UNIVERSIDADE DE SÃO PAULO
Instituto de Ciências Matemáticas de São Carlos
Departamento de Ciências de Computação e Estatística

**Representação de Atributos
com Características Gráficas em uma
Base de Dados Orientada a Objetos**

Osvaldo Kotaro Taka.

Orientador:
Prof. Dr. Caetano Traina Júnior

Dissertação apresentada ao Instituto de Ciências Matemáticas de São Carlos, da Universidade de São Paulo, como parte dos requisitos para a obtenção do título de Mestre em Ciências: Área de Ciências de Computação e Matemática Computacional.

São Carlos
1993

Dedico este trabalho aos meus pais Isao e Setsuko, que sempre me incentivaram a prosseguir a carreira acadêmica e, em especial, à Bete, minha companheira, que pacientemente aguardou o término deste trabalho.

AGRADECIMENTOS

Ao Prof. Dr. Caetano Traina Júnior, orientador e amigo, pelo apoio e atenção que sempre me dispensou durante a elaboração deste trabalho.

À Profa. Dr. Agma Juci Machado Traina, pela simpatia e atenção sempre concedida.

Ao Prof. Nivaldi Calônego Júnior pela atenção e amizade sempre concedida.

À Bete, Laura e Carmem, funcionárias do ICMSC, pela paciência e atenção com que sempre me atenderam.

Aos participantes do Grupo de Estudos de Bases de Dados - ICMSC-USP, que contribuíram para a realização deste trabalho, em especial aos professores Pedro Luis Pizzigatti Corrêa e Mauro Blajiz.

Aos amigos Paulo Fransisco Sprovieri, João de Lucca, Rose, Rinaldo e Regiane que sempre estiveram próximos à minha família.

Aos amigos do Departamento de Geologia, Física e Matemática da Faculdade de Filosofia Ciências e Letras de Ribeirão Preto, que proporcionaram um ambiente agradável e tranquilo para a finalização deste trabalho.

À CAPES, CNPQ e FAPESP, pelo apoio financeiro dado para o desenvolvimento deste trabalho.

Representação de Atributos com Características Gráficas em uma Base de Dados Orientada a Objetos

Autor: Osvaldo Kotaro Takai
Orientador: Prof. Dr. Caetano Traina Júnior

RESUMO

Este trabalho define, inicialmente, o conceito de Atributo com Características Gráficas para o Modelo de Representação de Objetos (MRO). Esta definição foi obtida através da análise dos sistemas de suporte gráfico mais significativos. Dessa forma, o MRO pode ser aplicado diretamente para modelar dados de aplicações que façam uso intenso de bases de dados para armazenar informações gráficas.

Em seguida, são estabelecidas as operações básicas de manipulação de atributos gráficos numa base de dados orientada ao objeto, as quais permitirão o armazenamento e recuperação de informações gráficas de modo organizado e consistente.

Com a intenção de desenvolver um Sistema Gerenciador de Gráficos e Base de Dados (SIGGBD) genérico, que sirva de sustentação à criação de novos aplicativos que necessitem de sistemas de base de dados bem como de sistemas gráficos, foi implementado um protótipo de um SIGGBD, com todos os recursos para dar suporte a um Editor de Esquemas (EE) para o MRO.

Finalmente, foi implementado um EE utilizando-se de um sistema gráfico específico e um sistema de base de dados orientado o objetos denominado Gerenciador de Objetos (GEO), ao qual foram incorporadas as operações básicas de manipulação de Atributos com Características Gráficas.

Representation of Attributes with Graphical Characteristics in a Object-Oriented Database

Author: Osvaldo Kotaro Takai

Adviser: Prof. Dr. Caetano Traina Júnior

ABSTRACT

This work starts defining what is an Attribute with Graphical Characteristics to the MRO (Object Representation Model), based in the most proeminent graphics support systems studied. Thus, it may be applied directly into the modeling of data for application that use databases intensively to store graphics information.

The basic operations are defined for the graphical attributes in object oriented databases, including the storage and retrieval of graphical information, in a consistent way.

The main goal of developing a generic SIGGBD (Graphics and Database Manager System) is to aid the creation of new applications that need the support of database and graphics systems. As a first step in order to fulfill this purpose, a proptotype SIGGBD was constructed, aimed at supporting a MRO-Schema Editor (MRO-EE).

The MRO-EE was constructed using a specifc graphics system and an object-oriented database system named GEO (Object Manager), based in the MRO, extended with Attributes of Graphical Characteristics. With this purpose, the GEO was extended with the basic operations of storage and retrieval of graphics.

CONTEÚDO

Capítulo 1: Introdução.....	1
1.1 - Considerações Iniciais.....	1
1.2 - Motivação do Trabalho.....	1
1.3 - Organização do Trabalho.....	4
Capítulo 2: Conceitos de BD e Revisão Bibliográfica.....	5
2.1 - Considerações Iniciais.....	5
2.2 - Os SGBDs Convencionais.....	5
2.2.1 - Modelos de Dados Tradicionais.....	6
2.2.2 - Modelo Conceitual Modelo Físico Modelo de Dados.....	6
2.3 - Vantagens do Uso de SGBD em Sistemas de CAD.....	7
2.4 - Requisitos do SGBD para Sistemas de CAD.....	8
2.5 - Pesquisas sobre SGBDs para Aplicações não Convencionais.....	12
2.6 - Considerações Finais.....	14
Capítulo 3: Classificações de SGBDOO e o MRO.....	15
3.1 - Considerações Iniciais.....	15
3.2 - Conceitos de Orientação a Objetos.....	15
3.4 - Descrição do MRO.....	17
3.3 - Classificações.....	23
3.3.1 - Classificação Segundo Dittrich.....	23
3.3.2 - Classificação Segundo Kim.....	23
3.3.3 - Classificação Segundo Atkinson.....	24
3.5 - Considerações Finais.....	26
Capítulo 4: Modelagem Geométrica.....	27
4.1 - Considerações Iniciais.....	27
4.2 - A Modelagem Geométrica.....	27
4.3 - A Modelagem de Sólidos.....	28
4.4 - Modelos Gráficos.....	30
4.5 - Considerações Finais.....	31
Capítulo 5: Modelagem do GKS e PHIGS em MRO.....	32
5.1 - Considerações Iniciais.....	32
5.2 - Modelagem GKS.....	32
5.3 - Modelagem PHIGS.....	34
5.4 - Proposta de um Modelo Geral.....	36
5.5 - Considerações Finais.....	36

Capítulo 6: Atributos com Características Gráficas	37
6.1 - Considerações Iniciais	37
6.2 - Composição dos Atributos com Características Gráficas	37
6.3 - Especificação dos Elementos que compõe Atributos com Características Gráficas	38
6.3.1 - Especificação das Primitivas Gráficas	38
6.3.2 - Especificação de Atributos de Primitivas Gráficas	40
6.3.3 - Especificação das Matrizes de Transformação	40
6.3.4 - Especificação da Estrutura Gráfica	41
6.4 - O Meta-Esquema do MRO	42
6.5 - Considerações Finais	43
Capítulo 7: Aspectos da Implementação	44
7.1 - Considerações Iniciais	44
7.2 - O Paradigma de Acesso Navegacional	44
7.3 - Definição das Operações sobre Atributos Gráficos	46
7.4 - Considerações sobre a Implementação	54
7.4.1 - Detalhamento das Estruturas de Dados	54
7.4.2 - Definição das Interfaces Programacionais	56
7.5 - Testes	60
7.5 - Considerações Finais	61
Capítulo 8: Especificação da Aplicação	64
8.1 - Considerações Iniciais	64
8.2 - As Formas Gráficas do DRO	64
8.3 - As Instâncias dos Elementos de um DRO	67
8.4 - Representação Interna das Instâncias dos Elementos de um DRO	68
8.4.1 - Representação Interna de Tipos de Objetos	69
8.4.2 - Representação Interna de Tipos de Relacionamentos	70
8.4.3 - Representação Interna de Tipos de Atributos	72
8.5 - A Arquitetura do Programa Editor de Esquemas	75
8.5.1 - A Abordagem Adotada para a Implementação da Aplicação (EE) e do SIGGBD	76
8.5.1.1 - A Aplicação (EE)	76
8.5.1.2 - O SIGGBD	78
8.6 - Considerações Finais	81
Capítulo 9: Conclusões e Propostas para Futuras Pesquisas	82
9.1 - Conclusões	82
9.2 - Propostas para Futuras Pesquisas	83
Referências Bibliográficas	84

Capítulo 1

Introdução

1.1 - Considerações Iniciais

Um fator comum dentro do universo dos sistemas para auxílio a projeto, tais como sistemas para CAD, CIM e CASE, é a necessidade de representar atributos gráficos numa base de dados. Este trabalho procura contribuir para o desenvolvimento de Sistemas Gerenciadores de Base de Dados (SGBD) no sentido de estabelecer métodos, algoritmos e estruturas de representação e manipulação de atributos gráficos em bases de dados orientadas a objeto.

1.2 - Motivação do Trabalho

O avanço tecnológico ocorrido nas últimas 5 décadas, principalmente na área de eletrônica, tem permitido que as indústrias recorram cada vez mais à automação para auxiliar não apenas o processo criativo, mas também o processo de fabricação. Por volta de 1940 era esperada uma nova revolução industrial para os anos vindouros, devido ao surgimento de máquinas-ferramentas controladas numericamente através de algum equipamento eletrônico programável [68]. Atualmente, a expectativa dos anos 40 parece estar bem próxima. Isso porque várias áreas de pesquisa vêm se consolidando desde então, principalmente as áreas de CAD ("Computer Aided Design") e CAM ("Computer Aided Manufacturing").

A área de Apoio Computadorizado para Atividades de Projeto preocupa-se em dar suporte à fase de projeto de algum produto, fornecendo ao projetista uma variedade de ferramentas de software e, dessa forma, liberando-o para questões ligadas ao raciocínio e à criatividade. O conjunto de hardware e software voltado para esta finalidade é denominado Sistema CAD.

A área de Apoio Computadorizado para Produção e Manufatura procura dar subsídios à fase de produção, a fim de aumentar a produtividade, diminuir custos e melhorar a qualidade dos produtos fabricados. Para tanto, esta área faz uso intensivo de máquinas-ferramentas controladas numericamente através de computador (ou NC - "Numerical Control"), robôs e outros equipamentos eletro-mecânicos. O sistema que aglutina esses componentes é denominado Sistema CAM.

Pode-se distinguir várias modalidades de sistemas CAD e CAM, dependendo do tipo de aplicação para o qual foram desenvolvidos: mecânica, elétrica, software, naval, arquitetura, etc.

A fase de projeto é seguida pela fase de produção, e ambas mantêm uma estreita relação, de forma que muitas pesquisas têm sido conduzidas, tanto pela comunidade de CAD quanto pela de CAM, para a obtenção de um único sistema capaz de integrar todos os requisitos destas duas áreas. Sistemas contruídos com essa finalidade são chamados de Sistemas CAD/CAM ou simplesmente CIM ("Computer Integrated Manufacturing"). Essa integração, segundo Groover [27], pode inovar o processo convencional de manufatura praticado pelas indústrias, onde essencialmente é adotado o procedimento de dois passos (as atividades de projeto são separadas das atividades de planejamento de processos), conseqüentemente consumindo mais tempo devido à duplicação de esforços. Num sistema CIM, uma ligação direta entre projeto e manufatura é estabelecida. Desse modo, pode-se dizer que a meta do CIM não é apenas automatizar certas partes do projeto e certas fases da manufatura, mas também automatizar a passagem de projeto para a manufatura.

Um outro aspecto diz respeito à integração entre várias ferramentas de software referentes a sistemas CAD existentes no mercado, tais como: sistemas de modelagem de sólidos (representação de objetos sólidos); análise de engenharia (análise de elementos finitos, análise das propriedades físicas em geral); revisão e avaliação de projeto (sobreposição de imagens, simulação); desenho automatizado (dimensionamento automático, geração de áreas hachuradas, colocação do desenho em escalas, capacidade de desenvolver visões de secções e ampliação das visões de um detalhe particular), entre outras.

O projeto CAD*I do programa ESPRIT [43] tem contribuído nesse sentido, criando uma padronização para a permuta de dados entre ferramentas de sistemas CAD, bem como entre sistemas CIM atualmente existentes, utilizando a estrutura de arquivos neutros como o elemento integrador.

Atualmente, várias pesquisas têm sido desenvolvidas pela comunidade de BD (Base de Dados), no sentido de construir SGBDs que possam servir como elemento integrador entre as várias ferramentas de sistemas CAD e CIM [20], [24], [30], [38], [45], [63], [65].

