ODE [AG89] é outro SGBDOO cujo projeto foi fortemente influenciado pela linguagem C++. A linguagem de programação do BD, O++, se baseia em C++, suportando encapsulamento e herança múltipla. O++ estende as classes C++, pela incorporação de facilidades para criação e manipulação de objetos persistentes e de versões desses objetos, e para associação de regras e gatilhos aos objetos. Além disso, O++ possui operadores para manipulação de conjuntos de objetos, que funcionam de modo similar às linguagens declarativas baseadas no cálculo relacional. O espaço de memória é dividido em duas partes: uma parte contém os objetos voláteis, e a outra armazena os objetos persistentes. Um oid é representado por um ponteiro para um objeto persistente.

A definição de objetos segue a sintaxe de C++, e consiste da especificação (tipo), que representa a interface da classe. Na definição da classe se encontram os corpos das funções (métodos) declaradas na sua especificação. Uma classe pode conter componentes públicos e privados. Os componentes privados representam detalhes internos das classes, e não podem ser vistos por seus usuários. Os componentes públicos podem ser itens de dados, construtores, destrutores, funções membro e funções "amigas". Estes componentes formam a interface do usuário com a classe, pois são os elementos que o usuário da classe pode referenciar.

Objetos similares podem ser modelados pela especificação da parte comum em uma superclasse, e pela derivação de subclasses que especializam a superclasse. A herança múltipla permite que novas classes sejam derivadas de diversas superclasses; as ambigüidades são resolvidas pelo uso de qualificação explícita.

A incorporação de persistência à linguagem de programação de ODE seguiu os seguintes princípios: 1) a persistência deve ser ortogonal à definição do tipo, de modo que ela seja uma propriedade de instâncias, e não de classes de objetos; 2) não deve haver perda

de desempenho para código que não utilize objetos persistentes; 3) a manipulação de objetos deve ser feita da mesma maneira para objetos persistentes e voláteis; 4) mudanças na linguagem base (C++) devem ser minimizadas.

Objetos persistentes podem ser copiados para objetos voláteis, e vice-versa, sendo que o modo de referência aos atributos é o mesmo para ambos os tipos de objetos. No entanto, os ponteiros para os dois tipos de objetos são diferentes, e a determinação do tipo de objeto referenciado (persistente ou volátil) é feita em tempo de execução. Qualquer objeto persistente pode ter versões associadas, sem um limite definido para o número de versões permitidas para cada objeto. A versão corrente de um objeto pode ser atualizada, mas versões anteriores são disponíveis apenas para consulta. Como a persistência, a versão é uma propriedade da instância e não da classe.

Todos os objetos persistentes de uma classe são agrupados em *clusters*, cujos nomes são idênticos aos das respectivas classes. Um *cluster* para uma classe deve ser explicitamente definido e criado antes de se poder criar objetos persistentes para a classe. A destruição de um cluster causa a eliminação de todos os objetos persistentes associados a ele. *Subclusters* são permitidos para agrupar objetos dentro de um *cluster*, de modo similar à definição de classes derivadas.

O++ suporta conjuntos cujos elementos podem ser duplicados. A declaração de um conjunto é semelhante à declaração de um vetor, mas o acesso indexado não é permitido. As operações sobre conjuntos suportadas são: união, diferença, atribuição, inserção e remoção. Comandos especiais incorporados a O++ permitem o acesso a elementos de conjuntos e de *clusters* (inclusive através de iteração em hierarquias de *clusters*).

2.4.4 ORION

Ao contrário dos SGBDOOs apresentados anteriormente, o sistema ORION [BCG+87] não foi desenvolvido a partir de uma determinada linguagem orientada a objetos. De fato, ORION consolida e modifica conceitos encontrados em diversos sistemas orientados a objetos. Assim, em ORION um objeto é uma região de memória privada que contém seu estado. Esta região é composta por uma coleção de variáveis de instância, onde o valor de cada variável é em si um objeto. Dois objetos podem ter variáveis de instância que se referem ao mesmo objeto. Um objeto primitivo (como um inteiro ou caractere) não possui variáveis de instância. O comportamento de um objeto é encapsulado em métodos.

Métodos, assim como as variáveis de instância, não são visíveis fora do objeto; objetos se comunicam através de mensagens. Para cada mensagem aceita por um objeto, existe um método correspondente que executa a mensagem. Objetos similares são agrupados em classes, e todos os objetos de uma classe são descritos pelos mesmos métodos e variáveis

de instância.

Uma hierarquia de classes no sistema ORION representa o relacionamento *É-Um* aplicado entre as classes, isto é, uma classe é a especialização de sua superclasse (classe *pai* na hierarquia de classes). Uma classe pode ter mais de uma superclasse, de modo que a hierarquia de classes é, na verdade, um grafo acíclico direcionado (rede de classes). Uma classe herda as propriedades (métodos e variáveis de instância) de suas superclasses (herança múltipla).

Visando controle de integridade, é útil que o domínio de uma variável de instância seja restrito a uma classe específica. Esta restrição, entretanto, é inaceitável para a modelagem de certos tipos de aplicações. Dessa forma, ORION suporta ambos os tipos de definição de domínio de uma variável de instância (restrito a uma só classe ou não).

O suporte à herança múltipla causa problemas de conflitos de nomes, e o sistema provê dois modos para resolução desses conflitos. O sistema assegura que todos os nomes herdados ou definidos para a classe são distintos. O primeiro modo de resolução de conflito consiste em deixar o usuário determinar a classe da qual cada atributo será herdado (conflito de nomes nas superclasses). No segundo modo, o usuário pode modificar o nome de um atributo herdado, de forma que ele não conflite com um atributo definido na classe (conflito entre superclasse e subclasse).

Uma classe especial, a classe *Object*, é usada como raiz da rede de classes, que inclui as classes definidas pelo sistema e pelo usuário. Além da classe raiz, o sistema define as seguintes classes: *PType*, que provê a base para definição de todas as classes que podem ser usadas como domínios primitivos de variáveis de instância; *Collection*, que consiste de objetos que são coleções de outros objetos; e *Class*, que agrupa todas as definições de classes do usuário.

O suporte à evolução de esquema oferecido por ORION envolve facilidades para mudanças em todos os aspectos da rede de classes: mudanças nos nós da rede (como inserção de classe e mudança de nome de classe); mudanças nos arcos da rede (inserção de superclasses e remoção de subclasses, por exemplo); e mudanças no conteúdo dos nós da rede (como inserção de variáveis de instância e modificações de métodos). Para isso o sistema implementa uma série de regras que impedem que operações de evolução do esquema deixem o BD em um estado inconsistente.

O modelo OO, em sua forma convencional, consegue representar coleções de objetos relacionados (objetos complexos). ORION estende o modelo OO, permitindo a modelagem de *objetos compostos*, através do relacionamento *É-Parte-De*. Um objeto composto não referencia seus subobjetos, mas "possui" subobjetos associados a ele. O conceito de objeto composto leva à definição de *objeto dependente*, que é aquele objeto cuja existência depende da existência de outro objeto. Um objeto dependente pertence a um único objeto

composto, enquanto subobjetos podem ser compartilhados por diversos objetos complexos.

ORION suporta diferentes operações sobre dois tipos de versões: versões transientes, que podem ser alteradas ou eliminadas pelo usuário que as criou, e versões de trabalho, que são estáveis, e não podem ser alteradas, mas apenas eliminadas pelo usuário que as criou. Uma versão transiente pode se transformar em versão de trabalho de duas maneiras: pela solicitação explícita do usuário, ou pela derivação de uma nova versão transiente, que transforma, automaticamente, a versão transiente anterior em versão de trabalho. A história das versões de um objeto é armazenada como uma hierarquia de versões.

2.4.5 O2

No sistema O2 [Deu91] o usuário pode definir não apenas objetos, mas também valores complexos. Um valor possui um tipo, definido recursivamente a partir de tipos atômicos e de construtores de tipos. Um objeto possui uma identidade, um valor e um comportamento definido por seus métodos. Além disso, um objeto pertence a uma classe, e tanto um objeto como um valor podem referenciar outros objetos através de suas identidades. O2 suporta classes cujas instâncias são objetos e que encapsulam dados e comportamento, e tipos, cujas instâncias são valores não encapsulados. A cada classe está associado um tipo que descreve a estrutura de suas instâncias [Deu90].

Os tipos atômicos de O2 são booleano, caractere, inteiro, real, cadeia de caracteres, *bits* e *bitmaps*; os construtores de tipos são lista, conjunto e tupla. Uma lista é uma coleção ordenada cujos elementos são referenciados através de índices. Campos do tipo cadeia de caracteres e *bits* se comportam, respectivamente, como uma lista de caracteres e como uma lista de *bytes*. Um conjunto é uma coleção não ordenada, que pode ou não conter elementos duplicados, de acordo com a necessidade do usuário. Uma tupla é uma agregação de atributos de diferentes tipos.

Um esquema O2 é um conjunto de classes relacionadas por ligações de herança e/ou composição, onde cada classe descreve a estrutura e o comportamento de um conjunto de objetos. A parte estrutural da classe é representada por seu tipo, enquanto os métodos da classe descrevem o seu comportamento. Uma classe pode herdar seu tipo e métodos de outras classes (herança múltipla), e conflitos de nomes são resolvidos pela redefinição explícita dos nomes conflitantes. Um tipo ou método herdado pode ser redefinido localmente, desde que seja respeitada a semântica de subtipos.

Para se tornar persistente, um objeto ou valor deve ser associado direta ou transitivamente a uma raiz persistente. Raízes persistentes são declaradas em um esquema através

da atribuição de um nome; todo objeto ou valor que possui um nome é persistente, bem como os objetos e valores que fazem parte de um objeto persistente. Objetos podem se tornar persistentes após a sua criação, e o processo de tornar o objeto persistente não altera sua identidade.

O2 provê encapsulamento em três níveis. O primeiro é o encapsulamento clássico, onde atributos e métodos são privados à classe a que pertencem. No entanto, O2 permite que o usuário torne públicos os atributos e métodos desejados. A segunda forma de encapsulamento estende o conceito para um conjunto de definições de classe. Dessa forma, um esquema pode exportar classes, permitindo que outros esquemas reutilizem suas definições. O último nível de encapsulamento diz respeito aos bancos de dados em si. Uma aplicação que atua sobre um determinado BD pode fazer acesso a outro BD, através da chamada de métodos que irão ser executados sobre o BD "remoto".

Uma classe definida pelo sistema, a classe *Object*, é a raiz da hierarquia de classes, onde são definidos métodos comuns a todas classes, como por exemplo o método para determinação da identidade do objeto. Em O2, todas as informações são armazenadas como objetos: os dados dos BDs, as classes, os métodos, etc. A linguagem de consulta do sistema, O2Query é um subconjunto da linguagem de programação do BD (O2C), e pode ser usada de modo independente, para consultas *ad hoc*, ou através de chamadas de funções em programas C ou C++. Todos os tipos de dados e operadores, inclusive métodos, são permitidos na formulação de uma consulta. Além disso, no modo *ad hoc*, a linguagem de consulta não é restrita pelo conceito de encapsulamento, podendo fazer acesso direto aos atributos de um objeto. Já na forma embutida em uma linguagem de programação o acesso aos dados privados do objeto só é possível através de seus métodos.

2.5 Análise Crítica do Modelo OO

2.5.1 Instrumentos de Avaliação do Modelo OO

Os sistemas discutidos acima se baseiam nas características propostas na seção 2.2 como representantes de um consenso entre as diferentes implementações de SGBDOOs. Todos os sistemas analisados apresentam, com pequenas variações, as seis características fundamentais introduzidas, apesar de seus modelos de dados serem bastante diferentes entre si.

Se por um lado as características fundamentais do modelo OO podem ser facilmente observadas, por outro lado, o conceito de completeza funcional proposto na seção 2.3 não pode ser diretamente avaliado a partir da descrição do modelo de dados do SGBD. A

análise da completeza funcional de um SGBDOO deve ser feita com base na descrição completa do sistema, o que, em geral, só pode ser encontrado nos manuais técnicos que definem o produto. Para que a avaliação seja mais precisa, é necessário ainda que o avaliador possa testar a funcionalidade descrita, o que exige o acesso a uma plataforma em que o SGBDOO esteja instalado. Assim, o conjunto de categorias de funções e a noção de completeza funcional introduzidas na seção 2.3 servem para orientar o avaliador que disponha dos recursos descritos acima, no sentido de determinar a qualidade de um SGBDOO.

2.5.2 Qualidades e Limitações do Modelo OO

Da discussão apresentada neste capítulo sobre o modelo de dados orientado a objetos, pode-se destacar algumas das suas principais qualidades:

- O modelo OO compartilha as vantagens apresentadas pelos modelos semânticos [HK87], tais como a presença de construtores específicos para a modelagem de diferentes tipos de relacionamentos (associação, generalização, especialização, etc) e os mecanismos de abstração para visualização e acesso às informações.

- A herança de dados e comportamento é uma ferramenta de modelagem muito poderosa, assim como o suporte ao relacionamento de classificação (conceito de classe) e a capacidade de definição de objetos complexos.

- O usuário não é limitado pelo sistema de tipos definido pelo SGBD, podendo definir seus próprios tipos de dados, bem como operações que atuam sobre estes tipos. Além disso, tipos definidos pelo sistema e pelo usuário possuem o mesmo *status*.

- O conceito de encapsulamento encoraja a modularidade e evita a redundância de funções, cujas definições são centralizadas na especificação da classe. A centralização das operações que afetam o estado do BD facilita ainda a detecção de erros e a manutenção da funcionalidade do sistema.

- Modificações sobre tipos e métodos (implementação da classe) podem ser feitas de maneira local à classe, garantindo independência de dados para as aplicações (desde que a interface da classe não seja afetada).

- A semântica do comportamento de um objeto não se encontra dispersa entre os programas de aplicação, mas sim na própria definição do objeto. Isto simplifica os programas de aplicação, que só precisam efetuar chamadas a um determinado conjunto de operações (os métodos).

Embora apresente muitas vantagens com relação aos demais modelos de dados, o modelo OO também possui limitações. A principal delas é a rigidez comportamental proveniente da necessidade de especificação prévia de todas as operações a serem efetuadas, através de um conjunto fixo de métodos. Esta característica é contrária à natureza tipicamente evolutiva das aplicações a que se destina o modelo de dados orientado a objetos [EN89]. Além disso, por ser uma área de pesquisa muito recente, existem diversos assuntos relacionados a SGBDOOs que precisam ser melhor estudados. Entre estes assuntos estão a otimização de consultas, principalmente para o caso de aplicação de métodos definidos pelo usuário [BM91]; o suporte eficiente a mudanças nos esquemas; o desenvolvimento de formalismos que representem o modelo; e a elaboração de metodologias para projeto físico e lógico de BDOOs.

Capítulo 3

Sistemas de Interface para Bancos de Dados

3.1 Introdução

A preocupação em oferecer aos usuários de bancos de dados uma interface adequada é bastante antiga, e muitos trabalhos têm sido publicados sobre este assunto. De fato, os SGBDs são ferramentas poderosas, que possuem uma vasta e complexa variedade de funções. O objetivo primordial das interfaces para sistemas de bancos de dados é facilitar o acesso a essas funções para a comunidade de usuários dos BDs [Wel88].

Entretanto, este objetivo é muito difícil de ser atingido, já que a comunidade de usuários é bastante heterogênea, e cada tipo de usuário deseja obter da interface funcionalidades diferentes (e às vezes conflitantes).

Em geral, usuários finais se interessam em obter acesso rápido e simples aos dados armazenados, através de mecanismos diretos de navegação ou consulta. Para estes usuários, é importante que a interface realize o trabalho de busca das informações, de criação de representações das informações, e de gerência de operações básicas sobre estas informações, de modo automático.

Programadores de aplicação, por outro lado, desejam obter acesso aos dados através de linguagens de programação computacionalmente completas, que lhes permitam definir o modo de busca e apresentação das informações, e que lhes garantam o acesso a todas as funções disponíveis no SGBD. Os programadores de aplicação enfatizam a importância do poder de expressão da linguagem que o sistema de interface oferece, em detrimento à simplicidade ou à automatização de funções específicas.

Um terceiro tipo de usuário, o administrador de bancos de dados (DBA), se interessa mais especificamente por um subconjunto de funções oferecidas pelo SGBD. Os DBAs são responsáveis pela segurança e pelo bom desempenho dos bancos de dados, e por isso trabalham constantemente com funções que permitem definir e atualizar esquemas e catálogos, criar e modificar visões e estabelecer autorizações de acesso, entre outras.

Nas próximas seções serão apresentados diversos sistemas de interface para bancos de dados existentes atualmente[1]. Poucos destes sistemas satisfazem as necessidades de toda a comunidade de usuários. As seções 3.2 e 3.3 discutem, respectivamente, sistemas de interfaces para o modelo de dados relacional e para os modelos de dados semânticos, precursores do modelo OO. Os sistemas de interface existentes para este último modelo de dados serão estudados na seção 3.4. Na seção 3.5 são analisados os pontos positivos e as deficiências dos sistemas de interface apresentados nas seções anteriores. A seção 3.6 encerra este capítulo, apresentando as características básicas de interfaces gráficas para sistemas de bancos de dados.

3.2 Interfaces para o Modelo Relacional

A maioria dos SGBDs relacionais possui como interface com o usuário linguagens declarativas de alto nível, através das quais são especificados os resultados desejados, deixando-se a cargo do sistema os detalhes de otimização de consulta e as decisões sobre a estratégia de execução. Estas linguagens de consulta textual (como SQL e QUEL) já foram bastante discutidas na literatura sobre bancos de dados.

Esta seção apresenta inicialmente um estudo sobre QBE, uma interface visual para sistemas relacionais. Em seguida é introduzido o sistema SDMS, uma interface que segue o paradigma de manipulação gráfica direta [Shn83]. Por fim é apresentado um sistema de consulta gráfica, usando o modelo de *relação universal*, uma extensão do modelo de dados relacional.

[1] As figuras contidas nas seções 3.2, 3.3 e 3.4 foram construídas através de um editor gráfico, e ilustram o tipo de informação representado em cada sistema de interface. O objetivo das figuras é fornecer subsídios para a comparação entre os diferentes padrões de representação de informações. As janelas gráficas usadas nos sistemas discutidos apresentam diversas características (decorativas e funcionais) que foram propositalmente omitidas nas figuras, a fim de facilitar essa comparação.

3.2.1 QBE

A linguagem QBE (*Query By Example*) [EN89] difere das linguagens de consulta declarativas pelo fato de o usuário não definir a estrutura da consulta explicitamente. QBE foi projetado para ser uma interface visual, não podendo ser utilizado de forma embutida em linguagens de programação. As solicitações do usuário e as respostas a estas solicitações são representadas de forma tabular. A idéia básica de QBE é que o usuário formule a consulta através de um exemplo de resposta possível, no local apropriado de uma tabela vazia. Assim, a especificação de consultas é feita através de exemplos, justificando o nome *Query By Example*.

Para realizar uma consulta, o usuário constrói a moldura da tabela através de teclas de função, e digita o nome da relação a ser utilizada. Em resposta, o sistema preenche a tabela com os nomes dos atributos da relação selecionada. O usuário não precisa conhecer os nomes dos atributos, e nem mesmo o nome das relações, pois existe uma função que lista os nomes das relações existentes.

O usuário especifica que o valor de um atributo deve aparecer no resultado da consulta digitando P na coluna em que o atributo aparece na tabela criada. Valores de exemplo para o atributo são precedidos pelo símbolo "_", enquanto valores constantes são digitados normalmente. Os valores de exemplo representam variáveis de valores arbitrários. Somente os valores constantes são usados para seleção de tuplas onde o valor do atributo especificado seja igual ao valor constante digitado pelo usuário. Operadores de comparação diferentes do operador de igualdade devem aparecer explicitamente na coluna apropriada, antes do valor constante. A regra geral é que todas as condições especificadas na mesma linha de uma tabela são relacionadas pelo conector lógico *And*, enquanto as condições especificadas em linhas distintas são conectadas pelo *Or* lógico. Para condições mais complexas, QBE oferece a *caixa de condições*, que é uma área onde o usuário digita as condições, podendo conter os conectores *And* e *Or*, mas não o operador de negação.

Os resultados da consulta são apresentados na tabela criada, nas colunas apropriadas. Se o resultado é maior do que o espaço da tela, teclas de funções permitem percorrer a tabela criada nos sentidos horizontal e vertical. O sistema permite ainda que o usuário especifique a ordem em que as tuplas devem aparecer na tabela resultante. Para isso, basta acrescentar os prefixos *AO* (ordem crescente) ou *DO* (ordem decrescente) na coluna do atributo de ordenação. Se mais de um atributo é usado para ordenar o resultado, a prioridade de ordenação é especificada através de números colocados entre parênteses. Uma operação de *junção* de duas ou mais relações é especificada em QBE pela utilização da mesma variável (valor de exemplo) nas colunas dos atributos de junção. Se atributos de diversas relações aparecem no resultado, o usuário deve construir a moldura da tabela resultante. Caso contrário, o sistema irá apresentar os resultados nas colunas das próprias

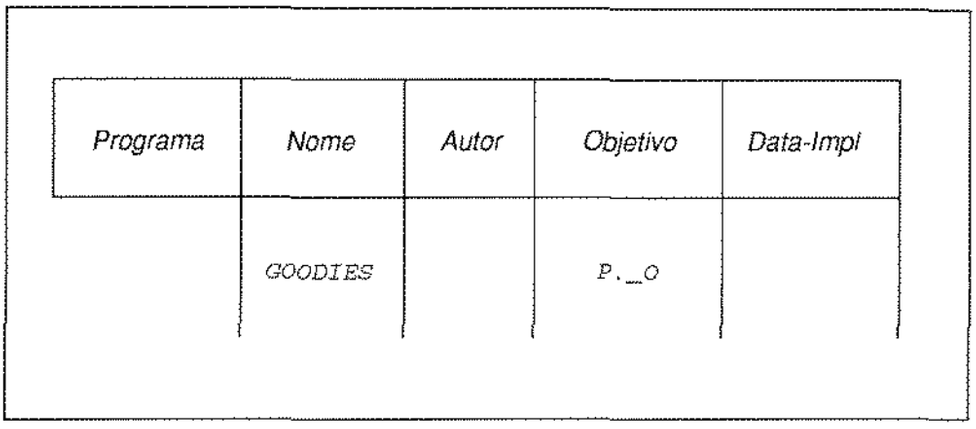


Figura 3.1: Exemplo de consulta em QBE

tabelas que representam as relações *juntadas*.

Os operadores QBE que permitem atualização de dados são representados pelos prefixos I, D e U, que fazem inserção, remoção e atualização de tuplas, respectivamente. Os prefixos I e D são digitados na coluna que contém o nome da relação; o prefixo U aparece na coluna do atributo a ser atualizado, seguido pelo novo valor deste atributo. Os três operadores requerem que as tuplas afetadas sejam selecionadas pelo mecanismo normal de consulta do QBE.

QBE parece ser mais fácil de ser compreendido que linguagens como SQL ou QUEL, além de livrar o usuário da memorização de nomes de relações e de atributos. A sintaxe de QBE é também mais flexível, e permite que uma consulta seja elaborada de forma progressiva. A consulta descrita na figura 3.1 visa representar o objetivo do programa **GOODIES**, sabendo-se que existe uma relação **Programa** contendo os atributos **Nome**, **Autor**, **Objetivo** e **Data-Impl**.

3.2.2 SDMS

O SDMS (*Spatial Data Management System*) [Her80] usa o paradigma de manipulação direta para prover o acesso às informações contidas em um SBD relacional. A gerência de dados espaciais, empregada em SDMS, é uma técnica para organização e recuperação de informações posicionadas em um espaço gráfico de dados (GDS ou *Graphical Data Space*). O GDS é visualizado através de monitores coloridos; um monitor mostra uma visão superficial e geral dos dados; em outro monitor é apresentada uma visão detalhada de uma determinada parte dos dados. A indicação dos dados que estão sendo detalhados

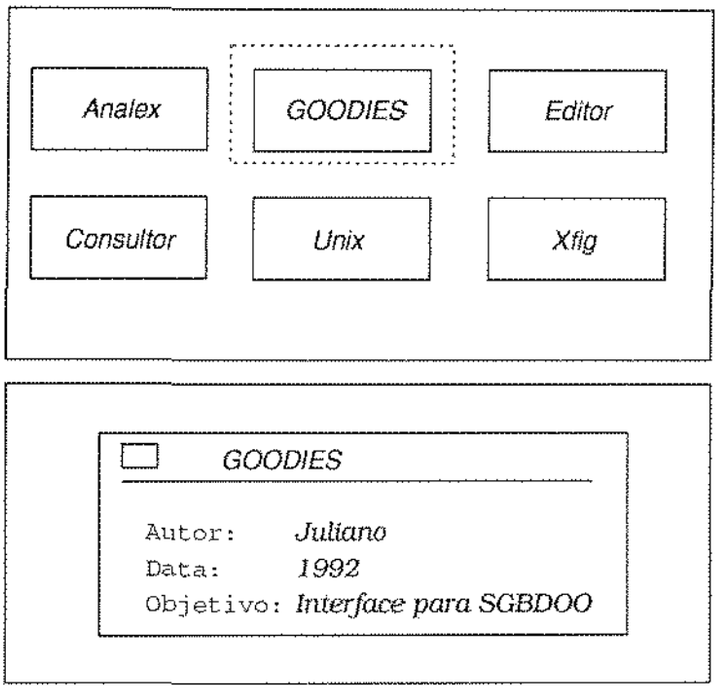


Figura 3.2: Representação de informações em SDMS

é feita por um retângulo brilhante no monitor da visão geral.

Através de um *joystick*, o usuário pode deslocar o retângulo pelo monitor da visão geral, controlando desse modo os dados que ele deseja ver de forma mais detalhada. Em ambas as visões, detalhada e geral, as informações são apresentadas como ícones. Na visão detalhada, porém, os ícones possuem formatos mais precisos, contendo ainda informações textuais. Para consultar uma informação, o usuário navega pela visão geral, indicando, através do *joystick*, a direção e sentido da navegação, até obter o dado desejado. Para conseguir mais informações, o usuário pode girar o *joystick* em sentido horário sobre um ícone. Essa operação aciona um processo de detalhamento do ícone, que apresenta mais informações textuais e formas mais específicas. O usuário pode voltar a navegar pelos dados com esse novo nível de visualização, ou pode girar o *joystick* em sentido anti-horário para voltar ao nível de detalhe anterior, e assim sucessivamente. Na figura 3.2, a visão geral da relação *Programa* definida anteriormente é apresentada na parte superior, e um nível de visão detalhada é mostrado na parte inferior. A visão detalhada corresponde à navegação na visão geral, buscando responder à consulta formulada na seção 3.2.1. O retângulo brilhante é representado na figura 3.2 como um retângulo pontilhado.

Uma linguagem de descrição de ícones permite que o DBA descreva a aparência dos ícones em cada nível de detalhamento. A linguagem permite ainda especificar o posicionamento dos ícones no monitor. A aparência de um ícone envolve a imagem do ícone propriamente dita, e mais a cor, o tamanho e os textos associados ao ícone. Após a criação do ícone, o DBA precisa associar a imagem formada a uma relação do BD, através do comando *Associate*. O SDMS irá buscar as tuplas da relação especificada, criando um ícone para cada tupla da relação que satisfaça um conjunto de condições estabelecidas na cláusula *Where* do comando *Associate*. Os parâmetros para criação dos ícones, como cor e tamanho, podem ser obtidos de valores de atributos das tuplas selecionadas.

Apesar do mecanismo de interação com o usuário seguir a manipulação gráfica direta, SDMS oferece, através de uma linguagem de consulta associada, comandos que aumentam a capacidade de expressão do sistema. A linguagem de consulta associada é uma extensão da linguagem QUEL, denominada SQUEL. Os comandos adicionais de SQUEL são: *Blink* e *Frame*, que destacam, na visão geral, os ícones que satisfazem determinadas condições. Estes ícones irão aparecer piscando ou dentro de uma moldura retangular, ao se usar, respectivamente, *Blink* ou *Frame*. O comando *Find* move a visão detalhada para a região ocupada por um determinado ícone. O comando *Associate*, como foi visto, associa tuplas de uma relação a um determinado tipo de ícone. Por fim, o comando *Change* permite a modificação de dados contidos no ícone apontado pelo usuário.

Algumas vantagens do SDMS podem ser destacadas com relação às interfaces apresentadas nas seções anteriores. SDMS permite a navegação pelos dados, sem conhecimento prévio das estruturas do BD, e através de movimentos simples e naturais. O nível de abstração em que os dados são apresentados também é controlado pela movimentação do *joystick*. Além disso, SDMS pode manipular tipos de dados diferentes, como *bitmaps*. No entanto, para que SDMS seja usado facilmente pelo usuário final, o DBA deve se esforçar bastante para conseguir, através da linguagem de descrição de ícones, definir uma transformação adequada dos dados relacionais em apresentações espaciais. Mais ainda, o SDMS trabalha com uma relação a cada momento, não sendo permitidas operações básicas, como a junção de relações.

3.2.3 PICASSO

PICASSO [KKS88] é uma linguagem gráfica para consulta em SBDs que seguem o modelo de *relação universal*. A grosso modo, este modelo estende o modelo relacional, provendo mecanismos para que o usuário não necessite determinar as relações onde se encontram os atributos que ele deseja. O próprio SGBD determina as relações que precisam ser recuperadas para atender uma determinada consulta.

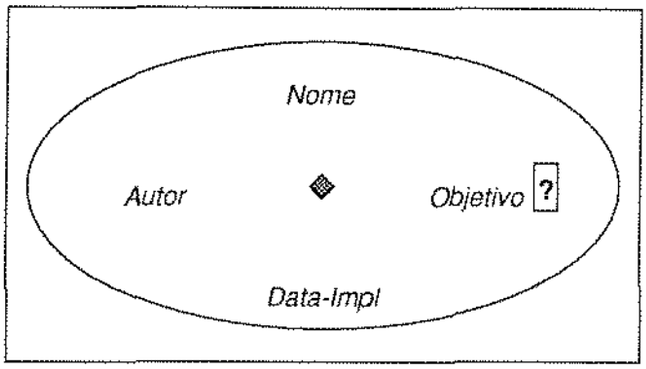


Figura 3.3: Exemplo de consulta em PICASSO

O sistema PICASSO interage com o usuário por meio de três janelas. A janela de mensagens indica o modo corrente do sistema, que pode ser: seleção de atributos; especificação de predicados; *Undo* e outros. As mensagens do sistema são também apresentadas nesta janela. A janela de opções contém comandos para mudança de contexto (modo). Através desta janela o usuário pode escolher um BD para consultar, obter uma apresentação tutorial sobre o uso do sistema, imprimir uma cópia da tela, navegar em um esquema de BD, e escolher entre os modos de consulta textual ou gráfica. A terceira janela, a janela gráfica, apresenta um hipergrafo construído pelo sistema a partir das relações do BD, e que representa o esquema do BD. Sobre este grafo são elaboradas as consultas no modo gráfico.

A formulação de consultas está baseada no uso de um *mouse* de três botões: o botão da esquerda é usado para seleção de atributos; o do meio para especificação de predicados; e o botão da direita é usado para a chamada do *menu* básico que contém as opções para o processamento de consultas. A figura 3.3 mostra a mesma consulta apresentada nas seções anteriores, elaborada de acordo com a descrição do sistema PICASSO.

Para selecionar um atributo, o usuário aponta o atributo desejado com o *mouse* e pressiona o botão da esquerda. O sistema responde, colocando um ponto de interrogação diante do atributo selecionado no hipergrafo. Um novo pressionamento do botão da esquerda causa o aparecimento do *menu* de operadores agregados (*Average*, *Sum*, etc) e dos operadores de conjunto ($\cap, \cup, \ldots$) permitidos pelo sistema.

A especificação de um predicado é feita pela seleção do atributo sobre o qual será aplicada a condição, com o botão central do mouse. Esta operação apresenta na tela o *menu* de operadores de comparação ($<, >$, e outros) e de operadores de conjunto (como $\supset$ e $\subset$) disponíveis. A escolha de um operador causa o aparecimento de um *gabarito* onde

pode ser digitado o valor do elemento a ser comparado com o atributo, se esse valor é uma constante. No caso de o segundo valor do predicado também ser um atributo, o usuário pode selecioná-lo com o botão central do *mouse*. O sistema irá desenhar uma flecha ligando o primeiro atributo ao segundo, com o operador selecionado aparecendo na própria flecha.

O menu básico de processamento de consulta é obtido com o botão da direita do *mouse*, e apresenta opções para carregar um novo BD, executar uma consulta, desfazer a última ação, formular consultas, mudar a fórmula de uma consulta, entrar em modo de atualização de dados, abrir uma janela para elaboração de predicados complexos (a chamada *caixa de conexões*), e para abandonar o sistema.

Via de regra, os predicados especificados sobre o hipergrafo são conectados pelo operador lógico *And*. Selecionando a opção para predicados complexos do *menu* de processamento de consulta, o usuário recebe uma caixa de diálogo, a caixa de conexões. Antes disso, o sistema rotula no hipergrafo os predicados formulados, atribuindo o valor *P1* para o primeiro predicado definido, *P2* para o segundo, e assim por diante. O usuário usa a caixa de conexões para estabelecer ligações entre os rótulos dos predicados, indicando o operador lógico de conexão entre eles.

Um ícone colocado no centro do hipergrafo possibilita a seleção de todos os atributos do esquema, através do pressionamento do botão da esquerda, com o *mouse* apontando para esse ícone. Se for pressionado o botão do centro, uma nova instância do hipergrafo é criada, permitindo-se, desse modo, a elaboração de consultas mais complexas, tais como as consultas aninhadas. Uma ferramenta de apoio permite ainda salvar e carregar o conteúdo de uma consulta, examinar os resultados da consulta, e ordenar estes resultados. Os valores apresentados como resultados das consultas podem ser utilizados para a construção de predicados, pela seleção de valores com auxílio do *mouse*.

PICASSO permite a visualização e navegação sobre o esquema do BD representado pelo hipergrafo. É também possível esconder ou mostrar partes do hipergrafo, reconfigurar a disposição de seus elementos, e abrir espaço para novas instâncias do hipergrafo. Entretanto, o sistema possui deficiências como a incapacidade de suporte a novos tipos de aplicações, como *CAD/CAM* e a ausência de consenso sobre o funcionamento do modelo de relação universal em aplicações reais. Além disso, a capacidade de atualização de dados não foi implementada, apesar de constar do *menu* básico do sistema.

3.3 Interfaces para os Modelos Semânticos

Os modelos semânticos de dados surgiram para atender as necessidades de aplicações que não podem ser modeladas, de maneira simples, pelas estruturas usadas no modelo relacional. Desse modo, os modelos semânticos tentam aumentar o poder de expressão do projetista, pela incorporação de um conjunto *rico* de características semânticas ao SBD [PM88]. Será demonstrado, nesta seção, que estas características semânticas tornam a representação de dados mais trabalhosa, de maneira que as interfaces aqui apresentadas superam aquelas introduzidas na seção 3.2 em termos de complexidade de construção. No entanto, estas interfaces conseguem representar informações e relacionamentos que não podem ser expressos, de forma direta, no modelo de dados relacional.

3.3.1 ISIS

O sistema ISIS (*Interface for a Semantic Information System*) [GGKZ85] permite a manipulação gráfica de um BD semântico, provendo ao usuário facilidades para construção e modificação de BDs, navegação sobre esquemas e dados, e facilidades de consulta gráfica. O modelo semântico usado em ISIS é o SDM. Neste modelo, uma entidade corresponde a qualquer objeto que possua significado semântico na aplicação modelada, e uma classe é uma coleção de entidades do mesmo tipo. Entidades possuem atributos, e para cada atributo é definido um nome único na classe e uma classe de onde os valores do atributo são obtidos (*value class*). Os relacionamentos entre classes incluem herança e agrupamento, representadas por *conexões interclasses*.

Embora o SDM permita a definição de herança múltipla, o sistema ISIS só aceita a especificação de uma superclasse para cada classe (herança simples). Dessa forma, ISIS trabalha com uma floresta de herança, definida pelas conexões interclasses de subclasse, e com uma rede de composição, estabelecida pelas *value classes* dos atributos.

ISIS opera com um conceito de visão, onde a visão corresponde a uma tela completa da estação de trabalho, e pode conter *menus*, janelas de texto e janelas gráficas. Os *menus* são consistentes, no sentido de que comandos com nomes idênticos possuem a mesma semântica em diferentes visões. As janelas de texto são usadas para solicitação de informações do usuário e para apresentação de mensagens do sistema. As janelas gráficas podem conter subconjuntos do esquema ou dos dados de um BD.

A operação do sistema ISIS pode ser feita no modo *esquema* ou no modo *dados*. No modo esquema, a floresta de herança é apresentada na janela gráfica. As classes são representadas como retângulos, que se dividem em três partes. Na parte superior é apresentado o nome da classe; logo abaixo é exibido um *padrão de preenchimento* único para

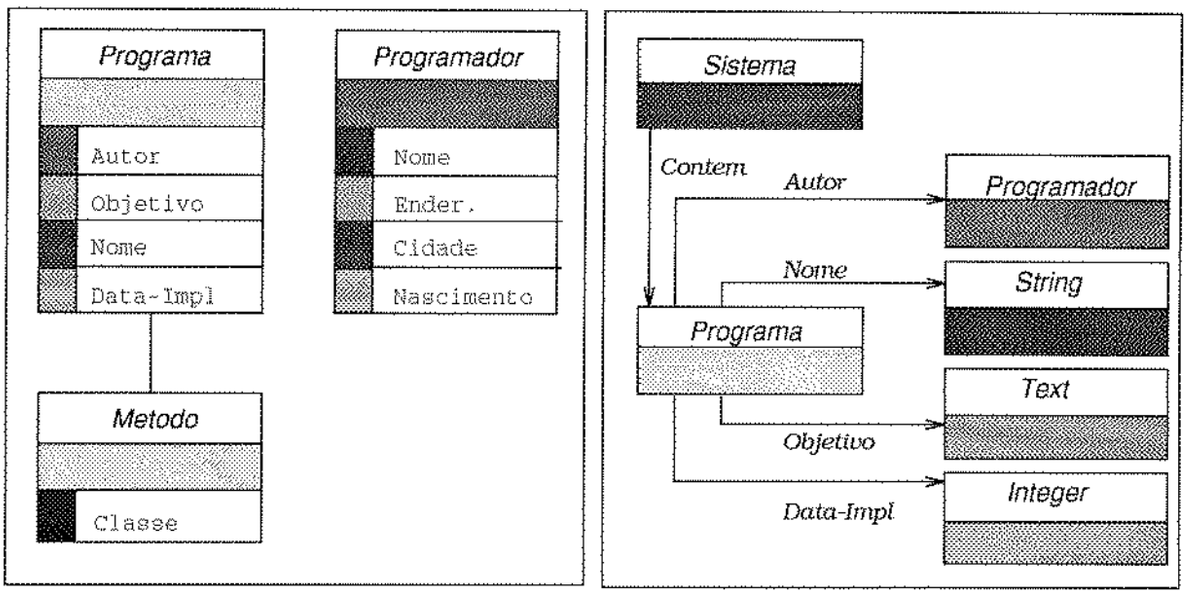


Figura 3.4: Representação de informações em ISIS

cada classe, provido pelo sistema. O restante do retângulo contém os atributos da classe. Cada atributo é representado por seu nome, precedido pelo padrão de preenchimento de sua *value class*. A floresta de herança é representada por linhas que ligam as classes às suas subclasses.

Um *menu* de edição associado à janela gráfica permite mover, modificar e eliminar classes e atributos. Os comandos deste menu variam de acordo com o componente selecionado do esquema (classe, atributo ou conexão). A seleção de um componente é feita pela indicação do elemento desejado com o *mouse*, seguida do pressionamento de um botão do *mouse*. O *menu* de edição contém ainda opções para troca de modo (esquema ou dados), e para visualização da rede de composição do esquema.

A rede de composição é uma visão alternativa do esquema, que pode ser usada para navegação sobre a estrutura do BD. A rede de composição é representada por uma *rede semântica*, contendo atributos ligados por flechas, de acordo com a *value class* definida para cada atributo. A rede de composição apresenta apenas os atributos da classe selecionada na floresta de herança, e a indicação das classes que são domínios desses atributos. A navegação no esquema é feita pela escolha de uma dessas classes, cuja rede de composição passa a ser representada na janela gráfica.

Para visualizar os dados contidos no BD, o usuário seleciona uma classe e, através do

menu de edição, passa para o modo *dados*. As informações de cada classe são apresentadas em janelas distintas, que se sobrepõem. A janela da classe selecionada na visão do esquema aparece sobreposta às demais. Em uma janela de dados são apresentadas todos os atributos da classe, inclusive os herdados. À direita de cada janela aparece uma lista de entidades que pertencem à classe, e à esquerda, o retângulo que representa a classe, conforme explicado anteriormente. A figura 3.4 ilustra o modo de representação de informações no sistema ISIS. A janela da esquerda mostra uma floresta de herança e na janela da direita aparece a rede de composição da classe *Programa*.

A navegação sobre os dados é possível pela seleção de um atributo, seguida do acionamento do botão *Follow*. Esta operação causa o aparecimento da janela de dados pertencente à classe correspondente à *value class* do atributo selecionado. Uma flecha liga o atributo ao seu valor, apresentado na janela de dados mais recentemente criada, e que se sobrepõe àquelas janelas de dados que já existiam.

No modo *dados* o usuário pode modificar valores de atributos, criar novas classes e entidades, e realizar consultas gráficas. Entretanto, o processo para realização dessas operações é complexo. O mecanismo de consulta gráfica, por exemplo, envolve a criação de uma subclasse temporária, que irá armazenar os resultados da consulta. Para tanto, o usuário precisa utilizar comandos da visão da floresta de herança para definir uma subclasse, e em seguida usar uma *janela de predicados* para especificar as condições da consulta. Esta janela contém várias subjanelas, que indicam regras e opções disponíveis para a elaboração de uma consulta.

3.3.2 SNAP

O sistema SNAP (*Schemas Notated As Pictures*) [BH86] é uma interface para SBDs que se baseiam no modelo semântico IFO. Este modelo permite a representação de esquemas por grafos, onde conjuntos de entidades (classes) são representadas por nós, e relacionamentos (funcionais e *ISA*) são descritos por diferentes tipos de arcos. SNAP provê uma série de facilidades, notadamente no que diz respeito à gerência de esquemas. O sistema permite a representação simultânea de todos os tipos de estruturas existentes no modelo de dados, por meio de diferentes níveis de abstração das informações apresentadas. Além disso, SNAP oferece mecanismos para reorganização visual do esquema, possibilitando um visão modular de suas estruturas.

No modelo IFO, um losângulo é usado para representar um tipo de dado abstrato. Retângulos contém os atributos de uma classe, e ícones colocados nos arcos que ligam uma classe a seus atributos indicam os atributos que são identificadores únicos de uma classe (*surrogates*). Se uma classe é usada em dois papéis distintos no esquema, um

dos papéis é representado por um nó derivado, de formato circular. Subtipos (classes derivadas) também são descritos como círculos, mas estes são maiores que aqueles usados para representar nós derivados.

O usuário se comunica com SNAP através de duas janelas gráficas, para representação do esquema e elaboração de consultas. São ainda utilizadas duas janelas de texto, para a apresentação de resultados de consultas, para mensagens do sistema e para entrada de dados por parte do usuário. Basicamente, a interação entre o usuário e o sistema é feita por meio de *menus*, que são apresentados sempre que o usuário seleciona, através do mouse, um objeto nas janelas gráficas. As opções contidas nos *menus* variam de acordo com o tipo de objeto selecionado.

A definição de uma nova classe no esquema pode ser feita da seguinte maneira. O usuário pressiona o botão do *mouse* na posição da janela de esquema em que a classe deverá ser criada (naturalmente esta posição deve estar vazia). O sistema apresenta então as opções dos tipos de nós disponíveis (nó de classe, nó derivado, etc). Ao Selecionar o tipo de nó desejado, o usuário é solicitado a informar o nome da nova classe. De maneira análoga são criados os atributos da nova classe. Pressionando o botão do *mouse* sobre o nó criado, o usuário obtém um *menu* com as seguintes opções: eliminar o nó; escondê-lo; movê-lo; editar o rótulo do nó; ou ligar o nó a outro componente do esquema. SNAP não verifica a consistência do esquema até que o usuário especifique o término de edição. A manipulação direta é o único paradigma de interação com o usuário; SNAP não suporta definição textual de esquema.

Durante a navegação sobre o esquema, o usuário dispõe de opções como: reposicionar, esconder ou mostrar objetos gráficos; modificar o nível de abstração das informações apresentadas; e reformatar as representações de hierarquias e de objetos. Esta operações podem ser aplicadas sobre um único objeto gráfico, sobre um conjunto de objetos, ou sobre uma região do esquema. Outra facilidade de navegação sobre esquema é busca de um determinado nó, bem como de nós relacionados a ele.

Consultas são especificadas graficamente, e de modo muito simples, comparando-se com o mecanismo de consulta gráfica do sistema ISIS. Mais de uma janela de consulta gráfica pode ser associada a cada esquema, e mais de uma janela textual pode apresentar os resultados de uma determinada consulta. Cada janela de consulta está fortemente associada a uma janela de esquema, de forma que operações nas duas janelas são efetuadas de maneiras muito semelhantes. A figura 3.5 ilustra a representação de um esquema, usando as convenções do sistema SNAP.

Uma consulta é formulada por um subgrafo, obtido a partir do grafo do esquema. Restrições podem ser diretamente associadas a nós do subgrafo de consulta. Entretanto, não se pode utilizar variáveis na especificação de restrições sobre os nós, ou seja, as

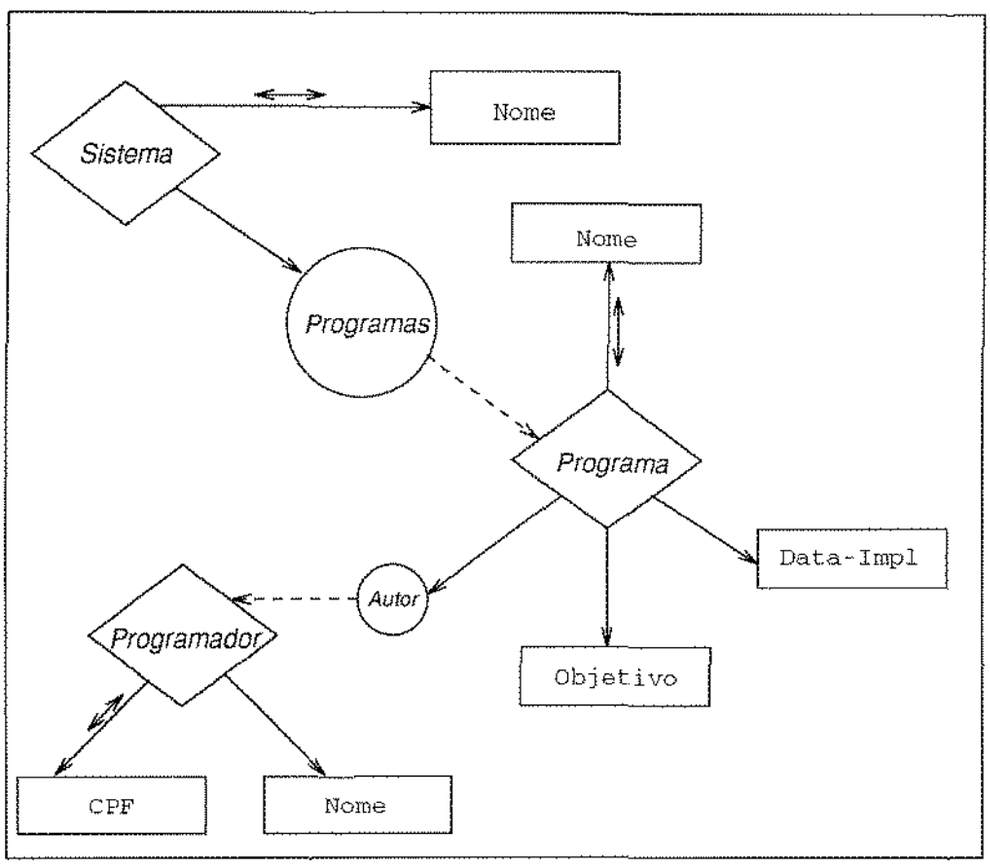


Figura 3.5: Representação de informações em SNAP

restrições devem ser especificadas por meio de valores absolutos. Arcos comparadores são especificados entre dois nós, para garantir que um certo relacionamento seja mantido entre os valores associados aos nós. O arco é rotulado com o operador desejado, e este operador pode ser de comparação ou de conjunto. Quando o usuário termina de especificar a consulta, SNAP escolhe um formato padrão para os resultados, e provê um mecanismo para que o usuário possa escolher outros formatos. O usuário pode, ainda, salvar um subgrafo de consulta para uso futuro.

3.3.3 SCHEMADESIGN e DATABROWSE

Rogers e Cattell apresentam em [RC88] uma interface para BDs que seguem o modelo Entidade-Relacionamento (E–R), introduzido por Chen [Che76]. A interface é composta por duas ferramentas que permitem a manipulação de esquemas e de dados. A ferramenta SCHEMADESIGN permite criar e representar graficamente o esquema de um BD, através de um diagrama E–R. SCHEMADESIGN possui três janelas: a janela gráfica para projeto de esquema, a janela de mensagens do sistema, e a janela de comandos. Nesta última janela o usuário pode acionar comandos através de botões, ou ainda fazer entrada de dados através do teclado.

Na modelagem do esquema, entidades com chave são representadas por retângulos normais, enquanto entidades que não possuem chave de acesso são representadas por retângulos de cantos arredondados. As flechas ligando as entidades representam o conceito de *chave estrangeira* do modelo de dados relacional. A criação de uma entidade é feita em uma janela de propriedades, que permite especificar o nome e os atributos da entidade. Os tipos dos atributos são escolhidos em um *menu* associado à janela de propriedades. Para criar relacionamentos entre entidades, seleciona-se a entidade de referência (no caso de relacionamentos do tipo *muitos-para-um*) com o *mouse*, e ativa-se a operação *connect* do *menu* da janela gráfica de esquema, desenhando-se, através do *mouse*, uma linha até a entidade referenciada. Relacionamentos do tipo *muitos-para-muitos* podem ser construídos pela repetição deste processo.

O usuário pode renomear ou eliminar entidades e atributos, selecionando comandos a partir do *menu* associado à janela de esquema. Da mesma forma, o usuário pode modificar a disposição dos componentes gráficos do esquema. SCHEMADESIGN provê ainda uma função para localização de uma entidade pelo seu nome. Esta função é aplicada em esquemas grandes, que não podem ser inteiramente representados na janela de esquema.

A segunda ferramenta da interface proposta em [RC88] chama-se DATABROWSE, e permite a visualização e a edição de entidades lógicas, e não apenas de registros ou tuplas. DATABROWSE possui uma janela de mensagens, uma janela de comandos, uma

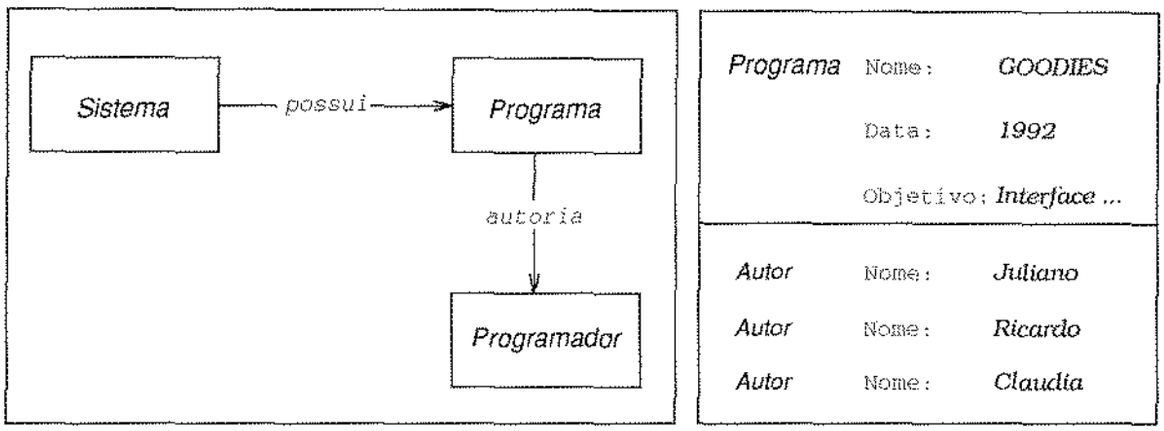


Figura 3.6: Representação de informações em SCHEMADESIGN e DATABROWSE

janela para navegação/edição de dados, uma janela de texto para apresentação/edição de campos do tipo *texto* e uma janela para apresentação de campos do tipo *bitmap*. Um *menu* associado à janela de navegação permite a seleção da entidade a ser visualizada. Os dados são apresentados na forma tabular, e cada nome de atributo é separado de seu valor por dois pontos (":"). Atributos do tipo *bitmap* aparecem como ícones.

A navegação pelas instâncias da entidade é feita por meio de *scrollbars* associadas à janela de dados. O usuário pode visualizar uma entidade lógica específica, selecionando-a com o *mouse*. O registro selecionado é mantido na tela, e os demais são eliminados da janela. Os registros referenciados pela entidade escolhida são apresentados no lugar dos registros eliminados. O mesmo processo de visualização pode ser repetido para as entidades referenciadas, e assim sucessivamente. No modo de edição, o usuário pode continuar a navegação, selecionando com o *mouse* os rótulos (nomes) de cada atributo. A seleção do valor de um atributo permite sua edição. Existem ainda opções para inserir e eliminar registros de entidades referenciadas.

DATABROWSE cria automaticamente a representação de uma entidade. No entanto, o usuário pode modificar esta representação, através de *menus* que permitem selecionar registros de referência, atributos desejados, e outros aspectos da representação da entidade. Os campos de tipos especiais (texto e *bitmap*) podem ser criados e editados por ferramentas externas, mantendo-se apenas o nome do arquivo correspondente no BD. A figura 3.6 mostra a maneira de se representar informações em SCHEMADESIGN (esquerda) e em DATABROWSE (direita).

As ferramentas SCHEMADESIGN e DATABROWSE apresentam características ge-

rais muito interessantes. Ainda assim, estas ferramentas poderiam ser melhoradas. Por exemplo, facilidades para movimentação de grupos de objetos poderiam ser incorporadas a SCHEMADESIGN, enquanto que DATABROWSE poderia ser estendido com mecanismos para formulação de consultas.

3.3.4 KIVIEW

KIVIEW é uma interface originalmente desenvolvida para o sistema KIWI, um pacote que provê acesso amigável para múltiplos BDs. Não obstante, KIVIEW pode servir como interface para qualquer modelo de dados a partir do qual uma rede semântica possa ser extraída [MDT89]. A motivação para a construção de KIVIEW veio da constatação que a maioria das ferramentas de navegação construídas para BDs oferece apenas operações triviais, que são praticamente inúteis em uma aplicação real.

O modelo interno de KIVIEW é uma rede semântica que consiste de objetos ligados por relacionamentos binários. O modelo externo (do usuário) é formado por uma estrutura de dados chamada de *visão*, a qual representa um objeto da rede semântica, bem como seus objetos adjacentes.

As seções de navegação de KIVIEW são intercalações de dois tipos de atividades: atividades de navegação propriamente ditas e atividades de manipulação de visões. Durante as atividades de navegação, o usuário explora o BD, visualizando os objetos referenciados nas visões. A manipulação é o processo pelo qual o usuário cria e modifica visões, onde são armazenados os resultados das operações de navegação efetuadas.

A rede semântica que define o modelo interno de KIVIEW pode ser entendida como um conjunto de triplas $< m, r, n >$, formadas por dois objetos (m, n) e por um relacionamento (r) entre eles. Estas triplas são denominadas *fatos*, e os objetos podem ser classes ou instâncias. Os relacionamentos entre objetos são de quatro tipos: entre classes (generalização), entre instância e classe (pertinência), entre instâncias (específico), e entre classe e instância (genérico). O par "r,n" de um fato é chamado de propriedade do objeto m.

A visão de um objeto inclui todos os fatos em que ele participa. Os fatos são classificados em quatro categorias — membro, superclasse, subclasse e propriedade — de acordo com o relacionamento representado. A visão de uma classe apresenta as quatro categorias de fatos, enquanto a visão de uma instância apresenta apenas as classes a que a instância pertence e as propriedades definidas para a instância. Os conteúdos de cada categoria de fatos são apresentados em janelas independentes, onde cada janela mostra um intervalo de objetos ou propriedades, em uma ordem particular. A figura 3.7 mostra a visão da classe *Programa*, com as suas quatro janelas de categorias de fatos.

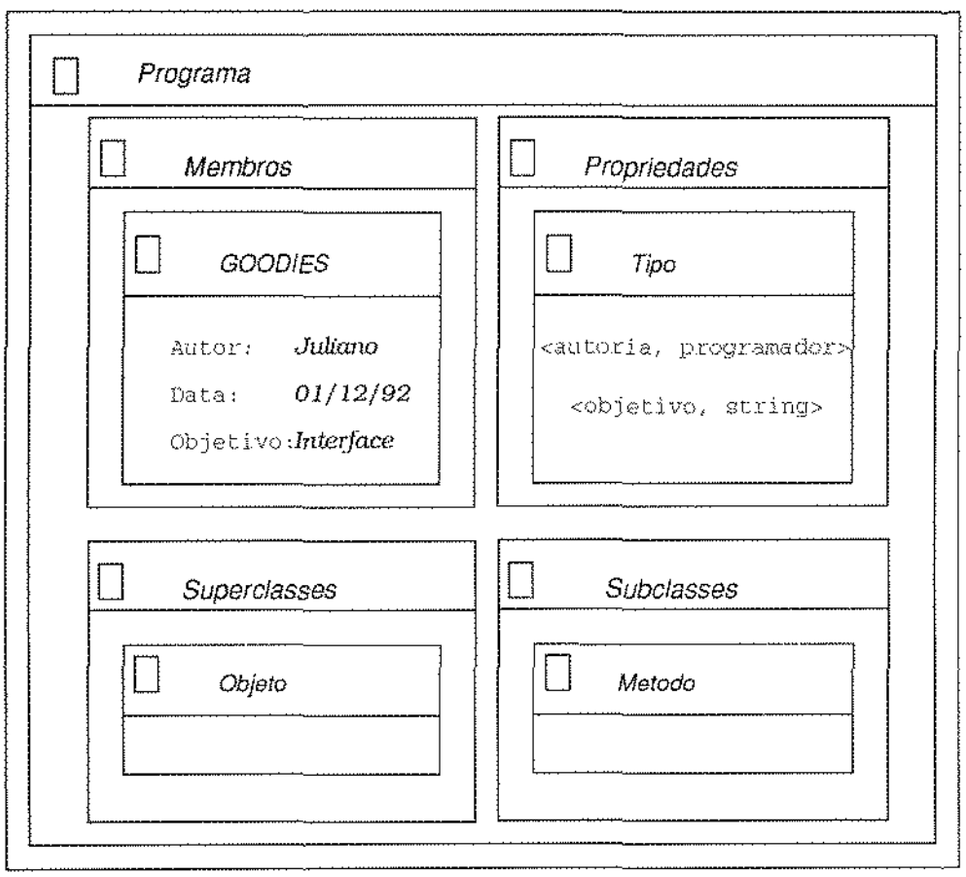


Figura 3.7: Representação de informações em KIVIEW

Para obter as janelas que compõem a visão de um objeto (classe ou instância), o usuário deve acionar o comando *Open* para o objeto desejado (o comando *Close* produz o efeito oposto, eliminando as janelas de visão do objeto). Em seguida, deve ser acionado o comando *Activate* para esse objeto, tornando a visão do objeto ativa, e desativando as outras visões. Dessa forma, existe no máximo uma visão ativa no sistema em um determinado instante. Na janela de visão ativa, o usuário pode usar os comandos *Order* para estabelecer o critério de ordenação dos elementos apresentados na janela, e *Scroll*, para controlar os dados visualizados na janela.

Na navegação normal, as diferentes visões abertas são independentes, e uma mudança em uma visão não afeta as demais visões. A sincronização é um mecanismo que permite a ligação de diversas visões em uma estrutura do tipo árvore, de modo que, ao mudar a visão raiz, os conteúdos dos seus filhos sejam automaticamente modificados. A operação de sincronização *Sync* (O, V) estabelece a visão V como filha do objeto O de outra visão, na árvore de sincronização. A sincronização pode ser desfeita pelo comando *Break* (V).

O efeito exato da sincronização depende do tipo de janela em que o objeto O é apresentado. Se O aparecia numa janela de membros ou de classes, a sincronização é posicional, isto é, se O ocupava a i-ésima posição na janela no momento da sincronização, então V irá representar sempre a visão do objeto que ocupa a i-ésima posição naquela janela. Se a janela que contém O é de propriedades, então a sincronização é semântica, ou seja, se a propriedade que ocupava a i-ésima posição naquela janela no instante da sincronização era $< r, O >$, então V mostrará a visão do objeto da propriedade que ocupa a posição "i" da janela, somente se o relacionamento dessa propriedade é r. De outro modo, V apresentará uma visão nula.

As operações de manipulação permitem a criação de classes virtuais, que não correspondem a objetos existentes no BD. Estas classes são usadas para guardar dados de interesse encontrados durante o processo de navegação. Assim, a manipulação pode ser considerada uma espécie de consulta ao BD. O operador *Create* cria uma classe virtual vazia no BD, e o operador *Include* torna a classe virtual criada uma generalização imediata de outra classe. O operador *Retain* torna a classe virtual uma especialização de outra classe, enquanto o operador *Restrict* estabelece uma restrição sobre uma propriedade da classe virtual. Existem ainda os operadores *Insert* e *Delete* que permitem, respectivamente, inserir ou remover uma instância específica de uma classe virtual.

3.3.5 PASTA-3

PASTA-3 é uma interface para KB2, um sistema de base de conhecimento que suporta o modelo E-R, estendido pelo conceito de herança e por regras dedutivas. O sistema

PASTA-3 foi projetado visando portabilidade, de modo que a adaptação a qualquer SBD que suporte o modelo E–R é feita com relativa facilidade [KM90]. PASTA-3 usa o paradigma de manipulação direta para diálogo com o usuário, e possui facilidades para manipulação de esquemas e de dados. Através de PASTA-3, o usuário pode projetar e modificar esquemas, navegar sobre esquemas e dados, consultar e atualizar informações. A execução destas tarefas é feita em um ambiente único, ao contrário de muitos sistemas onde as tarefas são realizadas através de ferramentas diferenciadas.

O paradigma de manipulação gráfica direta é usado, em PASTA-3, no seu sentido mais amplo: não apenas os comandos que operam sobre os dados são manipulados diretamente, mas as próprias representações dos dados também o são. As informações sobre os esquemas são apresentadas nos modos gráfico e textual (através de listas ordenadas de elementos do esquema). O acesso às funções do sistema é feito pelo seu *menu* principal, que contém opções para iniciar e terminar uma sessão de trabalho com um determinado BD, para iniciar o processo de consulta, para acionar comandos para navegação, para controlar e manipular as diversas janelas do sistema, entre outras.

A representação gráfica do esquema não contém as figuras tradicionais (retângulos e losângulos) do modelo E–R. PASTA-3 representa uma entidade pelo seu nome, escrito com letras normais; relacionamentos também são representados por seus nomes, escritos em negrito. Essa representação do diagrama E–R tenta reduzir a complexidade visual do esquema, que pode ser, dessa forma, representado em espaços menores da tela. Uma janela de herança é usada para refletir as derivações entre entidades. Nesta janela, as subclasses são identificadas por sua posição relativa às superclasses. Um meio simples de se obter estas posições relativas é pelo alinhamento das superclasses à esquerda da janela, deslocando-se as subclasses para a direita. As propriedades (atributos) das entidades não são apresentadas no diagrama, mas podem ser visualizados em janelas auxiliares.

Mais de um diagrama pode ser apresentado simultaneamente, em janelas distintas, o que permite a visualização de diferentes partes do esquema ao mesmo tempo. Além disso, as informações apresentadas em listas ordenadas podem ser manipuladas diretamente, ou com o auxílio de *menus*. Combinando estas duas capacidades, o sistema consegue representar esquemas de BDs bastante grandes, de maneira relativamente simples, ao contrário de muitas propostas que não se aplicam a esquemas que possuem um número considerável de entidades e relacionamentos.

Outras facilidades existentes para a manipulação de esquemas em PASTA-3 podem ser citadas. A manipulação direta do diagrama pode ser feita sobre um nó individual, ou sobre uma rede de nós, envolvendo entidades e relacionamentos. Mais ainda, o diagrama se ajusta continuamente às operações do usuário, de modo que a situação final possa ser visualizada previamente. Duas facilidades do sistema ajudam a construção do diagrama E–R, permitindo alinhar e agrupar entidades nas janelas gráficas.

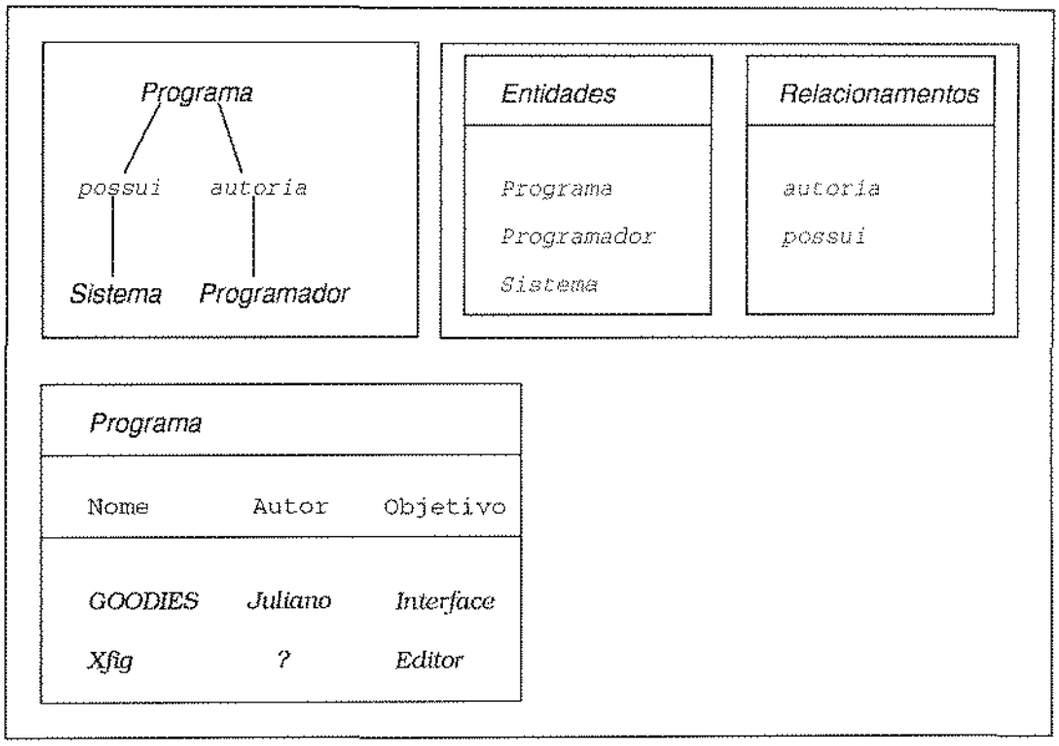


Figura 3.8: Representação de informações em PASTA-3

A navegação sobre o esquema pode ser feita de quatro modos distintos. Primeiramente, as janelas auxiliares de propriedades e o comando *Show Definition* permitem a visualização da definição de uma entidade (ou relacionamento) e de seus atributos. O segundo modo para navegação no esquema é através das listas ordenadas de elementos do esquema. A seleção de um nome nestas listas causa a apresentação do elemento correspondente na janela gráfica de esquema. Funções para referência cruzada de informações formam o terceiro modo de navegação sobre esquemas. O comando *Find ER itens*, por exemplo, apresenta na tela os elementos do esquema que possuem um determinado atributo. Finalmente, PASTA-3 oferece facilidades para identificar relacionamentos existentes entre duas entidades. Uma forma de se usar este tipo de navegação é a seleção de duas entidades no diagrama, o que leva o sistema a destacar todas as ligações (diretas ou não) existentes entre elas.