Seguindo este último enfoque, está em desenvolvimento no ICMSC-USP São Carlos, um SGBD (Sistema de Gerenciamento de Base de Dados) apoiado no MRO (Modelo de Representação de Objetos) [66], denominado GEO (GErenciador de Objetos) [65]. O MRO é um modelo que permite modelar os dados a serem armazenados na base em termos de Objetos.

De maneira geral, os Objetos e as associações entre objetos (Relacionamentos) podem possuir Atributos. Assim, a maior motivação deste trabalho é permitir que Atributos com Características Gráficas possam ser efetivamente armazenados na BD e reconhecidas pelo GEO. Isto porque a maioria das ferramentas de projeto voltadas para aplicações de Engenharia fazem uso de SGBDs e também de Núcleos Gráficos (NG). Para tais ferramentas, os SGBDs convencionais não fornecem um suporte adequado. Uma das características desejáveis é que a base sustente atributos com características gráficas, para tornar possível agregar tanto informações gráficas como não gráficas aos objetos manipulados pelo aplicativo. Assim, as informações gráficas de um objeto podem ser armazenadas pela base e repassadas para o NG quando precisarem ser traçadas. Para tal, torna-se importante estudar os NG existentes, pois são eles que tratam as informações dos objetos representadas como atributos gráficos, e consequentemente determinam a forma como tais atributos deverão ser adequadamente representados na BD.

Desde que a base suporte a representação de atributos gráficos, pode-se projetar um Subsistema Integrado de Gerenciamento de Gráficos e Base de Dados (SIGGBD), integrando um SGBD e um NG específicos. Esta abordagem traz vários benefícios:

1. Determinadas consultas à BD e/ou operações sobre o NG podem ser disparadas pelo aplicativo e efetuadas automaticamente, isto é, sem que os aplicativos que se utilizem do SIGGBD precisem atuar como intermediário entre o SGBD e o NG.
2. Aplicativos já desenvolvidos, que se utilizem de um SGBD e de um NG específicos, não necessitarão de grandes mudanças caso o SGBD ou o NG venham a ter suas interfaces modificadas.
3. Projetistas de aplicativos não necessitarão, a priori, conhecer ou desenvolver os SGBD e NG, mas apenas conhecer uma única interface SIGGBD.
4. Aplicativos que não façam uso do SIGGBD podem ainda fazer acesso ao SGBD.
5. Aplicativos que executam tarefas genéricas, tais como apresentação de gráficos, edição gráfica dependente de particularidades da aplicação, separação de informações gráficas para exportação, etc, podem ser construídos como ferramentas utilitárias do SIGGBD, e não precisam ser reconstruídas em cada sistema aplicativo. Assim, os aplicativos em geral podem se beneficiar da existência de tais utilitários.

1.3 - Organização do Trabalho

O presente trabalho está dividido em 9 capítulos. Os assuntos principais de cada capítulo são resumidos abaixo:

- **Capítulo 1:** são apresentadas as idéias que motivaram o desenvolvimento deste trabalho, bem como a sua organização.
- **Capítulo 2:** são abordados aspectos referentes às necessidades e aplicações de SGBDs em sistemas CAD.
- **Capítulo 3:** são apresentadas classificações de SGBDs Orientados ao Objeto, segundo vários autores. Além disso, é feita uma breve descrição do modelo de dados MRO, que servirá como base de estudo e experimentação para a representação e manipulação de atributos gráficos.
- **Capítulo 4:** são apresentados conceitos fundamentais da área de Modelagem Geométrica.
- **Capítulo 5:** são apresentadas duas modelagens de padrões gráficos, em termos do MRO, e definido um modelo de representação de informações gráficas genérico.
- **Capítulo 6:** são definidos e especificados os elementos que compõem os Atributos com Características Gráficas, de acordo com o modelo genérico obtido no capítulo 5, e ainda, é apresentado o DRO do meta-esquema com alterações que refletem a inclusão de Atributos com Características Gráficas.
- **Capítulo 7:** é definido um conjunto de operações que manipulam atributos gráficos, bem como são feitas considerações a respeito de sua implementação no GEO.
- **Capítulo 8:** é feita a especificação da aplicação, bem como considerações a respeito de sua implementação.
- **Capítulo 9:** São apresentadas as conclusões e propostas para futuras pesquisas.

Capítulo 2

Conceitos de BD e Revisão Bibliográfica

2.1 - Considerações Iniciais

O propósito deste capítulo é o de fornecer uma visão geral das definições e conceitos existentes na área de BD. Para tanto, as definições e conceitos são apresentados juntamente com as citações bibliográficas pertinentes.

Na seção 2.2 são expostos conceitos de SGBDs convencionais, explicando-se superficialmente seus modelos de dados. Na seção 2.3, são descritas as vantagens que podem ser obtidas pelo uso de SGBDs convencionais em sistemas de CAD. Na seção 2.4, são listados os requisitos que SGBDs devem atender a fim de que possam fornecer um suporte adequado para sistemas de CAD. Na seção 2.5, apresenta-se uma bibliografia sobre pesquisas na área de SGBDs para aplicações não convencionais.

2.2 - Os SGBDs Convencionais

Os primeiro SGBDs foram construídos com o intuito de dar suporte à aplicações comerciais, tais como contas a pagar/receber, gerenciamento de almoxarifado, folha de pagamento, etc. Estas aplicações, em geral, manipulam um volume muito grande de dados com formatos simples e bem definidos. Aos SGBDs voltados para essas aplicações é comum dar-se o nome de **SGBDs Convencionais**.

Os SGBDs seguem um **modelos de dados** para a sua implementação. Os modelos de dados utilizados pelos SGBDs convencionais são denominados **modelos de dados tradicionais**.

2.2.1- Modelos de Dados Tradicionais

Os **modelos de dados** tradicionalmente utilizados em SGBDs convencionais são: Rede, Hierárquico e Relacional.

O modelo de rede [14], [15] modela dados de aplicações utilizando-se do conceito de coleção de registros conectados uns aos outros através de "ligações", onde cada registro de um arquivo mestre é ligado a diversos registros do arquivo membro.

O modelo hierárquico [67], da mesma forma que o modelo de rede, representa dados de aplicações utilizando o conceito de coleção de registros. No entanto, as associações (ligações) entre registros são organizadas de maneira a estabelecer uma hierarquia entre os registros.

O modelo relacional [12] representa os dados de aplicações utilizando basicamente os conceitos de relações (tabela), tuplas (linhas da tabela), normalização e operações derivadas da álgebra relacional.

Kort classifica esses modelos de dados tradicionais como Modelos Lógicos Baseados em Registros [42].

2.2.2 - Modelo Conceitual × Modelo Físico × Modelo de Dados

De acordo com Heuser [28], o termo **modelo conceitual** foi estabelecido na década de 70 por um grupo de trabalho da associação norte-americana de normas técnicas (ANSI). No passado, os sistemas de processamento de dados eram construídos com o intuito de atender apenas a uma classe de usuários. Com o passar do tempo, surgiram BDs voltados para diversas classes de usuários com requisitos funcionais diferentes, as quais tornaram-se o meio de comunicação entre diferentes usuários. Para que esta comunicação seja confiável e eficaz, é necessário que todos os usuários tenham o mesmo entendimento sobre o significado dos dados armazenados na BD. A fim de suprir este entendimento, o **modelo conceitual** foi definido e utilizado como sendo o instrumento principal de toda a descrição do significado dos dados armazenados na BD.

A utilização do **modelo conceitual** foi gradativamente ampliada, desde a documentação principal de uma BD até o ponto em que o modelo foi utilizado para servir de base ao desenvolvimento de projetos de BDs. Segundo esta idéia, o **modelo conceitual** tornou-se um modelo abstrato para a organização dos dados a serem armazenadas na BD, sem conter detalhes de implementação. O processo de projeto de uma BD envolve adição gradativa de

detalhes de implementação no **modelo conceitual**, até chegar ao chamado **modelo físico**, que é uma descrição da BD mais próxima do nível de implementação.

Os modelos conceituais que são utilizados para representar apenas as propriedades estáticas das BDs são denominadas **modelos de dados semânticos**.

Apesar dos **modelos semânticos** utilizados em BDs terem evoluído para representar também as propriedades dinâmicas, muitos autores ainda conservam o termo original **modelo de dados**, muito embora tais modelos possam receber denominações específicas como **modelo de função** ou **modelo de comportamento**.

Referências a publicações sobre este assunto podem ser encontradas na parte introdutória do livro de Heuser [28].

2.3 - Vantagens do Uso de SGBD em Sistemas de CAD

A principal vantagem de se utilizar um SGBD em sistemas de CAD é a integração de todos os dados do projeto. Com isso consegue-se eliminar redundâncias além de permitir que diversas ferramentas de CAD compartilhem os dados de forma consistente e padronizada. Um exemplo disso é o sistema protótipo Pride ("PRototype Integrated DEsign system") [24] que utiliza uma BD relacional para gerenciar dados de engenharia.

Segundo Price [58], a utilização de SGBDs convencionais em sistemas de CAD traz outras vantagens, além da integração de dados do projeto. Estas vantagens são listadas a seguir:

- **Independência de dados:** Os SGBDs oferecem independência física e lógica de dados. A independência física libera as ferramentas de CAD da preocupação com a organização física dos dados, bem como com os meios de armazenamento utilizados. A independência lógica torna as ferramentas imunes a mudanças no crescimento e reestruturação dos dados. Assim, é possível, por exemplo, criar ou alterar as estruturas de armazenamento com o intuito de melhorar o desempenho, sem que as ferramentas já existentes tenham que ser modificadas.
- **Interface homogênea:** Os SGBDs oferecem uma interface padrão, através da qual todas as ferramentas e usuários devem interagir com a BD. O mecanismo de visões, oferecido pelos SGBDs, permite particularizar a visão dos dados para determinadas ferramentas, tornando mais fácil a construção de novas ferramentas.

- **Modelo de dados:** Todo SGBD oferece um modelo de dados que deve ser utilizado para mapear as construções da realidade em termos de conceitos pré-definidos. O modelo de dados é compreendido como um conjunto de estruturas de dados, um conjunto de operadores e um conjunto de regras de integridade implícitas, permitindo assim que as ferramentas expressem um certo grau da semântica existente na realidade. Entretanto, boa parte do significado dos dados ainda fica dependente da aplicação.
- **Controle de concorrência:** A BD é utilizada por diversos usuários e/ou ferramentas, tendo cada um a impressão de ser o único a fazer acesso aos dados. O SGBD apresenta mecanismos de sincronização para os diversos usuários.
- **Controle de integridade:** O controle centralizado permite verificar, mais facilmente, se os dados armazenados estão corretos. Além de garantirem as regras de integridade intrínsecas ao modelo, os SGBDs permitem definições de outras restrições que devem ser verificadas automaticamente. Quando os dados de projeto estão armazenados, com as dependências entre eles colocadas explicitamente, é possível verificar equivalências entre representações, bem como descobrir como propagar alterações feitas sobre estas.
- **Restauração da BD:** Os SGBDs oferecem mecanismos para garantir que suas bases de dados não serão danificadas em casos de falha de hardware ou software. O conceito de transação é normalmente utilizado como unidade de consistência e restauração da BD.
- **Segurança:** A base contém dados que não devem ser visíveis a todos os usuários. Com o mecanismo de controle de acesso (por exemplo, definição de visões da base para cada usuário), os SGBDs permitem a existência de dados confidenciais.

2.4 - Requisitos do SGBD para Sistemas de CAD

Os requisitos que um SGBD deve ter para ambientes de projeto, segundo Price [58], são descritos abaixo. O autor ilustra as necessidades de tais sistemas em termos de gerenciamento de dados, e apresenta exemplos tomando como base os requisitos de armazenamento e administração de dados em um sistema CAD para projeto de circuitos VLSI ("Very Large Scale Integration").

- **Modelo de dados:** Os modelos de dados tradicionais (rede, hierárquico e relacional) não possuem grande capacidade semântica para

representar, de modo natural, dados de projeto [20], [63]. Isto porque os objetos manipulados em projetos apresentam, basicamente:

1. **Estruturas complexas:** As atividades inerentes ao projeto normalmente necessitam manipular, como um todo, "Objetos Complexos" [30], [44], [63], tais como: formulários, documentos, diagramas e desenho de uma peça mecânica. O termo "Objeto Complexo" foi inicialmente introduzido por Haskins e Lorie [30], para descrever uma entidade lógica simples, composta por um conjunto de objetos. Como isso caracteriza uma hierarquia de composição, Kim [40] utiliza o termo "Objeto Composto" ao invés de "Objeto Complexo". Os modelos tradicionais (rede, hierárquico e relacional) possuem como unidade de processamento o registro, e as operações e suas restrições são centradas no conceito de registro e suas ligações. Assim, verifica-se que tais modelos não suportam o conceito de "Objeto Complexo".
 2. **Propriedades complexas:** Além dos objetos de projeto normalmente possuírem estruturas complexas, eles ainda podem ter propriedades complexas, tais como: estruturas do tipo árvore, conjunto, lista, matrizes e texto. Os modelos tradicionais não suportam atributos com essas características; isso porque eles foram feitos para modelar entidades com atributos bem definidos (inteiro, real, caracter e cadeia de caracteres com tamanho limitado).
- **Gerência de projeto:** Num sistema CAD existe a necessidade de armazenar as várias representações e configurações dos objetos de projeto. Assim, é desejável que um SGBD dê suporte ao gerenciamento de:
 1. **Representações:** Cada objeto de projeto consiste de várias representações, sendo que cada representação é uma perspectiva diferente de como o objeto pode ser visto [36]. As diversas representações de um objeto podem ser organizadas em termos de alternativas e versões. As alternativas representam as várias soluções possíveis para um objeto de projeto. Cada alternativa pode ter diversas versões, sendo que cada versão retrata um mesmo objeto a cada mudança sofrida. Mudanças em um objeto de projeto podem ocorrer devido a melhoramentos ou correções.
 2. **Configurações:** Para compor um objeto complexo (por exemplo, para efeitos de simulação ou processo de manufatura) é necessário especificar quais objetos do projeto serão empregados e, dentre estes objetos, quais versões deverão ser consideradas. Tal tipo de especificação corresponde a uma configuração.

- **Arquivamento e Manutenção de Históricos:** O arquivamento de um grande volume de documentos sobre um projeto pelo SGBD, por um longo período de tempo, é de grande interesse, sobretudo quando se deseja fazer estatísticas ou quando se pretende reutilizá-lo.
- **Controle de Integridade:** O controle de integridade dos dados, a serem armazenados na BD de projeto, vai além do controle de integridade suportado pelos SGBDs tradicionais. O controle de integridade, excluindo a verificação de tipos, pode em alguns casos ficar a cargo de alguma ferramenta de revisão e avaliação de projeto (sobreposição de imagens, cinemática). Em outros casos, por exemplo na alteração de partes de um projeto sendo compartilhado, pode ser interessante que o SGBD realize, condicionado ao tempo, a propagação de mudanças segundo regras pré-estabelecidas, de forma automática.
- **Transações de projeto:** Em SGBDs convencionais a transação é a unidade de consistência, restauração e sincronização, sendo que o seu tempo de duração é de, no máximo, alguns segundos. Uma transação é uma seqüência de operações de leitura e escrita que atualizam a BD. Para evitar que as atualizações levem a BD a um estado inconsistente, deve-se garantir que as transações sejam efetuadas atomicamente, isto é, ou todas as operações são realizadas ou nenhuma operação é realizada. As transações podem ser executadas de forma concorrente, desde que seus resultados sejam os mesmos que os produzidos por uma execução serial das mesmas.

Em contrapartida, uma transação de projeto, segundo Katz, começa quando o projetista ganha acesso a alguma parte do projeto [35]. A transação fica pendente enquanto o projetista estiver trabalhando (normalmente ocupando um longo período de tempo). A transação somente é efetivada quando as alterações forem validadas através de ferramentas de validação. Assim, enquanto o projeto não for validado o SGBD deve tolerar algumas inconsistências.

Como uma transação de projeto tem longa duração (transação longa), em caso de falha durante a execução de uma transação deve ser permitido que aquelas alterações realizadas, de forma ainda parcial, sejam efetuadas na BD, mesmo que a transação não tenha sido completada. Com isso evita-se perder dias ou até meses de trabalho. Dessa forma, as transações longas de um SGBD para sistemas de projeto não possuem a característica de atomicidade existente em SGBDs convencionais.

Quando várias pessoas trabalham em um mesmo projeto, há problemas de controle de concorrência. No entanto, devido a divisão de tarefas, a

"interferência" entre transações é pequena. Assim, uma negociação entre os projetistas pode ser suficiente para solucionar um conflito de acesso sobre dados compartilhados. Por outro lado, ainda é necessário que o SGBD faça o controle de concorrência para evitar qualquer desatenção. Verifica-se que a unidade de bloqueio ("locking"), ou seja, o menor tamanho do dado protegido, é de granularidade maior do que em SGBDs convencionais, que bloqueiam um registro ou um campo. Isso porque a unidade de bloqueio, em ambientes de projeto, é normalmente um objeto complexo ou uma de suas versões.

- **Cooperação entre projetistas:** Para que haja a cooperação entre os projetistas envolvidos em um mesmo projeto de forma organizada, é interessante que o SGBD ofereça mecanismos que estabeleçam esta cooperação, definindo formas de interação como, por exemplo, de BD públicas e particulares.

- **Arquitetura do sistema:** Como as atividades de projeto envolvem corriqueiramente vários projetistas, Dittrich [22] sugeriu a arquitetura ilustrada na figura 1.

Nessa arquitetura, os projetos são desenvolvidos nas estações de trabalho até que possam ser colocadas na BD pública. Sendo assim, o SGBD

deve oferecer maneiras de garantir a consistência geral dos dados da base, distribuídos em cada estação de trabalho remota.

Figura 1: Arquitetura segundo Dittrich

- **Desempenho:** Como os dados do projeto possuem estruturas complexas, diferindo portanto das aplicações convencionais, espera-se que o SGBD ofereça estruturas de armazenamento e mecanismos de acesso que permitam obter a eficiência desejada. Um exemplo disso é o agrupamento físico de dados ("clustering") que possuem alta probabilidade de serem acessados em conjunto.

2.5 - Pesquisas sobre SGBDs para Aplicações não Convencionais

A comunidade de BD tem feito esforços na busca de modelos de dados e técnicas apropriadas para SGBDs voltados à aplicações não convencionais. Atualmente há duas grandes linhas de pesquisa nessa área, segundo Melo [47]:

- Estender os modelos de dados convencionais;
- Descobrir um novo modelo expressivo e bem fundamentado.

Dentro da primeira linha de pesquisa estão, entre outras:

- **Extensões ao System-R [30]:** Haskins e Lorie estão entre os primeiros a estender um modelo de dados relacional, visando atender os requisitos impostos pelas aplicações não convencionais, tais como objetos complexos e campos longos. Para isso, utilizaram o System-R [2], incorporando os seguintes conceitos:
 - a) identificador de tuplas: idêntico ao conceito de "surrogates" apresentado por Codd [13]; isto é, o sistema atribui automaticamente a cada tupla de uma relação um código único não reutilizável, mesmo que alguma tupla seja eliminada da base.
 - b) atributo de composição em uma tupla: o valor de atributo é igual ao identificador de tuplas. Como um objeto complexo caracteriza uma hierarquia de composição, este atributo permite que esta hierarquia seja estabelecida;
 - c) atributo de referência: este atributo permite que tuplas se referenciem a outras tuplas; por exemplo, para criar uma lista;
 - d) campos longos: foram criadas instruções para manipular, como um todo, várias cadeias de caracteres; por exemplo, um texto.
- **Sistema AIM-II [20]:** Na busca da eficiência em SGBDs para aplicações de CAD, CAM e CIM, Dadam e outros desenvolveram um protótipo de um SGBD apoiado num modelo de dados relacional estendido AIM-II. O modelo chama-se NF² ("Non First Normal Form") [34], e procura generalizar o modelo de dados relacional, permitindo que relações possam ser manipuladas sem estarem necessariamente na primeira forma normal. A idéia chave do modelo é permitir que relações ocorram como valores de atributo de tuplas em relações. Além do modelo, é também definida uma linguagem de consulta, semelhante ao SQL [7], [11], e uma interface de programação de aplicações API, especialmente para manipular objetos complexos.

- **Sistema POSTGRES [64]:** Stonebraker e outros reconhecem que um SGBD, baseado simplesmente no modelo de dados relacional, não atende de forma eficiente às necessidades das aplicações de engenharia. Sendo assim, propõem um SGBD chamado POSTGRES que é considerado uma extensão do SGBD relacional INGRES [62]. As principais inovações estão refletidas na linguagem de consulta POSTQUEL. Em essência, o POSTQUEL é o QUEL do INGRES com duas novidades: 1) permite ao usuário trabalhar com tipos abstratos de dados e, 2) representa objetos complexos utilizando campos de tipo POSTQUEL e de tipo procedimento. Um campo do tipo POSTQUEL contém uma sequência de comandos de manipulação de dados, e um campo do tipo procedimento contém procedimentos escritos em alguma linguagem de propósito geral, com comandos de manipulação de dados embutidos. As extensões tiveram como objetivo:
 - a) fornecer melhor suporte a objetos complexos;
 - b) permitir ao usuário definir tipos de dados, operadores e métodos de acesso;
 - c) fornecer mecanismos como "regra" e "gatilho" para ativar a BD, bem como mecanismos de inferência;
 - d) simplificar o código do SGBD para recuperação de falhas;
 - e) fazer uso, tanto quanto possível, de novas tecnologias (por exemplo, discos óticos);
 - f) fazer o mínimo de mudanças no modelo relacional (de preferência nenhuma).
- **Sistema R²D² [38]:** Kemper e outros desenvolveram o SGBD R²D² ("A Relational Robotics Database System with Extensible Datatypes"). Os autores procuraram integrar as abordagens de orientação a objeto comportamental e estrutural [21], que serão discutidas mais adiante, buscando com isso maior facilidade de utilização do SGBD pelo usuário, bem como a eficiência necessária para utilização de SGBD em aplicações de projeto. O R²D² é um pré-processador para o SGBD AIM-II [20], fornecendo-lhe a característica estrutural. Obtem-se a característica comportamental estendendo-se a linguagem de consulta HDBL [57] do AIM-II, através de mecanismos para o usuário trabalhar com o conceito de Tipos Abstratos de Dados.

Dentro da segunda linha de pesquisa estão, entre outras:

- **Sistema GemStone [44]** O modelo de dados é derivado dos conceitos encontrados na linguagem Smalltalk [25], que incluem os conceitos de classe, mensagem, objeto, hierarquia de classes e herança de atributos e métodos. Na realidade, o sistema estende a linguagem Smalltalk com recursos de gerenciamento de objetos persistentes, ou seja,

gerenciamento de objetos que não são destruídos mesmo após a finalização da aplicação que os manipula.

- **Sistema ENCORE [73], [31]:** Esse sistema utiliza um modelo de dados semântico, de alto nível, onde o principal relacionamento entre tipos é o IS-A.
- **Sistema Iris [23].** A modelagem de dados nesse sistema é feita através de três construtores: objetos, tipos e funções. Os objetos são descritos pelos seus comportamentos e podem ser acessados e manipulados somente por meio de funções. Além disso, os objetos são classificados por tipos. Objetos que pertencem ao mesmo tipo compartilham as mesmas funções. Os tipos são organizados dentro de uma hierarquia de tipos com herança de funções. Os objetos podem ser argumentos ou resultado de alguma função. Uma função somente poderá ser aplicada a um objeto se no seu tipo estiver definida a função.
- **Sistema ORION [39].** O modelo, nesse sistema, admite conceitos como objeto, encapsulamento de comportamento através de métodos, classes, hierarquia de classes e hierarquia de composição de classes. A rigor, o modelo de dados de ORION segue estes cinco conceitos extraídos das Linguagens de Programação Orientadas a Objetos.
- **Sistema OZ+ [39]:** O sistema OZ+ foi desenvolvido utilizando como base os conceitos promovidos pelas linguagens de programação orientadas ao objeto, bem como os conceitos envolvidos em sistemas de base de dados. Sendo assim, o modelo de dados de OZ+ admite todos os conceitos da programação orientada a objetos, juntamente com o armazenamento de objetos persistentes. Além disso, o sistema faz o tratamento de acesso concorrente sobre objetos, utilizando o protocolo de comunicação por troca de mensagens bloqueantes.
- **Sistema GEO [65].** O Sistema GEO utiliza o Modelo de Representação de Objetos - MRO [66], que será descrito no próximo capítulo.

2.6 - Considerações Finais

Verifica-se que, apesar das vantagens introduzidas em SGBDs convencionais para aplicações de CAD, elas não são uma alternativa adequada. Isso porque os sistemas de CAD apresentam muitas necessidades adicionais àquelas apresentadas pelas aplicações ditas convencionais. Devido a esse fato, têm surgido muitas pesquisas e vários conceitos, principalmente no que diz respeito a Orientação a Objetos na área de BD.