A navegação sobre dados é feita de forma simples ou sincronizada, como no sistema KIVIEW, visto anteriormente. Os dados são apresentados na janela de dados, de forma tabular. O usuário pode escolher os atributos que deseja visualizar na janela de dados, e os valores apresentados são sensíveis ao *mouse*, podendo ser selecionados para uso em outras janelas. Os atributos multi-valorados são representados pelo seu primeiro

elemento, seguido de reticências ("..."). A opção *Expand* permite a apresentação dos demais elementos do conjunto, em uma janela auxiliar.

O mecanismo de sincronização de PASTA-3 permite a navegação simultânea em entidades que se relacionam. Uma das entidades é escolhida como primária, e a outra tem seus valores alterados em função da entidade primária. O processo é similar às planilhas eletrônicas, que recalculam todos os valores quando o usuário modifica uma valor chave. A figura 3.8 mostra a representação de um esquema através do diagrama E–R (canto superior esquerdo) e de listas de elementos do esquema (canto superior direito), e a representação de uma janela de dados para a entidade *Programa*, de acordo com as definições de PASTA-3.

3.4 Interfaces para o Modelo Orientado a Objetos

Como foi visto, o modelo OO é originário dos chamados modelos de dados semânticos. Esta seção vai mostrar que as características adicionais do modelo OO contribuem para a maior complexidade de suas interfaces, notadamente pela necessidade de representação de *comportamento*. Além disso, as interfaces que serão apresentadas a seguir comprovarão as divergências existentes entre diversos sistemas que implementam o modelo OO.

3.4.1 SIG

O sistema SIG (*SmallTalk Interaction Generator*) [MNG86] é um gerador de apresentações interativas para objetos complexos em sistemas de bancos de dados orientados a objetos. O conceito básico de SIG é o de apresentação interativa ou ID (*Interactive Display*). Um ID permite não só a visualização, mas também a atualização dos objetos, e reflete dinamicamente a estrutura do objeto representado. Um ID é construído de maneira declarativa, e IDs complexos podem ser criados a partir de IDs mais simples. Uma classe pode ter vários IDs associados, e mesmo um objeto pode ser representado em diferentes IDs ao mesmo tempo, de modo que as atualizações efetuadas em um ID são refletidas nos demais, automaticamente. Por fim, um ID pode referenciar outros IDs, inclusive de modo recursivo.

Objetos em SIG são tuplas NF^2 que possuem identidade, e que podem compartilhar valores de atributos. Classes agrupam objetos que possuem estrutura e comportamento similares. Um objeto é uma instância de sua classe, e todo objeto possui um protocolo de mensagens ao qual ele responde, através de uma mudança de estado, ou através do retorno de informações. Este protocolo encapsula o estado interno do objeto. Assim,

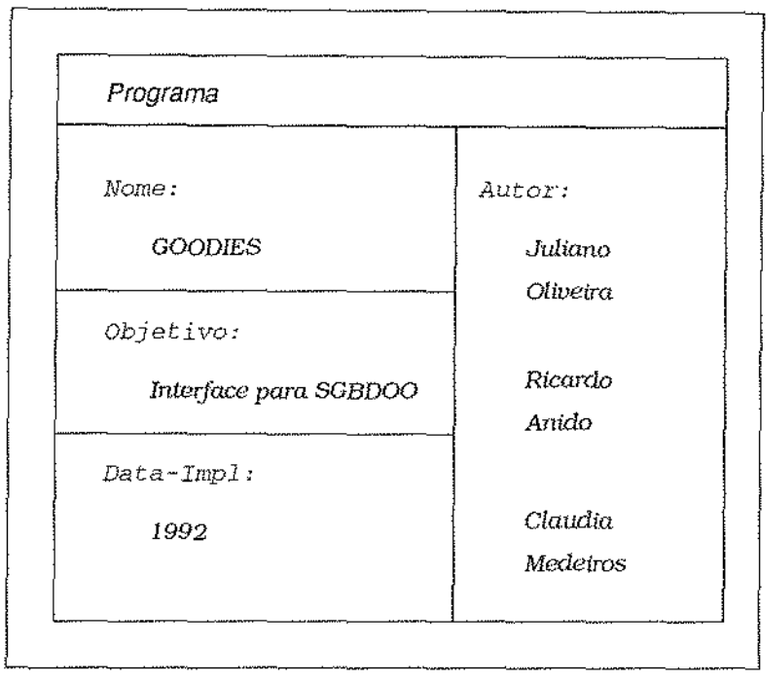


Figura 3.9: Representação de informações em SIG

um objeto, via de regra, não pode examinar ou modificar diretamente o estado de outro objeto.

Como foi visto, um ID pode ser composto por outros IDs. Estes IDs são chamados sub-IDs, e a cada sub-ID está associado um *menu* com os métodos próprios do sub-objeto representado. Dessa forma, um ID não viola o princípio de encapsulamento do estado interno do objeto. Uma janela gráfica é usada para representação de cada ID, e os sub-IDs são representados dentro da janela de seu ID. Entretanto, sub-IDs podem representar conjuntos de objetos, e SIG impõe algumas limitações sobre a representação de conjuntos. De fato, apenas quatro elementos de um conjunto podem ser representados em um sub-ID. São apresentados os três primeiros elementos, seguidos por reticências ("...") e pelo último elemento do conjunto. O comando *Expand* é utilizado para gerar um novo ID para representar todo o conjunto contido no sub-ID.

Todo ID está associado a uma classe, e SIG provê um editor para construção de novos IDs, baseado nos IDs já definidos para a classe. No entanto, o primeiro ID de cada classe deve ser inteiramente definido pelo usuário, mesmo que já exista um ID semelhante para outra classe do esquema. Dessa forma, o paradigma de manipulação direta não se aplica à definição dos IDs, que deve ser feita através de uma linguagem de comandos. Outra deficiência de SIG é a inexistência de mecanismos para mudar o nível de abstração das

informações apresentadas em um ID. A figura 3.9 mostra um ID para um objeto da classe *Programa*, de acordo com as definições do sistema SIG.

3.4.2 Interfaces do sistema O_2

O_2 é um SGBDOO que apresenta um sistema de interface bastante amigável, funcionalmente completo, e que pode atender aos diversos níveis de usuários de um SBD. Nesta seção, serão apresentadas três ferramentas de interface para o O_2: os sistemas LOOKS, TOONMAKER e OOPE.

LOOKS

O sistema LOOKS [Mam91] suporta a manipulação gráfica e interativa de objetos e valores complexos no SGBDOO O_2. LOOKS permite a visualização e atualização de objetos por meio de representações genéricas que fornecem suporte navigacional ao usuário, através da execução de métodos. Uma representação genérica é formada por um editor, uma barra de comandos, e um ou mais *menus* associados [Alt90b]. O editor de um objeto complexo é composto de vários editores subsidiários. Toda a apresentação do objeto é construída pelo editor, exceto a moldura que envolve a apresentação e a barra de comandos. O editor mais utilizado é o editor de tupla, onde uma barra de título contém o nome do objeto (ou da classe, se o objeto não possui um nome). À esquerda da apresentação são mostrados os nomes dos campos, em negrito, e à direita de cada campo, seu respectivo valor, que por sua vez, é também um editor. Os elementos de conjuntos são apresentados como filas de editores, em forma de ícone.

Os editores que compõem uma apresentação podem ser copiados, movidos ou eliminados por manipulação direta, e a sintaxe destas operações segue a definição tradicional dos comandos *Copy-Cut-Paste*. Para mover ou copiar editores, o sistema verifica a compatibilidade entre os tipos de origem e destino da operação. Toda apresentação possui um *menu* associado, que contém opções para imprimir o conteúdo da apresentação e para modificar o nível de abstração em que as informações são apresentadas, além dos métodos públicos associados ao objeto apresentado.

Os tipos de editores disponíveis em LOOKS são: tupla, caractere, *booleano*, lista, conjunto, *bitmap* e texto. O editor de tupla foi discutido acima; o editor de caractere engloba os editores de números inteiros e de números reais, que são apresentados da forma convencional. O editor de *booleanos* apresenta seu valor em uma caixa, contendo um "X", se o valor é verdadeiro, ou vazia, se é falso. O editor de listas mostra uma seqüência de sub-editores do mesmo tipo, separados por ícones de flechas, que indicam a ordem da

lista. A mesma forma de apresentação é usada no editor de conjuntos, eliminando-se, porém, as setas, pois não existe ordem entre elementos de um conjunto. Os editores de texto e de *bitmap* apresentam textos e figuras, respectivamente.

O sistema LOOKS, como foi visto, pode ser usado como uma interface direta para o usuário. Não obstante, LOOKS pode também ser embutido em aplicações, através de primitivas que executam funções LOOKS [Alt90a]. Como o sistema se baseia em uma arquitetura cliente/servidor, uma aplicação deve usar a primitiva *Prologue* para estabelecer uma conexão com o servidor LOOKS. A primitiva *Epilogue* tem efeito contrário, desconectando a aplicação do servidor.

A primitiva *Present* define uma apresentação para um objeto ou valor. Ela retorna um identificador de apresentação para o programa chamador, mas não mostra a apresentação na tela. *Present* recebe como parâmetros a identificação do objeto a ser exibido, e a indicação de permissão para atualização do objeto através da apresentação, entre outros. Para mostrar a apresentação na tela é usada a primitiva *Map*, enquanto as primitivas *Unmap* e *Destroy* produzem resultados opostos aos das primitivas *Map* e *Present*, respectivamente.

A aplicação transfere o controle da apresentação para o usuário através da primitiva *Lock*. A execução da aplicação fica bloqueada até que o usuário acione um comando para gravar ou abandonar a apresentação, ou até que outro método ou aplicação chame a primitiva *Unlock* para a aplicação, desfazendo o efeito da primitiva *Lock*. LOOKS permite ainda especificar títulos para apresentações, ativar ou inibir a execução de métodos a partir de uma apresentação, obter as apresentações criadas para um determinado objeto, entre outras facilidades. A figura 3.10 mostra a apresentação do objeto *GOODIES* da classe *Programa*, segundo os critérios do sistema LOOKS.

TOONMAKER

O sistema TOONMAKER [Mam91] permite a adaptação e modificação das apresentações geradas pelo sistema LOOKS. As vantagens adicionais das apresentações de TOONMAKER sobre as apresentações genéricas de LOOKS são: 1) possibilidade de limitar ou expandir a quantidade de informações exibidas em uma apresentação; 2) maior facilidade para ajuste dos parâmetros gráficos (tipo de letra, cores) e da decoração das apresentações; 3) capacidade de atribuir uma estrutura gráfica para os dados; e 4) possibilidade de adaptar o modo de interação entre o usuário e a aplicação, de acordo com a representação gráfica escolhida.

TOONMAKER possui um módulo de especificação, onde o usuário descreve a natureza dos componentes gráficos de uma apresentação. Uma especificação define um modelo de

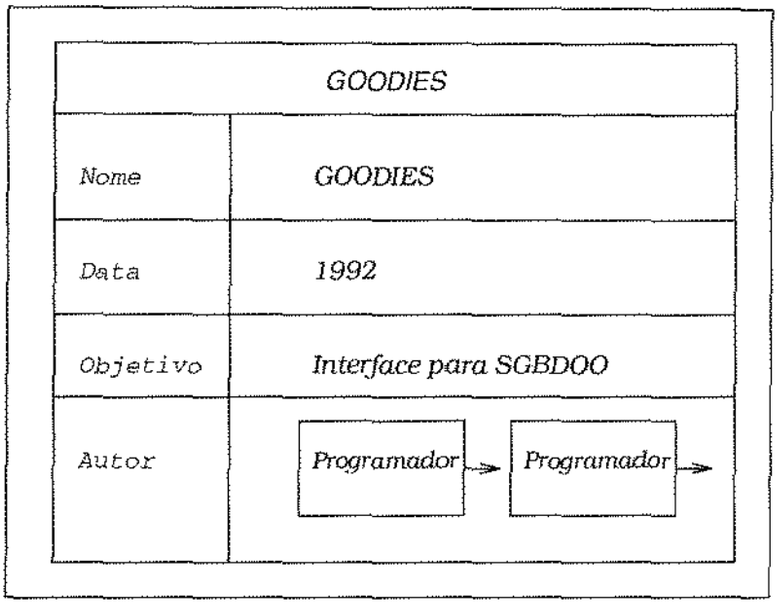


Figura 3.10: Representação de informações em LOOKS

apresentação que pode ser aplicado aos objetos de uma classe. Uma classe pode possuir diversas especificações de apresentação, de modo que o usuário pode escolher a que mais se adapte a cada aplicação. Mais ainda, bibliotecas de especificações de apresentação podem ser armazenadas no próprio BD.

A especificação de uma apresentação é feita através de um editor gráfico, onde a apresentação pode ser diretamente manipulada. A definição da apresentação pode tomar como base outra definição já existente, ou a apresentação genérica, criada pelo LOOKS. O editor de apresentação permite eliminar campos que não devem ser apresentados, e também expandir campos que são objetos, de modo que seja apresentada a estrutura do sub-objeto, e não um ícone, como na apresentação padrão.

O sistema permite a modificação da estrutura organizacional dos dados apresentados. Por exemplo, elementos de uma lista podem ser agrupados em um só campo. Pode-se acrescentar novos campos na apresentação, representando informações derivadas, ou seja, informações obtidas a partir dos dados efetivamente armazenados no BD. O editor interativo de apresentações de TOONMAKER trabalha com duas árvores: a árvore de edição, que apresenta a estrutura da classe editada, e a árvore de *toons*, que mostra a composição de construtores gráficos associados a cada nó da árvore de edição.

Um *toon* é um componente gráfico que pertence a uma classe de objetos gráficos,

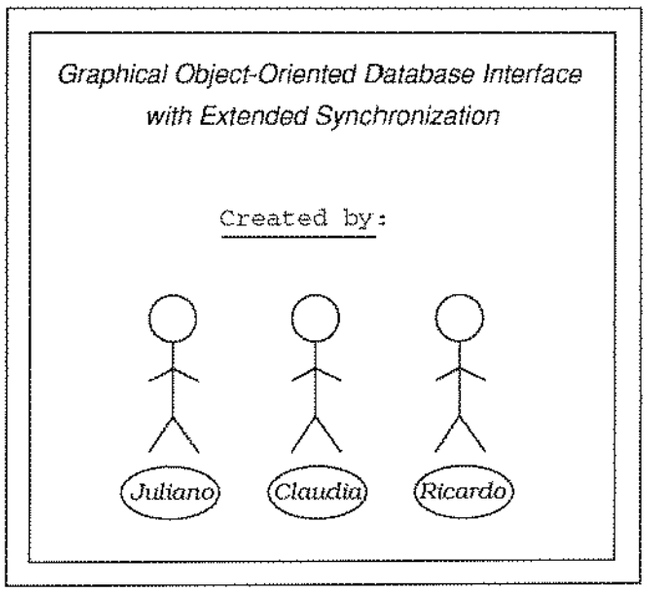


Figura 3.11: Representação de informações em TOONMAKER

possuindo métodos que implementam seu aspecto e seu comportamento. Um *toon* pode ser composto ou terminal. Os *toons* compostos são: *Form*, *Frame*, *Column*, e *Row*, e são nós internos da árvore de *toons*. *Column* dispõe seus descendentes verticalmente, e *Row* horizontalmente. Eles possuem recursos para definir a reação da estrutura a uma inserção, remoção ou substituição de elementos, e permitem definir atributos gráficos como margens, separadores de elementos, e outros. O *toon Frame* envolve seus descendentes em uma moldura cuja cor e estampa podem ser modificadas. *Form* dispõe seus descendentes em uma zona retangular, em coordenadas específicas.

Toons terminais são as folhas da árvore de *toons*. São eles: *Text*, usado para representar campos textuais; *Bitmap*, que mostra figuras na apresentação; *Button*, que pode ter um texto ou *bitmap* como rótulo, e permite a execução de métodos O_2; *Line, Rectangle, Ellipse* e *Polygon*, que permitem definir figuras e desenhos na apresentação.

Além do editor, TOONMAKER possui uma interface de programação, que permite criar ou modificar apresentações, dentro de um programa de aplicação. Para isso, TOONMAKER provê funções visando a definição da árvore de *toons*. A apresentação criada pelo editor ou pela interface de programação pode ser utilizada em uma aplicação, através da primitiva *Present*, do mesmo modo usado no sistema LOOKS. A figura 3.11 mostra a apresentação da figura 3.10, após uma sessão de edição em TOONMAKER.

OOPE

OOPE (*Object-Oriented Programming Environment*) [Alt90c] é outro sistema que provê facilidades de interface para o sistema O_2. As ferramentas de OOPE fornecem funções para: navegação sobre esquemas e dados; criação de estrutura de classes, objetos e métodos; depuração e teste de métodos e aplicações; e formulação de consultas, entre outras. OOPE usa as apresentações criadas pelo sistema LOOKS para mostrar e editar objetos. A própria definição dos esquemas é armazenada em classes do BD (meta-dados). OOPE pode ser configurado para satisfazer diferentes tipos de usuários. Os aspectos que podem ser modificados incluem: posicionamento e tamanho de janelas das ferramentas, ícones e títulos associados às janelas, entre outros.

As facilidades de manipulação de esquema de OOPE incluem navegação, criação, visualização e modificação de estruturas de tipo de classes (e de seus métodos). A ferramenta *Browser* possui a função *Display Class*, que apresenta a lista de todas as classes do esquema de um BD, em ordem alfabética. A seleção de uma classe nesta lista causa o aparecimento da apresentação da estrutura da classe. De modo semelhante, pode-se visualizar estruturas de objetos, a hierarquia de classes, e as instâncias das classes. A figura 3.12 mostra uma janela de apresentação da estrutura da classe *Programa*, de acordo com as definições usadas em OOPE.

A criação de uma nova classe é feita pela aplicação do método *Add-Subclass* na sua superclasse. Este método aparece no *menu* de métodos de toda apresentação de classe. A definição de uma classe é composta pelo nome, tipo, superclasses, subclasses, métodos públicos, métodos privados da classe, além da indicação de classe pública, e de informações sobre a classe. O método *Add-Subclass* cria uma apresentação de estrutura de classe contendo estes campos vazios. O usuário preenche os campos da apresentação, e pode, a seguir, compilar a classe, para que ela se torne conhecida no restante do sistema, ou apenas salvar a definição da classe.

Do mesmo modo, um método pode ser criado em uma classe pela execução de métodos pré-definidos para adicionar um método à classe. É criada uma apresentação da estrutura do método, contendo os campos: nome do método, indicador de método público, classe recebedora, informações, parâmetros, resultados, linguagem e corpo. Os campos indicador de método público e classe recebedora são pré-inicializados em função da classe sobre a qual foi acionado o método para criação de método. Os demais devem ser preenchidos, como na inserção de uma nova classe. O usuário pode compilar apenas a assinatura do método, tornando-o conhecido no sistema, ou compilar também o seu corpo, para que ele possa ser não só referenciado, mas também executado.

O método *Run-Methods* associado a toda apresentação de estrutura de classe, permite a execução de um método da classe descrita. É criada uma instância de teste da classe, e

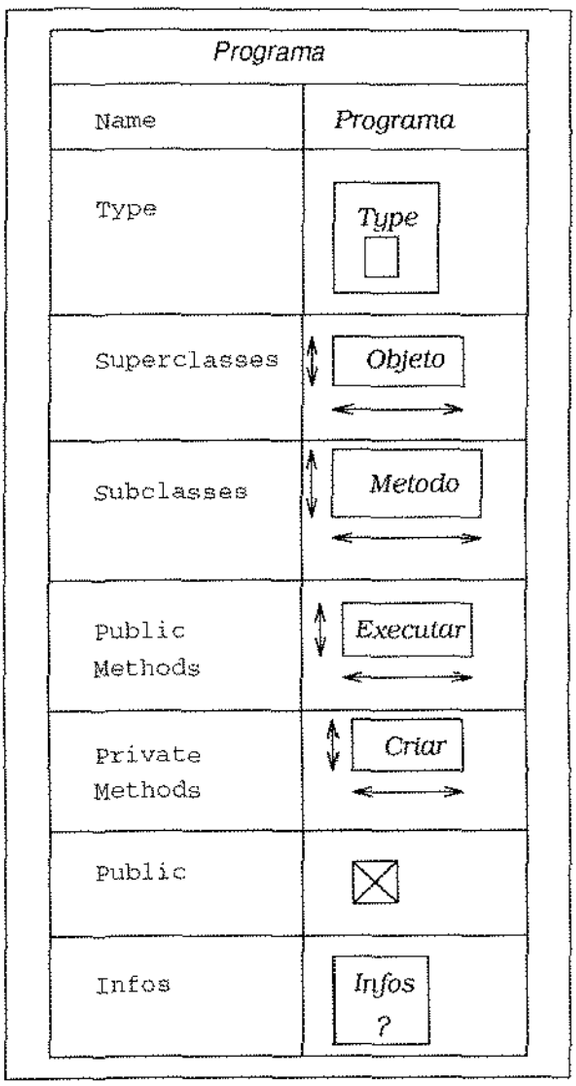


Figura 3.12: Representação de informações em OOPE

uma apresentação para essa instância, cujo *menu* contém os métodos compilados da classe. A instância de teste criada pode ser editada e salva no BD. A escolha de um método na apresentação desta instância fará com que ele seja executado sobre a instância, e se o método possuir parâmetros, o sistema irá solicitá-los.

O sistema O_2 não permite a modificação do tipo de uma classe. Dessa forma, OOPE fornece opções apenas para eliminar, compilar, e tornar uma classe pública ou privada. Para eliminar uma classe, o sistema verifica: 1) se ela não possui subclasses; 2) se ela não possui instâncias; e 3) se ela não participa da definição de tipo de outra classe do esquema. Métodos, por outro lado, podem ser livremente modificados e eliminados.

OOPE permite a criação de uma instância em uma classe, a partir de uma opção do *menu* da apresentação da classe. O sistema cria uma apresentação da estrutura do objeto, contendo: nome do objeto, classe, métodos públicos e privados, e informações. O usuário deve digitar o nome do objeto e utilizar o comando *Compile* do *menu* da apresentação da estrutura do objeto para inserir o objeto no BD. A opção *Initialize* deste *menu* cria a apresentação dos dados do objeto, que o usuário pode, então, editar. Métodos são criados para objetos da mesma maneira como é feito para as classes.

O campo *informações*, comum às apresentações de estrutura de classes, métodos e objetos, é, na verdade, um objeto composto pelos seguintes campos: *doc*, um texto contendo a documentação sobre a entidade; *errors*, um texto com as últimas mensagens de erro geradas pelo sistema para o elemento descrito; *compiled*, que indica se o elemento foi ou não compilado; e *bitmap*, que contém a imagem do ícone associado ao objeto, classe, ou método.

Além da ferramenta *Browser*, OOPE possui diversas outras ferramentas importantes. A ferramenta *Applications* permite definir uma aplicação sobre o BD, através de facilidades para criação de programas e de variáveis globais, e de opções para compilar, executar e depurar os programas e métodos. A ferramenta O_2 *Shell* é um editor de texto que permite a entrada e a execução de comandos do O_2, sendo útil para implementar comandos não disponíveis no OOPE. Um *workspace* pode ser usado como uma visão do esquema. Objetos podem ser inseridos ou retirados do *workspace*, e as modifica, coes podem ser salvas para futura utilização. Outra ferramenta (*Checker*), efetua a tradução de elementos do esquema criados fora do OOPE, para a definição armazenada no BD de meta-objetos do OOPE. Uma última ferramenta, *Queries*, possibilita a especificação de consultas *ad hoc* ao BD.

3.4.3 ODEVIEW

ODEVIEW [AGS90] é a interface gráfica para o SGBDOO ODE, provendo facilidades para navegação sobre esquemas e dados. O sistema ODEVIEW é apropriado para usuários que não desejam programar em O++, a linguagem de programação de ODE, que é uma extensão da linguagem C++. ODEVIEW se baseia no paradigma de manipulação gráfica direta, de forma que o usuário interage com o sistema pela seleção de itens em *menus* e botões, e pela movimentação e seleção de ícones.

Novas classes podem ser incluídas em ODE através de sua interface textual (O++), sem nenhuma mudança ou compilação em ODEVIEW, porque nenhum *meta-BD* é mantido pelo sistema. Ao contrário, ODEVIEW usa ligação dinâmica (*dynamic linking*) para chamar funções especiais das classes, que são responsáveis pela representação dos objetos pertencentes à classe. Cada classe deve prover essas funções (métodos) especiais, que são chamadas sempre que um objeto da classe for apresentado em ODEVIEW.

A janela inicial de ODEVIEW contém os ícones que representam os BDs gerenciados por ODE. Com o auxílio do *mouse*, o usuário seleciona o BD desejado, causando a abertura de uma janela contendo a descrição gráfica da hierarquia de classes para o BD selecionado. Esta hierarquia é representada por um grafo acíclico direcionado, cujos nós e arestas representam classes e relacionamentos de especialização, respectivamente. O usuário pode examinar o grafo de hierarquia em diferentes níveis de detalhe, através de operações de *Zoom*. Ele pode ainda escolher uma classe para visualizar em detalhe, selecionando-a com o *mouse* no grafo. Esta seleção causa o aparecimento de uma janela contendo informações sobre a classe.

A janela de informações sobre uma dada classe possui três subjanelas: duas janelas para mostrar superclasses e subclasses, e uma janela contendo meta-dados associados à classe, tais como o número de instâncias de objeto pertencentes à classe correntemente armazenados no BD. A janela de informações sobre classe contém um botão que causa a abertura de uma quarta subjanela, contendo a descrição textual do tipo da classe.

A navegação no esquema é feita pela seleção de outro nó do grafo de hierarquia, ou pela seleção de uma classe nas subjanelas de super/sub classes. A janela de informações sobre uma classe contém ainda um botão que permite o início da navegação sobre os dados. O botão *Objects* faz aparecer a janela onde são apresentados os objetos pertencentes à classe. Esta janela se divide em três partes. Na parte de controle estão os botões que realizam operações de navegação sobre os dados; no painel de objetos aparecem os botões para visualização de objetos; e na terceira parte, a parte de informações, aparecem os dados do objeto propriamente ditos.

Em ODEVIEW, um objeto pode ser apresentado de diferentes formas, dependendo da

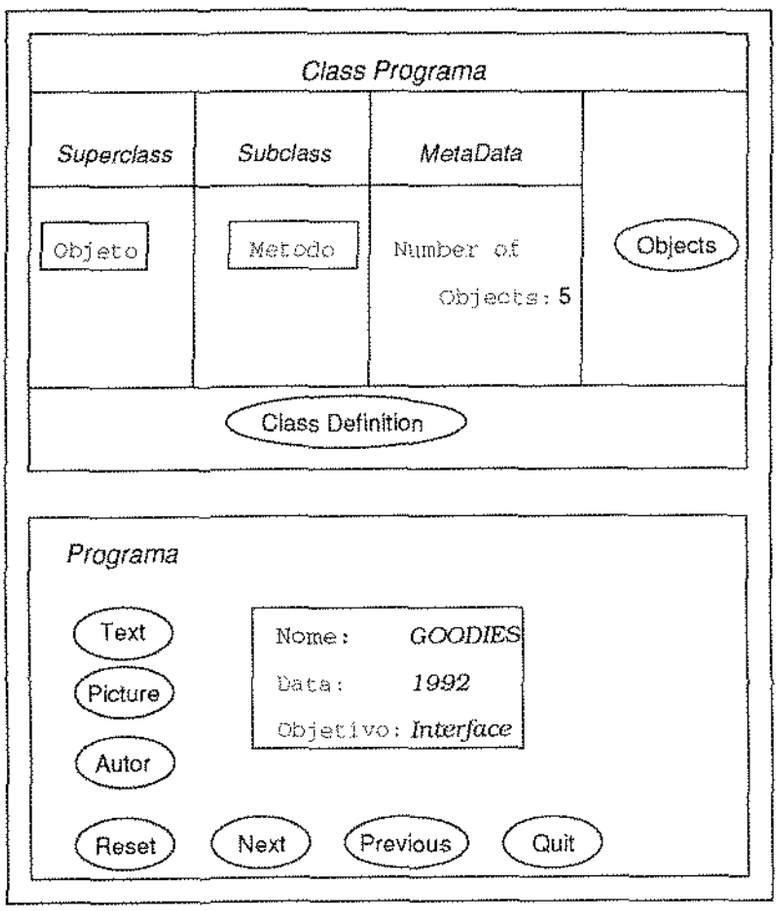


Figura 3.13: Representação de informações em ODEVIEW

semântica dos métodos que gerenciam as apresentações de uma classe. Estes métodos são especificados pelo projetista da classe, e a janela de objetos provê um botão para cada formato de apresentação definido para a classe, como por exemplo, os formatos textual e pictórico. A figura 3.13 mostra a representação de uma classe (acima) e de um objeto (abaixo), segundo as definições de ODEVIEW.

Objetos complexos são formados por outros objetos, que são visualizados em ODEVIEW através de botões situados no painel de objetos da janela de apresentação de objetos. A cada sub-objeto corresponde um botão, cujo acionamento causa a criação e exibição da janela de apresentação de objeto para o sub-objeto selecionado. Os botões situados na parte de controle da janela de apresentação de objeto são três: *Reset, Next* e *Previous*. Estes botões permitem a navegação sobre os dados, apresentando na janela de objetos, o primeiro objeto da extensão da classe, o próximo objeto, ou objeto anterior, respectivamente.

Como os sistemas PASTA-3 e KIVIEW, discutidos anteriormente, ODEVIEW permite navegação sincronizada: o usuário seleciona uma rede de sub-objetos e aplica, através dos botões da área de controle, uma operação de navegação sobre um objeto. Esta operação é automaticamente propagada por toda a rede de objetos definida. No entanto, mesmo com a capacidade de sincronismo, a navegação em ODEVIEW precisa de extensões para se tornar mais poderosa. Os próprios projetistas da interface indicam, em [AGS90], duas capacidades adicionais que deverão ser incorporadas ao sistema, que são a projeção e a seleção. A primeira capacidade permitirá a escolha dos atributos de um objeto que devem ser apresentados, e a segunda permitirá estabelecer condições a serem satisfeitas por atributos do objeto para que ele seja apresentado.

3.4.4 GS DESIGNER

O sistema GS DESIGNER (*GemStone Visual Schema Designer*) [Alm91] permite a definição interativa de classes e de relacionamentos entre classes, através de um projeto gráfico de esquema. GS DESIGNER é o editor gráfico de esquema para o SGBDOO GemStone, e oferece ao usuário facilidades para criar, modificar e remover definições de esquemas por manipulação gráfica direta.

O princípio básico de organização de informações em GS DESIGNER é o conceito de grafo de classes (*class graph*). Um grafo de classes pode ser considerado como uma coleção de classes que se relacionam entre si, de diversas maneiras. Um esquema contém vários grafos de classes, e uma classe pode pertencer a mais de um grafo de classes. O conjunto de grafos de classes pode ser entendido como a visão que a aplicação possui do BD, ou seja, o esquema conceitual da aplicação.

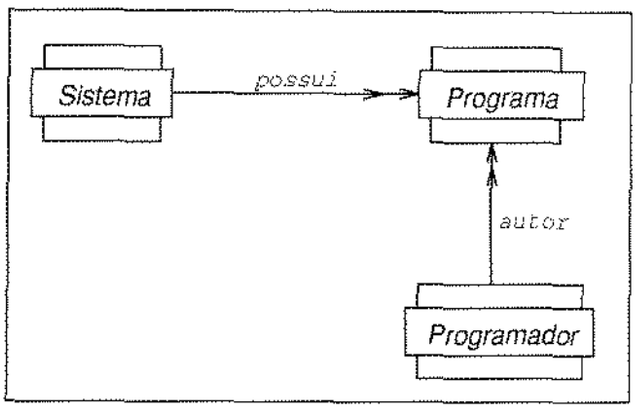


Figura 3.14: Representação de informações em GS DESIGNER

Todas as classes pertencem a uma única hierarquia de herança, cuja raiz é a classe *Object*, definida pelo sistema. Dessa forma, qualquer grafo de classes criado pelo usuário é um subgrafo conectado à hierarquia de classes. Os grafos de classes são úteis para particionar as classes em divisões lógicas, para cada aplicação específica. Em um grafo de classes, relacionamentos podem ser escondidos, diminuindo-se a complexidade visual de grandes esquemas. Uma característica importante de GS DESIGNER é que ele não permite que o usuário crie um grafo de classes inválido. Assim, todo grafo de classes representa um estado consistente do BD.

GS DESIGNER possui três janelas principais. A janela de esquema contém um ícone para cada grafo de classes definido para o esquema do BD, e inclui os quatro grafos de classes criados pelo GemStone. Estes grafos criados pelo SGBD contêm as classes embutidas, como *Integer* e *String*. A janela de definição de classe permite a definição textual de uma classe do esquema. Modificações realizadas sobre uma janela são imediatamente refletidas nas demais.

A terceira janela é a janela de grafo de classes, que contém retângulos, representando as classes, e flechas ligando os retângulos, que indicam os relacionamentos entre as classes. Uma flecha com o rótulo *IsA* representa o relacionamento de especialização entre uma subclasse e sua superclasse. Flechas indicando subobjetos são rotuladas com o nome do atributo cujo domínio é a classe apontada pela flecha. Flechas com pontas duplas simbolizam atributos multi-valorados.

As principais operações que podem ser executadas em GS DESIGNER são a criação e remoção de grafos de classes, a definição e modificação de classes, o salvamento do esquema definido como um objeto do BD, a importação de classes definidas em O++,

transformando-as em grafos de classes, a geração de relatórios com as definições das classes, e a exportação da definição de classes ou do esquema de um BD, para uso em outros BDs.

Em GS DESIGNER, classes e relacionamentos podem ser representados de diversos modos. A representação padrão para um grafo de classes, como já foi dito, é feita com retângulos e flechas. No entanto, outras representações podem ser especificadas, e as mesmas operações de manipulação direta podem ser aplicadas em qualquer representação. A figura 3.14 apresenta um grafo de classes, de acordo com a representação padrão de GS DESIGNER.

3.4.5 GOOD

O sistema GOOD (*Graph-Oriented Object Database*) [PBA+92] permite a visualização do esquema de um BDOO, através de um grafo rotulado e direcionado, cujos nós representam classes de objetos, e cujos arcos representam relacionamentos (ou propriedades) que podem existir entre os objetos dessas classes. Nós retangulares são usados para representar classes abstratas (definidas pelo usuário), enquanto nós ovais representam classes básicas (definidas pelo SGBD).