Capítulo 3

Classificações de SGBDOO e o MRO

3.1 - Considerações Iniciais

Os objetivos deste capítulo vão além de simplesmente fazer a descrição do MRO. Conforme foi comentado no capítulo anterior, as buscas por novos modelos de dados para atender aos requisitos de Aplicações não Convencionais podem levar em consideração aspectos de Orientação a Objetos. Uma questão que está em voga é: "Quando um SGBD é Orientado a Objeto?". Para tentar responder a esta questão e comprovar que qualquer SGBD apoiado no MRO é Orientado a Objeto, na seção 3.2 são definidos alguns conceitos de Orientação a Objetos. Estes conceitos fornecerão subsídios às classificações, que serão feitas posteriormente na seção 3.4, segundo as definições de alguns autores. Na seção 3.3, uma descrição bastante resumida do MRO é apresentada.

3.2 - Conceitos de Orientação a Objetos

Recentemente, tem aumentado o interesse em se aplicar os conceitos de orientação a objetos em várias áreas de pesquisas da Ciência da Computação, tais como: Base de Dados, Linguagens de Programação, Representação do Conhecimento e Arquiteturas de Computadores. No entanto, ainda não existe um consenso geral sobre o que seja, precisamente, um objeto e, conseqüentemente, o que significa o termo "Orientado a Objeto". Talvez isso esteja acontecendo devido aos fundamentos conceituais de orientação a objeto terem se desenvolvido em quatro áreas de pesquisas distintas: inicialmente em Linguagens de Programação, então em Inteligência Artificial e gerenciadores de Interface Homem/Computador e, por fim, em BD.

Simula-67 [6] é geralmente apontada como a primeira linguagem de programação orientada a objetos. A partir de Simula-67, pesquisadores em linguagens de programação adotaram duas abordagens diferentes de pesquisa. A primeira, foi a de desenvolver novas linguagens orientadas ao objeto como, por exemplo: Smalltalk [25] e Eiffel [48]. A outra, foi a de estender as linguagens de programação convencionais, resultando em linguagens como ObjectLISP [52], Clascal [60] e C++ [64].

Em Inteligência Artificial, o esquema de frames para representação do conhecimento, desenvolvido por Minsky [50], tem permitido o desenvolvimento

de linguagens de representação do conhecimento baseadas em *frames*; por exemplo, KEE da Intellicorp e ART da Inference [29].

Em Bases de Dados, as pesquisas em modelos de dados semânticos [32], [53] têm produzido conceitos similares aqueles encontrados em Linguagens de Programação e Inteligência Artificial.

Vários conceitos de orientação a objetos surgiram a partir dessas pesquisas. Esses conceitos individualmente não são suficientes para sugerir que um sistema que os adote seja considerado Orientado a Objetos; no entanto, vários autores têm considerado que nem todos são necessários para garantir isso. Apresenta-se a seguir um conjunto significativo de conceitos, em geral associados à Orientação a Objetos:

- **Objeto e Identificador de Objetos:** Em sistemas e linguagens orientadas a objeto, qualquer entidade do mundo real é modelada uniformemente como um objeto. Este tratamento uniforme simplifica a visão do usuário do mundo real. O identificador de objetos é utilizado para fazer acesso a instâncias de objetos;
- **Atributos e Métodos:** Todo objeto tem um estado e um comportamento. O estado de um objeto é o conjunto de valores de seus atributos, e o comportamento é o conjunto de métodos (procedimentos e/ou funções) que operam sobre o estado de um objeto. Além disso, um atributo de um objeto pode tomar valores simples (por exemplo, inteiro, Boolean) ou um conjunto de valores (por exemplo, uma lista);
- **Classe:** Uma classe é o agrupamento de todos os objetos que compartilham um mesmo conjunto de atributos e métodos. Um objeto deve pertencer somente a uma classe, como uma instância dela. O relacionamento entre um objeto e sua instância é conhecido como "instance-of". Uma classe é semelhante a um tipo abstrato de dados, pois inclui o encapsulamento da implementação e dos dados internos de um módulo, somente permitindo o acesso ao módulo através da sua interface. O estado e o comportamento, encapsulados num objeto, são acessíveis somente através do envio de mensagem explícita ao objeto (ou chamadas de rotinas);

- **Hierarquia de Classes e Herança:** Os sistemas orientados a objeto permitem ao usuário criar uma nova classe a partir de uma classe existente; a nova classe, chamada subclasse da classe existente, herda todos os atributos e métodos da classe existente, chamada a superclasse da nova classe. Esta forma de herança é denominada herança simples. Por outro lado, se uma classe for definida a partir de duas ou mais superclasses existentes, a herança é denominada herança múltipla;

3.3 - Descrição do MRO

O MRO é um modelo de dados criado para representar dados de quaisquer aplicações, convencionais ou não, que necessitem utilizar-se de uma BD. Para essa finalidade, o modelo permite que objetos (entes, eventos, coisas e conceitos) e suas associações, pertinentes a alguma aplicação, sejam representadas dentro do esquema da BD.

Alguns conceitos relevantes ao modelo são listados a seguir:

OBJETOS: No MRO, o mundo real é modelado através de **Objetos**, sendo que Objetos semelhantes são caracterizados através de **Tipos de Objeto**. Além do seu tipo, um Objeto pode ter **Identificadores**, sendo que em um determinado instante existe apenas um identificador que é denominado corrente.

RELACIONAMENTOS: As associações entre os Objetos do mundo real, não representáveis através de construções específicas do MRO, são caracterizadas através de **Relacionamentos** definidos pelo usuário. Relacionamentos semelhantes são caracterizados através de **Tipos de Relacionamento**, sendo que a semântica de todos os relacionamentos de mesmo tipo é denominada **Modalidade de Relacionamento**. A Modalidade estabelece o significado de um Relacionamento, independente dos tipos dos objetos relacionados. Os Tipos de Relacionamentos podem envolver dois ou mais Tipos de Objetos. Quando são dois os Tipos de Objetos envolvidos, o Tipo de Relacionamento é chamado **Relacionamento Binário**, e deve possuir sempre um **Tipo de Relacionamento Oposto**, onde os Tipos de Objetos origem e destino trocam de posição. Quando o Tipo de Relacionamento envolve mais que dois Tipos de Objetos é denominado **Relacionamento Triplo**, sendo compostos de duas Modalidades de Relacionamento.

ATRIBUTOS: Objetos e Relacionamentos podem ter **Atributos** e todo Atributo deve ter seu **Tipo de Atributo**. Todo tipo de Atributo possui um **Tipo**

de Dado e uma Característica. A Característica determina o conjunto de operações que o SGBD deve suprir para atributos daquele tipo. Todo atributo e todo relacionamento pode possuir uma multiplicidade mínima, uma multiplicidade máxima e uma ordinalidade.

COLÔNIAS: Os Objetos da base são divididos em partições, denominadas **Colônias de Objetos**, ou simplesmente, **Colônias**. As Colônias são classificadas em **Tipos de Colônias**, sendo que cada Tipo de Colônia é um dos subconjuntos de tipos de Objetos, feita no Conjunto de tipos de Objetos. Uma consequência imediata disso é que todo Objeto deve **Habitar** exatamente uma Colônia.

Os Tipos de Colônias são estruturados em uma Hierarquia, em que o tipo supremo é o **Tipo de Colônia Global**. Na BD existe somente uma Colônia Global, controlada pelo SGBD, e na qual todos os Objetos estão sempre disponíveis. Num dado instante, estarão **Disponíveis** apenas Objetos de uma Colônia de cada Tipo, a qual é chamada a **Colônia Aberta**, dentre as Colônias daquele tipo. As demais Colônias estarão **Fechadas**. São chamados **Objetos Disponíveis** os Objetos que habitam Colônias abertas.

Toda Colônia, exceto a global, é **Constrita** por um Objeto que habita uma Colônia do tipo imediatamente superior ao tipo dessa Colônia. Objetos Disponíveis que **Constringem** uma Colônia podem ser **Considerados** para abrir tais Colônias. O conjunto de Objetos que devem ser considerados para abrir uma Colônia constitui o **Contexto** dessa Colônia. Qualquer identificador de Objeto somente será reconhecido em seu Contexto.

VERSÕES: Uma Colônia pode ser instanciada, gerando múltiplas **Variantes** de uma mesma Colônia. As variantes podem ser usadas temporariamente, como **Alternativas** de um aspecto da informação, ou podem vir a se tornar **Versões**, quando então ficam "congeladas", não podendo mais ser alteradas ou eliminadas da BD.

META-BASE DE DADOS: Uma BD Completa usando o MRO é chamada **Base Unificada**. Esta é composta de uma **Meta-Base de Dados**, que armazena o esquema da base da aplicação, e de uma **Base de Dados** propriamente dita.

A BD armazena as Colônias de Objetos, os Objetos, Relacionamentos e Atributos. A Meta-Base de Dados armazena os Tipos de Colônias, Tipos de Objetos, Tipos de Relacionamentos, Tipos de Atributos, etc, o que caracteriza o **Esquema** da BD. A Meta-Base de Dados é armazenada como uma Colônia, cujo tipo é controlado pelo Sistema, e denominada **Colônia Esquema**. O esquema da própria Colônia Esquema é pré-definido e constitui o **Meta-Esquema** do MRO. O Objeto que constribe a Colônia Esquema é

controlado pelo sistema e habita a Colônia Global. Este Objeto é do tipo denominado **BESQUEMA**, também controlado pelo sistema. Podem existir múltiplas Variantes da Colônia Esquema. As variantes que são promovidas a Versões podem passar a atuar como esquemas da BD. Cada Colônia é definida apenas por uma Versão do Esquema. Um Esquema Conceitual pode ser armazenado em partes, em diversas colônias esquemas. Nesse caso, uma única colônia esquema, denominada Colônia Esquema Inicial, deverá estar sempre ativa. As demais Colônias Esquemas podem ser atingidas ou desativadas livremente, modificando o esquema disponível num dado instante para um usuário.

O acesso à Meta-Base de Dados é feito da mesma maneira que o acesso à BD, permitindo que o esquema de dados (Variante) possa ser dinamicamente atualizado.

O modelo fornece meios para se representar graficamente o esquema e as possíveis instâncias de objetos, quais sejam: o DRO (Diagrama de Representação de Objetos) e o DRI (Diagrama de Representação de Instâncias). O DRO é utilizado para modelar graficamente o esquema da BD. Sendo assim, permite ao projetista representar as associações entre tipos de objetos e tipos de relacionamentos. O DRI serve para representar instâncias de objetos armazenados na BD, sendo normalmente utilizado para atender ao propósito de exemplificação ou indicação de objetos complexos.

Para que o modelo possa ser melhor compreendido, é interessante que se apresente a modelagem dos dados de uma aplicação bastante simplificada.

Considere-se uma aplicação onde se queira armazenar as informações sobre os alunos de graduação e as matérias que irão cursar em cada departamento de uma universidade. O DRO desta aplicação aparece na figura 2.

Figura 2: DRO de uma aplicação

Os círculos com traços horizontais representam os **Tipos de Objetos**. No exemplo tem-se quatro Tipos de Objetos: Universidade, Departamento, Matéria e Aluno. Abaixo do Tipo de Objeto, tem-se o nome da colônia na qual instâncias desse tipo de objeto irão Habitar. Os Tipos de Objetos: Departamento e Aluno, estão **constritos na colônia Universidade**. O Tipo de Objeto Universidade **constringe a colônia Universidade**. O tipo de objeto universidade, por sua vez, está **constrito à colônia global**. A colônia global é uma colônia pré-estabelecida no modelo e é a partir dela que tipos de objetos são definidos.

Os arcos entre os tipos de objetos representam os **Tipos de Relacionamentos**. No MRO, todos os Tipos de Relacionamentos devem possuir a indicação do seu oposto. Tal fato é representado, no DRO, pelo arco interceptado por uma reta; colocando-se à esquerda e à direita dessa reta as respectivas Modalidades de primeiro grau (Cursa e Cursada).

Os **tipos de atributos** que objetos do tipo Matéria possuem são: nome, sala e professor. O círculo precedendo o tipo de atributo nome, indica que o mesmo é um **sinônimo** para o identificador de objetos daquele tipo. Sendo assim, objetos do tipo Matéria podem ser acessados através do seu identificador ou pelo seu sinônimo. Embora não ocorra no exemplo, os Tipos de Relacionamentos, da mesma forma que os tipos de objetos, também podem ter atributos. No entanto, não possuem sinônimos, uma vez que os Relacionamentos somente podem ser acessados via objetos, e não através de identificadores.

Apresenta-se, na figura 3, um possível DRI para o DRO acima.

Neste DRI, tem-se apenas uma instância de objeto do tipo Universidade, Departamento e Matéria, e duas instâncias de objetos do tipo Aluno. É importante observar que o sinônimo para o identificador SCE-144 é Tópicos de Computação, e que o acesso a esse objeto pode ser feito tanto pelo identificador quanto pelo seu sinônimo. O acesso a outros objetos pode ser feito de forma semelhante. Uma outra alternativa para se acessar um objeto é através do mecanismo de acesso navegacional, isto é, uma vez que um objeto é acessado, pode-se fazer acesso a outros objetos que estejam associados através de relacionamentos.

Figura 3: Um possível DRI para o DRO da figura 2

Assim como sinônimo, os tipo de atributos podem ter outras Características. A seguir, são listadas todas as Características definidas no MRO juntamente com seus símbolos gráficos associados.

- **Comentário.** Corresponde a informações organizadas de forma textual.
- ▣ **Estrutura de Dados.** São agrupamentos de informações com significado completo que normalmente são representados em estruturas de arranjo de dados (vetores, matrizes, pilhas, etc.), sobre os quais podem ser definidas operações pertinentes.
- △ **Gráfico.** São informações que possuem significado geométrico.
- ▤ **Imagem.** São informações normalmente estruturadas em mapa de bits.

- ♦ **Procedimento.** São fórmulas ou parâmetros associados a atividades que devem ser exercidas de uma forma pré-determinada pelo sistema.
- **Propriedade.** Quando um tipo de atributo não possuir nenhuma característica especial, que o distingue dos demais, será considerado como tipo de atributo normal, referenciado simplesmente como Propriedade.
- ◇ **Regra.** São atributos que possuem uma condição e uma ação associadas.
- **Sinônimo.** Indica um sinônimo para o identificador de um objeto.
- ≈ **Som.** Representa uma informação sonora.
- 🕒 **Tempo.** Indicação de tempo absoluto ou relativo associado aos objetos e relacionamentos. Pode ser indicado de várias formas, desde especificações de frações de unidades de tempo até indicações de data, ou faixas de tempo.
- 😊 **Visualização.** São atributos que tratam de informações inerentes à interface para visualização.

É apresentado a seguir o Meta-Eschema do MRO. Convém destacar que o Meta-Eschema apresentado, na figura 4, ainda não descreve em detalhes os conceitos sobre Atributos com Características Gráficas que foram desenvolvidos neste trabalho. A descrição detalhada será apresentada no capítulo 6.

Figura 4: Meta-Esquema do MRO

3.4 - Classificações

3.4.1 - Classificação Segundo Dittrich

Segundo Dittrich [21], um SGBD será orientado a objetos se possuir a capacidade de representar uma entidade do mundo real, qualquer que seja sua estrutura e/ou complexidade, por exatamente um objeto. Além disso, classifica-os como possuindo **orientação estrutural** e **orientação operacional**.

Um SGBD orientado a objeto é **orientado estruturalmente** se possuir a capacidade de definir entidades arbitrariamente complexas, que serão acessadas através de operadores genéricos, sem que haja restrições em qualquer nível de abstração. É **orientado operacionalmente** se possuir a capacidade de definir operadores próprios que acessam um objeto complexo como um todo, enquanto que os tipos básicos componentes do objeto complexo não são acessíveis para o usuário externo do objeto. A união dessas duas classificações é denominada **orientação comportamental**.

De acordo com esta classificação, um SGBD apoiado no MRO, é um SGBD orientado a objeto estruturalmente. A orientação comportamental pode ser obtida no nível da aplicação, já que o comportamento de um objeto complexo é dependente da aplicação.

3.4.2 - Classificação Segundo Kim

Segundo Kim [39], um SGBDOO (SGBD Orientado a Objeto) deve possuir um modelo de dados orientado a objetos com, no mínimo, os seguintes conceitos: Objeto e Identificador de Objeto, Atributos e Métodos, Classes, Hierarquia de Classes e Herança. Sendo assim, segundo esta classificação, um SGBD apoiado no MRO é orientado a objeto. Isto, porque o MRO possui conceitos equivalentes aos citados por Kim:

- **Objeto e Identificador de Objetos:** o MRO admite estes conceitos sem nenhuma modificação;
- **Atributos e Métodos:** no MRO, um objeto pode possuir atributos com diversas características, quais sejam: de sinônimo, de texto, de tempo, de regra, de procedimento, etc. Assim, os métodos pertencentes a um objeto são representados, no MRO, através de atributos com características de regra ou procedimento. Um melhor detalhamento a respeito de características de atributo será descrito na próxima seção (3.4 - Descrição do MRO);

- **Classes:** Uma classe especifica o agrupamento de todos os objetos que compartilham um mesmo conjunto de atributos e métodos. No MRO, um tipo especifica o agrupamento de todos os objetos que compartilham um mesmo conjunto de atributos. Levando-se em conta que atributos podem possuir características de regra ou procedimento, os conceitos de classe e tipo são equivalentes;
- **Hierarquia de Classes e Herança:** Dado que classes e tipos são equivalentes, a hierarquia de classes torna-se hierarquia de tipos, subclasse torna-se subtipo e superclasse torna-se tipo mestre do subtipo considerado.

3.4.3 - Classificação Segundo Atkinson

Atkinson e outros [3] descrevem as principais características que um SGBD deve ter para ser considerado um SGBDOO. Essas características foram separadas em três grupos:

- **Obrigatórias:** são as características mínimas que um SGBD deve ter para ser considerado um SGBD orientado a objeto. Estas características são: objetos complexos, identificador de objeto, encapsulamento, tipos ou classes, herança, "overriding" combinado com "late binding", extensibilidade, completeza operacional, persistência, gerenciamento do armazenamento secundário, concorrência, restauração e facilidades de consultas não planejadas;
- **Opcionais:** são características que, somadas às obrigatórias, melhoram o sistema. São elas: herança múltipla, verificação de tipos e inferência, distribuição, transações de projeto e versões;

- **Abertas:** são aquelas características que podem ser incorporadas ao sistema de acordo com o projetista, tais como: o paradigma de programação, sistema de representação, sistema de tipos e uniformidade.

Considerando o grupo das obrigatórias, pode-se verificar o seguinte:

- (1) o sistema de BD orientado a objeto deve ter, inicialmente, as características de qualquer SGBD convencional, tais como: persistência, gerenciamento do armazenamento secundário, concorrência, restauração da BD e facilidades de consultas não planejadas; e
- (2) características que estão associadas às linguagens de programação orientadas a objetos, tais como: objetos complexos, identificador de objetos, encapsulamento, tipos ou classes, herança, "overriding" combinado com "late binding", extensibilidade e completeza computacional.

As características descritas no item (1) são bastante contundentes; por outro lado, algumas características descritas no item (2) são um tanto duvidosas. O fato é que os conceitos ali expostos dizem respeito a três coisas distintas, quais sejam: modelo de dados, implementação do modelo e linguagem de consulta. Sendo assim, pode-se fazer uma subclassificação da seguinte forma:

- (a) modelo de dados: fornece características como: objetos complexos, identificador de objetos, tipos ou classes e herança;
- (b) implementação do gerenciador apoiado no modelo: fornece características como: encapsulamento, "overriding" combinado com "late binding" e extensibilidade;
- (c) linguagem de consulta: fornece a característica de completeza computacional e consultas não planejadas.

O MRO, como um modelo de dados, possui todas as características listadas em (a). Quanto à implementação do modelo, o artigo sugere que se utilize uma linguagem de programação orientada a objeto e o paradigma de orientação a objeto. Isto pode ser um fator benéfico para a codificação, manutenção e extensibilidade de SGBDs; no entanto, as características de encapsulamento e "overriding" combinado com "late binding", listadas em (b), talvez não sejam de fundamental importância para que um SGBD seja orientado a objeto. Isso porque um sistema de BD pode ser implementado, com maior ou menor dificuldade, utilizando-se de alguma linguagem de programação não orientada a objeto e, ainda assim, fornecer aos usuários do sistema, todas as facilidades e conceitos de orientação a objetos descrito na seção 3.2 (Conceitos de Orientação a Objetos).

A completeza computacional, em (c), transmite a idéia de que uma linguagem de consulta possa, por exemplo, expressar qualquer função computável, isto é,

funções que ao serem executadas num computador sempre terminam e, ainda, que a linguagem permita acessar uniformemente todos os conceitos manipulados. Esta característica é bastante desejável; no entanto, pode-se colocar em dúvida a sua necessidade para que um SGBD seja considerado orientado a objeto. Convém ressaltar que, linguagens de consulta a BD devem, a princípio, dar suporte à manipulação de duas coisas: elementos descritos no esquema e dados, segundo um modelo de dados. Sendo assim, acredita-se que um SGBD apoiado no MRO seja orientado a objetos, mesmo que sua linguagem de consulta não forneça mecanismos para expressar todas as funções computáveis.

3.5 - Considerações Finais

Verificou-se neste capítulo que um SGBD apoiado no MRO será Orientado a Objeto e, como tal, permite que dados de Aplicações não Convencionais sejam representados e manipulados coerentemente.

Os Atributos com Características Gráficas serão discutidas com maiores detalhes a partir do capítulo 6.

Capítulo 4

Modelagem Geométrica

4.1 - Considerações Iniciais

O objetivo deste capítulo é buscar na Modelagem Geométrica as necessidades de armazenamento de informações gráficas. Para tanto, inicialmente, na seção 4.2, é feita uma pequena discussão sobre a área de Modelagem Geométrica. Na seção 4.3, discutem-se dois modelos para representação de sólidos muito utilizados em sistemas de CAD. Na seção 4.4, os principais objetivos dos modelos gráficos são discutidos, e aspectos a respeito da padronização dos modelos gráficos são também abordados.

4.2 - A Modelagem Geométrica

O termo **modelagem geométrica** vem sendo empregado desde 1970 após o rápido desenvolvimento das tecnologias de computação gráfica, CAD e CAM. A modelagem geométrica tem como objetivo principal utilizar métodos para definir a forma e as características geométricas de um objeto, produzindo como resultado um **modelo geométrico** do objeto. O termo **objeto** aqui é empregado como sinônimo de entidades físicas do mundo real, que possuem alguma forma de representação gráfica. Os métodos da modelagem geométrica são uma síntese de técnicas encontradas em vários campos de pesquisas, tais como: Geometria Analítica e Descritiva, Topologia, Análise Numérica, Cálculo Vetorial, Métodos Matriciais e outros.

O modelo geométrico do objeto faz parte de uma representação mais completa de um objeto, denominada **modelo do objeto**. Um modelo do objeto é necessário quando a experimentação para se estudar o objeto é onerosa, trabalhosa ou impraticável. O modelo do objeto substitui o próprio objeto, facilitando a análise do objeto sendo estudado. Segundo Mäntylä [46], um modelo do objeto é composto por **dados não geométricos** e pelo **modelo geométrico**. Sendo assim, a modelagem geométrica supre as necessidades do modelo de objeto fornecendo métodos para que o **modelo geométrico** do objeto possa ser descrito.

Existem quatro tipos de modelos geométricos: 1) modelo de sólido, 2) modelo gráfico, 3) modelo de forma e 4) modelo de superfície. Cada um procura atender às necessidades específicas do projetista quanto à representação do objeto:

- **Modelos de Sólidos:** são representações completas de objetos sólidos, isto é, representações que são adequadas para responder algoritmicamente a questões geométricas, sem a ajuda interativa do usuário. Estes modelos são muito utilizados em sistemas CAD e CAM para Engenharia. Exemplos de sistemas gráficos que utilizam os modelos de sólidos são: PADL-2 [8], GDP [72] e SYNTHAVISION [26];
- **Modelos Gráficos:** representam o desenho do objeto ao invés do próprio objeto. A maioria dos sistemas que fornecem auxílio à desenho, tal como AutoCad [4], adotam este modelo;
- **Modelos de Formas:** Representam imagens de objetos. Eles podem ser coleções desestruturadas de elementos de imagens, ou podem ter alguma estrutura interna para auxiliar as operações de processamento de imagens. Este modelo é utilizado em sistemas voltados à área de Processamento de Imagens, tais como sistemas de análise térmica, automação e visão de robôs, tomografia computadorizada e outros [1];
- **Modelos de Superfícies:** Fornecem informações detalhadas sobre superfícies curvas, mas nem sempre retornam informações suficientes para determinar todas as propriedades do objeto contornado pela superfície. No entanto, as técnicas utilizadas para se criar modelos de superfície estão se tornando parte importante do processo de modelagem de objetos sólidos para projeto, análise, manufatura e simulação [71].

Dentre os modelos citados acima, os Modelos de Sólidos e os Modelos Gráficos são os mais utilizados em sistemas de CAD. Devido a esse fato, esses dois modelos serão individualmente estudados nas próximas seções.