Em GOOD, mesmo uma instância de uma classe pode ser vista como um grafo, onde cada sub-objeto e cada valor são representados por nós únicos, de acordo com o conceito de identidade de objetos do modelo OO. Cada nó é rotulado com um nome de classe, e nós de classes básicas são rotulados adicionalmente com seu respectivo valor. Os arcos do grafo de instância são rotulados de acordo com os relacionamentos apresentados no grafo de esquema. A figura 3.15 apresenta um esquema de BD, conforme a representação definida pelo sistema GOOD.

Consultas são formuladas através de subgrafos obtidos a partir dos componentes do grafo de esquema. A estrutura do grafo de consulta especifica as partes do BD que devem ser recuperadas na consulta. Um grafo de consulta se assemelha a um grafo de instância, exceto pelo fato de que, na instância, os nós de classes básicas são rotulados com seus valores, enquanto no grafo de consulta isto não ocorre. A aplicação do grafo de consulta a um grafo de esquema resulta em um certo número de *casamentos de padrão*, que correspondem às instâncias das classes que são iguais (exceto pelos seus valores de rótulo) ao grafo de consulta.

Assim como a consulta, todas as operações de GOOD possuem o efeito correspondente a uma transformação em grafos. Uma operação, portanto, consiste em um subgrafo e uma ação aplicada às instâncias que *casam* com o padrão estabelecido pelo subgrafo. As operações definidas em GOOD incluem adição e remoção de arcos e nós do grafo de

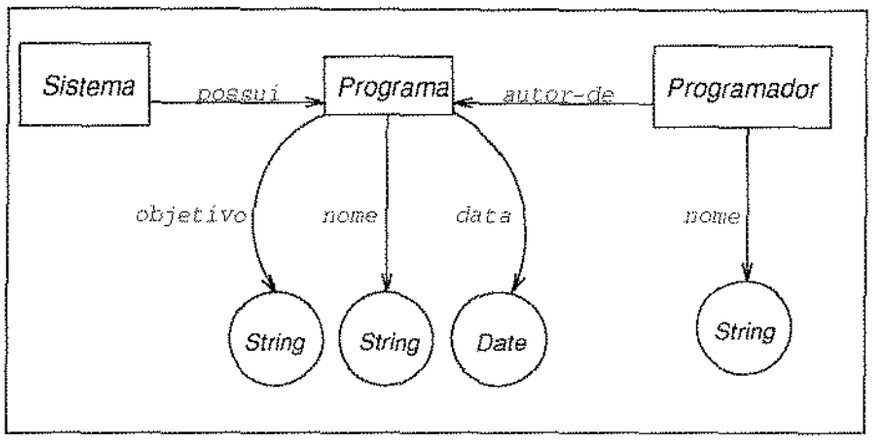


Figura 3.15: Representação de informações em GOOD

esquema. Para isso, duas facilidades adicionais são utilizadas: composição e decomposição de esquemas. A decomposição permite a fragmentação do esquema, para que se destaque dele uma determinada classe. A composição tem efeito contrário, conectando uma classe, através de um arco, ao grafo de esquema. Dessa forma, o sistema é flexível, e garante ao usuário a capacidade de alteração de esquemas.

A montagem de um subgrafo para aplicação de operações é feita por manipulação direta. O usuário pode copiar, identificar e remover nós e arcos do grafo de esquema, selecionando o elemento desejado com o *mouse*. Uma vantagem deste tipo de construção é que ela evita grande parte dos erros que poderiam ser cometidos se o subgrafo fosse especificado por meio de uma linguagem de programação. Além disso, um subgrafo obtido por manipulação direta pode ser utilizado em programas GOOD. Mais ainda, a interface provê mecanismos para visualização e navegação nos resultados produzidos por estes programas.

Embora possua um nível elevado de características gráficas, algumas aplicações em BDs não podem ser modeladas em GOOD de forma natural, por dois motivos principais. Primeiro, os arcos dos grafos não possuem qualquer informação associada a eles, a não ser os seus rótulos, de modo que relacionamentos com semântica devem ser modelados como nós. Em segundo lugar, o modelo de GOOD é *plano*, no sentido de não permitir a representação de informações aninhadas, o que dificulta, por exemplo, a representação de diferentes níveis de abstração para os dados.

3.4.6 FACEKIT

O sistema FACEKIT [KN92] é um conjunto de ferramentas que suporta o projeto de interfaces para BDOOs, combinando técnicas de *UIMSs*[2] com o conhecimento embutido sobre os projetos de SGBDOOs. FACEKIT é um sistema gráfico interativo, baseado em janelas, e embora possa ser considerado um UIMS, se destina a um grupo específico de interfaces, que são aquelas que se relacionam a sistemas de bancos de dados orientados a objetos. Para isso, FACEKIT engloba conhecimentos sobre esquemas, hierarquias de herança, métodos e ferramentas para definição de dados, entre outros. Dessa forma, o sistema incorpora conhecimentos específicos sobre SGBDOOs a um UIMS, permitindo que uma interface projetada em FACEKIT seja integrada ao BD, utilizando o seu modelo de dados, e tendo acesso ao seu esquema, e às linguagens de consulta do SGBD.

FACEKIT utiliza o modelo de dados e as ferramentas do SGBDOO CACTIS para manipulação de dados e esquemas. Os objetos de uma interface possuem a mesma estrutura dos objetos dos BDs, e são armazenados nos próprios BDs. FACEKIT constrói, mantém e executa métodos que produzem a representação visual dos objetos. Esta abordagem permite a mudança de comportamento de uma interface, à medida que os dados do BD são modificados. A figura 3.16 mostra uma janela para definição gráfica de representações, contendo os botões das funções disponíveis em FACEKIT.

UIMSs tradicionais permitem que o usuário especifique o formato da tela, relacionando cada ação possível a uma rotina da aplicação. FACEKIT define uma interface em termos de aparência e funcionalidade. A funcionalidade da interface é definida por métodos, e a sua aparência envolve objetos de interface (*widgets*) e objetos do BD propriamente dito. Definir a aparência de objetos do BD envolve a especificação de representações para classes de objetos. Uma representação pode ser idêntica para todos os objetos de uma classe, ou pode ser dependente dos dados do objeto, de maneira que o tipo de resultado de uma consulta, por exemplo, determine a aparência de sua representação. Na verdade, a aparência de uma representação pode depender inclusive de dados externos (como o recebimento de mensagens, ou a variação do *clock* do sistema).

Qualquer tipo de dado que não tenha uma representação definida pelo usuário emprega a representação padrão, criada pelo sistema. Esta representação é criada pela associação de um método à definição da classe. Os dados que este método utiliza são, do mesmo modo, acrescentados à definição da classe. O objetivo desse encapsulamento é controlar as várias representações disponíveis para um determinado tipo de objeto, favorecendo, ainda, a construção de bibliotecas de representações, e conseqüentemente, facilitando a reutilização dessas representações. O armazenamento como objetos do BD permite a fácil

[2]UIMS é um acrônimo para *User Interface Management System*, e representa uma classe de software que visa suportar o projeto, implementação e utilização de sistemas de interface.

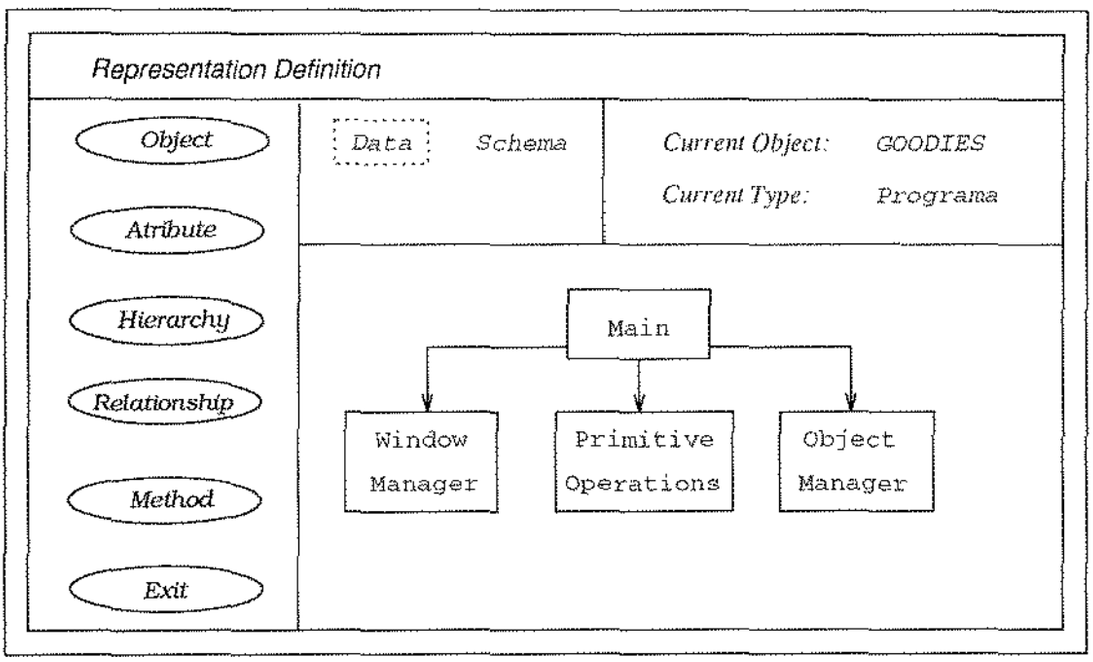


Figura 3.16: Definição de uma interface em FACEKIT

manutenção das representações, pois todas as facilidades e ferramentas do SGBD podem ser utilizadas para manipulação das representações.

Representações podem ser relacionadas, formando representações complexas, e podem ser redefinidas e modificadas, para a criação de novas representações. O armazenamento das apresentações, com já foi dito, não é feito na forma de imagens, mas sim como métodos dependentes de um conjunto de atributos. Esses atributos não precisam ser explicitados na chamada do método, porque eles são definidos no esquema, e portanto encapsulados na classe do objeto ao qual a representação se relaciona. Assim, o método sabe *a priori* onde encontrar seus argumentos.

3.5 Estudo Comparativo das Interfaces

Os primeiros SGBDs que implementaram o modelo relacional ofereciam apenas linguagens de comandos como mecanismo de interface com o usuário. Estas interfaces se tornaram, com o domínio dos sistemas relacionais sobre o mercado mundial, o mecanismo padrão de interação entre um usuário e o SGBD, com destaque para as linguagens SQL e QUEL. Com a popularização dos SBDs, a figura do usuário ganhou importância, e os projetistas de BDs atentaram para a necessidade de interfaces mais amigáveis. O sistema QBE foi uma primeira tentativa de facilitar o diálogo entre usuário e SGBD. Apesar de ter representado um avanço considerável com relação às linguagens de comandos, QBE não provê a facilidade de utilização desejável em uma interface para BDs.

O desenvolvimento da tecnologia de *hardware* permitiu o surgimento de estações de trabalho de grande capacidade de processamento, e de visores gráficos de alta resolução. Acompanhando esse desenvolvimento, surgiram, na área de *software*, sistemas gráficos poderosos, capazes de representar informações pictoricamente. Os sistemas relacionais foram beneficiados por estes avanços tecnológicos, através de trabalhos como o pioneiro SDMS [Her80], e mais recentemente com o sistema PICASSO, desenvolvido por Kim, Korth e Silberschatz [KKS88], que já se baseia em uma extensão do modelo relacional.

SDMS, embora seja um sistema muito limitado, tem importância histórica, por se tratar de uma das primeiras propostas de interface gráfica para BDs. O sistema só permite a navegação sobre os dados de uma relação do BD, sem a possibilidade de visualização simultânea de outras relações. Já o sistema PICASSO utiliza um conjunto de janelas para apresentar, ao mesmo tempo, informações sobre as diversas relações contidas em um BD. Os pontos positivos deste sistema são a capacidade de formulação gráfica de consultas; a possibilidade de se construir consultas complexas a partir de resultados de outras consultas; e o suporte à navegação sobre resultados de diversas consultas já realizadas. Como pontos negativos podem ser destacados a impossibilidade de atualização das in-

formações visualizadas através da interface, e a dificuldade de construção e representação do hipergrafo de esquema, para BDs complexos.

Apesar de os sistemas relacionais (e suas extensões) terem sido beneficiados com o advento dos recursos gráficos, são os modelos semânticos, e de forma destacada o modelo OO, que necessitam e utilizam em grande escala as novas capacidades gráficas dos computadores. A visualização de objetos complexos, que não podem ser representados de maneira simples como as tabelas utilizadas no modelo relacional, é uma das aplicações evidentes dos recursos gráficos em interfaces para os novos modelos de dados.

Os sistemas ISIS, SNAP e SIG representam as primeiras implementações de interfaces gráficas para os modelos semânticos surgidos a partir da década de oitenta. Estes sistemas influenciaram os trabalhos posteriores, pelas características de utilização de representações gráficas para esquemas e dados, pelo uso de sistemas de janelas para apresentar diferentes aspectos das informações, e pela preocupação em tornar o diálogo com o usuário o mais natural possível, através do paradigma de manipulação gráfica direta. Além disso, as deficiências apresentadas por estes sistemas motivaram o desenvolvimento de novas interfaces.

O sistema ISIS [GGKZ85] se destaca por ser uma interface completa, no sentido de prover não só acesso às funções de navegação e consulta, mas também permitindo a atualização de dados e esquemas. Entretanto, ISIS restringe a capacidade de expressão do seu modelo de dados (SDM), por não permitir a representação do conceito de herança múltipla. Outro ponto negativo de ISIS é a representação das classes do esquema através de padrões gerados pelo sistema. Este tipo de representação só é eficiente para esquemas muito simples, de forma que os sistemas mais recentes não adotam esta abordagem usada em ISIS.

Já os sistemas SNAP e SIG perdem para o sistema ISIS em termos de facilidades. SNAP [BH86] provê pouco suporte à navegação sobre dados; a única maneira de se visualizar as informações de uma determinada entidade é especificar uma consulta. Não existem, porém, facilidades para reutilização de resultados em consultas subseqüentes. A ênfase de SNAP é o projeto de esquemas, embora a representação utilizada para os componentes de um esquema seja deficiente, devido à utilização de um número excessivo de símbolos, que tornam o diagrama muito complexo.

Abordagem oposta é utilizada no sistema SIG [MNG86], onde apenas objetos do BD podem ser representados. Dessa forma, SIG não oferece funções para visualização e manipulação de esquemas de BDs. A representação de objetos complexos é o ponto forte de SIG, tanto que diversos sistemas recentes adotam representações muito semelhantes para esse tipo de informação. Além do formato das representações, SIG introduz uma maneira de se manter o encapsulamento dos subobjetos que compõem um objeto com-

plexo, através do aninhamento de representações, onde cada representação traz o menu de métodos definidos para a classe de objetos que ela representa.

Das interfaces apresentadas nas seções anteriores, duas utilizam o modelo E–R para representação de suas informações. O sistema proposto por Rogers e Cattell em [RC88] representa um esquema através de retângulos e flechas, enquanto o sistema PASTA-3 , de Kuntz e Melchert [KM90] elimina totalmente as figuras geométricas do diagrama E–R. Esta abordagem diminui a complexidade visual do diagrama, sendo mais apropriada para representação de esquemas com grande número de entidades e relacionamentos. PASTA-3 supera ainda a interface apresentada em [RC88] pela exploração da semântica do modelo E–R em seus mecanismos de navegação. A facilidade oferecida por PASTA-3 para navegação sincronizada em diversas entidades justifica esta afirmativa. Outra vantagem de PASTA-3 é a apresentação de informações sobre esquemas nos modos gráfico e textual, onde ambas as representações podem ser diretamente manipuladas. Por outro lado, a representação de dados na interface descrita em [RC88] utiliza um método de representação de entidades mais adequado que o o formato tabular convencional utilizado em PASTA-3.

O sistema KIVIEW [MDT89] não pretende ser uma interface completa para SGBDs. O objetivo de KIVIEW é facilitar a pesquisa exploratória dos BDs, por parte de usuários novatos ou casuais. Logo, KIVIEW não provê funcionalidade para construção de esquema, nem para atualização de informações. Sem embargo, KIVIEW apresenta uma grande facilidade para navegação sobre esquemas e dados. De fato, o processo de navegação em KIVIEW é tão poderoso quanto um mecanismo de consulta. A maior contribuição de KIVIEW está na definição do conceito de navegação sincronizada, que foi posteriormente empregado em diversos sistemas, como PASTA-3 e ODEVIEW. Também a classificação dos tipos de informações que devem ser apresentadas em uma visuallização de estrutura de classe, introduzida em KIVIEW, constitui uma contribuição expressiva.

Assim como KIVIEW, ODEVIEW é uma ferramenta para navegação, não provendo suporte para atualização de esquemas ou dados. Aliás, o mecanismo de navegação utilizado em ODEVIEW se inspira em grande parte nos conceitos de KIVIEW, inclusive no que diz respeito à sincronização. Entretanto, ODEVIEW não oferece facilidades para armazenar resultados do processo de navegação. O projeto de ODEVIEW foi também influenciado pelo sistema SIG, no que concerne à representação de objetos, embora a representação utilizada em ODEVIEW seja mais consistente. Em SIG, a representação de um subobjeto pode ser especificada tanto em um método da classe que define o objeto complexo quanto em um método associado à própria classe do subobjeto; já em ODEVIEW um subobjeto utiliza sempre a representação definida em sua própria classe. Assim, em ODEVIEW cada classe deve ter um método específico para criar as representações de seus objetos. O problema é que o sistema não oferece suporte para que o projetista faça a

definição gráfica da apresentação, que poderia ser, subseqüentemente, convertida para a especificação textual do método que cria a apresentação. Ao contrário, as representações de objetos em ODEVIEW são definidas através da linguagem de comandos do SGBD ODE.

A tendência de se construir ferramentas de interface separadas para cada tipo de tarefa executada em um SBD é seguida pelo sistema GS DESIGNER [Alm91]. O sistema PASTA-3 adota abordagem oposta, propondo a implementação de todas as ferramentas em um único ambiente integrado. No entanto, um dos autores de PASTA-3, em um trabalho mais recente ([Kun92]), reconhece que uma única ferramenta não pode ser adaptada, de maneira ótima, a todas as tarefas e tipos de informações existentes em um SBD. A tarefa a que GS DESIGNER se dedica é a construção e evolução de esquemas. Portanto, nenhum suporte à navegação sobre dados é oferecido, nem qualquer tipo de manipulação de objetos dos BDs propriamente ditos é permitida. Em se tratando de manipulação de esquemas, entretanto, GS DESIGNER é mais que um editor gráfico que permite a manipulação direta de componentes para a definição de esquemas. O sistema incorpora conhecimento sobre as regras de construção de esquemas, de modo a garantir que um esquema projetado através de GS DESIGNER seja correto. Outro ponto interessante de GS DESIGNER é a possibilidade de representar elementos do esquema em diferentes formatos, em um único diagrama. A representação de uma classe, por exemplo, pode ser feita por um retângulo, ou por uma moldura contendo determinados atributos da classe. Dessa forma, pode-se destacar elementos importantes de um esquema.

As ferramentas LOOKS, TOONMAKER e OOPE, se consideradas de forma isolada, apresentam as mesmas deficiências apontadas para diversas interfaces já estudadas, notadamente no que se refere ao conjunto de funções. LOOKS [Alt90b] provê uma estrutura para representação de objetos complexos, baseada em editores especializados, que supre, em grande parte, as necessidades de visualização e manipulação de informações dos usuários. TOONMAKER [Mam91] oferece facilidades para transformar as representações de LOOKS, de acordo com as necessidades de aplicações específicas. No entanto, estes sistemas não suportam manipulação ou visualização de esquemas. Por outro lado, OOPE oferece facilidades para manipulação de esquemas, mas não possui um mecanismo eficiente para navegação sobre objetos dos BDs. A associação das três ferramentas torna o sistema de interface do SGBD O_2 bastante poderoso, constituindo-se em um dos principais destaques do O_2, de acordo com diversos trabalhos recentemente publicados ([Deu90], [Deu91], [BMP+92], e [Sol92]) sobre este sistema.

Embora já existam ferramentas de alta qualidade implementadas, como as do sistema O_2, a área de pesquisa sobre interfaces para BDs ainda está em aberto. É grande o esforço empregado em busca de novos modelos e representações, que tornem o diálogo com o usuário o mais natural possível, e que reflitam, da mesma maneira, a estrutura e a

semântica das informações. Os sistemas GOOD e FACEKIT são amostras de resultados atuais de pesquisas nesta direção. GOOD [PBA+92] é uma tentativa de uniformização de representações de esquemas e dados por meio de grafos. A representação de esquemas por grafos já demonstrou ser adequada para muitos tipos de aplicações. No entanto, o formato específico de representação proposto por GOOD parece ser impróprio para BDs com muitos tipos de classes, devido à complexidade visual resultante da utilização de três tipos de figuras geométricas, ligadas por flechas rotuladas. Argumentação semelhante pode ser aplicada com relação à representação de objetos complexos.

FACEKIT [KN92], por sua vez, tenta conciliar as características de UIMSs e SGBDs, para facilitar a construção de interfaces. O sistema utiliza as facilidades de definição de apresentações oferecidas pelos UIMSs, armazenando estas apresentações como objetos do BD. Dessa forma, é possível utilizar todas as facilidades de manipulação de dados do SGBD para a manutenção e controle das apresentações. O ponto negativo deste tipo de abordagem é a forte interdependência existente entre o UIMS e o SGBD utilizados. Entre outros problemas, a evolução de uma das ferramentas pode ser limitada pela outra. O ideal é que a ferramenta de interface seja independente, tanto do SGBD quanto da aplicação para a qual se destina.

3.6 Características de Interfaces para SGBDs

A interface com o usuário é um aspecto do projeto de sistemas que vem ganhando importância cada vez maior. É notório que o sucesso de um produto pode ser medido pela sua aceitação por parte de seus usuários, e por essa razão, alguns sistemas chegam a conter mais linhas de código para gerência da interface do que para realizar a atividade específica da aplicação a que se destinam [MvD91].

Esta seção propõe algumas diretivas que devem orientar o projeto de uma interface gráfica para sistemas de bancos de dados. Em seguida, os sistemas de interface gráfica apresentados nas seções anteriores são analisados, sob o ponto de vista das diretivas propostas.

3.6.1 Diretivas para Projeto de Interfaces

Grande parte dos trabalhos publicados na área de interfaces apresentam regras, princípios e critérios para projeto, objetivos a serem atingidos, e problemas a serem resolvidos por um sistema de interface com o usuário. Algumas destas propostas são genéricas; outras analisam o problema do desenvolvimento de interfaces sob o ponto de vista de uma

aplicação em particular.

Serão introduzidas, a seguir, dez diretivas que devem ser consideradas no projeto de uma interface gráfica para sistemas de bancos de dados. As quatro primeiras diretivas representam uma síntese de características introduzidas por diversos trabalhos anteriores, entre os quais podem ser destacados [Shn87], [Wel88], [Sch90], [Gim90], [Sun90], [MvD91] e [Mam91]. As demais características foram identificadas à luz do estudo realizado na seção anterior.

1. Transparência e retroalimentação:

   Uma interface é transparente quando o usuário, a qualquer instante, sabe o que está se passando no sistema. Este conceito pode ser estendido pela noção de contexto, que representa o estado do sistema em um determinado momento. O usuário deve ter conhecimento das ações possíveis em cada contexto, e da maneira como essas ações podem ser ativadas.

   Para isso, a interface deve prover retroalimentação de informações, através de mensagens do tipo *"Execução do comando em progresso"*, indicando ao usuário a ação que o sistema está desenvolvendo. É importante que o usuário seja orientado sobre mudanças de estado do sistema, através da troca de cores, ou de mudanças na forma do cursor. Em interfaces para SGBDs, o problema de retroalimentação é crítico, notadamente nos processos de consulta ou navegação, quando se deseja investigar o BD de forma progressiva, com base em resultados intermediários obtidos.

2. Concisão e qualidade de apresentação:

   Uma interface gráfica deve ser concisa em dois aspectos: a) na apresentação de opções e informações ao usuário; e b) no volume de entrada de dados exigido para que o usuário expresse suas necessidades. Cada tela ou janela da interface deve conter apenas as informações necessárias. É um erro apresentar dados que nunca serão utilizados pelo usuário. Além disso, o número de opções apresentadas nos menus deve ser limitado, e a memória do usuário não pode ser utilizada como parte da interface. Dessa forma, ao mudar de contexto, o sistema deve permitir a recuperação das informações anteriores, se o usuário for precisar delas para qualquer operação, ou para entendimento do novo contexto.

   Do mesmo modo, a interface deve ser sucinta, ou seja, o usuário deve poder expressar seus desejos e ações de maneira rápida e direta. A resposta do sistema deve seguir o mesmo estilo, para que o usuário sinta que está dialogando com um parceiro inteligente. Ainda no que diz respeito às respostas do sistema, é útil que o usuário consiga filtrá-las de alguma maneira, selecionando a quantidade e a qualidade das informações a serem visualizadas.

O fator "apresentação" é sem dúvida muito importante em uma interface. No entanto, questões estéticas não devem sobrecarregar a tela, ou prejudicar a ordenação dos elementos nela contidos. O excesso de cores e de formas gráficas pode ser um fator prejudicial, desviando a atenção do usuário. A decoração da interface deve ser, dessa forma, bastante criteriosa.

3. Adaptabilidade e auxílio ao usuário:

   Uma interface deve possuir a qualidade de ser tutorial, ou seja, em qualquer momento, o usuário pode perguntar ao sistema sobre o contexto em que ele se encontra, sobre opções disponíveis nesse contexto, enfim sobre o funcionamento do sistema em si. A resposta do sistema a essas perguntas deve ser clara, objetiva e sucinta, garantindo a ajuda desejada pelo usuário, sem sobrecarregá-lo com informações em excesso. Além disso, a interface deve se adaptar ao modo de trabalho do usuário. O sistema deve permitir que o usuário escolha as características que compõem seu ambiente de trabalho, tais como fontes de letras, cores, mecanismos de confirmação, entre outras.

4. Coerência e integridade:

   A previsibilidade e a coerência de uma interface se resumem no fato de ações idênticas produzirem resultados idênticos em qualquer contexto do sistema. Estas características tornam o aprendizado mais fácil, e dão ao usuário uma sensação de segurança, aumentando sua confiança no sistema. A intuitividade da interface também decorre de sua previsibilidade e coerência, de modo que o usuário possa ter a intuição da ação a ser tomada pela interface, mesmo que ele esteja utilizando um comando pela primeira vez. A manutenção de um estilo único ao longo de todos os contextos é fundamental para que o sistema apresente estas características.

   O termo *integridade* possui uma semântica diferenciada no contexto de interfaces gráficas, e se refere aos mecanismos de confirmação e às rotinas de tratamento de erros. Mecanismos de confirmação devem estar presentes, para garantir a integridade dos dados manipulados pela interface. No entanto, a utilização indiscriminada de mecanismos de confirmação pode tornar a operação do sistema cansativa, afetando o bom desempenho do usuário, pela repetição de ações idênticas. Outro modo de se garantir a integridade das informações é através de mecanismos que permitam desfazer (ou reverter) certas operações. Estes mecanismos, apesar de muito úteis, devem ser, no caso geral, limitados à última operação efetuada, sob pena de se tornarem muito onerosos para o sistema.

5. Completeza funcional e suporte à comunidade de usuários:

   Uma interface gráfica deve prover acesso a todas as funções disponíveis no SGBD. Esta completeza funcional pode ser atingida através de uma única ferramenta de

propósito geral. Entretanto, esta ferramenta tende a ser muito complexa, tanto para os projetistas que a desenvolvem, quanto para os usuários que com ela trabalham. Por essa razão, a tendência que se nota é a construção de ferramentas especializadas para cada grupo particular de usuários, e para cada conjunto de tarefas específicas em um BD. Estas ferramentas podem ser posteriormente integradas em um ambiente homogêneo, que atenda a toda a comunidade de usuários, sem penalizar usuários finais com excesso de funcionalidade, e ao mesmo tempo, garantindo aos usuários especialistas facilidades para explorar todo o potencial do SGBD.

6. Geração automática de apresentações:

   É forte a tendência das interfaces para SGBDs em adotar um estilo de interação baseado na edição de, e navegação sobre, objetos dos BDs [FM92]. Estes objetos podem representar informações sobre a estrutura dos BDs (esquemas) ou sobre os seus conteúdos. Nos modelos de dados tradicionais, a aplicação é a responsável pelo apresentação de objetos, porque a semântica de composição não está embutida nas estruturas dos modelos. Os novos modelos de dados (semânticos e orientados a objetos) permitem a modelagem direta de objetos estruturados, e a interface deve tirar proveito desta propriedade para criar as representações dos objetos, independentemente da aplicação. É claro que a aplicação pode especificar como e quando uma representação é criada, mas o modo de operação da representação é definido pelo sistema de interface, que passa a assumir uma tarefa que era, anteriormente, do programador de aplicações do BD.

7. Suporte a diferentes níveis de abstração nas apresentações:

   O suporte a diferentes níveis de abstração é uma característica complementar à geração automática de apresentações. A interface deve prover facilidades para que o usuário manipule as representações criadas, de modo a ajustá-las às suas necessidades. Devem existir mecanismos para deslocar, esconder e destacar as estruturas de um objeto complexo. De forma semelhante, deve ser possível associar representações de objetos, através, por exemplo, de hierarquias, ou de outro tipo de dependência, de modo que as próprias representações de objetos possam ser relacionadas para formar representações complexas, assim como um objeto complexo é composto pelo relacionamento de outros objetos mais simples. Enfim, o ponto essencial é que a interface permita que o usuário escolha o nível de detalhamento desejado para as informações apresentadas.

8. Acesso aos dados feito através do SGBD:

   A geração automática de representações propicia o isolamento entre o código da aplicação e o código destinado à gerência da interface. No entanto, para que a interface seja capaz de criar e gerenciar apresentações, é necessário que ela tenha acesso

aos objetos do BD. Alguns sistemas de interface fazem acesso direto aos objetos, pelo conhecimento embutido nestes sistemas sobre os mecanismos de armazenamento utilizados pelo SGBD. Esta abordagem apresenta dois grandes problemas. Primeiro, o sistema de interface se torna dependente da implementação do SGBD, não podendo ser utilizado com outro tipo de SGBD, mesmo que os seus modelos de dados sejam idênticos. O segundo problema é que o sistema de interface viola, neste caso, uma regra clássica dos SBDs, segundo a qual o acesso aos dados é feito exclusivamente através do SGBD. A violação desta regra causa a perda de todas as vantagens do controle logicamente centralizado dos dados.

Portanto, a interface deve fazer acesso aos dados através do próprio SGBD. Um módulo específico do sistema de interface deve ser o responsável pelo envio e recebimento de informações do SGBD. É óbvio que este módulo é dependente do SGBD utilizado, no que se refere à sintaxe de comandos e ao formato de respostas. Sem embargo, se a implementação destas particularidades for bem projetada no módulo de comunicação com o SGBD, todo o sistema de interface pode ser *portado* para outros SGBDs, através de modificações relativamente simples, feitas num único local do sistema.

9. Independência entre ações:

   Uma interface gráfica deve garantir que cada ação do usuário produza um resultado completo e perceptível. A cada ação deve corresponder uma resposta ou sinalização da interface, de modo que o usuário entenda que sua ação foi aceita e processada pelo sistema. Não obstante, o conceito de independência entre ações extrapola a simples retroalimentação, e deve ser entendido como uma oposição ao uso de *"modos de processamento"*. Nas interfaces que usam diferentes modos de processamento, o usuário seleciona um desses modos, e a partir dessa seleção, um grupo diferenciado de funções se torna ativo no sistema, tornando temporariamente inacessíveis as demais funções do sistema. Em geral, uma das ações permitidas em um "modo" qualquer é a volta ao modo normal de processamento.

   Este tipo de comportamento torna a interface bastante difícil de ser compreendida por um usuário inexperiente, pois cada "modo" apresenta um conjunto específico de ações, e mais, ações semelhantes em "modos" distintos nem sempre possuem semânticas similares. Em interfaces que possuem a característica de ações independentes, por outro lado, o conjunto de operações permitidas é o mesmo em qualquer contexto, e a semântica das ações mantém sua coerência ao longo de qualquer situação. A principal vantagem desta abordagem é que o usuário sempre vê um conjunto consistente de ações, e sabe que qualquer ação sua não levará o sistema a um estado que o usuário pode desconhecer.

10. Integração de BDs e representação do modelo de dados:

A nova geração de SBDs, que começa a surgir, deve possuir capacidade de integração com outros tipos de aplicação, como planilhas de cálculo e linguagens de programação. Mais ainda, os novos SBDs devem ser capazes de se comunicarem, cooperando entre si, para que os usuários de diferentes SBDs consigam manipular dados, independentemente do SBD que contém esses dados. Este tipo de integração é a proposta dos Sistema de Bancos de Dados Heterogêneos (SBDHs) [Oli93]. De modo semelhante, as interfaces para SGBDs devem possibilitar a visualização e a manipulação simultânea da estrutura e dos objetos de diferentes bancos de dados.

Uma interface para sistemas de bancos de dados deve ser construída de acordo com o modelo de dados específico do SBD, para que a interface consiga expressar, de modo natural, todas as informações nele contidas. É inconcebível que a interface restrinja, sob qualquer ponto de vista, o poder de expressão do modelo de dados definido para o SGBD. Mesmo em SBDHs, existe um modelo de dados comum, para o qual todos os modelos dos SBDs componentes são mapeados. Este modelo comum deve, dessa forma, ser suficientemente rico para representar todos os modelos subsidiários. O sistema de interface para SBDHs pode ser baseado no modelo de dados comum, ou ainda concordar com o modelo de dados de cada SBD componente. Em ambos os casos, a interface deve mapear as estruturas do modelo comum para estruturas correspondentes no modelo específico de cada SBD utilizado.

Mamou e Medeiros mostram, em [MM90] e [MM91], que o conceito de visão, tradicionalmente empregado na literatura sobre bancos de dados, pode ser estendido e empregado para a construção de sistemas capazes de gerar, automaticamente, uma representação contendo diversos objetos, pertencentes à classes distintas. O mesmo princípio pode ser aplicado para representar, em uma visão, informações provenientes de diferentes BDs.