4.3 - A Modelagem de Sólidos

Uma área especificamente voltada ao estudo de modelos de sólidos é a área de Modelagem de Sólidos. Segundo Miller [49], muitas abordagens de representação têm sido desenvolvidas no campo da Modelagem de Sólidos. Duas destas abordagens dominam o campo por causa da generalidade e a disponibilidade de um grande número de algoritmos. Essas abordagens são CSG e B-rep:

- **Geometria Construtiva de Sólidos:** Esta abordagem é normalmente conhecida como CSG ("Constructive Solid Geometry"). A técnica CSG

adota a abordagem de **blocos construtores** para modelar sólidos, isto é, o usuário desse modelo utiliza apenas instâncias de primitivas de sólidos (blocos) e um conjunto de operadores booleanos (construtores). Uma maneira de se representar um modelo de sólido CSG é através da árvore-CSG [59], que pode ser definida em BNF ("Backus Naur Form") como:

```

<árvore-CSG> ::= <primitiva>
                | <árvore-CSG> <operador de conjunto> <árvore-CSG>
                | <árvore-CSG> <deslocamento rígido>
<primitiva> ::= cubo
                | cilindro | cone | esfera | torus | ..
<operador de conjunto> ::= união
                | intersecção | diferença
<deslocamento rígido> ::= rotação | translação

```

Na figura 5 é mostrada uma árvore-CSG de uma peça. As subárvores que não são folhas representam o resultado da aplicação de operador de conjunto e de deslocamento rígido sobre um tipo de primitiva sólida, no caso um paralelepípedo. A raiz da árvore-CSG representa a peça como um todo.

Figura 5: Árvore CSG

- **Representação por Fronteira:** Esta abordagem é conhecida com B-rep ("Boundary Representation"). Um objeto sólido é representado através de uma estrutura de dados gráfica, onde é descrito o seu contorno em termos de faces, arestas e vértices. Para se compreender melhor, a figura 6 ilustra o exemplo de uma representação B-rep de uma cantoneira.

Uma questão que pode ser levantada é: "Qual a melhor abordagem a ser utilizada para Ambientes de Projeto?". Segundo Miller [49], do ponto de vista da construção do modelo, nenhuma das abordagens é claramente melhor que a outra. As operações Booleanas e de "sweeping" (deslocamento ou rotação de uma forma geométrica planar para se obter um sólido), por exemplo, são duas técnicas de construção poderosas que tanto a

Figura 6: Representação B-rep de uma peça

abordagem CSG quanto a B-rep podem admitir. No entanto, do ponto de vista da edição do modelo (interface com o usuário), a situação é um pouco diferente. Existem técnicas eficientes de edição baseadas no CSG que não podem ser implementadas de maneira razoável em abordagens B-rep, pois estas não mantêm a informação de como o objeto sólido foi construído. A habilidade de modificar, realocar e remover primitivas ou subárvores numa árvore CSG, por exemplo, permitem maiores modificações no modelo de sólido com poucas entradas ("input"). Por outro lado, existem também métodos poderosos baseados em B-rep que não podem ser razoavelmente suportados num contexto puramente CSG. Por exemplo, para fazer mudanças locais no modelo, como a operação de chanfrar arestas e vértices, o usuário necessita manipular faces, vértices e arestas, que são precisamente os elementos de uma estrutura de dados B-rep. Dessa maneira, o autor sugere uma arquitetura híbrida, composta pelas duas abordagens, CSG e B-rep com o objetivo de obter num único sistema, o potencial de ambos.

4.4 - Modelos Gráficos

Os modelos gráficos representam as informações geométricas, em última instância, em termos de pontos e retas. Como muitas das técnicas para processar modelos gráficos são de interesse para a Modelagem de Sólidos, tornou-se necessário estudá-las.

A construção de um modelo gráfico é baseada na seleção das primitivas de modelagem, e na coleção de procedimentos de modelagem disponíveis para sua instanciação e manipulação. O conjunto de primitivas e procedimentos podem variar de acordo com o padrão gráfico que está sendo utilizado. Segundo Cunha [18], existem vários padrões gráficos adotados internacionalmente, além de outros em estudo. A seguir, são listados alguns deles, com suas principais características.

- GKS ("Graphical Kernel System"):
 - ⇒ Interface gráfica padrão para programas aplicativos;
 - ⇒ Adotada como norma americana pelo ANSI ("American National Standards Institute"), em 1985;
 - ⇒ Adotado como norma internacional pela ISO ("International Organization for Standards"), em 1985.
- GKS-3D ("Graphical Kernel System for Three Dimensions"):
 - ⇒ Extensão tridimensional do GKS;
 - ⇒ Proposta de norma internacional em estudo na ISO.

- CGI ("Computer Graphics Interface"):
 - ⇒ Especificação de técnicas de interfaceamento para uso padronizado de dispositivos gráficos;
 - ⇒ Objeto de estudo no âmbito da ISO e ANSI.
- PHIGS ("Programmer's Hierarchical Interactive Graphics System"):
 - ⇒ Interface gráfica bi e tridimensional para programas aplicativos;
 - ⇒ Adotado como norma na ISO e ANSI.

4.5 - Considerações Finais

Como foi discutido, os modelos de sólidos e os modelos gráficos são os mais utilizados em sistemas de CAD, sendo que os modelos de sólidos sustentam-se nos modelos gráficos para que suas primitivas de modelagem possam ser construídas. Sendo assim, verifica-se que as informações gráficas que devem ser armazenadas são, em última instância, as primitivas de modelagem que compõem os Modelos Gráficos. Os Modelos Gráficos, por sua vez, sustentam-se em padrões gráficos tais como GKS, PHIGS e CGI.

Logo, o procedimento de trabalho, a ser adotado no capítulo a seguir, será o de modelar alguns padrões gráficos, a fim de capturar os conceitos e a semântica dos dados manipulados, e tentar abstrair um modelo genérico para armazenamento de informações gráficas.

Capítulo 5

Modelagem do GKS e PHIGS em MRO

5.1 - Considerações Iniciais

O objetivo deste capítulo é modelar dois padrões gráficos através do MRO e, assim, determinar um modelo geral de representação de informações gráficas. O modelo geral, incorporado ao Meta-esquema do MRO, pode definir uma nova Característica de Atributo (Característica Gráfica), para que informações gráficas possam ser convenientemente representadas e manipuladas pela BD.

Nas seções 5.2 e 5.3, foram modelados os padrões gráficos GKS ("Graphical Kernel System") [33] e PHIGS ("Programmer's Hierarchical Interactive Graphics System") [19]. O objetivo de modelar, em termos do MRO, esses dois padrões gráficos (GKS e PHIGS) foi o de abstrair um modelo genérico de representação de informações gráficas para o GEO-MRO. Este modelo genérico encontra-se na seção 5.4.

5.2 - Modelagem GKS

O GKS é um sistema que permite ao usuário definir gráficos através de alguma linguagem de alto nível como FORTRAN, BASIC ou C, fazendo chamadas a rotinas da biblioteca do GKS. A modelagem inicialmente realizada do GKS (figura 7) demonstra que alguns gráficos usuais, tais como: Barra e Setor, podem ser facilmente confeccionados. Um aspecto importante desse sistema é que a forma de acesso a periféricos é normalizada através do Objeto VDI ("Virtual Device Interface"). Através deste Objeto, os Objetos Gráficos podem ser editados, impressos ou armazenados. O VDI gerencia os Drivers individuais de cada periférico, fazendo com que os comandos de acesso normalizados sejam convenientemente interpretados pelos periféricos, tais como: impressora, Plotter e Display.

Dentre os periféricos, o Metafile permite que os gráficos sejam armazenados num formato padrão VDM ("Virtual Device Metafile"). Este formato fornece ao GKS maior portabilidade dos gráficos, pois armazena informações de como construir o gráfico, ao invés de armazenar dados do gráfico.

Figura 7: Modelagem Inicial do GKS

Além dos gráficos usuais, o GKS permite que o usuário defina os seus próprios gráficos (vide figura 8). Para tanto, o sistema fornece algumas Primitivas Geométricas (PG), tais como: "PolyLine", "PolyMaker" e Texto, que podem ser livremente selecionadas, através do "Pick", para compor um Segmento. O Segmento é um Objeto que pode ser composto por PG e por outros Segmentos. Uma vez que Segmentos estejam definidos, eles são tratados como gráficos que podem ser armazenados em Metafile, visualizados ou impressos.

5.4 - Proposta de um Modelo Geral

Apesar das diferenças de abordagem encontradas nos padrões gráficos, verifica-se que existem similaridades entre o objeto Segmento do GKS e o objeto Estrutura do PHIGS. Tais similaridades fornecem elementos para a elaboração de um modelo genérico para representação de informações gráficas para o GEO-MRO, como mostra a figura 10.

Figura 10: Modelo geral de um núcleo gráfico

5.5 - Considerações Finais

Neste capítulo, foram modelados os padrões gráficos GKS e PHIGS em termos do MRO. Obteve-se, a partir dessas duas modelagens, um Modelo Geral capaz de representar os principais tipos de objetos encontrados em ambos os padrões.

Capítulo 6

Atributos com Características Gráficas

6.1 - Considerações Iniciais

Neste capítulo, define-se o que são Atributos com Características Gráficas, incorporando-se ao MRO os conceitos identificados como fundamentais na modelagem genérica de sistemas gráficos apresentada no capítulo 5. Dessa forma, o MRO passa a prover recursos de modelagem gráfica de maneira padronizada e integrada aos recursos já existentes.

6.2 - Composição dos Atributos com Características Gráficas

De acordo com o Modelo Geral de um Núcleo Gráfico apresentado na figura 10, pode-se dizer que a base de dados deve ter condições de manipular os recursos, presentes na modelagem, para que ela possa interagir com a maioria dos núcleos gráficos existentes. Os recursos ali presentes são: "Bundles" (maço) de Atributos, Primitiva, Estrutura, Periférico de Entrada e Periférico de Saída.

Os periféricos de Entrada e Saída referem-se, na realidade, à controladores de periféricos ("Drivers") que são normalmente partes de uma configuração de um aplicativo gráfico específico, sendo permitido, portanto, desconsiderá-los como componentes de um atributo com características gráficas. Por outro lado, os recursos restantes devem ser considerados, pois são elementos que definem os objetos gráficos quanto ao seu aspecto ("Bundles" de Atributo), a sua forma (Primitiva) e sua organização (Estrutura). Um outro elemento ainda não citado, porém de igual importância, é a matriz de transformação, presente no Modelo Geral de um Núcleo Gráfico na forma de um atributo de relacionamento. Este elemento é essencial para que se possa especificar a localização, orientação e escala, tanto de primitivas quanto de estruturas no momento de sua visualização.

A seguir, será feita uma descrição sucinta dos elementos que devem compor um atributo com características gráficas.

- **Primitiva.** As Primitivas especificam uma abstração de classificação para tipos de gráficos tais como: Ponto, "Polyline" e Arco.

- **"Bundle" de Atributo.** Os "Bundles" de Atributos definem um conjunto de atributos de primitivas ou estruturas. Por exemplo, estes atributos podem indicar a espessura e o estilo de uma linha.
- **Matriz de Transformação.** Matrizes de Transformação especificam a classe de matrizes denominadas homogêneas que são utilizadas para realizar transformações geométricas de escala, rotação e translação de primitivas ou estruturas gráficas.
- **Estrutura.** As estruturas definem o agrupamento de Primitivas, Estruturas, "Bundles" de Atributos e Matrizes de Transformação, organizadas de maneira que possam realizar-se operações de inserção, modificação, remoção e visualização de seus componentes.

6.3 - Especificação dos Elementos que compõe Atributos com Características Gráficas

Nesta seção é especificado em detalhes cada um dos elementos que compõe um Atributo Gráfico (ou Atributo com Característica Gráfica) definidos anteriormente.

6.3.1 - Especificação das Primitivas Gráficas

Os tipos de primitivas gráficas podem variar conforme o núcleo gráfico que estiver sendo considerado. Dessa forma, fez-se a escolha de alguns tipos de primitivas que farão parte do MRO, de acordo com os tipos de primitivas encontrados na maioria dos núcleos gráficos existentes atualmente. São eles: Ponto, "Polyline", Arco, Texto e Elipse. Além destes, escolheu-se ainda Bezier pois curvas e superfícies de Bezier estão sendo empregadas cada vez mais em aplicações de projeto [56].

A primitiva gráfica **Ponto** é definida pelas coordenadas do sistema cartesiano (x, y), onde x corresponde à abscissa e y à ordenada.

A primitiva **"Polyline"** é definida por uma seqüência finita de pontos para formar um polígono.

A primitiva **Arco** é subentendida como sendo um arco de circunferência definido por três parâmetros:

- (x_c, y_c) - centro da circunferência
- R - raio
- a_i - ângulo inicial em relação à abscissa (sentido anti-horário)
- a_f - ângulo final em relação à abscissa (sentido anti-horário)

A primitiva **Texto** é definida pelos seguintes parâmetros:

- n - número de caracteres no texto
- (x, y) - ponto que especifica o início do texto

A primitiva **Ellipse** é definida através das coordenadas de um retângulo que envolve a ellipse. Para se definir um retângulo, é necessário apenas dois pontos P_1 e P_2 , conforme ilustra a figura 11.

Figura 11: Definição de uma elipse

A primitiva **Bezier** é definida por n pontos de controle e pode ser implementada por uma rotina que obedece o seguinte algoritmo [54]:

```

Algoritmo Bezier dados  $n$  pontos de controle ( $P_0, P_1, \dots, P_{n-1}$ )
  posicione em  $P_0$ 
   $u=0.01$ 
  repita
    para  $j=1$  até  $(n-1)$  faça
      para  $i=0$  até  $(n-j-1)$  faça
         $P[i] = u \cdot P[i+1] + (1-u) \cdot P[i]$ 
      fim
    fim
    trace um seguimento de reta da posição corrente até  $P[i]$ 
     $u=u+0.01$ 
  até  $u=1$ 
fim
  
```

6.3.2 - Especificação de Atributos de Primitivas Gráficas

As primitivas gráficas podem ser apresentadas, visualmente, de diversas formas. Por exemplo, um segmento de reta pode aparecer com espessuras diferentes, tracejado ou pontilhado; um texto pode ser desenhado em diferentes tamanhos, espessuras, fontes, e todos com várias cores. Os "Bundles" de Atributos servem exatamente para esse propósito, ou seja, no momento em que um texto ou uma linha for mostrada, os atributos utilizados serão aqueles definidos como correntes no "Bundle". Para atingir o propósito deste trabalho, os atributos escolhidos foram cor e estilo. O atributo cor pode especificar a cor tanto de primitivas gráficas quanto de texto. O atributo estilo pode especificar, por exemplo, a espessura ou o tipo de linha de um segmento de reta ou especificar o tipo de fonte para textos. Como o GEO não "mostra" os gráficos, em um dispositivo físico, mas apenas armazena todas as suas informações, repassando-as para um Núcleo Gráfico, caberá à aplicação que faz uso deste atributo, escolher o significado conveniente para o atributo estilo.

6.3.3 - Especificação das Matrizes de Transformação

Uma das famílias de manipulação mais utilizadas em programas gráficos é das transformações geométricas [69]. As transformações geométricas são operações matemáticas que permitem alterações uniformes de uma imagem sendo definida sobre um sistema de coordenadas. Estas alterações afetam o aspecto que o desenho irá assumir ao ser apresentado sob diferentes orientações e escalas.

Uma maneira sistemática de se realizar transformações geométricas é através de Matrizes de Transformação. Este método faz com que a transformação geométrica possa ser obtida pela multiplicação das coordenadas do ponto a ser transformado, dispostas em uma matriz de linha, pela matriz de transformação. De acordo com os valores dos elementos da matriz de transformação, o resultado obtido pode ser uma rotação, uma escala, uma translação ou qualquer combinação destas transformações.

Nesse trabalho, adota-se Matrizes de Transformação para coordenadas bidimensionais homogêneas. Assim, as matrizes de transformação de escala, translação e rotação são, respectivamente:

$$\text{Escala} = \begin{bmatrix} E_x & 0 & 0 \\ 0 & E_y & 0 \\ 0 & 1 & 1 \end{bmatrix}$$

E_x e E_y são os fatores de escala para as coordenadas x e y respectivamente.

$$\text{Translação} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ T_x & T_y & 1 \end{pmatrix}$$

T_x e T_y indicam o deslocamento de um ponto nas direções x e y respectivamente.

$$\text{Rotação} = \begin{pmatrix} \cos \alpha & \sin \alpha & 0 \\ -\sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

O ângulo α indica a rotação de um ponto em torno da origem das coordenadas cartesianas.

A combinação de operações sucessivas pode ser obtida pela matriz produto das matrizes de cada operação elementar, na ordem em que as operações devem ser aplicadas. O resultado é uma matriz cuja terceira coluna é sempre o vetor $(0, 0, 1)$. Dessa forma, armazena-se na base apenas as duas primeiras colunas.

É importante destacar que a matriz de transformação é armazenada e utilizada pelo gerenciador de base de dados em operações de recuperação de dados. Assim, para a manipulação de atributos gráficos, a base mantém sempre uma **matriz de transformação corrente**, que é aplicada sobre todas as coordenadas recuperadas (ou gravadas) pela base. Isso é importante em particular para as operações de busca baseada em valores de atributos gráficos.

6.3.4 - Especificação da Estrutura Gráfica

Um atributo com características gráficas é na realidade uma seqüência de primitivas, matrizes de transformação e "bundles" de atributos. Partes dessa seqüência podem ocorrer repetidas vezes em um mesmo atributo ou em outros atributos. Assim, foi criado o conceito de estrutura gráfica, a qual nada mais é do que a lista ordenada de primitivas gráficas, matrizes de transformação e "bundles" de atributos que compõem um gráfico.

Note-se que uma estrutura gráfica pode ser composta por outras, ou compor outras, não estando necessariamente vinculada a um particular valor de atributo gráfico. No entanto, cada valor de atributo gráfico consiste de exatamente uma estrutura gráfica.

6.4 - O Meta-Esquema do MRO

Definidos os elementos que compõe Atributos com Características Gráficas, tornou-se possível estender o Meta-Esquema, cujo DRO é apresentado na figura 4 do capítulo 3, com esta nova característica de atributo. A figura 12 ilustra este novo Meta-Esquema, porém, com destaque apenas para os conceitos de Atributos com Características Gráficas.

Figura 12: O novo Meta-Esquema do MRO

6.5 - Considerações Finais

Neste capítulo, foi definido em detalhes o que é um Atributo com Características Gráficas para o Modelo de Representação de Objetos. A partir dessa definição, foi elaborado um novo Meta-Esquema. Este novo Meta-Esquema, incorporado ao GEO, irá permitir às aplicações que fizerem uso deste gerenciador, o acesso aos recursos de modelagem e manipulação de Atributos com Características Gráficas suportados agora pelo modelo.

Capítulo 7

Aspectos da Implementação

7.1 - Considerações Iniciais

Neste capítulo são definidas as operações de manipulação dos elementos que compõem os atributos com características gráficas, estabelecidos no capítulo anterior.

Para isso, foi levado em consideração o fato de que tais operações não fariam parte de um sistema isolado, mas sim de um sistema maior denominado GEO, apoiado no modelo MRO. Conseqüentemente, foram feitos esforços para que a forma de se fazer acesso, entre os vários conceitos do MRO implementados no GEO, continuasse homogênea após a implementação das rotinas de acesso à atributos com características gráficas. Assim, para a definição do conjunto de operações sobre atributos com características gráficas guiou-se, tanto quanto possível, pela forma de se fazer acesso já definida para as outras características de atributo.

7.2 - O Paradigma de Acesso Navegacional

Um dos critérios que norteiam a implementação do GEO é o paradigma de acesso navegacional, citado brevemente no capítulo 3. Para ilustrar este conceito, considere uma operação de consulta no GEO, que possui o seguinte esquema (DRO) e as correspondentes instâncias (DRI).

Seguindo-se esse paradigma, verifica-se que a Pessoa João Trabalha na Empresa IBM. Neste exemplo, João e IBM são identificadores de objeto; e Pessoa e Empresa os tipos correspondentes.

Utilizando o identificador João, pode-se realizar a operação de posicionamento sobre o objeto para torná-lo corrente. Uma vez que exista um objeto corrente, neste caso o objeto (do tipo Pessoa) João, tornam-se factíveis quaisquer operações sobre atributos e relacionamentos que este objeto tenha. No caso de se fazer uma consulta para se obter uma lista de endereços de empresas onde João trabalha, a seqüência necessária de operações seria a seguinte:

Algoritmo Listar todos os endereços de empresas onde o João trabalha

Posicionar no objeto João;

Posicionar no primeiro relacionamento Trabalha;

Enquanto relacionamento = Trabalha faça

Obter o objeto destino do relacionamento;

Salvar o estado corrente;

Posicionar no objeto destino;

Posicionar no atributo do tipo Endereço;

Ler a propriedade Endereço;

Mostrar formatado o valor da propriedade Endereço;

Restaurar o estado anterior;

Avançar para o próximo relacionamento;

fim-se

Fim-Algoritmo.

Este paradigma é adotado também para atributos, conforme pode ser observado a seguir, na lista de operações já existentes no GEO para atributos com característica de propriedade:

1. Colocar uma propriedade no objeto ou relacionamento corrente;
2. Retirar a propriedade corrente;
3. Posicionar na propriedade indicada;
4. Ler a propriedade corrente;
5. Ler o tipo de dado da propriedade corrente;
6. Posicionar na primeira propriedade do tipo de atributo indicado;
7. Avançar para a propriedade seguinte;
8. Codificar um valor de uma propriedade a partir de uma cadeia;
9. Formatar um valor de propriedade como uma cadeia.

7.3 - Definição das Operações sobre Atributos Gráficos

Como pretende-se definir a interface de programação do GEO para atributos gráficos de maneira semelhante às já existentes para outras características, foi elaborada a seguinte lista de operações sobre atributos com característica gráfica:

1. Colocar um gráfico no objeto ou relacionamento corrente;
2. Retirar o gráfico corrente;
3. Posicionar no gráfico indicado;
4. Ler o gráfico corrente;
5. Ler o tipo do elemento gráfico corrente;
6. Posicionar no primeiro gráfico do tipo de atributo indicado;
7. Avançar para o gráfico seguinte;
8. Codificar um valor de um gráfico a partir de uma cadeia;
9. Formatar um valor gráfico como uma cadeia.

As operações listadas acima sugerem que os valores dos atributos gráficos (ou atributos com característica gráfica) tenham uma organização, ou seja, uma estrutura gráfica de dados. Esta estrutura gráfica, citada no capítulo 6 como um dos elementos que compõe um atributo gráfico, é exemplificada na figura 13.

Figura 13: Estrutura de dados de atributos com característica gráfica

Esta estrutura é conveniente se for considerado que a visualização dos valores dos atributos gráficos pela aplicação, de um objeto ou relacionamento, necessitará apenas percorrer as listas executando as rotinas apropriadas (de impressão, transformação geométrica ou de posicionamento) para cada elemento a ela pertencente. O formato do registro da lista de Tipos de Atributo

Gráficos, bem como da lista de Valores de Atributos Gráficos pode ser especificada, em sua essência, da seguinte forma:

Registro Tipos de Atributos Gráficos

Tipo de atributo;

Referência para o início da lista de Valores de atributo;

Fim-Registro.

Registro Valores de Atributos Gráficos

Tipo do valor gráfico;

União dos valores

Ponto;

Polilinha;

Elipse;

Bezier;

Arco;

Texto;

Atributo ("Bundles");

Matriz de Transformação;

Consiste; // continuação da lista em um outro objeto ou relacionamento

Fim-União

Fim-Registro.

O designador União indica a escolha de uma das alternativas indicadas. Corresponde à declaração "union" da linguagem C.

O conteúdo do registro de Tipos de Atributos Gráficos foi estabelecido de acordo com o que existia para suportar tipos de atributos de outras características. No entanto, o conteúdo do registro de Valores de Atributos Gráficos é totalmente novo e foi estabelecido de acordo com o Meta-Eschema definido no capítulo anterior.

Pode-se observar que, no registro de Valores de Atributos Gráficos, tanto o campo **Matriz** quanto **Consiste** compartilham o mesmo espaço de memória que os campos destinados à primitivas gráficas. Isto foi feito para facilitar a operação de visualização pelo aplicativo; que simplesmente executará a operação de transformação ou de posicionamento da mesma forma que executa a operação de impressão de primitivas. Este registro possui um campo **Tipo do valor gráfico**, fora da união, para que seja possível fazer a distinção entre os campos que a compõem.

Assim, as operações de manipulação de atributos gráficos, sobre a estrutura de dados acima, podem ser desempenhadas conforme os algoritmos descritos a seguir.

Parâmetros de entrada: CTA (Código do Tipo de Atributo), VG (Valor Gráfico)

Parâmetros de saída: Nenhum

Algoritmo Colocar atributo gráfico no objeto ou relacionamento corrente

Obter o tipo CTA e o valor gráfico VG que se deseja inserir;

Se CTA não é do tipo definido no esquema então
retornar erro;

Fim-Se

Se operação é sobre objeto então

Se não existe objeto corrente então
retornar erro;

Fim-Se

Inserir na lista de Tipos de Atributos Gráficos o tipo CTA;

Se VG é primitiva gráfica então

Transformar os pontos de VG segundo a matriz corrente;

Fim-Se

Inserir na lista de Valores dos Atributos Gráficos o valor VG;

Senão

// Repete-se a mesma seqüência de operações indicada acima, porém
para relacionamentos

Fim-Se

Tornar o atributo gráfico inserido como o corrente;

Fim-Algoritmo.