3.6.2 Análise das Características em Algumas Interfaces

As tabelas 3.1 e 3.2 analisam as interfaces gráficas apresentadas no início deste capítulo, com relação às dez características básicas de interfaces introduzidas nesta seção. A convenção de símbolos adotada para elaboração das tabelas denota pelo sinal "$\surd$" uma característica presente na interface, e implementada de acordo com os comentários efetuados acima para a respectiva característica. O símbolo "$\approx$" indica a presença da característica no sistema de interface, todavia, sem o atendimento completo dos requisitos anteriormente citados. A ausência de uma característica é representada por uma coluna vazia.

Sistema de Interface	Características				
	Transparência/ Retroalimentação	Concisão/ Apresentação	Adaptabilidade/Auxílio	Coerência/ Integridade	Suporte à Comunidade
QBE	≈			≈	≈
SDMS	≈	√	≈	√	
PICASSO	√	√	√	√	
ISIS	≈	≈	≈	≈	≈
SNAP	≈	≈	√	√	
DATABROWSE/ SCHEMADESIGN	√	√	≈	√	
KIVIEW	≈	≈	≈	√	
PASTA-3	√	≈	≈	√	√
SIG	√	√	≈	√	
LOOKS/OOPE/ TOONMAKER	√	√	√	√	√
ODEVIEW	√	√	≈	√	
GS DESIGNER	≈	√	≈	√	
GOOD		≈		√	√
FACEKIT	√	√	≈	√	
Legenda	√ - Característica Plenamente Presente ≈ - Característica Parcialmente Presente □ - Característica Ausente				

Tabela 3.1: Características básicas em interfaces existentes (Parte 1)

Sistema de Interface	Características				
	Geração Automática de Apresentações	Níveis de Abstração	Acesso Através do SGBD	Independência entre ações	Integração/ Representação do Modelo
QBE	√	≈	√	√	
SDMS		≈	≈	√	
PICASSO	√		≈		≈
ISIS	√	≈	√		
SNAP		≈	√		
DATABROWSE/ SCHEMADESIGN	√	≈	√	√	≈
KIVIEW	√	√	√	√	≈
PASTA-3	≈	≈	√		≈
SIG	√	≈	√	√	
LOOKS/OOPE/ TOONMAKER	√	≈		√	≈
ODEVIEW	√	≈	√	√	
GS DESIGNER		√	√	√	
GOOD	≈	√	√	√	
FACEKIT		√	√	√	
Legenda	√ - Característica Plenamente Presente ≈ - Característica Parcialmente Presente □ - Característica Ausente				

Tabela 3.2: Características básicas em interfaces existentes (Parte 2)

Capítulo 4

GOODIES: Modelo Externo

4.1 Introdução

Este capítulo apresenta um novo sistema de interface para SGBDOOs. O sistema *GOODIES*[1] foi desenvolvido com base nas características essenciais de uma interface para SGBD, introduzidas na seção 3.6 do capítulo anterior. GOODIES combina e expande a funcionalidade de diversos sistemas de interface existentes, acrescentando uma série de novas funções para navegação em SGBDOOs.

Descrever-se-á a seguir o modelo externo do novo sistema, ou seja, a visão do BD que o sistema oferece ao usuário. A próxima seção apresenta uma discussão introdutória sobre o sistema desenvolvido. A seção 4.3 ilustra o modo de exibição das informações do BD no sistema. A seção seguinte mostra o mecanismo de interação entre o usuário e o sistema de interface. Na seção 4.5 é explicado o funcionamento das operações de navegação e consulta. A seção 4.6 apresenta algumas funções que tornam mais fácil e eficaz a utilização do sistema. Na última seção deste capítulo são apresentados comentários a respeito do modelo externo do sistema GOODIES.

4.2 Uma Nova Interface para SGBDOOs

Ao contrário dos sistemas relacionais, os SGBDOOs não são desenvolvidos com base em um modelo formal específico. Como foi discutido no capítulo 2, existe um conjunto de ca-

[1]GOODIES é um acrônimo para *Graphical Object Oriented Database Interface with Extended Synchronism* (Interface Gráfica para Bancos de Dados Orientados a Objetos com Sincronismo Estendido).

racterísticas e funcionalidades aceitas pelos pesquisadores como sendo fundamentais para que um BD seja considerado orientado a objetos. Assim, os SGBDOOs implementam características semelhantes, mas sem o compromisso de se adequarem a um determinado conjunto de regras rígidas. Analogamente, os sistemas de interface para SGBDOOs são construídos de maneira *ad hoc*, de acordo com a implementação escolhida em cada SGBDOO para os fundamentos do modelo OO.

O sistema GOODIES introduz uma nova abordagem para a construção de interfaces para SGBDOOS. Abandonando a idéia de acoplamento total entre a implementação do SGBDOO e o desenvolvimento da interface, o sistema foi construído com base nas características essenciais do modelo OO, identificadas na seção 2.2 do capítulo 2, e de modo independente de uma implementação específica destas características.

Duas vantagens dessa nova abordagem podem ser apresentadas, com relação à abordagem anterior. Primeiramente, ela permite a validação das características básicas que definem o modelo OO, e ao mesmo tempo, a verificação da adequação de um SGBD a estas características. A segunda vantagem é a facilidade de adaptação do sistema de interface a um SGBD que implemente, de alguma forma, estas características.

A especificação atual do sistema GOODIES só permite o acesso para consulta a dados e a esquemas. Dessa forma, o sistema não satisfaz os requisitos de uma interface completa para SGBDs (a seção 3.6 do capítulo 3 desta dissertação apresenta estes requisitos). No entanto, o projeto do sistema foi concebido com a preocupação de facilitar a sua extensão neste sentido. Acredita-se que a capacidade de atualização das informações dos BDs pode ser incorporada ao sistema, sem afetar a visão que o usuário possui do sistema atual (modelo externo), e com um número reduzido de modificações no modelo interno do sistema (o modelo interno descreve a interação entre a interface e o SGBD, e será discutido no próximo capítulo).

4.3 Visualização de Informações

Em GOODIES, todas as informações são apresentadas em janelas construídas segundo o padrão *OpenLook* [Sun90] para desenvolvimento de interfaces gráficas. Assim, as janelas são compostas de três partes: cabeçalho, corpo e rodapé.

No cabeçalho aparece o título da janela, ou seja, a identificação do tipo de informação apresentado na janela. O corpo da janela contém os controles e as representações de informações associadas à janela. Os rodapés das janelas se dividem em duas partes: direita e esquerda. Na parte direita aparece o nome ou identificação da informação representada na janela. Por exemplo, em uma janela que representa uma classe do esquema de um BD,

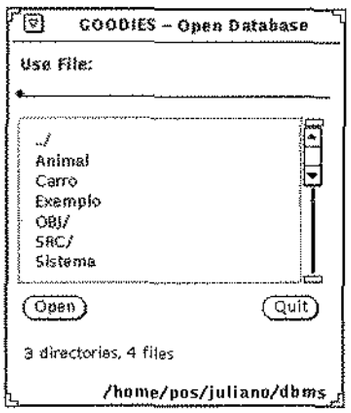


Figura 4.1: Janela de Diretório

o rodapé direito deve conter o nome do BD e da classe. A parte esquerda do rodapé é reservada para mensagens do sistema com relação aos dados apresentados ou operações realizadas sobre a janela (vide figura 4.3).

O sistema possui quatro tipos de janelas básicas, onde são representadas as informações sobre os esquemas e dados dos BDs, e um conjunto de janelas auxiliares, que garantem ao usuário o acesso às funções do sistema. GOODIES permite que o usuário trabalhe com um número arbitrário de janelas abertas simultaneamente.

4.3.1 Visualização de Esquemas

Três janelas básicas contêm informações sobre esquemas: a *janela de diretório*, que permite a navegação pelos BDs existentes; a *janela de BD*, onde é apresentado o rol de classes que compõem o esquema de um determinado BD; e a *janela de classe*, onde são apresentados os itens que definem uma classe do BD.

A janela de diretório provê acesso aos BDs existentes. Esta janela permite a navegação entre diretórios, para a escolha dos BDs desejados. O usuário pode visualizar diferentes BDs ao mesmo tempo, pois cada escolha de BD na janela de diretório causa o aparecimento de uma janela de BD correspondente ao BD selecionado. Os BDs existentes em um diretório são visualizados em uma lista existente na janela de diretório. Esta lista contém ainda os subdiretórios do diretório que está sendo visualizado e uma opção, sempre colocada no topo da lista, para subir um nível na hierarquia de diretórios. A figura 4.1 mostra uma janela de diretório.

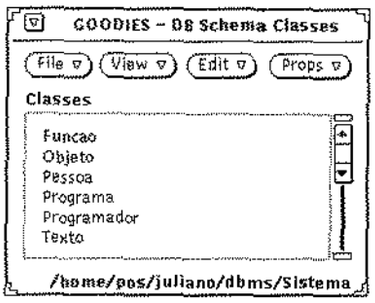


Figura 4.2: Janela de BD

Além dos quatro BDs (Animal, Carro, Exemplo e Sistema), o diretório apresentado possui três diretórios associados: o diretório pai na árvore de diretórios (representado por ../), e dois subdiretórios (SRC e OBJ). Na figura 4.2 aparece a janela de BD contendo as classes do BD *Sistema*.

A terceira janela básica para visualização de esquemas é a janela de classe. Esta janela apresenta a definição de uma classe do esquema de um BD, e é composta pelos seguintes itens:

1. **Type**: uma descrição textual do tipo da classe, isto é, da composição dos objetos pertencentes à classe;
2. **Superclasses**: uma lista de superclasses das quais a classe descrita herda atributos e métodos;
3. **Subclasses**: uma lista de subclasses que herdam os atributos e métodos da classe definida;
4. **Methods**: uma lista de métodos associados à classe descrita;
5. **Objects**: uma lista de instâncias de objetos pertencentes à classe especificada, ou seja, a extensão da classe.

A figura 4.3 apresenta uma janela de classe descrevendo a classe *programa* do BD representado na figura 4.2. Os controles deslizantes localizados à esquerda dos itens da janela de classe permitem o redimensionamento da representação de um item, com relação aos demais. Em outras palavras, o usuário pode, através desses controles, aumentar (ou diminuir) o tamanho de um item, sem afetar o tamanho da janela em si. O sistema diminui (ou aumenta) automaticamente os demais itens, de modo que todos os itens continuem

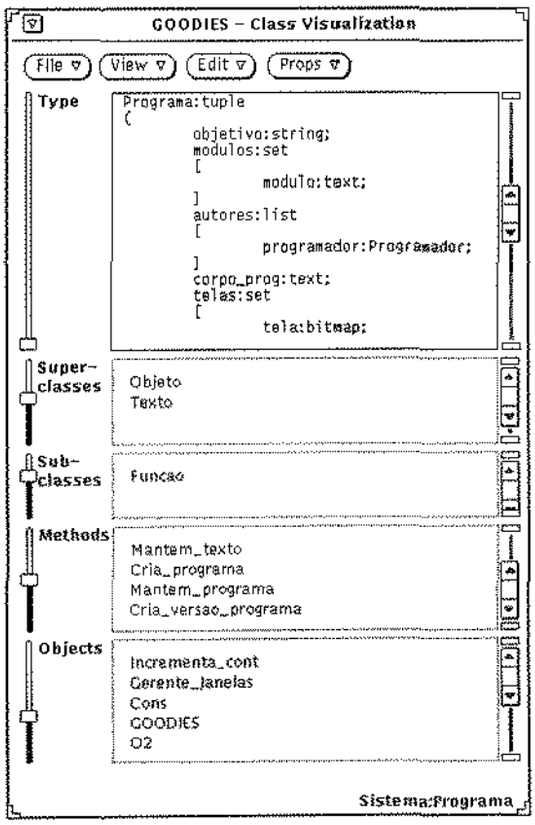


Figura 4.3: Janela de Classe

sendo apresentados no espaço disponível no corpo da janela. Este mecanismo é útil para destacar os itens mais importantes, ou para permitir a visualização de mais elementos de determinados itens da janela de classe.

4.3.2 Visualização de Dados

As três janelas básicas descritas anteriormente (janela de diretório, janela de BD e janela de classe) servem para visualização e navegação sobre os esquemas dos diversos BDs controlados por um SGBDOO. A quarta janela básica permite a visualização e a navegação sobre os dados propriamente ditos, isto é, sobre os objetos dos BDs.

A *janela de objeto* contém os valores dos atributos que compõem a instância do objeto, de acordo com a descrição da composição da classe apresentada na janela de classe. A

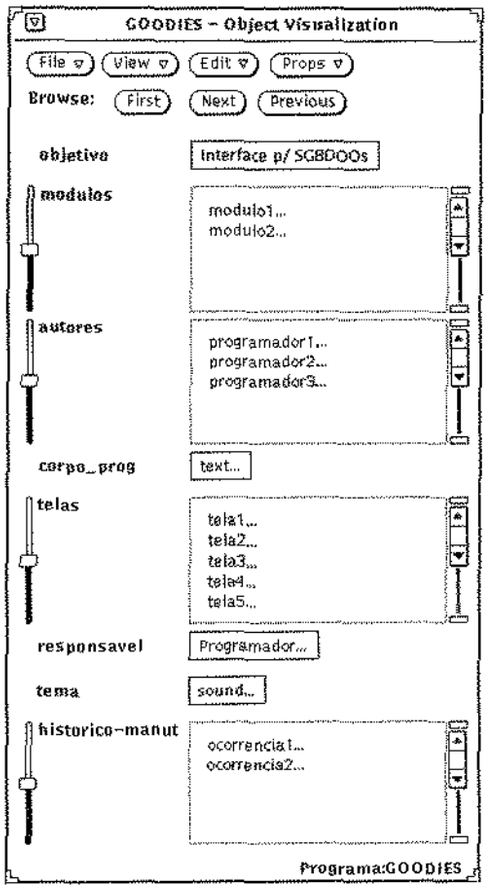


Figura 4.4: Janela de Objeto

figura 4.4 representa um objeto da classe *programa*, definida na figura 4.3.

Os atributos de objetos são divididos, conforme sua representação no sistema, nos seguintes grupos:

- Atributos simples: são os que podem ser representados por cadeia de caracteres de comprimento máximo 128, e que são elementares, isto é, não são compostos de outros elementos. Os números (reais, inteiros), os valores booleanos e as *cadeias de caracteres*[2] com menos de 128 caracteres são exemplos de atributos simples. Estes atributos são diretamente representados na janela de objeto. O atributo *objetivo* da figura 4.4 é um atributo simples.

[2]cadeias de caracteres não são consideradas como compostas por elementos do tipo caractere, pois neste caso os caracteres individuais não possuem semântica própria.

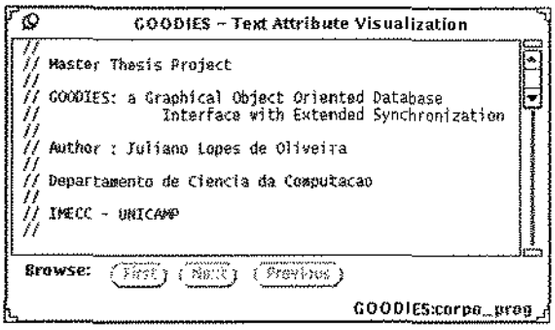


Figura 4.5: Janela de Texto

- Atributos textuais: são considerados atributos textuais os atributos elementares, como definido acima, que possuem mais de 128 caracteres. Estes atributos são representados em janelas de texto auxiliares associadas à janela de objeto que contém o atributo textual. A figura 4.5 mostra a representação do atributo textual *corpo_prog*, do objeto apresentado na figura 4.4.

- Imagens: uma imagem é uma seqüência de *bytes* que definem a representação gráfica de uma figura. Imagens, como textos, são apresentados em janelas gráficas auxiliares, associadas à janela de objeto que contém o atributo do tipo imagem. Na figura 4.6 aparece a janela gráfica para o atributo *foto* de um objeto da classe *Programador*, que aparece no esquema do banco de dados *Sistema* (figura 1.1).

- Sons: atributos do tipo som se aplicam a gravações sonoras, cuja representação é feita pela reprodução do som armazenado no atributo. Os tipos som e imagem provêm capacidade de armazenamento e manipulação de objetos *multimeios*, suportados por um grande número de sistemas orientados a objetos. O atributo *tema* do objeto apresentado na figura 4.4 é do tipo som.

- Listas: os atributos do tipo *lista* são os que representam coleções de elementos, onde todos os elementos pertencem ao mesmo tipo. Os elementos de uma lista podem ser atributos simples, e neste caso, são representados diretamente na janela de objeto, como itens da lista. Se os elementos da lista não forem atributos simples, os itens da lista apresentada na janela de objeto referenciam os objetos complexos correspondentes. Estes podem ser, por sua vez, visualizados em novas janelas, até atingir valores atômicos. A figura 4.7 exibe o quinto elemento do atributo *telas*, que é definido como um conjunto de atributos de tipo atômico *bitmap* (figura 4.3). *Bitmaps*, por sua vez são representados em GOODIES através de atributos do tipo imagem.

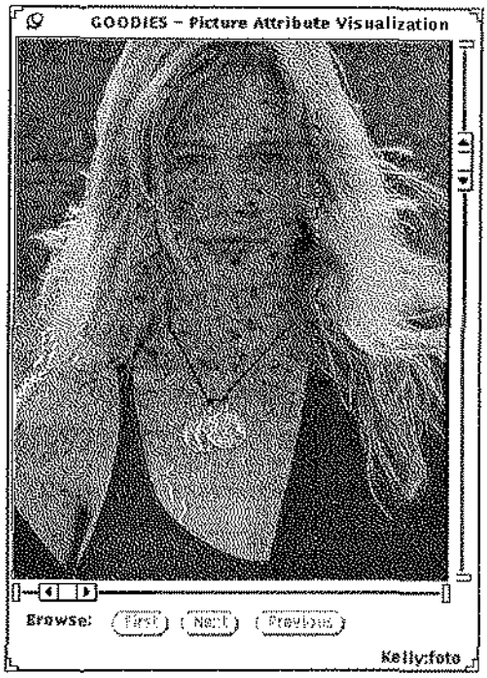


Figura 4.6: Janela de Imagem (1)

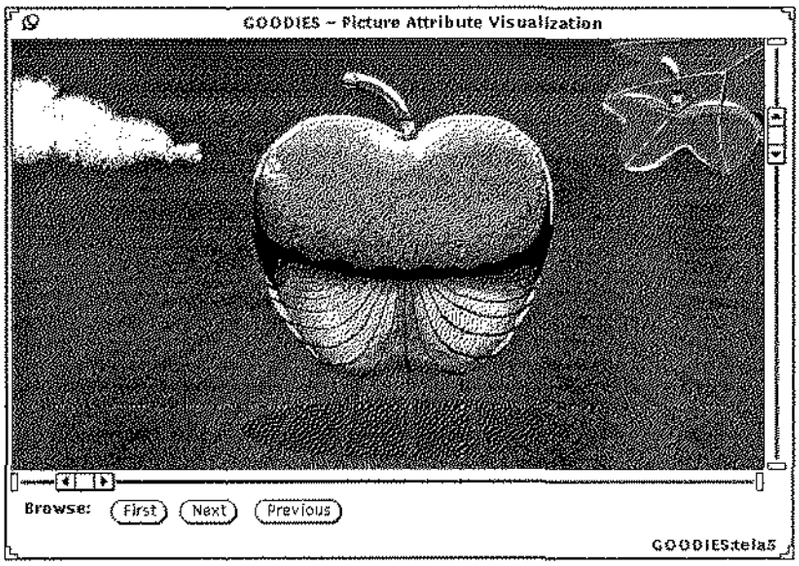


Figura 4.7: Janela de Imagem (2)

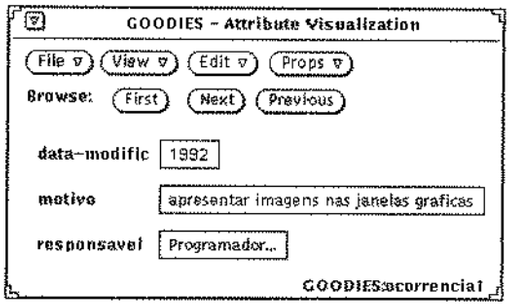


Figura 4.8: Janela de Tupla

- Tuplas: atributos do tipo tupla representam agrupamentos de elementos de tipos heterogêneos. As tuplas exigem a criação de uma janela auxiliar para a sua representação, já que seus elementos podem pertencer a qualquer dos tipos definidos. A figura 4.8 mostra a representação do atributo *historico-manut*, do tipo tupla, definido no objeto da figura 4.4.

- Sub-objetos: são atributos usados para representar o conceito de *objeto complexo*, segundo o qual um objeto pode ser formado por um conjunto arbitrário de outros objetos. Os sub-objetos são apresentados em janelas de objeto auxiliares, associadas à janela de objeto básica. A construção e apresentação das janelas auxiliares é idêntica à das janelas básicas. A única distinção é que, por *default*, a janela de objeto auxiliar é sincronizada com a janela de objeto básica. O conceito de sincronismo entre janelas de objetos será detalhado posteriormente, ainda neste capítulo.

As janelas auxiliares associadas à janela de objeto seguem o mesmo padrão de representação de atributos desta janela. Desse modo, é possível representar um número arbitrário de objetos e valores aninhados, satisfazendo-se assim as recomendações existentes para a construção de objetos no modelo OO (conforme discutido na seção 2.2 do capítulo 2). Os atributos que devem ser representados em janelas distintas podem ser facilmente identificados, pois aparecem seguidos por reticências ("...").

4.4 Interação com o usuário

O paradigma de manipulação direta [Shn87] foi adotado como principal mecanismo de interação com o usuário. Como foi visto na introdução desta dissertação, este mecanismo reduz e simplifica as ações requeridas para que o usuário expresse a ação desejada do sistema, diminuindo a ocorrência de erros e o esforço exigido do usuário.

Existem dois modos básicos para o acionamento de funções do sistema. O primeiro modo é o tradicionalmente utilizado em interfaces gráficas: o usuário seleciona as informações e, em seguida, indica a ação a ser realizada sobre estas informações, através de botões de comando situados no interior da janela que contém as informações selecionadas.

A seleção de uma informação é feita pelo posicionamento do *mouse* sobre a informação desejada, seguido do *click* com o botão de seleção do mouse. O termo *click* é empregado neste texto indicando a ação de pressionar um botão e soltá-lo imediatamente.

O segundo modo de ativação serve como um caminho abreviado para determinadas operações, notadamente as operações de navegação. Sempre que o usuário desejar selecionar uma informação e em seguida aplicar uma operação visando criar ou abrir uma janela referente à informação selecionada, o segundo modo de interação pode ser utilizado. Ao invés de selecionar a operação em um *menu* associado a algum botão da janela, o usuário precisa apenas realizar a operação de seleção (através do click com o botão de seleção do mouse) duas vezes seguidas. Esta operação, que será chamada de *click* duplo, indica que o usuário deseja abrir uma janela para visualização de informações relacionadas à informação selecionada.

Como exemplo dos mecanismos de interação com o usuário, a janela auxiliar apresentada na figura 4.5 poderia ter sido exibida de duas maneiras. O usuário poderia ter selecionado o valor do atributo *corpo prog* da janela apresentada na figura 4.4 e, em seguida, ter acionado o botão *View* desta janela, que apresentaria um menu cuja opção *Open* causaria a criação e apresentação da janela da figura 4.5. Da mesma forma, o click duplo sobre o valor do atributo *corpo prog* na janela da figura 4.4 produziria efeito análogo. A partir deste ponto, o termo *seleção* será empregado com o sentido de especificar a ação completa de escolha de informação e aplicação de operação sobre a informação, seja através de menus ou de click duplo, já que ambos os mecanismos produzem efeitos idênticos.

Através dos botões de comando, o usuário tem acesso a toda a funcionalidade da interface. Já o segundo modo de interação permite, basicamente, a ativação de funções de navegação sobre esquemas e dados dos BDs. É importante ressaltar que, em qualquer janela do sistema, ambos os mecanismos de interação apresentam resultados análogos para os mesmos tipos de operações. A coerência entre ações e resultados obtidos foi uma preocupação constante no projeto do sistema, e garante uma rápido compreensão do funcionamento da interface por parte do usuário final.

Outra preocupação do projeto foi garantir ao usuário um sistema flexível. Assim, GOODIES permite ao usuário organizar livremente seu *workspace*, através do redimensionamento, reposicionamento, abertura, fechamento, criação e destruição de janelas. O sistema não limita o número de janelas abertas (de fato este número é limitado pelo

Window Manager[3] e pela quantidade de memória disponível no equipamento), nem faz qualquer restrição quanto ao tamanho ou posicionamento das janelas.

4.5 Mecanismo de Navegação e Consulta

Já foram apresentadas as janelas disponíveis no sistema GOODIES. As próximas seções demonstram as maneiras existentes para a visualização dos esquemas e dados contidos em um SGBDOO, com o auxílio destas janelas.

4.5.1 Navegação sobre Esquemas

Uma sessão de trabalho em GOODIES é iniciada com a janela de diretório para a escolha do BD com o qual se deseja interagir. A seleção de um BD faz surgir a janela de BD, contendo uma lista de classes que compõem o esquema do BD. Para efetuar a escolha do BD, o usuário pode visualizar o conteúdo dos diretórios existentes no sistema de arquivos. A seleção de um diretório na lista de arquivos da janela de diretório causa a mudança do conteúdo da lista, que passa a conter os BDs e subdiretórios do diretório escolhido.

O usuário pode escolher um subdiretório, navegando para baixo na hierarquia de diretórios; pode subir um nível nesta hierarquia, através da primeira opção da lista de arquivos (../); pode ainda selecionar o BD desejado na lista de arquivos. Se o usuário conhece o caminho do diretório raiz até o diretório ou BD desejado, ele pode digitar o nome completo do diretório ou BD no campo de texto da janela de diretório, eliminando o processo de navegação pelos diretórios intermediários. Na seção 4.6 será demonstrado um mecanismo pelo qual o usuário pode estabelecer os BDs desejados, de modo que o sistema busque automaticamente estes BDs a cada início de seção de trabalho, tornando dispensável o emprego da janela de diretório.

Obtida a janela de BD, o usuário pode selecionar as classes do esquema a partir da lista de classes daquela janela. Selecionando as classes, o usuário obtém as respectivas janelas de classe, contendo a descrição de cada classe do esquema. A seção 4.3.1 apresenta e explica o conteúdo completo da janela de classe.

De modo semelhante, a partir da janela de classe, o usuário pode continuar a navegação sobre o esquema do BD, selecionando classes nas listas de superclasses e de subclasses, ou selecionando métodos na lista de métodos da classe. É possível ainda o início da navegação

[3] *Window Manager* representa o sistema gerenciador de janelas, em um ambiente que permite múltiplas janelas.

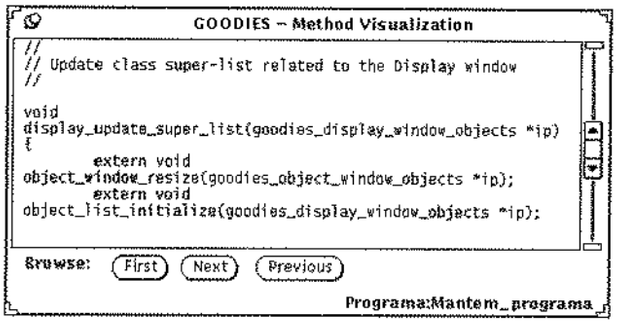


Figura 4.9: Janela de Método

sobre os dados da classe, pela seleção de instâncias na lista de objetos (extensão) da classe.

A seleção de superclasses ou de subclasses na janela de classe representa uma operação análoga à seleção de uma classe na janela de BD. Estas operações causam a criação e apresentação da janela de classe para a classe selecionada.

Já a seleção de um método na janela de classe dispara o processo de criação e apresentação de uma janela auxiliar associada àquela janela, que é a janela de descrição de método. Esta janela contém a descrição textual (corpo) do método selecionado, e cada seleção de método causa a criação de uma nova janela. A figura 4.9 ilustra o modo de visualização de um método da classe apresentada na figura 4.3.

4.5.2 Navegação sobre Dados

A navegação sobre os dados tem início com a seleção de um objeto na lista de objetos da janela de classe. Esta operação causa o surgimento da janela de objeto para o objeto selecionado e, a cada nova seleção de objeto na janela de classe, uma nova janela de objeto é criada. Dessa maneira, o usuário pode trabalhar com várias instâncias de objetos de uma mesma classe ao mesmo tempo.

Por outro lado, o usuário pode utilizar uma única janela de objeto para ter acesso aos dados de uma instância de objeto a cada vez, através das *operações de seqüenciamento*. Estas operações são ativadas pelos botões *next*, *previous* e *first* da janela de objeto. O botão *next* faz com que o conteúdo apresentado na janela de objeto seja obtido do próximo objeto da extensão da classe a que pertence o objeto. Por exemplo, se o objeto visualizado na janela de objeto corresponde ao primeiro elemento da lista de objetos da janela de classe, a ativação do botão *next* mostrará o segundo elemento da extensão da

classe.

O botão *previous* é análogo ao botão *next*, exceto que, ao invés de usar o próximo elemento da lista de objetos, toma-se o elemento anterior desta lista. O botão *first* causa a apresentação do primeiro elemento da lista de objetos da classe na janela de objeto, independente da posição do objeto que está sendo visualizado na lista de objetos.

Convém ressaltar que as operações *next* e *previous* consideram a lista de objetos como circular, de modo que a ativação de *next* no último elemento da lista de objetos causa a apresentação do primeiro elemento, e a ativação de *previous* neste elemento exibe o último elemento da lista de objetos na janela de objeto.

4.5.3 Facilidades de Consulta

Nas seções anteriores foram apresentados os mecanismos básicos de navegação em GOODIES, mecanismos estes que também estão presentes em grande parte das interfaces existentes para sistemas de bancos de dados. Nesta seção serão discutidas as capacidades adicionais que tornam o mecanismo de navegação de GOODIES mais poderoso, podendo ser considerado como um processo simplificado de consulta ao BD.

É importante distinguir neste ponto a terminologia adotada; *navegação* é o processo de visualização seqüencial de informações de um determinado tipo; *consulta* é o processo de seleção e restrição das informações, de modo que apenas as informações explicitamente solicitadas sejam recuperadas e apresentadas ao usuário.

Predicados

A primeira facilidade de consulta disponível em GOODIES é a especificação de *predicados*. O botão de propriedades da janela de objeto exibe um *menu* que contém a opção *predicate...*. Esta opção cria uma janela auxiliar associada à janela de objeto, a *janela de predicados*, onde podem ser definidos predicados a serem aplicados sobre o objeto apresentado na janela de objeto. Um predicado é formado por três elementos:

Atributo: Um atributo do objeto representado na janela de objeto para a qual foi solicitada a criação da janela de predicados;

Operador: Um operador de comparação ($=, <, >, \leq, \geq, \neq$) ou de conjunto ($\supset, \subset$);

Referencial: Um valor ou atributo de objeto, cujo tipo seja igual ao do atributo correspondente ao primeiro elemento do predicado.

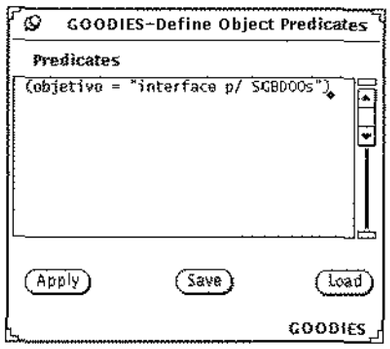


Figura 4.10: Janela de Predicado

Um predicado pode, ainda, ser composto pela associação de predicados, através de conectores lógicos (*And, Or*) e do uso de parênteses para especificar a ordem em que os predicados serão avaliados. A figura 4.10 mostra um predicado definido para o objeto apresentado na figura 4.4.

Uma vez definido um predicado para uma janela de objeto, as operações de seqüenciamento têm sua semântica alterada. A ativação de *next* não mais buscará o próximo elemento da lista de objetos da classe, mas sim o próximo elemento desta lista que satisfaça o predicado especificado. O mesmo acontece com a operação *previous*, que faz a busca na lista em ordem reversa, e com a operação *first*, que traz o primeiro elemento, a partir do início da lista de objetos, que satisfaz o predicado definido.

Sincronismo

Outra facilidade de consulta em GOODIES é a sincronização de janelas de objeto. A opção *Synchronism...*, contida no *menu* de propriedades da janela de objeto, cria e apresenta uma janela auxiliar associada à janela de objeto: a *janela de sincronismo*. Esta janela permite a criação e modificação de uma árvore de janelas de objetos denominada *árvore de sincronismo*. O mecanismo de sincronização garante que qualquer operação de seqüenciamento aplicada sobre uma janela de objeto seja refletida na sub-árvore que tem como raiz a janela de objeto onde ocorreu a operação.

Uma ligação de sincronismo estabelece uma hierarquia entre duas representações de objetos, mas não é permitida entre dois objetos quaisquer. Essas ligações devem ser feitas de acordo com a composição dos objetos, determinadas pelos tipos das classes. Uma janela de objeto só pode ser *pai* de outra janela na árvore de sincronismo se o objeto representado naquela janela contém algum atributo que referencia este objeto.

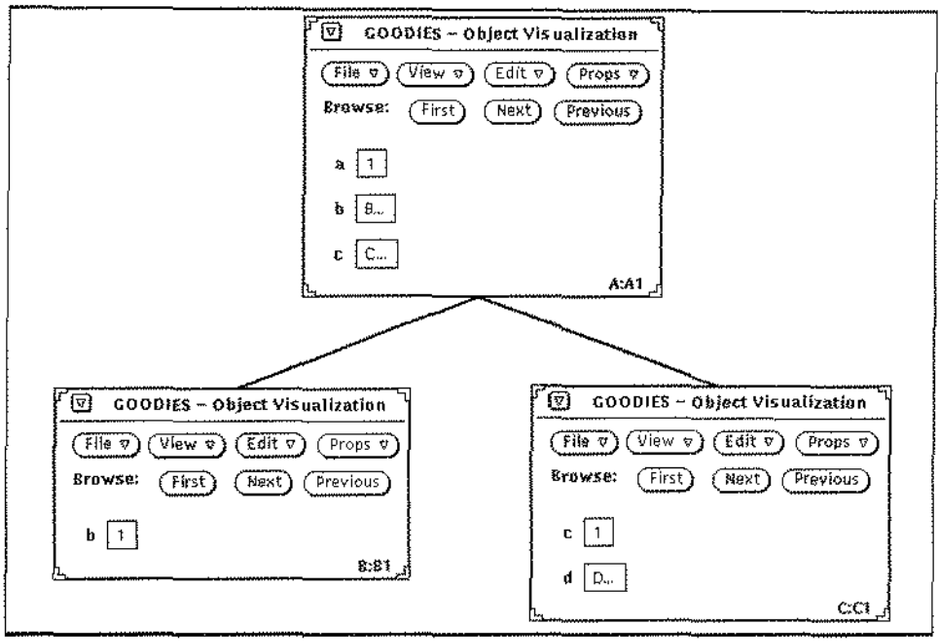


Figura 4.11: Árvore de Sincronismo

Uma árvore de sincronismo permite a visualização simultânea, em janelas distintas, do conteúdo de um objeto complexo e de seus componentes. A árvore de sincronismo pode ter altura arbitrária, dependendo do nível de aninhamento de um objeto complexo e das operações especificadas pelo usuário. O exemplo apresentado a seguir torna mais claro o mecanismo de sincronização.

Seja uma classe **A** de componentes **B** e **C**. Suponha que o objeto **A1** da classe **A** esteja disponível em uma janela de objeto. Esta janela mostra que **A1** tem subobjetos de nomes **B1** (do tipo **B**) e **C1** (do tipo **C**). Se o usuário seleciona **B1**, será criada uma nova janela de objeto (agora para a classe **B**), na qual aparecerá o conteúdo de **B1**. Analogamente, a seleção de **C1** cria uma janela para este objeto da classe **C**. Com isto, está criada uma árvore de sincronismo de raiz **A1** e folhas **B1** e **C1** (figura 4.11).

Seja **A2** o próximo objeto de **A**, obtido através da operação *next* aplicada na janela de **A1**. Como, neste caso, existe uma árvore de sincronismo, ao mesmo tempo aparecerão nas outras duas janelas os componentes **B** e **C** de **A2**, de maneira automática, ou seja, sem que o usuário precise especificar qualquer operação adicional (figura 4.12).

Dessa maneira, uma única operação de seqüenciamento pode afetar e atualizar os objetos visualizados em diversas janelas, pela aplicação dos mecanismos de sincronização. É relevante observar que os predicados definidos nas janelas de cada objeto continuam sendo aplicados quando a janela de objeto está sincronizada com outras janelas. A asso-

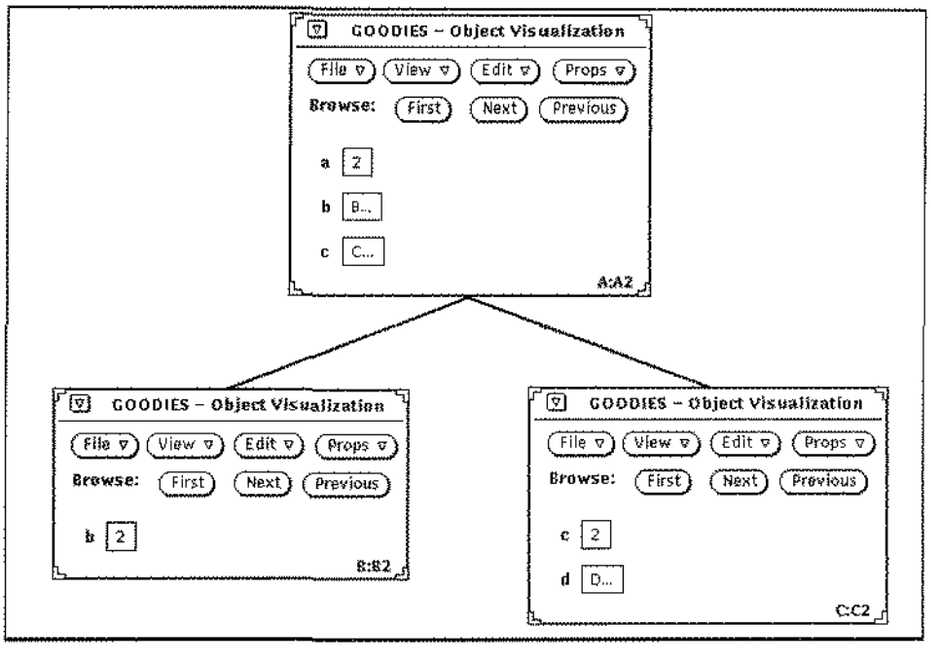


Figura 4.12: Árvore de Sincronismo após operação *next*

ciação de predicados com o mecanismo de sincronização torna o processo de navegação em GOODIES semelhante a um processo de consulta, onde o usuário seleciona e restringe as informações desejadas. Somente as ferramentas de consulta gráfica possuem tais facilidades, que não se encontram reunidas em nenhum dos sistemas de navegação apresentados no capítulo 3.

4.6 Outras Facilidades

4.6.1 Salvamento de Contexto

Além das facilidades de navegação e consulta anteriormente citadas, o usuário de GOODIES conta com uma série de operações que visam facilitar ainda mais o seu trabalho, e lhe permitem ajustar o seu ambiente de acordo com o seu gosto pessoal, ou com a necessidade da tarefa que ele irá realizar.

A opção *Save Workspace* associada ao botão *File* da janela de BD é uma dessas operações. Ao ser acionada, o sistema salva o contexto corrente em que o usuário está trabalhando. A partir deste ponto, sempre que o usuário ative GOODIES para uma seção de trabalho, o sistema irá automaticamente apresentar o contexto que o usuário estava

visualizando, no momento em que ativou a operação *Save Workspace*.

O termo *contexto* é aqui empregado para representar as janelas básicas (janela de diretório, janela de BD e janela de classe), excluindo-se as janelas de objeto, já que objetos são dinamicamente inseridos e removidos dos BDs, ao passo que os esquemas são modificados em escala muito menor.

4.6.2 Nível de Visualização

De acordo com o conceito de *herança*, a definição de uma classe herda métodos e atributos de suas superclasses. Além disso, a hierarquia de herança pode ter um nível arbitrário de profundidade. No caso de herança múltipla ser permitida, uma classe herda atributos e métodos de todas as suas superclasses. Desse modo, a definição do tipo de uma classe pode ter um único atributo ou método próprio, herdando um número ilimitado de atributos e métodos. É possível que, ao visualizar uma classe, o usuário esteja interessado apenas nos atributos e métodos próprios da classe, e não naqueles herdados.

GOODIES oferece facilidades para definição do nível de visualização da hierarquia de classes. O usuário pode selecionar o nível de visualização desejado, através das seguintes opções:

Display Superclasses : através de uma janela auxiliar que contém a lista de superclasses de uma determinada classe, o usuário seleciona as superclasses cujas propriedades (atributos e métodos) devem ser representadas. Esta seleção afeta os campos *Type*, *Superclasses* e a *Methods* da janela de classe, bem como os atributos visualizados nas janelas de objetos pertencentes à classe.

Display Subclasses : analogamente, o usuário pode selecionar as subclasses que deseja visualizar a partir de uma determinada classe. As subclasses selecionadas são eliminadas da lista de subclasses da janela de classe, assim como as instâncias daquelas subclasses são eliminadas da lista de objetos desta janela.

4.6.3 Seleção de Atributos

Em um banco de dados orientado a objetos real, a definição de uma classe pode ter um número muito grande de atributos explicitamente definidos. Dessa forma, a escolha do nível de visualização não é suficiente para que o usuário defina os atributos que ele deseja visualizar. GOODIES permite, através de uma janela auxiliar associada à janela de objeto, escolher os atributos desejados. A *janela de atributos* contém a lista de atributos

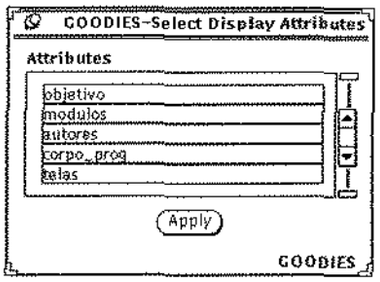


Figura 4.13: Janela de Atributos

do objeto, de acordo com o nível de visualização corrente, onde o usuário seleciona os atributos que devem aparecer na janela de objeto.

De modo semelhante, o usuário pode definir os itens que serão exibidos na janela de classe. Como foi visto na seção 4.3.1, os itens que compõem a janela de classe são cinco: a definição textual da classe (*Type*) e quatro listas (*Superclasses*, *Subclasses*, *Methods* e *Objects*). Do ponto de vista do usuário, o processo de seleção de atributos é idêntico para ambas as janelas. A figura 4.13 mostra a janela de atributos para a janela de objeto apresentada na figura 4.4.

4.7 Comentários sobre o Modelo Externo

Este capítulo apresentou a funcionalidade do sistema GOODIES, e descreveu o mecanismo básico de interação entre o sistema e seus usuários. Como foi visto, muitas funções da interface foram adaptadas de operações disponíveis em diversos sistemas existentes. Esta seção mostra a fonte original destas operações.

O estilo de construção das janelas segue as recomendações do padrão OPENLOOK [Sun90], mas o fundamento de se dividir a apresentação de um objeto complexo em várias janelas teve como ponto de partida a proposta de McDonald, Stuetzle e Buja [MSB90]. De acordo com estes autores, a tendência natural de se apresentar informações complexas em uma única figura não é sempre possível, além de ser freqüentemente ineficiente. De modo geral, é melhor apresentar uma informação complexa (por exemplo instâncias de objetos em um BDOO) em um número de representações separadas, mais simples, enfocando aspectos particulares da informação global.

O mecanismo de navegação básico de GOODIES se inspira no sistema descrito em

[RC88], uma interface para bancos de dados que seguem o modelo de Entidades e Relacionamentos. O conceito de sincronismo usado no processo de navegação foi, por sua vez, fortemente influenciado pelo modelo do sistema KIVIEW, descrito em [MDT89].

A idéia de se aplicar predicados para tornar o processo de navegação mais poderoso foi tomada dos sistemas de consulta gráfica, notadamente do sistema PICASSO [KKS88]. A definição de predicado utilizada em GOODIES é muito semelhante àquela do sistema PICASSO, apesar deste sistema ser dirigido ao modelo de dados de relação universal.

Por fim, o conceito de seleção de atributos desejados na apresentação de objetos se assemelha aos conceitos apresentados no sistemas LOOKS, descrito por [Mam91].

Cabe observar, no entanto, que nenhum dos sistemas dos quais GOODIES *herdou* características é independente da implementação dos sistemas para os quais eles servem de interface. Essa importante característica, a independência de um SGBDOO específico, será discutida no próximo capítulo.

Capítulo 5

GOODIES: Modelo Interno

5.1 Introdução

O sistema GOODIES é uma camada de software desenvolvida com o objetivo de envolver um SGBDOO, facilitando o acesso dos usuários aos dados contidos nos BDs. Desse modo, a funcionalidade do sistema GOODIES é determinada pelo seu relacionamento com os usuários e com o SGBD. Esta funcionalidade pode ser compreendida, de modo bastante geral, como um processo dividido em duas etapas fundamentais:

1. A tradução de operações de manipulação direta realizadas pelo usuário em consultas que podem ser processadas pelo SGBDOO;

2. A conversão dos resultados das consultas em informações audio-visuais, cuja semântica seja compreensível para o usuário.

No capítulo anterior foi apresentado o modelo externo do projeto do sistema GOODIES. O modelo externo reflete as etapas citadas acima sob o ponto de vista do usuário do sistema. Este capítulo analisa o modelo interno do sistema, que trata dos aspectos relacionados ao SGBDOO e de sua influência na funcionalidade do sistema GOODIES.

É importante observar que o próprio projeto de GOODIES reflete o seu papel de intermediário entre usuário e SGBD. Os modelos externo e interno, que descrevem a interação de GOODIES com o usuário e o SGBD, respectivamente, são complementares entre si. Desta forma, a combinação dos modelos interno e externo resulta em um modelo completo de interação entre um usuário e um sistema automatizado.

A análise do modelo interno do sistema GOODIES está dividida da seguinte forma: a seção 5.2 discute a dependência do sistema de interface com relação ao SGBD utilizado; na seção 5.3 é apresentada a formalização adotada para representar as características do modelo de dados orientado a objetos; a seção 5.4 estuda a interação entre GOODIES e o SGBD, que é feita com base em operações primitivas definidas pelo modelo interno de GOODIES; a seção seguinte analisa esta mesma interação, considerando, porém, fatores relacionados a transações, e não apenas a operações isoladas; a última seção deste capítulo destaca e resume as principais características do modelo interno do sistema GOODIES.

5.2 Acoplamento entre Interface e SGBD

A diretiva número oito apresentada na seção 3.6 para a construção de interfaces para BDs aborda o acoplamento entre sistema de interface e SGBD. Aquela diretiva mostra que o acesso às informações de um BD deve ser feito unicamente através do SGBD, e que, por isso, o sistema de interface é dependente do SGBD utilizado, embora esta dependência possa ser isolada em um módulo do sistema de interface, de forma que o sistema se adapte com facilidade a diversos SGBDs.

A maior parte dos sistemas de interface discutidos no capítulo 3 apresenta uma forte dependência do SGBD utilizado, e não respeita a recomendação de isolamento da parte do código da interface que é dependente do SGBD. Pelo contrário, aqueles sistemas utilizam-se do conhecimento de detalhes de implementação do SGBD para obter maiores facilidades na execução dos serviços de interface a que se propõem.

O projeto de GOODIES, por outro lado, apresenta um grau mínimo de dependência em relação ao SGBD utilizado. Ainda assim, todos os acessos aos dados dos BDs necessários para o funcionamento de GOODIES são feitos através do SGBD. Um único módulo do sistema, o módulo de operações primitivas, contém todo o código que é dependente do SGBD, organizado de maneira que a substituição de um SGBD por outro SGBD que segue o mesmo modelo de dados seja feita de modo simples e direto. No caso geral, as únicas ações requeridas são a mudança no nome dos comandos de acordo com a sintaxe aceita pelo SGBD e a modificação do mecanismo interpretador de respostas de GOODIES, conforme o formato de resultados provido pelo SGBD.

O acoplamento entre GOODIES e o SGBDOO associado é efetuado através de *operações primitivas*, cuja semântica é definida pelo modelo interno de GOODIES, e que devem ser convertidas em comandos do SGBDOO utilizado. As operações primitivas serão estudadas ainda neste capítulo, na seção 5.4. O processo de associação de um SGBD ao sistema GOODIES, isto é, a conversão de operações primitivas em comandos do SGBD, será discutido no capítulo 6 desta dissertação.

A principal diferença de GOODIES para a maior parte dos sistemas de interface existentes para SGBDOOs está no fato de GOODIES se basear nas características essenciais do modelo OO, de acordo com a proposta apresentada no capítulo 2, e não em uma implementação específica daquelas características [OA92]. A próxima seção mostra a maneira pela qual estas características foram representadas no modelo interno do sistema GOODIES.

5.3 Representação do Modelo OO

Um SGBDOO pode controlar diversos BDs, e GOODIES define um BD através de um esquema e de um conjunto de dados. O conjunto de dados representa os objetos do BD, enquanto o esquema é representado pelos grafos de herança e de composição, e por um conjunto de métodos.

5.3.1 Grafo de Herança

O *grafo de herança* é um grafo orientado e acíclico, cujos nós representam todas as classes do esquema, e são rotulados com o nome de cada classe. Não são permitidos nós com rótulos idênticos, isto é, cada classe do esquema de um BD é univocamente identificada pelo seu nome. Cada nó contém, além do rótulo que identifica a classe, uma lista que define os métodos associados àquela classe.

As arestas do grafo de herança são de dois tipos: arestas de *especialização* e arestas de *generalização*. Os rótulos das arestas contém seu tipo.

As arestas de generalização são direcionadas das subclasses para as superclasses, e representam o fato de a superclasse, destino da aresta, ser uma *generalização* da subclasse, em que a aresta se origina.

As arestas de especialização, por sua vez, são orientadas das superclasses para as subclasses, representando o fato de a subclasse, destino da aresta, ser uma especialização da superclasse em que a aresta tem origem.

É fácil notar que uma aresta ligando dois nós no grafo de herança é sempre acompanhada por outra aresta de tipo diferente e direção oposta: a superclasse é uma *generalização* da subclasse, e esta é uma *especialização* da primeira.

O modelo interno de GOODIES provê suporte para a representação do conceito de herança múltipla, pois um nó do grafo de herança pode possuir um número arbitrário

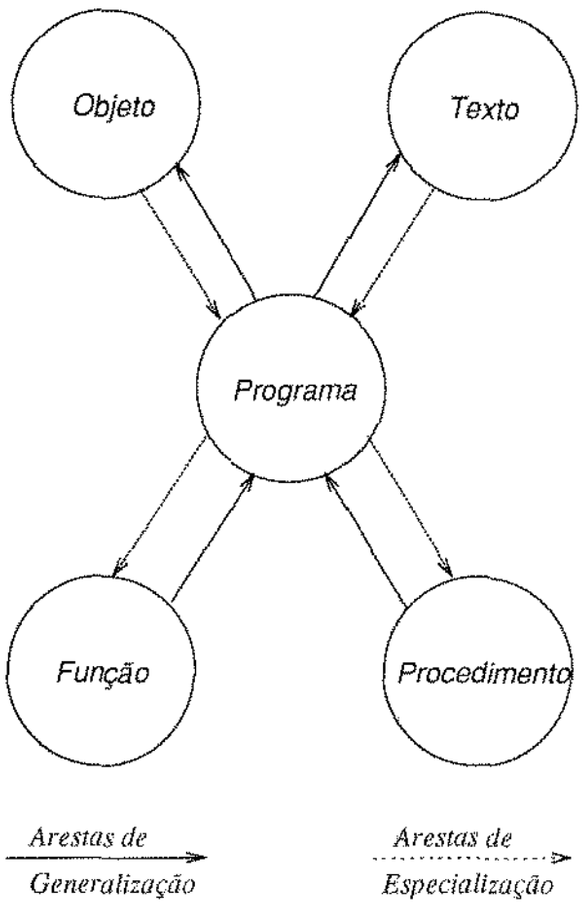


Figura 5.1: Exemplo de Grafo de Herança

de arestas originárias e/ou incidentes de qualquer tipo. Assim, uma classe pode, por exemplo, ser a especialização de duas outras; neste caso, duas arestas de generalização teriam origem no nó que representa a classe. De acordo com o modelo OO a classe especializada herda atributos e métodos de suas superclasses.

A mesma classe que representa uma especialização de outras duas pode, a seu turno, ser a generalização, por exemplo, de três outras subclasses; neste caso, o nó que representa a classe no grafo de herança possuiria três arestas de generalização incidentes. A figura 5.1 apresenta uma parte do grafo de herança construído a partir da classe visualizada na figura 4.3.

5.3.2 Grafo de Composição

O *grafo de composição* é um grafo orientado, podendo conter ciclos. Existem três tipos básicos de nós no grafo de composição: nós de classe, nós de construtor e nós de atributo. Um nó de classe é rotulado com o nome da classe que ele representa no esquema, e para cada classe do esquema existe um, e somente um, nó correspondente no grafo de composição. Portanto, não existem rótulos repetidos para nós de classe.

Os nós de construtor e de atributo são subordinados aos nós de classe, e representam a composição do tipo da classe. Um nó de atributo é rotulado com o tipo do atributo representado, e podem haver rótulos repetidos para este tipo de nó. GOODIES utiliza as seguintes representações para atributos atômicos:

1. atributo simples;
2. texto;
3. imagem;
4. som.

Estes quatro tipos de atributos são usados como rótulos dos nós de atributo. Sub-objetos, como já foi dito, representam o conceito de objeto complexo, e funcionam como referências a classes de objetos definidas no esquema. Logo, sub-objetos são representados no grafo de composição como referências a nós de classe.

Apenas nós de classe e nós de construtor podem dar origem a arestas no grafo de composição. Um nó de classe define o tipo da classe através dos nós de atributo e dos nós de construtor ligados a ele. Nós de construtor podem originar arestas porque eles representam os construtores de tipo utilizados pelo modelo interno de GOODIES, que são as listas e as tuplas. Os demais nós são utilizados apenas como destino de arestas naquele grafo.

Uma aresta no grafo de composição liga um nó *"compositor"* a um nó *"componente"* (sendo orientada do compositor para o componente), e é rotulada com o nome do atributo descrito no tipo da classe. As arestas que ligam nós subordinados a um mesmo nó de classe não podem ter rótulos repetidos, ou seja, dois atributos da mesma classe não podem possuir nomes idênticos. A figura 5.2 mostra a visão parcial de um grafo de composição construído a partir do tipo da classe apresentada na figura 4.3.

Um nó de classe dá origem a uma aresta para cada atributo definido no tipo da classe que ele representa. De modo similar, um nó de classe recebe uma aresta incidente para

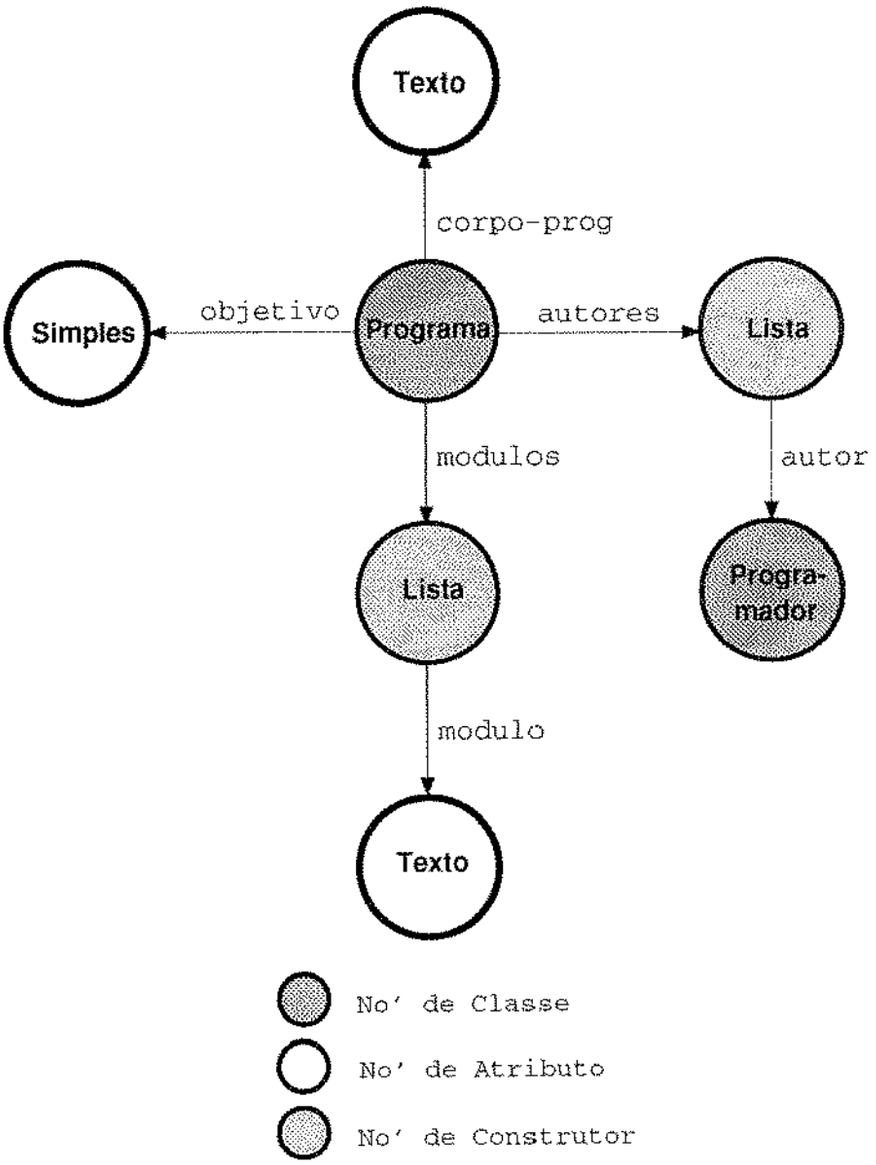


Figura 5.2: Exemplo de Grafo de Composição

cada atributo cujo tipo seja a classe representada pelo nó, independente deste atributo ter sido definido na própria classe ou em outra classe do esquema. Com isso GOODIES permite que um objeto complexo possua como atributo um objeto de seu próprio tipo.

Um nó de construtor do tipo "lista" possui sempre uma, e apenas uma, aresta incidente, pois cada lista faz parte da composição do tipo de uma só classe. Da mesma forma um nó de "lista" origina sempre uma única aresta que aponta para um nó que indica o tipo dos elementos da lista, já que a definição do construtor "lista" exige que todos os seus elementos sejam do mesmo tipo.

Pelo mesmo motivo descrito para o nó de "lista", um nó de construtor do tipo "tupla" possui sempre uma única aresta incidente. No entanto, um nó de "tupla" pode possuir um número arbitrário de arestas originando-se dele, pois o construtor "tupla" permite o agrupamento de elementos de tipos heterogêneos. Cada aresta originária de um nó de "tupla" aponta para o tipo de atributo de um dos componentes da tupla.

Os nós do grafo de composição que não podem dar origem a arestas representam atributos elementares (atômicos) de GOODIES (os quatro tipos de atributos descritos acima). Estes nós possuem sempre uma única aresta incidente, rotulada, como já foi visto, com o nome do atributo cujo tipo é identificado pelo nó em que a aresta incide.

5.4 Operações Primitivas

A interação entre GOODIES e o SGBDOO é feita através de um número reduzido de *operações primitivas*. A semântica destas operações é definida por GOODIES, enquanto o modo de implementá-las é próprio de cada SGBD. Portanto, as operações primitivas definem o módulo do sistema GOODIES que é dependente do SGBD utilizado. Note-se, porém, que apenas a implementação das operações é variável; suas semânticas são definidas no modelo interno de GOODIES, independentemente de qualquer SGBD.

As operações primitivas foram projetadas como um conjunto mínimo de funções que devem ser supridas por um SGBDOO para que um usuário possa ter acesso e controle sobre os dados armazenados nos BDs. Na verdade, as operações primitivas podem ser interpretadas como consultas exclusivamente textuais sobre esquemas e sobre objetos dos BDs. As operações primitivas definidas por GOODIES são:

Get-Schema: A semântica desta operação consiste em obter do SGBDOO a lista de classes definidas em um determinado esquema. Para isso a operação recebe um nome de BD como parâmetro. A resposta obtida do SGBD é processada por GOODIES, armazenada em suas estruturas de dados, e apresentada ao usuário do sistema através

da lista de classes da janela de BD (figura 4.2). Esta operação é solicitada sempre que o usuário escolhe um novo BD para interagir, a partir da janela de diretório (figura 4.1).

Get-Class: Através desta operação, GOODIES obtém do SGBDOO a descrição completa de uma determinada classe de um dado esquema. A classe e o esquema são recebidos como parâmetros da operação. A descrição de uma classe, como já foi visto no capítulo 4, é composta pela definição da composição da classe (tipo da classe) e pelas listas de superclasses, subclasses, métodos e objetos pertencentes à classe. Esta operação é chamada quando o usuário escolhe uma classe para visualizar, a partir da janela de BD (figura 4.2). Os resultados desta operação são utilizados por GOODIES para a construção dos grafos de herança e de composição, e são apresentados ao usuário através da janela de classe (figura 4.3).

Get-Object: Esta operação representa uma consulta aos dados de um BD, no sentido comumente empregado para este termo na área de BDs. A operação deve receber como parâmetros o esquema, a classe e a identificação do objeto que se deseja consultar. Como resposta, o SGBDOO envia ao sistema os valores dos atributos do objeto solicitado. Através dos grafos de herança e de composição o sistema interpreta os valores recebidos, guardando-os em sua memória privada. O resultado da consulta efetuada é apresentado ao usuário através da janela de objeto (figura 4.4). A operação *Get-Object* é tipicamente utilizada para implementar as operações de seqüenciamento, definidas no capítulo anterior.

É óbvio que tanto o comando a ser submetido quanto o formato dos resultados produzidos são dependentes do SGBDOO utilizado. A adaptação do sistema GOODIES a diferentes SGBDs consiste, portanto, em: a) determinar a sintaxe dos comandos que possuem a semântica descrita em cada SGBD; e b) adaptar a função de recebimento de respostas do sistema GOODIES ao formato utilizado pelo SGBD específico.

5.5 Seqüências de Operações: Transações

Como foi visto, as operações primitivas definidas no modelo interno de GOODIES são, por um lado, dependentes da implementação adotada para essas operações em cada SGBDOO, e por outro lado, conceitualmente independentes de um SGBDOO específico, já que a semântica das operações é modelada pelo sistema. Assim, a dependência de um SGBDOO específico fica limitada à sintaxe das operações que correspondem à semântica definida no modelo interno de GOODIES.

A nível de transação, isto é, de seqüências de operações, o sistema de interface é completamente independente do SGBD utilizado. GOODIES não impõe bloqueios (*locks*) sobre os dados dos BDs, e também não efetua qualquer tipo de restrição sobre as operações enviadas para o processamento do SGBD.

Para garantir esta independência, o modelo interno de GOODIES trabalha com o conceito de *snapshot*, de modo que cada operação é realizada sobre uma cópia dos dados armazenados no BD. Antes de acionar uma operação, o sistema verifica a presença dos dados necessários na memória privada da interface, realizando uma consulta, através das operações primitivas, sempre que os dados exigidos não estejam disponíveis na memória privada.

O modelo interno de GOODIES parte da premissa de que o usuário está manipulando BDs estáveis. Mudanças no esquema de um BD durante uma sessão de consulta em GOODIES podem causar resultados inesperados, já que a execução de uma operação se baseia, na maioria dos casos, em resultados obtidos de operações anteriores. Por exemplo, se um objeto está sendo apresentado e, através de uma alteração no esquema do BD, um de seus atributos é removido, ocorrerá um erro na próxima operação de seqüenciamento sobre o objeto, pois a estrutura de dados obtida a partir do grafo de composição será diferente da estrutura na qual o SGBD enviará os dados do objeto.

Alterações realizadas sobre os dados de um objeto apresentado não são refletidas na apresentação do objeto, porque, para isso, uma das seguintes alternativas deveria ser escolhida: solicitar ao SGBD a sinalização de mudanças ocorridas nos objetos, ou consultar continuamente o objeto para garantir que a apresentação contém os valores atuais do objeto.

No primeiro caso, a interface se tornaria dependente de uma capacidade especial do SGBD, a capacidade de enviar uma mensagem para sinalizar qualquer modificação ocorrida nos dados. Essa capacidade, ao contrário daquelas exigidas pelas operações primitivas propostas (*Get-Schema, Get-Class, Get-Object*), não é comumente provida pelos SGBDOO's existentes.

No caso de se consultar continuamente os objetos do BD, surgem problemas de excesso de comunicação entre a interface e o SGBD, e de sobrecarga do sistema com operações que, na maioria absoluta das vezes, são desnecessárias. É fácil verificar que esta opção causaria uma queda acentuada no desempenho do SGBD. Desta forma, ambas as alternativas representam custos excessivos, diante de um benefício que pode ser considerado pequeno, que é a visualização imediata de modificações efetuadas sobre o objeto.

É preciso observar ainda que uma operação de exclusão aplicada sobre um objeto de uma classe pode causar uma situação inconsistente, se a classe está sendo visualizada no momento em que a operação é executada. A lista de objetos apresentada na janela de

classe não é afetada pela operação, por motivos semelhantes aos que desaconselham a reflexão automática de atualizações de atributos na janela de objeto. Já a inclusão de objetos não causará problemas, pois a lista de objetos não é alterada, e desse modo a interface irá simplesmente ignorar o objeto incluído.

5.6 Considerações sobre o Modelo Interno

GOODIES é um sistema de interface com o usuário que se aplica a SGBDs que seguem o modelo de dados orientado a objetos. O modelo interno de GOODIES rege a interação entre o sistema de interface e o SGBDOO utilizado.

Dois aspectos principais podem ser destacados com relação ao modelo interno de GOODIES: concisão e independência de SGBD. Ambos os aspectos se influenciam mutuamente, de modo que a independência de um SGBD específico evita a modelagem de detalhes de implementação desse sistema, enquanto a concisão do modelo contribui para a independência de SGBD, justamente por não representar estes detalhes.

Os principais fatores que contribuem para a concisão do modelo interno do sistema GOODIES são:

- A utilização de um número reduzido de operações primitivas para especificar a interação com o SGBD;
- A representação de características básicas do modelo OO através dos grafos de herança e de composição.
- O princípio de estabilidade dos BDs assumido, evitando a preocupação com detecção e tratamento de inconsistências, e eliminando a necessidade de imposição de bloqueios sobre os dados.

Da mesma forma, os principais aspectos que possibilitam a relativa independência de GOODIES com relação ao SGBD utilizado podem ser destacados:

- O emprego do conceito de *snapshot*, copiando para uma memória privada os dados utilizados;
- O estabelecimento das características do modelo OO que são comuns aos diversos SGBDOOs existentes;

- O isolamento das partes dependentes do SGBD em um único módulo, preparado para se adaptar a diversos tipos de sintaxe para consulta;
- A modelagem das funções básicas exigidas do SGBD, através da definição das operações primitivas.

A característica de independência relativa de um SGBDOO específico destaca GOODIES da maioria das interfaces existentes para esses sistemas. Essa independência permite, por exemplo, que o aspecto interface com o usuário seja considerado um problema resolvido em novos projetos de SGBDOOs, já que GOODIES pode ser facilmente empregado para esta finalidade. Portanto, o desenvolvimento de GOODIES deve contribuir para a simplificação dos futuros projetos de SGBDOOs.

Capítulo 6

GOODIES: Implementação

6.1 Introdução

Este capítulo discute o mapeamento dos modelos interno e externo do sistema GOODIES para um conjunto de programas que executam a funcionalidade proposta por estes modelos.

A implementação do sistema permitiu a validação dos modelos propostos, garantindo a viabilidade de se prover um mecanismo de navegação poderoso, e ainda assim, independente das características específicas de um determinado SGBDOO. Além disso, a implementação de GOODIES propiciou o estudo de detalhes que são abstraídos dos modelos do sistema, em nome da concisão e da independência de SGBD.

A experiência de desenvolver GOODIES possibilitou a análise de outros fatores envolvidos na construção de sistemas de interface. A utilidade das diretivas propostas no capítulo 3 para a construção de interfaces gráficas para SGBDs, por exemplo, foi comprovada. O emprego de diretivas mais genéricas, como as recomendadas pelo padrão OPENLOOK, também contribuiu para o desenvolvimento do sistema, principalmente na manutenção da consistência em todos os aspectos da interação com o usuário. O estudo do paradigma de orientação a objetos, aplicado no contexto de projetos de sistemas, foi outro fator relevante para a implementação de GOODIES.

O restante deste capítulo aborda diversos aspectos relativos à implementação do sistema GOODIES. A seção 6.2 apresenta a plataforma de *hardware* e *software* utilizada pelo sistema. Na seção seguinte são apresentados os padrões gerais que refletem o estilo de GOODIES. A seção 6.4 analisa a arquitetura modular do sistema. A seção 6.5 encerra este capítulo, abordando as técnicas de projeto orientado a objetos utilizadas no

desenvolvimento dos programas que compõem GOODIES.

6.2 Plataforma de Hardware e Software

GOODIES foi desenvolvido em estações de trabalho *SPARCstation* (modelos *SPARCstation SLC*, *SPARCstation 1+*, e *SPARCstation 2*) fabricadas pela *Sun Microsystems, Inc.*. A estação utilizada para executar o sistema deve ser equipada com monitor gráfico de alta resolução, podendo ser colorido ou não. Além disso, a estação precisa possuir a capacidade de reprodução de gravações sonoras.

O sistema operacional utilizado para executar GOODIES deve ser compatível com o sistema UNIX, e precisa suportar a execução de um sistema gerenciador de janelas (*Window Manager*) que obedeça às definições utilizadas pelo *Sistema X* (*X_Window System*), e que siga a especificação funcional OPENLOOK (*OPENLOOK Graphical User Interface Functional Specification*).

O sistema X se baseia em um protocolo de rede (o *protocolo X*) que utiliza o modelo de arquitetura cliente/servidor. O protocolo X gerencia a comunicação entre processos clientes e processos servidores. Um processo servidor monitora recursos do usuário (como tela e teclado) e torna estes recursos disponíveis para as aplicações (processos clientes). Diversos processos clientes podem estar conectados a um único processo servidor. Além disso clientes e servidores podem estar executando em estações diferentes de uma rede de estações de trabalho. Entre outras tarefas, o servidor X é responsável pelo controle de acesso à tela pelos diversos clientes; pela interpretação das mensagens dos clientes; pelo atendimento dos pedidos contidos nessas mensagens; e pela transmissão de entradas do usuário aos clientes, na forma de eventos do protocolo X.

O sistema Xlib provê o nível mais baixo de interface entre uma aplicação escrita em linguagem C e o protocolo X. Xlib traduz estruturas de dados e funções C para eventos do protocolo X, e ao mesmo tempo converte pacotes de informações recebidas do protocolo X em estruturas de dados no formato usado pela linguagem C. Através de Xlib, um usuário pode ter acesso a toda a funcionalidade do protocolo X. Entretanto, o conjunto de funções é extenso, e sua programação não é simples. Por isso, *toolkits* são empregados na simplificação do acesso às funções mais comuns, facilitando ainda a codificação da parte da aplicação referente aos componentes de interface com o usuário.

XView [Hel90] é um dos *toolkits* baseados em Xlib que provê estas facilidades. XView é um sistema orientado a objetos que fornece componentes de interface configuráveis e reutilizáveis, como janelas e botões. XView facilita o desenvolvimento de aplicações que executam sob o protocolo X, provendo uma estrutura que permite a combinação dos

componentes de interface pré-existentes com o código específico da aplicação. XView interpreta os eventos recebidos do protocolo X e decide quanto à necessidade de propagar esses eventos para a aplicação. Uma importante característica de XView é que ele implementa as diretivas propostas pelo padrão OPENLOOK [Sun90].

GOODIES utiliza as facilidades oferecidas por XView para se adequar ao padrão OPENLOOK, e para usar os recursos do sistema X. Como XView é um sistema orientado a objetos, escolheu-se a linguagem de programação C++ para a implementação de GOODIES, de modo que tanto o código referente à interface com o usuário quanto o código da aplicação propriamente dita seguem o mesmo estilo de programação. A versão atual do sistema contém cerca de quatorze mil linhas de código fonte.

6.3 Padrões Adotados

A implementação de GOODIES foi orientada pelas recomendações propostas pelo padrão OPENLOOK para desenvolvimento de interfaces gráficas. O padrão OPENLOOK [Sun90] especifica o *look and feel*[1] de uma aplicação baseada em janelas, incluindo os tipos de objetos apresentados ao usuário e as convenções básicas que definem a maneira pela qual ele interage com estes objetos.

OPENLOOK define três princípios básicos para projeto de aplicações: simplicidade, consistência e eficiência.

A simplicidade deve ser obtida através de um bom projeto visual, com controles rotulados de maneira clara e precisa, refletindo, o mais fielmente possível, o mundo real da aplicação.

O princípio da consistência permite que o usuário utilize conhecimentos previamente adquiridos em novas áreas e funções da aplicação. Para isso deve haver um padrão determinado para uso de *mouse* e teclado, bem como uma convenção única para nomes e posições de controles em todas as janelas da aplicação. Além disso, OPENLOOK sugere que sejam oferecidos dois métodos para manipulação de informações: *select-then-operate* (selecionar o objeto antes de indicar a operação desejada) e manipulação direta (discutida na introdução desta dissertação).

Com relação à eficiência, a aplicação deve minimizar o número de passos necessários para realizar uma operação. É conveniente prover "atalhos" para as tarefas executadas com maior freqüência. Controles mais poderosos e/ou sofisticados devem estar disponíveis,

[1] A expressão *look and feel* é largamente utilizada na literatura sobre interfaces e representa a aparência e a funcionalidade de uma aplicação.

mas isolados em menus e janelas *pop-up*.

6.3.1 Simplicidade e Clareza de Controles

Os três princípios básicos do estilo OPENLOOK foram empregados na implementação de GOODIES. O projeto visual do sistema utiliza nove tipos de janelas, sendo quatro janelas básicas (*base windows*) e cinco janelas auxiliares (*pop-up windows*). Apesar do número razoavelmente elevado de janelas, o sistema atinge o objetivo de simplicidade, já que a aplicação a que se destina é bem representada por estas janelas.

As quatro janelas básicas permitem quatro níveis distintos para a realização das operações de navegação e consulta. Assim, o usuário pode navegar pelos diretórios do sistema, pelos BDs de um diretório, pelas classes de um BD, e pelos objetos de uma classe. Estas operações são inerentemente hierárquicas, e esta hierarquia é representada pela subordinação existente entre os quatro tipos de janelas básicas.

Com relação aos controles, as janelas básicas, com exceção da janela de diretório, seguem a recomendação do padrão OPENLOOK, definindo quatro tipos de menus de funções. As funções que afetam a aplicação como um todo são reunidas no menu rotulado *"File"* (funções de arquivo). As funções que afetam o modo de visualização de um determinado aspecto da aplicação são agrupados no menu *"View"* (funções de visualização). O menu *"Edit"* contém as funções que afetam o estado de objetos que o usuário pode selecionar (funções de edição). Finalmente, o menu *"Props"* reúne as funções que alteram as propriedades de partes da aplicação (funções de propriedades).

A janela de diretório propicia apenas um meio confortável para a escolha de BDs. Ela não deveria ser implementada como janela básica, e sim como janela auxiliar. Isto não foi feito por limitações impostas pelo XView. Este *toolkit* não admite que uma janela auxiliar, no caso uma *pop-up window*, seja utilizada como única janela de uma aplicação. Como GOODIES inicialmente só apresenta a janela de diretório, esta foi implementada como uma janela básica (*base window*). Ela contém dois controles, posicionados como se estivessem em uma janela auxiliar (pois conceitualmente, a janela de diretório é uma janela auxiliar). Um dos controles descarta a janela, enquanto o outro causa a abertura do BD ou diretório selecionado. Estes controles são rotulados, respectivamente, *"Quit"* e *"Open"*.

A janela de objeto contém outros controles, além dos quatro menus definidos acima. Ainda segundo o padrão OPENLOOK, a janela de objetos apresenta três botões que provêem acesso às funções que devem ser as mais utilizadas em uma aplicação de navegação em BDs: as operações de seqüenciamento. Os botões que representam estas operações possuem rótulos explícitos: *"First"* para o primeiro objeto de uma classe, *"Next"* para o

próximo elemento da classe e *"Previous"* para o elemento anterior na lista de objetos da classe.

Usando apenas as quatro janelas básicas, o usuário pode realizar navegações completas sobre diversos BDs. As janelas auxiliares de GOODIES servem para duas finalidades: visualização de dados não-estruturados e acesso a funções menos comuns. As janelas que servem para a primeira finalidade (janela de texto e janela de imagem) apresentam os mesmos controles para seqüenciamento definidos na janela de objeto. As demais janelas auxiliares (janela de arquivo, janela de predicado e janela de atributos) possibilitam o uso mais elaborado do sistema.

A janela de predicado permite, como já foi visto, associar um predicado a uma janela de objeto, de forma que o processo de navegação seja similar a um processo de consulta. Três botões de controle existem nesta janela. Os rótulos desses botões indicam claramente as suas respectivas finalidades: *"Apply"* efetiva a associação entre o predicado definido e a janela de objeto; *"Save"* salva o predicado definido em um arquivo em disco; e *"Load"* carrega na janela de predicado o conteúdo de um arquivo.

O arquivo a ser utilizado pelas operações *"Save"* e *"Load"* é escolhido através da janela de arquivo. Esta janela é muito semelhante à janela de diretório; a principal diferença é que a janela de diretório considera válidos arquivos do tipo BD, enquanto a janela de arquivos considera válidos os arquivos do tipo texto.

Finalmente, a janela de atributos possui um único controle. Um botão com o rótulo *"Apply"* permite que o usuário efetive a escolha efetuada na lista de atributos.

6.3.2 Consistência e Eficiência

O princípio de consistência foi enfatizado no modelo externo de GOODIES, e foi implementado exatamente segundo a definição deste modelo. A principal maneira de interação entre o usuário e o sistema é através do *mouse*, e existe uma convenção rígida que controla esta interação. A convenção se baseia no método *select-then-operate*. Facilidades de manipulação direta são também oferecidas. GOODIES utiliza os recursos do *Window Manager* para prover algumas dessas facilidades como, por exemplo, a movimentação e a *"iconização"* de janelas. O próprio GOODIES trata os eventos de modificação do tamanho das janelas, adaptando o conteúdo de cada janela à dimensão desejada pelo usuário.

Também o princípio de eficiência foi considerado no projeto de GOODIES, e a sua implementação seguiu fielmente as definições do modelo externo do sistema. As operações mais freqüentemente utilizadas, que são as que criam janelas a partir de referências existentes em outras janelas podem ser feitas pelo método *select-then-operate*, ou podem ser

executadas por um "atalho" representado por um *click* duplo no campo de referência. Além disso, e como já foi dito acima, janelas auxiliares provêem acesso a funções que amplificam o poder do sistema.

6.4 Módulos do Sistema

A implementação de GOODIES foi dividida em três módulos principais. Dois destes módulos implementam o modelo externo e o modelo interno do sistema. O terceiro módulo controla a funcionalidade do próprio sistema. Cada módulo principal possui outros módulos subordinados, formando a hierarquia representada na figura 6.1.

6.4.1 Módulo de Interação com o Usuário (*MIU*)

Toda a funcionalidade descrita pelo modelo externo de GOODIES é implementada pelo MIU. Este módulo é totalmente independente do SGBDOO utilizado, e contém as definições dos componentes gráficos da interface (janelas, menus, botões, listas, entre outros).

O MIU é composto por dois submódulos que cuidam, respectivamente, das funções associadas às janelas básicas e às janelas auxiliares de GOODIES. Também esses submódulos possuem outros módulos que os compõem. São apresentados a seguir nove módulos; os quatro primeiros são componentes do Módulo de Janelas Básicas (MJB), e os cinco módulos restantes formam o Módulo de Janelas Auxiliares (MJA).

Módulo de Gerência de Diretórios (*MGD*)

Este módulo implementa a funcionalidade da janela de diretório, que é criada na abertura de uma sessão de trabalho em GOODIES. Apenas uma instância da janela de diretório é criada ao longo de uma sessão de trabalho. Todas as outras janelas utilizadas são hierarquicamente subordinadas à janela de diretório. O MGD controla a presença de janelas subordinadas, só permitindo a destruição da janela de diretório (e conseqüentemente o encerramento da sessão de trabalho) quando não existam outras janelas no sistema.

O MGD responde ainda pela interpretação de entradas de dados textuais na janela de diretório. Diversos formatos de entrada podem ser utilizados, e o MGD interpreta e provê respostas adequadas a cada tipo de entrada. A tabela 6.1 apresenta os tipos de entrada aceitas pelo MGD. Se o arquivo digitado for um diretório, ele passa a ser o

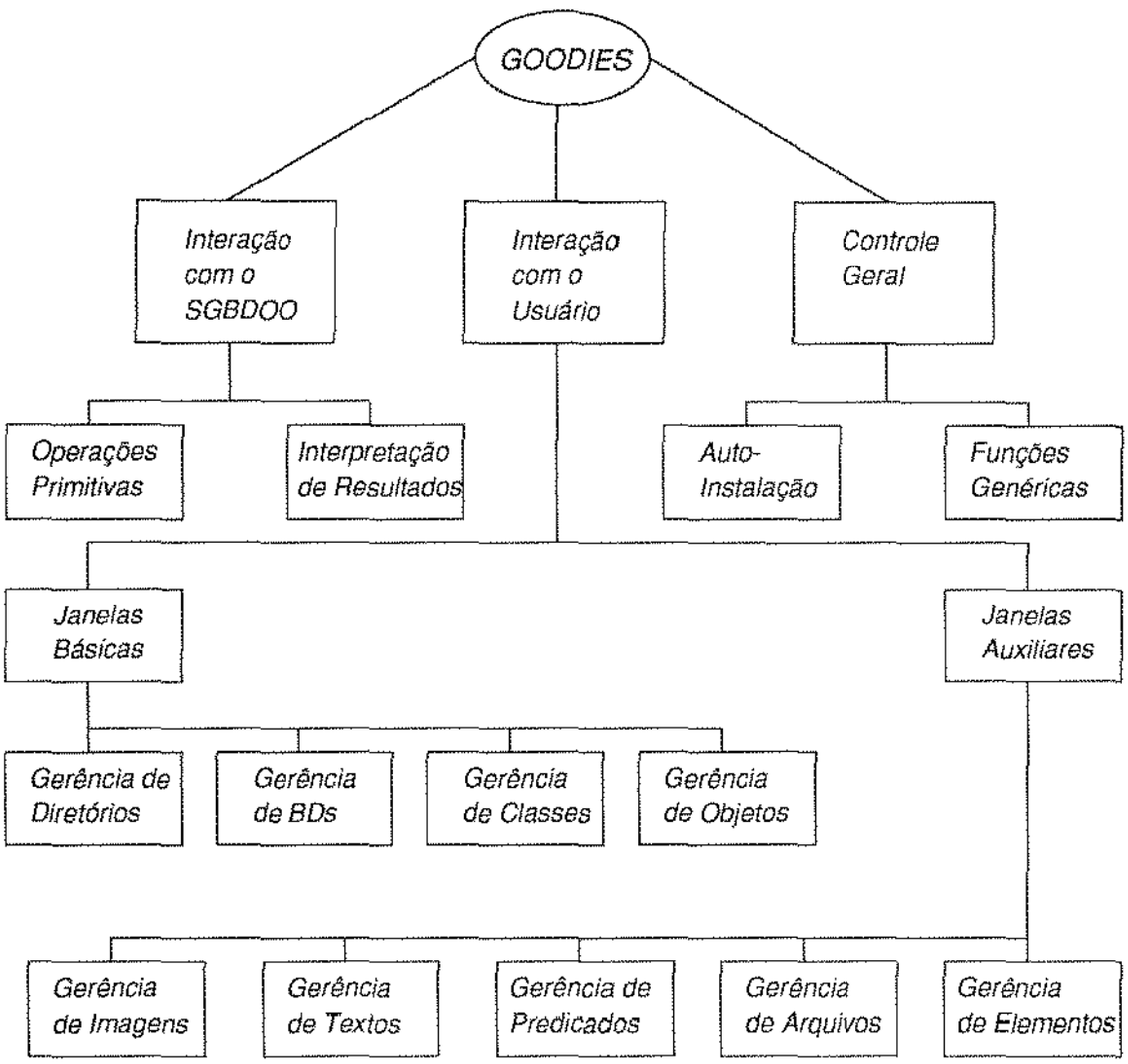


Figura 6.1: Arquitetura Modular de GOODIES

Caracteres Iniciais	Interpretação
./	Caminho que se inicia no diretório corrente
../	Caminho que se inicia no diretório imediatamente superior na hierarquia de diretórios
/	Caminho que se inicia no diretório raiz do sistema de arquivos
~	Caminho que se inicia no diretório *"home"* do usuário corrente
~*nome-de-usuário*	Caminho que se inicia no diretório *"home"* de um usuário cujo *"login-name"* é *nome-de-usuário*
Outros caracteres	Nome de arquivo ou diretório existente no diretório corrente

Tabela 6.1: Entradas aceitas pelo Módulo de Gerência de Diretórios

diretório corrente; caso contrário, se o arquivo é um BD, o MGD encaminha o arquivo para o Módulo de Gerência de BDs.

Módulo de Gerência de BDs (*MGB*)

O MGB é chamado para tratar a seleção de um BD na janela de diretório. O módulo recebe do MGD o BD selecionado, e verifica se já existe uma janela para este BD. Caso exista, o MGB irá simplesmente garantir que esta janela seja visível, trazendo-a para a frente de todas as outras janelas existentes no *workspace* do usuário. Caso contrário, o MGB cria a janela de BD e faz uma chamada ao Módulo de Operações Primitivas para obter as classes que compõem o esquema do BD selecionado.

Assim, um BD está associado a no máximo uma janela de BD. O usuário pode, todavia, trabalhar com diversos BDs distintos simultaneamente, já que a seleção de um novo BD causa a criação de uma nova instância da janela de BD. Este é o mecanismo básico de navegação sobre BDs. A navegação sobre as classes do esquema de um BD, por sua vez, é iniciada com a seleção de uma classe na janela de BD. Esta seleção faz com que o MGB passe o controle para o Módulo de Gerência de Classes, responsável pela criação da janela de classe para a classe selecionada.

O MGB controla todas as suas janelas subordinadas, que são as janelas de classe dos diversos BDs. A destruição de uma janela de BD é propagada para as suas janelas subordinadas, que também são destruídas.

Duas outras funções executadas pelo MGB são dignas de nota: a chamada da janela de diretório e o salvamento de contexto. Como já foi visto, a janela de diretório só é destruída quando não existem janelas subordinadas. Se o usuário seleciona o botão de destruição da janela de diretório antes de eliminar as demais janelas, aquela janela não é destruída, mas apenas desaparece do *workspace* do usuário. Este pode, então, recuperar a janela de diretório através de uma função disponível na janela de BD.

A função para salvar o contexto de uma sessão de trabalho está disponível através do menu *"Props"* da janela de BD. O MGB salva, em um arquivo denominado *"goodies.init"*, situado no diretório *home* do usuário, as janelas de BD e de classe definidas no instante da execução da função. Este arquivo é utilizado pelo Módulo de Auto-Instalação para configurar o contexto inicial de uma sessão de trabalho em GOODIES.

Módulo de Gerência de Classes (*MGC*)

Como já foi dito, a seleção de uma classe em uma janela de BD causa a chamada do MGC. Assim, como o MGB, o MGC verifica a existência da janela de classe, pois apenas uma instância de janela de classe pode existir para cada classe de um esquema. Se a janela de classe existe, o procedimento adotado é análogo ao descrito para o MGB. Caso contrário, a janela de classe é criada, e uma chamada ao Módulo de Operações Primitivas é necessária para preencher os campos desta janela.

A navegação sobre as classes é feita de forma semelhante à navegação sobre BDs. O usuário abre uma janela distinta para cada classe do esquema que ele deseja visualizar. No entanto, existem dois modos de seleção de classes. O primeiro modo é através da janela de BD, onde o usuário pode selecionar qualquer classe do esquema. O segundo modo é a navegação na hierarquia de classes: a partir da janela de classe, o usuário pode selecionar superclasses ou subclasses da classe descrita pela janela, usando as listas nela existentes. Além disso, o usuário pode selecionar métodos da classe para serem visualizados.

Também a navegação sobre os dados é iniciada na janela de classe, através da seleção de um objeto na lista de objetos. A seleção de métodos e de objetos causa a transferência do controle para o Módulo de Gerência de Textos e para o Módulo de Gerência de Objetos, respectivamente.

O MGC controla suas janelas subordinadas (janelas de método e janelas de objeto), de modo que a destruição de uma janela de classe provoca a destruição de todas as janelas subordinadas a ela.

O menu *"Props"* da janela de classe provê acesso a três funções que permitem o controle do nível de abstração em que as classes e seus objetos são apresentados em GOODIES.

As funções *"Display Attributes"*, *"Display Superclasses* e *"Display Subclasses"* utilizam as facilidades do Módulo de Gerência de Elementos para selecionar, respectivamente, os atributos a serem visualizados na janela de classe, as superclasses que devem entrar na composição da classe, e as subclasses cujas instâncias devem ser incluídas na lista de objetos da janela de classe. A flexibilidade visual oferecida por estas funções é um dos pontos fortes do sistema GOODIES.

Módulo de Gerência de Objetos (*MGO*)

O MGO é o responsável pela criação e pelo controle das janelas de objeto, pelos mecanismos de sincronização e pelas operações de seqüenciamento. Ao contrário do que ocorre nas janelas de BD e de classe, podem existir diversas janelas de objeto representando um único objeto de uma classe. Esta possibilidade torna o processo de navegação bastante poderoso, permitindo, por exemplo, a visualização simultânea de um mesmo objeto em diferentes níveis de abstração. Todas estas facilidades contribuem para a complexidade da implementação do Módulo de Gerência de Objetos.

A navegação sobre objetos pode ser feita da mesma forma que a navegação sobre esquemas. O processo se inicia com a seleção de um objeto na janela de classe. Se já existe a janela de objeto para o objeto selecionado, ela é apresentada ao usuário; caso contrário ela é criada, efetuando-se uma chamada ao Módulo de Operações Primitivas para obter os valores dos atributos (pois a estrutura de composição do objeto já está disponível na janela de classe). O usuário pode abrir uma janela de objeto distinta para cada novo objeto selecionado na janela de classe. No entanto, isto não é necessário, devido às operações de seqüenciamento disponíveis na janela de objeto (botões *First*, *Next* e *Previous*). Utilizando apenas uma janela de objeto e as operações de seqüenciamento, o usuário pode navegar sobre toda a extensão de uma classe.

São as operações de seqüenciamento que impõem a necessidade de janelas de objeto distintas poderem representar o mesmo objeto. Considere, por exemplo, a situação em que um usuário seleciona, na lista de objetos de uma janela de classe, os objetos que ocupam a primeira e a segunda posição. A seleção do botão *Previous* na janela do objeto correspondente à segunda posição daquela lista irá atualizar o conteúdo desta janela, que passará a ser o mesmo da janela criada pela seleção do primeiro elemento da lista de objetos. O MGO deve reconhecer e controlar as instâncias de janelas que representam o mesmo objeto.

GOODIES é um sistema de navegação com sincronismo estendido, no sentido de que a navegação entre objetos que formam um objeto complexo é, por *default*, sincronizada. O fluxo de controle exigido, tanto internamente ao MGO quanto entre o MGO e os módulos

que gerenciam suas janelas subordinadas é bastante complexo. Uma hierarquia de sincronização, como já foi visto, pode ter um tamanho arbitrário, e o MGO precisa garantir que todos os componentes dessa hierarquia sejam coerentes entre si. A propagação de operações, neste caso, não se limita ao processo de destruição de janelas, mas abrange ainda as operações de seqüenciamento, as quais podem, por sua vez, gerar replicações de janelas que o MGO deve controlar.

Entre as diversas funções existentes no MGO, três ainda merecem ser comentadas. A função de escolha de atributos a serem visualizados na janela de objeto permite um ajuste fino no nível de abstração desejado para a visualização do objeto. Esta função utiliza as facilidades providas pelo Módulo de Gerência de Elementos.

A segunda função permite a quebra de um relacionamento de sincronismo. O efeito desta função é o de desligar uma janela de objeto de seu ancestral direto na hierarquia de sincronização, de modo que a janela de objeto passa a se comportar como se o objeto houvesse sido selecionado diretamente da lista de objetos de sua janela de classe.

Por fim, a terceira função mencionada possibilita a representação de atributos sonoros. O MGO considera que o valor de um atributo do tipo "som" é um nome de arquivo completo (isto é, um nome de arquivo precedido pelo caminho do diretório raiz até o diretório que contém o arquivo). O arquivo deve estar no formato *audio* utilizado pelas estações de trabalho da *Sun Microsystems, Inc.*. Uma chamada de função do sistema operacional permite que o arquivo indicado seja passado ao dispositivo de áudio da estação de trabalho utilizada, causando a reprodução do som armazenado.

Módulo de Gerência de Imagens (*MGI*)

O MGI é ativado durante o processo de navegação sobre os dados, quando o usuário seleciona um atributo do tipo "imagem" em uma janela de objeto. O módulo considera que o valor obtido para esse atributo é um nome de arquivo completo. Além disso, o arquivo deve utilizar o formato *raster*.

O formato raster é o padrão utilizado pela *Sun*, embora suas estações suportem outros formatos para arquivos de imagem, como por exemplo os formatos *Bitmap*, *Postscript* e *GIF*. A vantagem do formato raster é que o arquivo de imagem contém um cabeçalho que identifica o conteúdo do arquivo, facilitando a manipulação da imagem (identificação das cores e das dimensões da imagem, por exemplo).

Ao receber um nome de arquivo completo, o MGI verifica se o arquivo existe, e se ele segue o formato raster. Caso o arquivo satisfaça estas condições, o MGI testa a compatibilidade entre a imagem e o monitor utilizado pela estação de trabalho. Este

teste é necessário porque o sistema X suporta diversas classes de monitores, divididas basicamente entre monitores coloridos e monitores monocromáticos. A quantidade de cores representáveis pode variar enormemente (de 256 a 16 milhões de cores) conforme o tipo de monitor utilizado. Mesmo os monitores monocromáticos podem ser diferentes, suportando tons de cinza ou apenas as cores preta e branca.

Após constatar que a imagem pode ser representada no monitor da estação, o MGI utiliza recursos de Xlib para apresentar a imagem na janela de imagem. Convém ressaltar que este é o único módulo do sistema GOODIES que faz uso direto de rotinas de Xlib; todos os outros módulos utilizam exclusivamente XView.

Módulo de Gerência de Textos (*MGT*)

As janelas de texto GOODIES são utilizadas com dois fins: a apresentação de métodos de uma classe e a apresentação de atributos textuais. A funcionalidade da janela, em ambos os casos, é semelhante. No entanto, os valores dos atributos textuais a serem apresentados é obtido dos valores de objetos, enquanto as definições de métodos fazem parte da descrição das classes. O MGT controla os dois tipos de valores representados na janela de texto.

De acordo com a origem da chamada, o MGT descobre se o nome de arquivo recebido por ele representa um método ou um atributo textual. A diferenciação entre os tipos de texto representados é necessária por diversas razões. A principal delas é que janelas de texto que representam métodos são subordinadas às suas respectivas janelas de classe, ao passo que as janelas de texto que representam atributos textuais são subordinadas às suas respectivas janelas de objeto. Como já foi visto anteriormente, as janelas subordinadas são destruídas se a janela superior na hierarquia de janelas é eliminada. Portanto, ao se eliminar uma janela de objeto, deseja-se eliminar todas as janelas de texto que representam atributos da janela de objeto, mas não as janelas que representam métodos da classe a que o objeto pertence.

Módulo de Gerência de Predicados (*MGP*)

O MGP trata da janela de predicado associada a cada janela de objeto. A ativação da opção *"Predicate"* do menu *"Props"* da janela de objeto, faz com que o MGP assuma o controle do sistema. Se a janela de predicado já existe para a janela de objeto (ou seja, se o usuário já havia ativado a opção *"Predicate"* desta janela), ela é apresentada ao usuário. Caso contrário, o MGP cria a janela de predicado, subordinando-a à janela de objeto.

A janela de predicado contém uma subjanela de edição, onde o usuário pode formular

o predicado desejado, e três botões de controle (*"Apply"*, *"Save"* e *"Load"*). O botão *"Apply"* faz com que o MGP analise o predicado definido na subjanela de edição. Se o predicado foi corretamente formulado, ele é enviado pelo MGP ao Módulo de Operações Primitivas, onde o predicado é convertido para a sintaxe do SGBDOO e associado à operação *Get-Object*. Os dois outros botões causam a chamada do Módulo de Gerência de Arquivos.

Módulo de Gerência de Arquivos (*MGA*)

O MGA recebe o controle quando o usuário seleciona as opções *"Load"* ou *"Save"* na janela de predicado. A funcionalidade da janela de arquivo, criada pelo MGA, é análoga à da janela de diretório. O usuário navega pelos diretórios em busca do arquivo desejado. Ao encontrar este arquivo, o usuário seleciona o botão correspondente à opção ativada na janela de predicado (*"Save"* ou *"Load"*).

A opção *"Save"* faz com que o MGA armazene o conteúdo da subjanela de edição da janela de predicado no arquivo indicado pelo usuário, criando-o, caso ele não exista.

Na opção *"Load"* o MGA faz uma cópia do arquivo indicado para a subjanela de edição da janela de predicado. O usuário pode alterar o predicado carregado, ou pode associá-lo diretamente à janela de objeto, através da opção *"Apply"* da janela de predicado.

A capacidade de salvar a definição de um predicado e posteriormente reutilizá-la, associando o predicado a uma janela de objeto é um ponto forte de GOODIES. Esta capacidade provê suporte ao conceito de *visão* em BDOOs, de acordo com a definição de visão proposta em [MM91].

Módulo de Gerência de Elementos (*MGE*)

O MGE é responsável pela funcionalidade da janela de atributos. Esta janela é utilizada em GOODIES para quatro diferentes fins: 1) escolha de atributos a serem apresentados em janelas de objeto; 2) escolha de atributos a serem apresentados em janelas de classe; 3) escolha de superclasses para a formação da composição de uma classe; 4) escolha de subclasses cujas instâncias devem ser visualizadas na lista de objetos de uma classe. No primeiro caso, a janela de atributo é subordinada a uma janela de objeto; nos três outros casos aquela janela se subordina a uma janela de classe.

A tarefa do MGE é identificar o tipo de janela ao qual ele deve subordinar a janela de atributos, identificando ainda o modo de ativação, se o MGE foi ativado a partir de uma janela de classe. Identificados a janela e o modo de ativação, o MGE cria a janela de

atributos que contém, em todos os casos, uma lista de elementos e um botão *"Apply"*. Por *default*, todos os elementos, nos quatro casos, aparecem inicialmente selecionados, indicando que GOODIES está apresentando o máximo nível de detalhamento das informações. O usuário pode, por exemplo, reduzir este nível de detalhamento até encontrar um objeto de interesse, voltando a selecionar todos os elementos do objeto, a fim de observar seus diferentes aspectos. Cabe observar que o MGE não atua sobre as janelas de classe ou de objeto. Ele apenas informa ao MGC e ao MGO, respectivamente, sobre as alterações efetuadas pelo usuário nas janelas de atributos correspondentes.

6.4.2 Módulo de Interação com o SGBDOO (*MIS*)

O MIS é o módulo de GOODIES que implementa o seu modelo interno. Neste módulo está contido o código dependente do SGBDOO utilizado. O MIS é composto por dois submódulos fortemente interligados. Um deles recebe pedidos de informações dos demais módulos do sistema e traduz estes pedidos, formulando consultas que utilizam a sintaxe da linguagem de consulta do SGBDOO. O segundo módulo recebe a resposta textual do SGBDOO, e converte os resultados da consulta efetuada para estruturas internas de GOODIES. Como ambos os submódulos são dependentes do SGBDOO utilizado, a discussão a seguir está baseada na integração de GOODIES com o sistema O2 [O2 92].

Módulo de Operações Primitivas (*MOP*)

Como foi visto, o modelo interno de GOODIES define a semântica de três operações primitivas: *Get-Schema*, *Get-Class* e *Get-Object*. Estas operações são suficientes para especificar todas as informações que GOODIES precisa obter do SGBDOO. Em O2, estas operações seriam escritas da seguinte forma:

- *Get-Schema*(**nome-de-BD**):

  set base **nome-de-BD**: estabelece o esquema e a base de dados a serem utilizadas;

  display classes: apresenta a hierarquia de classes do esquema corrente.

- *Get-Class*(**nome-de-BD**, **nome-de-classe**):

  set base **nome-de-BD**;

  display class **nome-de-classe**: apresenta o tipo (composição) da classe;

display methods in **nome-de-classe**: apresenta os métodos definidos para a classe;

select x from x in **nome-de-classe**: retorna um conjunto de ponteiros (oids) para objetos que são instâncias da classe.

- *Get-Object*(**nome-de-BD**, **nome-de-classe**, **oid**):

  set base **nome-de-BD**;

  select **composição** *from x in* **nome-de-classe** [*where* **predicado**]: no campo **composição** da consulta devem ser especificados os atributos selecionados para visualização na janela de objeto. Este tipo de consulta retorna um conjunto de objetos, e o parâmetro **oid** é utilizado para selecionar o objeto desejado neste conjunto. A cláusula *where* da consulta é opcional, e só é utilizada quando a janela de objeto possui um predicado associado.

Módulo de Interpretação de Resultados (*MIR*)

Uma vez efetuada a consulta ao SGBDOO, o MOP passa o controle do sistema para o MIR, indicando a consulta efetuada. O MIR se mantém em estado de espera até que o SGBDOO forneça uma resposta à consulta efetuada. Esta resposta é recebida como um texto, contendo delimitadores específicos entre os diversos valores que podem estar contidos na resposta. O MIR analisa o texto recebido em função da consulta efetuada pelo MOP, identificando delimitadores e atualizando as estruturas de dados internas de GOODIES com os valores apropriados provenientes do texto analisado.

No caso de utilização do sistema O2, as operações primitivas são convertidas nas consultas apresentadas anteriormente, e os resultados são fornecidos no seguinte formato:

Valores atômicos : cada linha do texto de resposta contém um único valor atômico, ou seja, valores atômicos são delimitados por *"line feeds"*;

Tuplas : os valores dos campos do tipo tupla são precedidos pela palavra-chave *"tuple"*. Cada atributo da tupla aparece em uma linha separada (valor atômico), e cada linha contém o nome e o valor do atributo, separados por dois pontos (":"). Parênteses marcam o início e o final dos atributos de uma tupla;

Listas e Conjuntos : para estes tipos de campos a resposta se inicia com as palavras-chave *"list"* e *"set"*, respectivamente. A seguir são apresentados os valores dos elementos, em linhas separadas. Parênteses marcam o início e o final dos elementos de um conjunto ou lista;

Sub-objetos : campos que representam referências a outros objetos são enviados entre colchetes. O valor enviado é o nome da classe a que pertence o objeto referenciado.

6.4.3 Módulo de Controle Geral (*MCG*)

GOODIES é um sistema de interface entre usuário e SGBDOO. Por essa razão, a maior parte de seu código é destinada à gerência da interação entre o sistema e cada uma das partes dialogadoras (usuário e SGBDOO). No entanto, a interação com cada parte deve ser feita de modo coordenado, e para exercer esta coordenação foi desenvolvido o MCG. Este módulo define o ambiente inicial de uma sessão de trabalho em GOODIES, estabelece a conexão com o servidor X, e provê funções de uso geral. Os dois submódulos que compõem o MCG são discutidos a seguir.

Módulo de Auto-Instalação (*MAI*)

O MAI é o primeiro módulo executado ao se ativar GOODIES. O trabalho realizado por este módulo consiste em verificar a presença do arquivo de configuração do sistema (arquivo *~/goodies.init*), e instalar uma sessão de trabalho. Se o arquivo de configuração não existe, o MAI executa três passos para instalar uma sessão de trabalho: determinar o valor de algumas variáveis globais (como por exemplo os valores de chave das estruturas *XV_KEY_DATA*), chamar o MGD para criar a janela de diretório, e estabelecer a conexão com o servidor X.

No caso de o arquivo de configuração estar presente, o MAI verifica se o arquivo está no formato utilizado por GOODIES, já que um arquivo do usuário pode ter, por coincidência, o mesmo nome. Se o arquivo é realmente o arquivo de configuração de GOODIES, isto significa que o usuário utilizou a função de salvamento de contexto em uma sessão anterior. O MAI executa os dois primeiros passos para o estabelecimento da sessão de trabalho, mas antes de estabelecer a conexão com o servidor X, ele faz chamadas ao MGB e ao MGC, solicitando a criação das janelas salvas no arquivo de configuração.

Pelo discussão acima, observa-se que pelo menos uma janela é criada no início de uma sessão de trabalho (a janela de diretório). Uma vez criadas as janelas, o MAI estabelece a conexão com o servidor X, através da chamada à função *xv_main_loop()* de XView. O módulo *Notifier* de XView passa então a receber, processar e propagar eventos X para GOODIES. A chamada de módulos, a partir da inicialização do *Notifier*, é controlada pelos eventos por ele recebidos do protocolo X. Cabe lembrar, entretanto, que um módulo ativado por um evento (o pressionamento de um botão, por exemplo) pode chamar diretamente outros módulos do sistema.

Módulo de Funções Genéricas (*MFG*)

Existem algumas funções de GOODIES que são utilizadas por diversos módulos do sistema. O MFG contém estas funções, dentre as quais se destacam as funções para processamento de listas.

Muitas janelas de GOODIES possuem componentes representados por listas de tamanho variável (*scrolling lists*). Estas listas suportam quantidades arbitrárias de elementos, onde cada elemento é representado por uma cadeia de caracteres. Ao invés de implementar funções para manipulação de listas em cada módulo do sistema, optou-se por reunir estas funções no MFG. As principais funções oferecidas pelo MFG para manipulação de listas são:

- Seleção dos elementos de uma lista;
- Inserção de um elemento em uma lista;
- Remoção de um elemento de uma lista;
- Remoção de todos os elementos de uma lista;
- Recuperação do elemento que ocupa a *i-ésima* posição de uma lista;
- Recuperação da posição de um elemento em uma lista.

Todas estas funções recebem como um de seus parâmetros o *"handle"* (identificador de objeto de interface gerado por XView) da lista a ser manipulada.

6.5 Técnicas de Projeto Orientado a Objetos

O paradigma de orientação a objetos é uma presença constante no sistema GOODIES. Em primeiro lugar, o sistema em si é uma interface para SGBDs que utilizam este paradigma nos seus modelos de dados. Além disso, o sistema foi desenvolvido com base em duas ferramentas orientadas a objetos: C++ e XView. Assim, as técnicas de projeto orientado a objetos foram largamente utilizadas em GOODIES, e possibilitaram o desenvolvimento de um sistema razoavelmente complexo, em um período relativamente curto, por um único programador.

O primeiro benefício conseguido com o uso de uma linguagem orientada a objetos foi a construção de funções independentes, as quais podem ser facilmente reutilizadas. O

Módulo de Gerência de Arquivos, por exemplo, pode ser utilizado por qualquer aplicação X que necessite de uma ferramenta para navegação em diretórios. O GEOLAB, um dos projetos recentemente desenvolvidos no DCC[2], faz uso efetivo deste módulo.

Durante o desenvolvimento de GOODIES, a facilidade de experimentação provida por C++ foi muito importante. Novas classes de objetos puderam ser inseridas, sem afetar a funcionalidade das classes já existentes, devido ao encapsulamento do comportamento das mesmas. Além disso, novas funcionalidades puderam ser obtidas de classes já existentes, através do mecanismo de herança. As janelas de texto, usadas para representação de métodos e de atributos textuais, fazem uso do mecanismo de herança de C++ para obter esta dupla funcionalidade.

Em um sistema orientado a objetos, diz-se que a interface de uma classe descreve as funções suportadas pela classe, enquanto a implementação da classe define como estas funções trabalham. A utilização de interfaces idênticas, com diferentes implementações aplicadas a classes distintas, recebe o nome de polimorfismo. A utilização de polimorfismo tornou o conjunto de funções de cada módulo do sistema bastante homogêneo. Por exemplo, todos as classes definidas em GOODIES para manipulação de janelas possuem funções, com interfaces idênticas, para obter as janelas subordinadas. O polimorfismo foi especialmente útil quando houve a necessidade de alteração da funcionalidade de algumas dessas classes.

A sobreposição (*overloading*) de funções, isto é, a utilização de um mesmo nome de função com diferentes tipos de parâmetros em uma única classe, foi utilizada no desenvolvimento de GOODIES com bons resultados. A função que atualiza os valores dos atributos de uma janela de objeto após uma operação de seqüenciamento age de modo diferente ao receber cada tipo de atributo (simples, texto, imagem, som, tupla, lista ou objeto). As funções que criam as janelas também deveriam ser sobrepostas, para atender à possibilidade de receber ou não a posição da janela como parâmetro. Todavia isto não foi necessário, porque C++ provê facilidades para uso de valores *default* em parâmetros de funções.

[2]Departamento de Ciência da Computação da Universidade Estadual de Campinas.

Capítulo 7

Conclusões

7.1 Considerações Finais

Esta dissertação abordou a construção de interfaces gráficas aplicadas a SGBDOOs. Foi apresentado um novo sistema de interface, o sistema GOODIES, que provê facilidades para navegação sobre os esquemas e sobre os dados gerenciados por um SGBDOO. Além de oferecer um poderoso mecanismo de navegação, que chega a ser similar a um processo de consulta, GOODIES oferece uma grande vantagem sobre outros sistemas de interface existentes, que é a independência com relação ao SGBDOO utilizado.

A versão final do sistema GOODIES contém cerca de quatorze mil linhas de código em C++, incluindo as definições dos componentes gráficos da interface. A experiência de desenvolver um sistema de interface com o poder de GOODIES em um período aproximado de um ano, e utilizando um único programador, foi possível por duas razões. Primeiro devido à utilização de técnicas de projeto orientado a objetos, notadamente dos mecanismos de herança, sobreposição e polimorfismo. Segundo porque o projeto do sistema foi ao mesmo tempo conciso, claro e abrangente, servindo como um guia seguro para a implementação dos programas.

O objetivo de desenvolver GOODIES não foi a produção de um sistema de interface pronto para ser utilizado (apesar de a versão final do sistema ter atingido este ponto), mas sim a validação de dois aspectos que envolvem a base teórica desta tese. Um dos aspectos validados pela implementação de GOODIES foi a definição do conjunto de características fundamentais do modelo de dados OO, estabelecida no capítulo 2 desta dissertação. O segundo aspecto foi a comprovação da utilidade e da possibilidade de aplicação prática das diretivas para projeto de interfaces gráficas para SGBDs propostas no capítulo 3 desta

dissertação.

São apresentadas, a seguir, as conclusões obtidas do trabalho realizado durante a elaboração desta tese. A seção 7.2 faz uma análise crítica do projeto de pesquisa desenvolvido. Na seção 7.3 o sistema GOODIES é comparado a outros sistemas de interface existentes para SGBDs. Em seguida são discutidas as contribuições alcançadas com a presente tese, e a seção 7.5 encerra esta dissertação, propondo extensões que podem melhorar o sistema desenvolvido.

7.2 Avaliação do Projeto

O primeiro desafio do projeto de pesquisa envolvido nesta tese foi o dimensionamento do sistema a ser desenvolvido. Dois interesses conflitantes contribuíram para tornar mais difícil este desafio. Por um lado, o tempo e o número de programadores disponíveis exigiam que o sistema fosse trivial. Por outro lado, o estudo preliminar das características de interfaces gráficas para SGBDOOs demonstrava que um sistema muito simples não permitiria a análise e a experimentação dos diversos componentes envolvidos.

Para equilibrar as duas demandas, optou-se por projetar um sistema de interface que abordasse apenas uma tarefa entre as diversas possíveis em um SGBD, mas que oferecesse um bom suporte à execução desta tarefa específica. GOODIES provê acesso apenas para navegação sobre os BDs, mas o mecanismo de navegação desenvolvido pode ser comparado aos utilizados nos melhores *browsers*[1] existentes na atualidade. A dimensão final do projeto de GOODIES conseguiu atender aos objetivos de pesquisa envolvidos, utilizando apenas os recursos humanos e de tempo alocados inicialmente.

A partir do estabelecimento do escopo do sistema, teve início a fase de projeto propriamente dita. Esta fase envolveu inicialmente o estudo de sistemas de bancos de dados e de sistemas de interface em geral. Um resumo dos resultados desse estudo inicial pode ser encontrado na introdução desta dissertação. Em seguida foram abordados temas mais intimamente ligados com o projeto de pesquisa, que foram os SGBDOOs e os sistemas de interface para SGBDs, apresentados, respectivamente, nos capítulos 2 e 3 desta dissertação.

Seguindo na fase de projeto, o conhecimento adquirido dos estudos realizados foi utilizado para produzir as definições dos modelos externo e interno do sistema GOODIES. O modelo externo, apresentado no capítulo 4 desta dissertação, estabelece as normas de interação entre usuário e interface. Já o modelo interno, discutido no capítulo 5, rege a interação entre o SGBDOO e o sistema de interface. Os dois modelos são o principal

[1] Ferramentas para navegação em Sistemas de Bancos de Dados.

produto da fase de projeto. Foram eles que orientaram a implementação dos módulos de GOODIES, determinando seus relacionamentos e suas funcionalidades.

É conveniente ressaltar ainda a importância do estudo dos sistemas já existentes. Este estudo evitou a repetição de erros de projeto ocorridos nestes sistemas, e indicou alguns pontos básicos adotados por todos eles. Por essa razão, o projeto final de GOODIES foi obtido através de alguns poucos refinamentos sobre o projeto original.

De acordo com a discussão efetuada no capítulo 6, a implementação do projeto constituise, basicamente, em um mapeamento direto da funcionalidade descrita pelo modelo externo para programas escritos em C++. Já o modelo interno de GOODIES não pôde ser tão diretamente implementado, devido aos aspectos dependentes do SGBD. No entanto, o modelo interno ofereceu uma estrutura para a organização de um módulo de GOODIES que se dedica à implementação destas dependências.

Com o estudo realizado e com a implementação do sistema GOODIES ficaram comprovadas diversas hipóteses formuladas ao longo do projeto de pesquisa desenvolvido, entre as quais pode-se destacar: a existência de um conjunto básico de características comuns aos diversos SGBDOOs existentes; a eficiência obtida com o uso das diretivas que orientam o projeto de interfaces, em oposição à abordagem de projeto *"ad hoc"*; a possibilidade de construção de um sistema de interface independente do SGBDOO utilizado; e a importância das técnicas de projeto orientado a objetos para o desenvolvimento e manutenção de sistemas não triviais.

7.3 Comparação com Outros Trabalhos

O sistema GOODIES não pode ser diretamente comparado aos sistemas QBE ([EN89]), SDMS ([Her80]), PICASSO ([KKS88]), PASTA-3 ([KM90]) e à interface descrita por [RC88], pois esses sistemas não se direcionam a modelos que enfrentam o problema de representação de objetos complexos.

Já os sistemas ISIS ([GGKZ85]), SNAP ([BH86]) e SIG ([MNG86]) são interfaces para modelos semânticos e OO, que permitem a definição de objetos complexos. Os sistemas SNAP e SIG não suportam navegação sobre esquemas e dados, e portanto, deixam a desejar. Esta deficiência aparece também no sistema GS DESIGNER ([Alm91]). O sistema ISIS suporta este tipo de navegação, mas não suporta o conceito de herança múltipla, que é muito importante no modelo OO. O sistema apresentado nesta dissertação permite navegação sobre dados e esquemas dos BDs, e suporta herança múltipla das classes do esquema.

A representação de atributos, objetos e classes em um só diagrama, proposta no sistema ZOO ([Rie87]) é impraticável para sistemas que possuem uma quantidade razoável de classes ou objetos complexos. A abordagem adotada em GOODIES provê a visualização dos componentes de um esquema em janelas separadas. Este tipo de abordagem é adotada pela maior parte dos trabalhos existentes, e parece ser a mais adequada para o modelo OO.

O sistema KIVIEW ([MDT89]) apresenta muitas características positivas, entre as quais pode-se destacar a navegação sobre dados e esquemas, a navegação sincronizada e o armazenamento de resultados do processo de navegação. O sistema ODEVIEW ([AGS90]) possui características semelhantes ao KIVIEW, mas não possui a facilidade de armazenar resultados de navegações. Entretanto, duas importantes características do sistema GOODIES não estão presentes nestes sistemas: o suporte para a visualização de métodos, e a seleção de objetos através da especificação de predicados no processo de navegação.

O sistemas GOOD ([PBA+92]) e FACEKIT([KN92]) representam novas tendências de representação do modelo OO. GOOD adota a representação de esquemas e dados através de grafos. Este tipo de representação é superior àquela utilizada em GOODIES somente para esquemas extremamente simples. Em um esquema real, a complexidade da representação utilizada em GOOD seria excessiva. Já a comparação entre FACEKIT e GOODIES é mais complicada, pois FACEKIT não é um sistema de interface para navegação, e sim para construção de representações de objetos. Todavia existe uma deficiência de FACEKIT – a dependência de um SGBDOO específico – que torna o projeto deste sistema mais restrito do que o projeto de GOODIES.

Um excelente mecanismo de visualização de objetos é proposto pelos sistemas LOOKS ([Alt90a, Alt90b]) e TOONMAKER ([Mam91]). No entanto, estes sistemas não suportam visualização de esquemas. O sistema OOPE ([Alt90c]), ao contrário, suporta manipulação de esquema, mas não provê um mecanismo adequado para navegação sobre objetos de um BD. Os três sistemas, em conjunto, formam o sistema de interface do banco de dados O2. Esta interface é superior àquela proposta nesta dissertação, pois permite atualização de dados, e não somente navegação e consulta. Entretanto, a facilidade de definição de predicados nas janelas de objetos, e a capacidade de sincronização entre objetos tornam o mecanismo de navegação de GOODIES mais poderoso que o do sistema O2.

A tabela 7.1 sintetiza as tabelas 3.1 e 3.2, acrescentando uma linha onde aparece o sistema GOODIES. Cada linha da tabela 7.1 reflete a presença das características básicas de um sistema de interface para SGBDs (identificadas no capítulo 3) em cada um dos sistemas apresentados.

Sistema	Características									
	1	2	3	4	5	6	7	8	9	10
QBE	≈			≈	≈	√	≈	√	√	
SDMS	≈	√	≈	√			≈	≈	√	
PICASSO	√	√	√	√		√		≈		≈
ISIS	≈	≈	≈	≈	≈	√	≈	√		
SNAP	≈	≈	√	√			≈	√		
SCHEMADESIGN e DATABROWSE	√	√	≈	√		√	≈	√	√	≈
KIVIEW	≈	≈	≈	√		√	√	√	√	≈
PASTA-3	√	≈	≈	√	√	≈	≈	√		≈
SIG	√	√	≈	√		√	≈	√	√	
LOOKS, TOONMAKER e OOPE	√	√	√	√	√	√	≈		√	≈
ODEVIEW	√	√	≈	√		√	≈	√	√	
GS DESIGNER	≈	√	≈	√			√	√	√	
GOOD		≈		√	√	≈	√	√	√	
FACEKIT	√	√	≈	√			√	√	√	
GOODIES	√	√	≈	√		√	√	√	√	√
Legenda	√ - Característica Plenamente Presente ≈ - Característica Parcialmente Presente □ - Característica Ausente									

Tabela 7.1: Comparação das Características: GOODIES χ outros sistemas

7.4 Principais Contribuições

O sistema de interface desenvolvido durante o projeto de pesquisa representa uma contribuição, tanto para a área de bancos de dados orientados a objetos quanto para a área de interfaces com o usuário. Acredita-se que o sistema GOODIES possa ser utilizado na prática, aumentando a eficiência nos processos de navegação sobre BDOOs. Espera-se que a utilização de GOODIES em substituição às tradicionais linguagens de consulta torne os BDs disponíveis para um número consideravelmente maior de usuários, notadamente para aqueles que não possuem conhecimentos sobre linguagens de consulta. É importante, todavia, realizar experimentos que comprovem estas expectativas.

Em contrapartida, algumas contribuições apresentadas por esta dissertação já estão comprovadas. Entre essas contribuições pode-se destacar:

- A definição de um conjunto básico de características apresentadas pelos SGBDOOs atuais. Este conjunto de características foi obtido através da análise comparativa de diversas definições do modelo de dados OO, e pelo estudo dos modelos de dados de alguns dos principais SGBDOOs existentes;
- A identificação de dez características básicas de sistemas de interfaces para SGBDs, que levaram à definição de dez diretivas para a construção deste tipo de interface. Estas diretivas servem, não apenas para orientar novos projetos, mas também como termo de comparação entre sistemas existentes;
- O estudo e a utilização prática de técnicas de projeto orientado a objetos no desenvolvimento de um sistema de interface. As conclusões obtidas devem incentivar a utilização destas técnicas em outros projetos.

Uma última contribuição esperada é a motivação para novos estudos envolvendo SGBDOOs e interfaces gráficas. Essas duas áreas de pesquisa são bastante recentes, e necessitam de um grande volume de trabalhos para explorar todo o seu potencial.

7.5 Possíveis Extensões e Melhoramentos

Apesar de o sistema GOODIES ser hoje uma ferramenta pronta para uso, ele pode ser estendido e melhorado em vários aspectos. A primeira possibilidade de extensão seria o acréscimo de funções para atualização de BDs. Um sistema com as facilidades de navegação de GOODIES e com a capacidade de atualização de dados e de esquemas pode ser considerado um sistema completo de interface para SGBDOOs.

Outra extensão desejável seria a introdução da capacidade de representação gráfica de esquemas. É um costume consagrado entre usuários de BDs a representação de esquemas através de diagramas de blocos, e este tipo de representação não é factível, atualmente, em GOODIES. Vale observar que o projeto do sistema foi preparado para receber estas duas extensões, através de modificações mínimas. Também o código do sistema, escrito em C++, foi desenvolvido com a preocupação de ser extensível.

Um aspecto em que o sistema GOODIES precisa ser melhorado se refere ao auxílio ao usuário. Apesar de possuir mecanismos intuitivos, pode ocorrer que o usuário tenha alguma dúvida sobre o funcionamento de uma operação específica. GOODIES não provê funções que permitam ao usuário obter ajuda sobre este tipo de dúvida. No entanto o sistema incentiva a aprendizagem por experimentação, já que não existem operações de GOODIES que não possam ser revertidas. Ainda assim, a implementação de mensagens de auxílio, conforme a recomendação OPENLOOK [Sun90], seria um importante fator de melhoria para GOODIES.

Por fim, GOODIES pode ser melhorado em questão de desempenho. As estruturas de dados e os algoritmos que as manipulam foram escolhidos principalmente sob o ponto de vista da simplicidade. É possível melhorar o desempenho do sistema, pela substituição das estruturas e dos algoritmos utilizados atualmente por outros mais sofisticados e eficientes. Vale ressaltar, no entanto, que o ganho de desempenho dificilmente será notado em termos de tempo de resposta, pois o processamento interno do sistema é muito menor que aquele realizado pelo SGBD para responder às consultas. Além disso, considerando-se uma situação de uso típica, onde o usuário manipula um número relativamente pequeno de janelas, as estruturas usadas atualmente apresentam desempenho satisfatório.

Índice

T

V

X

Bibliografia

[ABD+89] Malcolm Atkinson, François Bancilhon, David DeWitt, Klaus Dittrich, David Maier, and Stanley Zdonik. The Object-Oriented Database System Manifesto. In *Proceedings of the First International Conference on Deductive and Object-Oriented Databases*, pages 40–57, Kyoto, Japan, December 1989.

[AG89] R. Agrawal and N. Gehani. ODE (Object Database and Environment): The Language and the Data Model. In *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data*, pages 36–45, Portland, USA, June 1989.

[AGS90] R. Agrawal, N. Gehani, and J. Srinivasan. Odeview: The Graphical Interface to Ode. In *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data*, pages 34–43, Atlantic City, USA, May 1990.

[Alm91] Jay Almarode. Issues in the Design and Implementation of a Schema Designer for an OODBMS. In *ECOOP'91 Proceedings*, pages 200–218, July 1991.

[Alt90a] Altaïr. The Looks Programmer Manual. Technical Report, Gip Altaïr, January 1990. Printing Revision 1.1, 9/01/1990.

[Alt90b] Altaïr. The Looks User's Manual. Technical Report, Gip Altaïr, January 1990. Printing Revision 1.1, 9/01/1990.

[Alt90c] Altaïr. OOPE: The Object-Oriented Programming Environment. Technical Report, Gip Altaïr, January 1990. Printing Revision 1.1, 9/01/1990.

[BCG+87] Jay Banerjee, Hong-Tai Chou, Jorge Garza, Won Kim, Darrell Woelk, Nat Balou, and Hyoung-Joo Kim. Data Model Issues for Object-Oriented Applications. *ACM Transactions on Office Information Systems*, 5(1):3–26, 1987.

[BH86] Daniel Bryce and Richard Hull. Snap:A Graphics-based Schema Manager. In *IEEE Proceedings of the International Conference on Data Engineering*, Los Angeles, USA, February 1986.

[BM91] Elisa Bertino and Lorenzo Martino. Object-Oriented Database Management Systems: Concepts and Issues. *IEEE Computer*, 24(4):33–47, April 1991.

[BMP+92] P. Borras, J. C. Mamou, D. Plateau, B. Poyet, and D. Tallot. Building User Interfaces for Database Applications: The O2 Experience. *SIGMOD Record*, 21(1):32–38, March 1992.

[BOS91] Paul Butterworth, Allen Otis, and Jacob Stein. The Gemstone Object Database Management System. *Communications of the ACM*, 34(10):64–77, October 1991.

[BR92] John E. Bell and Lawrence A. Rowe. An Exploratory Study of Ad Hoc Query Languages to Databases. *IEEE Proceedings of the International Conference on Data Engineering*, pages 606–613, February 1992.

[Cat91] R. G. G. Cattell. Next-generation Database Systems. *Communications of the ACM*, 34(10):30–33, October 1991.

[Che76] Peter Pin-Shan Chen. The Entity-Relationship Model — Toward a Unified View of Data. *ACM Transactions on Database Systems*, 1(1):9–36, 1976.

[CM84] George Copeland and David Maier. Making SmallTalk a Database System. In *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data*, pages 316–325, Boston, USA, June 1984.

[Com90] The Committee for Advanced DBMS Function. Third-Generation Database System Manifesto. *SIGMOD Record*, 19(3):31–44, September 1990.

[CRR91] P. K. Chrysanthis, S. Raghuran, and K. Ramamritham. Extrating Concurrency from Objects: A Methodology. In *Proceedings of the 1991 ACM SIGMOD International Conference on Management of Data*, May 1991.

[Dat86] C. J. Date. *Introdução a Sistemas de Bancos de Dados.* Editora Campus Ltda, Rio de Janeiro, 1986. Terceira Edição.

[Deu90] O. Deux. The Story of O_2. *IEEE Transactions on Knowledge and Data Engineering*, 2(1):91–108, March 1990.

[Deu91] O. Deux. The O_2 System. *Communications of the ACM*, 34(10):34–48, October 1991.

[EN89] Ramez Elmasri and Shamkant Navathe. *Fundamentals of Database Systems.* The Benjamin/Cummings Publishing Company, 1989.

[FM92] Berlinda Flynn and David Maier. Suporting Display Generation for Complex Database Objects. *SIGMOD Record*, 21(1):18–24, March 1992.

[GGKZ85] Kenneth J. Goldman, Sally A. Goldman, Paris C. Kanellakis, and Stanley B. Zdonik. ISIS: Interface for a Semantic Information System. In *Proceedings of the 1985 ACM SIGMOD International Conference on Management of Data*, Austin, USA, May 1985.

[Gim90] Rainer Gimnich. Implementing Direct Manipulation Query Languages Using an Adequate Data Model. *Lecture Notes in Computer Science*, 1(439):227–240, 1990.

[Hel90] Dan Heller. *XVIEW Programming Manual*, volume 7 of *The X Window System Series*. O'Reilly & Associates, April 1990. Second Printing.

[Her80] Christopher F. Herot. Spatial Management of Data. *ACM Transactions on Database systems*, 5(4):493–513, December 1980.

[HH89] H. Rex Hartson and Deborah Hix. Human-Computer Interface Development: Concepts and Systems for its Management. *ACM Computing Surveys*, 21(1):5–92, March 1989.

[HK87] Richard Hull and Roger King. Semantic Database Modeling: Survey, Applications, and Research Issues. *ACM Computing Surveys*, 19(3):201–260, September 1987.

[HM81] Michael Hammer and Dennis McLeod. Database Description with SDM: A Semantic Database Model. *ACM Transactions on Database Systems*, 3(6):351–386, 1981.

[Jac91] Mike S. Jackson. Tutorial on object-oriented databases. *Information And Software Technology*, 33(1):4–12, January 1991.

[Kat90] Randy H. Katz. Toward a Unified Framework for Version Modeling in Engineering Databases. *ACM Computing Surveys*, 22(4):375–408, December 1990.

[KKS88] Hyoung-Joo Kim, Henry F. Korth, and Avi Silberschatz. Picasso: A Graphical Query Language. *Software Practice and Experience*, 18(3):169–203, March 1988.

[KM90] Michel Kuntz and Rainer Melchert. Ergonomic Schema Design and Browsing with More Semantics in the Pasta-3 Interface for E-R DBMSs. In F. H. Lochovsky, editor, *Entity-Relationship Approach to Database Design and Querying*, pages 419–433. Elsevier Science Publishers B.V. (North-Holland), 1990.

[KN92] Roger King and Michael Novak. Building Reusable Data Representations with Facekit. *SIGMOD Record*, 21(1):11–17, March 1992.

[Kun92] Michel Kuntz. The Gist of GIUKU: Graphical Interactive Intelligent Utilities for Knowledgeable Users of Database Systems. *SIGMOD Record*, 21(1):58–64, March 1992.

[LF91] Ernst Lippe and Gert Florijn. Implementation Techniques for Integral Version Management. In *ECOOP'91 Proceedings*, pages 342–359, July 1991.

[LLOW91] Charles Lamb, Gordon Landis, Jack Orenstein, and Dan Weinreb. The ObjectStore Database System. *Communications of the ACM*, 34(10):50–63, October 1991.

[LLPS91] Guy Lohman, Bruce Lindsay, Hamid Pirahesh, and K. Bernhard Schiefer. Extensions to StarBurst: Objects, Types, Functions, and Rules. *Communications of the ACM*, 34(10):94–109, October 1991.

[Mam91] Jean-Claude Mamou. *Du Disque à le ecran: Genération D'Interfaces Homme–Machine Pour Objects Persistants.* PhD thesis, Université de Paris - Sud - Centre d'Orsay, May 1991.

[MDT89] Amihai Motro, Alessandro DÁtri, and Laura Tarantino. The Design of KIVIEW: An Object-Oriented Browser. In Larry Kerschberg, editor, *Proceedings of the Second International Conference on Expert Database Systems*, pages 107–131. The Benjamin/Cummings Publishing Company, Inc., 1989.

[ME92] Priti Mishra and Margaret Eich. Functional Completeness in Object-Oriented Databases. *SIGMOD Record*, 21(1):71–83, March 1992.

[MM90] Jean-Claude Mamou and Claudia Bauzer Medeiros. Visões em Sistemas Orientados a Objetos: Problemas e Implementação. In *Conferência Latino-Americana de Informática*, Assunção, Paraguai, 1990.

[MM91] Jean-Claude Mamou and Claudia Bauzer Medeiros. Interactive Manipulation of Object-Oriented Views. In *IEEE Proceedings of the International Conference on Data Engineering*, pages 60–69, Kobe, Japan, April 1991.

[MNG86] David Maier, Peter Nordquist, and Mark Grossman. Displaying Database Objects. In *Proceedings of the First International Conference on Expert Database Systems*, pages 15–30, April 1986.

[Mra91] Hamid El Mrabet. Outils de Generation de Interfaces: Etat de La Art et Classification. Technical report, Institut National de Recherche en Informatique et en Automatique, February 1991. Rapport Techiniques 126.

[MSB90] John Alan McDonald, Werner Stuetzle, and Andreas Buja. Painting Multiple Views of Complex Objects. In *OOPSLA'90 Proceedings*, pages 245–257, Ottawa, Canada, October 1990.

[MvD91] Aaron Marcus and Andries van Dam. User-Interface Developments for the Nineties. *IEEE Computer*, 24(9):49–57, September 1991.

[Nor91] Kent L. Norman. Models of the Mind and Machine: Information Flow and Control between Humans and Computers. *Advances in Computers*, 32(1):201–254, 1991.

[O2 92] O2 Technology. *The O2 User's Manual*, April 1992. Version 3.3.

[OA92] Juliano Lopes de Oliveira and Ricardo de Oliveira Anido. Browsing and Querying in Object Oriented Databases. Technical Report 12/92, Universidade Estadual de Campinas - Departamento de Ciência da Computação, December 1992.

[Oli93] Ronaldo Lopes de Oliveira. Transparência de Modelos de Dados em Sistemas de Bancos de Dados Heterogêneos. Master's thesis, Universidade Estadual de Campinas - Departamento de Ciência da Computação, 1993. To be published.

[PBA+92] Jan Paredaens, Jan Van Den Bussche, Marc Andries, Marc Gemis, Marc Gyssens, Inge Thyssens, Dirk Van Gucht, Vijay Sarathy, and Lawrence Saxton. An Overview of GOOD. *SIGMOD Record*, 21(1):25–31, March 1992.

[PH91] Thiagarajan Palanivel and Martin Helander. Human-Factors Issues in Dialog Design. *Advances in Computers*, 33(1):115–171, 1991.

[PM88] Joan Peckham and Fred Maryanski. Semantic Data Models. *ACM Computing Surveys*, 20(3):153–189, September 1988.

[RC88] T. R. Rogers and R. G. G. Cattell. Entity-Relationship Database User Interfaces. In Michael Stonebraker, editor, *Readings in Database Systems*. Morgan Kaufmann Publishers, Inc., 1988.

[Rie87] Wolf-Fritz Riekert. The ZOO Metasystem: A Direct-Manipulation Interface to Object-Oriented Knowledge Bases. In *ECOOP'87 Proceedings*, Paris, France, June 1987.

[Sch90] Matthias Schneider. Integrating Visual Aids into an Object-Oriented Programming Environment. *Lecture Notes in Computer Science*, 439(1), 1990.

[Shi81] David Shipman. The Functional Data Model and the Data Language DAPLEX. *ACM Transactions on Database Systems*, 6(1):140–173, 1981.

[Shn83] Ben Shneiderman. Direct Manipulation: A Step Beyond Programming Languages. *IEEE Computer*, 16(8):57–69, August 1983.

[Shn87] Ben Shneiderman. *Designing the User Interface: Strategies for Effective Human-Computer Interaction.* Addison-Wesley, 1987.

[SK91] Michael Stonebraker and Greg Kemnitz. The Postgres Next-Generation Database Management System. *Communications of the ACM*, 34(10):78–92, October 1991.

[Sol92] Valery Soloviev. An Overview of Three Commercial Object-Oriented Database Systems: ONTOS, ObjectStore, and O2. *SIGMOD Record*, 21(1):93–104, March 1992.

[SRH90] M. Stonebraker, L. A. Rowe, and M. Hirohama. The implementation of postgres. *IEEE Transactions on Knowledge and Data Engineering*, 2(1):125–142, March 1990.

[Sun90] Sun Microsystems. *OPENLOOK - Graphical User Interface Applications Style Guidelines.* Addison-Wesley, June 1990. Third Printing.

[SZ87] Karen E. Smith and Stanley B. Zdonik. Intermedia: A Case Study of the Differences Between Relational and Object-Oriented Database Systems. In *OOPSLA '87 Proceedings*, pages 452–465, Orlando, USA, October 1987.

[Wel88] Jay-Louise Weldom. Database Interfaces. *Journal of Information Systems Management*, pages 80–82, 1988.

[Zic91] Roberto Zicari. A Framework for Schema Updates in an Object-Oriented Database System. In *IEEE Proceedings of the International Conference on Data Engineering*, pages 2–13, Kobe, Japan, April 1991.