Parâmetros de entrada: Nenhum

Parâmetros de saída: Nenhum

Algoritmo Retirar o gráfico corrente

Se não existe gráfico corrente então
retornar erro;

Fim-Se

Se operação é sobre objeto então

Se não existe objeto corrente então
retornar erro;

Fim-Se

Se não existe gráfico corrente então
retornar erro;

Fim-Se

Remover da lista de Valores dos Atributos Gráficos o gráfico corrente;

Se lista de Valores ficou vazia então

Remove da lista de Tipos de Atributos Gráficos o CTA correspondente
ao Valor;

Fim-Se

Senão

// Repete-se a mesma seqüência de operações indicada acima, porém
para relacionamentos

Fim-Se

Fim-Algoritmo.

Parâmetros de entrada: CTA (Código do Tipo de Atributo), VG (Valor Gráfico)

Parâmetros de saída: Nenhum

Algoritmo Posicionar no gráfico indicado

Obter o tipo CTA e o valor gráfico VG em que se deseja posicionar;

Se CTA não é do tipo definido no esquema então

retornar erro;

Fim-Se

Chamar **Posicionar no primeiro gráfico do tipo de atributo indicado**;

Se posicionamento mal sucedido então

retornar erro;

Fim-Se

Ler o tipo CTAC e valor VGC do grafico corrente;

Enquanto (CTA $\neq$ CTAC) e (VG $\neq$ VGC) faça

Chamar **Avançar para o gráfico seguinte**;

Se não existe o seguinte então

retornar erro;

Fim-Se

Ler o tipo CTAC e valor VGC do grafico corrente;

Se VGC é primitiva gráfica então

Transformar os pontos de VGC segundo a matriz MT;

Fim-Se

Fim-Enquanto

Fim-Algoritmo.

Parâmetros de entrada: Nenhum

Parâmetros de saída: CTA (Código do Tipo de Atributo), VG (Valor Gráfico)

Algoritmo Ler o gráfico corrente

Se operação é sobre objeto então

Se não existe objeto corrente então
retornar erro;

Fim-Se

Se não existe gráfico corrente então
retornar erro;

Fim-Se

Ler o tipo CTA e valor VG;

Senão

// Repete-se a mesma seqüência de operações indicada acima, porém
para relacionamentos

Fim-Se

Se VG é primitiva gráfica então

Transformar os pontos de VG segundo a matriz corrente;

Fim-Se

retornar CTA e VG;

Fim-Algoritmo.

Parâmetros de entrada: Nenhum

Parâmetros de saída: TV (Tipo de Valor)

Algoritmo Ler o tipo do elemento gráfico corrente

Se operação é sobre objeto então

Se não existe objeto corrente então
retornar erro;

Fim-Se

Se não existe gráfico corrente então
retornar erro;

Fim-Se

Ler o tipo do valor TV;

Senão

// Repete-se a mesma seqüência de operações indicada acima, porém
para relacionamentos

Fim-Se

retornar TV;

Fim-Algoritmo.

Parâmetros de entrada: CTA (Código do Tipo de Atributo)

Parâmetros de saída: Nenhum

Algoritmo Posicionar no primeiro gráfico do tipo de atributo indicado

Obter o tipo CTA;

Se operação é sobre objeto então

Se não existe objeto corrente então

retornar erro;

Fim-Se

Se CTA não é do tipo definido no esquema então

retornar erro;

Fim-Se

Posicionar no primeiro elemento da lista de Tipos de Atributos Gráficos do objeto corrente;

Ler o tipo CTAL do elemento posicionado na lista Tipos de Atributos Gráficos;

Enquanto (CTA $\neq$ CTAL) faça

Avançar para o próximo elemento da lista de Tipos de Atributos Gráficos do objeto corrente;

Se fim de lista então

retornar erro;

Fim-Se

Ler o tipo CTAL do elemento posicionado na lista Tipos de Atributos Gráficos;

Fim-Enquanto

Senão

// Repete-se a mesma sequência de operações indicada acima, porém para relacionamentos

Fim-Se

Tornar o atributo posicionado corrente;

Inicializar MT como uma matriz identidade;

Fim-Algoritmo.

Parâmetros de entrada: Nenhum

Parâmetros de saída: Nenhum

Algoritmo Avançar para o gráfico seguinte

Se operação é sobre objeto então

Se não existe objeto corrente então

retornar erro;

Fim-Se

Se não existe gráfico corrente então

retornar erro;

Fim-Se

Se fim da lista de Valores de Atributos Gráficos do objeto corrente então

retornar erro;

Fim-Se

Posicionar no próximo elemento da lista;

Senão

// Repete-se a mesma seqüência de operações indicada acima, porém para relacionamentos

Fim-Se

Tornar o atributo posicionado corrente;

Ler o valor VGC do gráfico corrente;

Se VGC é uma matriz de transformação então

$MT = MT * VGC;$

Fim-Se

Fim-Algoritmo.

Parâmetros de entrada: CAD (Cadeia)

Parâmetros de saída: RVG (Registro do Valor Gráfico)

Algoritmo Codificar um valor de um gráfico a partir de uma cadeia

Obter a cadeia CAD;

Se formato de CAD incorreto então

retornar erro;

Fim-Se

Preencher o registro de valor gráfico RVG de acordo com o tipo do valor gráfico

retornar RVG;

Fim-Algoritmo.

Parâmetros de entrada: RVG

Parâmetros de saída: CAD

Algoritmo Formatar um valor gráfico como cadeia

Obter o registro de valor gráfico RVG;

Se formato de RVG incorreto então
retornar erro;

Fim-Se

Formatar a cadeia CAD de acordo com o valor gráfico RVG;
retornar CAD;

Fim-Algoritmo.

Convém destacar novamente que estas operações, bem como as estruturas de dados associadas foram definidas a partir do entendimento de quais elementos deveriam compor um atributo com características gráficas e, posteriormente, pelo paradigma de acesso navegacional e interface homogênea impostos pela implementação do GEO-MRO.

Especificadas as operações básicas e as estruturas de dados que irão compor o núcleo do GEO para atributos com características gráficas, serão feitas a seguir algumas considerações a respeito de sua implementação.

7.4 - Considerações sobre a Implementação

As estruturas de dados, assim como as interfaces programacionais de rotinas serão, quando possível, descritas em linguagem C, que é a linguagem adotada para a implementação do GEO.

7.4.1 - Detalhamento das Estruturas de Dados

As estruturas de dados para atributos com características gráficas foram incorporadas no arquivo de cabeçalho MRO.H. Este arquivo é definido como sendo o local de especificação de todas as estruturas de dados básicas existentes na implementação do GEO que podem ser acessadas por programas de aplicação. As estruturas, assim como as definições de algumas constantes incluídas são as seguintes:

```
// Constantes quantificadores
#define MAX_PONTOS      25
#define MAX_CARACTERES 100
#define MAX_PRIM        9
```

```

// Constantes dos especificadores de tipos de primitivas
enum t_tipogra {
    GEO_PONTO      = 12601,
    GEO_PLINE      = 12602,
    GEO_ELIPSE     = 12603,
    GEO_BEZIER     = 12604,
    GEO_ARCO       = 12605,
    GEO_TEXTO      = 12606,
    GEO_ATRPRIM    = 12607,
    GEO_MATRIZ     = 12608,
    GEO_CONSISTE   = 12609
};

typedef struct {
    double x, y;
}ponto;

typedef struct {
    pal2 n; // pal2 especifica um tipo inteiro de 2 bytes.
    ponto p[MAX_PONTOS];
}pline;

typedef struct {
    ponto p[2];
}elipse;

typedef pline bezier;

typedef struct {
    ponto p;
    double ai, af;
    double r; // Considerado como o ponto (r,0)
}arco;

typedef struct {
    ponto p; // Posicao para o inicio do texto
    double a; // Angulo de inclinacao
    unsigned char t[MAX_CARACTERES + 2];
} texto;

typedef struct {
    pal2 estilo;
    pal2 cor;
} atrprim;

```

```

typedef struct {
    double m11;
    double m12;
    double m13;
    double m21;
    double m22;
    double m23;
} matriz;

typedef struct {
    t_tipogra          v_tipo;
    union {
        ponto          v_ponto;    // 12601
        pline           v_pline;    // 12602
        elipse          v_elipse;   // 12603
        bezier          v_bezier;   // 12604
        arco            v_arco;     // 12605
        texto           v_texto;    // 12606
        atrprim         v_atrprim;   // 12607
        matriz          v_matriz;    // 12608
        codiobjeto      v_consiste;  // 12609 - codiobjeto é do tipo long.
    };
} ValorGRA;

```

Observa-se que a estrutura ValorGRA não é a ideal, pois possui campos do tipo **pline**, **bezier** e **texto** que são definidas estaticamente. Estes tipos permitem armazenar quantidades de pontos ou caracteres variáveis. O ideal seria que esta estrutura gastasse apenas o espaço de memória realmente necessário para armazenar as informações. Esta alternativa foi levada em consideração, porém não adotada, pois implicaria na utilização e adaptação de um conjunto de rotinas para gerenciar registros de tamanho variável da base de dados ainda não testadas para esta finalidade, o que poderia elevar consideravelmente o tempo desta dissertação. Acredita-se, no entanto, que uma vez adaptadas tais rotinas a transição desta alternativa de implementação para a ideal será extremamente simples.

7.4.2 - Definição das Interfaces Programacionais

Estabelecidas as estruturas de dados, passa-se agora à definição das interfaces programacionais das rotinas gráficas, que foram descritas sob a forma de algoritmos na seção 7.3.

Para tal, define-se a seguir os seus nomes e, posteriormente os seus argumentos.

Operação	Nome da Rotina
Colocar um gráfico no objeto ou relacionamento corrente	CTGRA
Retirar o gráfico corrente	RNGRA
Posicionar no gráfico indicado	PNGRA
Ler o gráfico corrente	BDGRA
Ler o tipo do elemento gráfico corrente	BNGRA
Posicionar no primeiro gráfico do tipo de atributo indicado	TAGRA
Avançar para o gráfico seguinte	VNGRA
Codificar um valor de um gráfico a partir de uma cadeia	GTGRA
Montar uma cadeia a partir de um valor gráfico.	FTGRA

A adoção de tais nomes é justificada devido a uma padronização de nomenclatura para todas as rotinas que fazem parte do núcleo do GEO. As regras para se compor o nome são as seguintes:

1. A primeira letra dos nomes das rotinas deve indicar o grupo:

- A - Inicialização
- B - Ler dados onde se está posicionado
- C - Colocar na Base de Dados
- F - Formatar
- G - Tornar permanente um dado temporário
- L - Retirar (Limpa) tudo
- P - Posicionar com indicação precisa
- R - Retirar da Base de Dados
- T - Posicionar indicando seqüência de busca
- V - Avançar desta posicao na seqüência indicada
- X - Operações variadas
- Z - Encerrar operações

2. A segunda letra é usada para distinguir mais de uma operação distinta possível dentro de um mesmo grupo, e tem o seguinte significado:

- A - Primeiro - indica o primeiro de uma seqüência
- D - Dados - indica uma referência a dados fundamentais
- E - Dados - indica uma referência a dados completos
- F - Dados - indica uma referência a campos intrínsecos
- N - Número - indica uma referência a um número, código ou tipo
- T - Título - indica uma referência a um nome ou valor manipulado pelo usuário

3. As três letras seguintes indicam sobre que peça da base a operação deve ser efetuada:

ATR - Atributos em geral
 COL - Colônia
 COM - Comentários
 GRA - Gráfico
 MRO - Base de Dados como um todo
 MST - Manipulação de Estados
 OBJ - Objetos
 PRO - Propriedades
 REL - Relacionamentos
 RGR - Regras
 SEL - Seleção de Objetos
 SIN - Sinônimos
 TOI - Tabelas de Operações Implícitas
 TRI - Relacionamentos Triplos

Estabelecido os nomes das rotinas, o próximo passo é definir seus parâmetros e valores de retorno, segundo algoritmos descritos na seção anterior, para completar a definição das interfaces.

pal2 CTGRA (ValorGRA Valor, coditipo CTA);

Retorno:

0 : Operação solicitada foi executada corretamente
 5 : Código de tipo especificado e inválido
 7 : Não existe OBJ corrente
 8 : Não existe REL corrente
 9 : Não existe GRA corrente

pal2 RNGRA (void);

Retorno:

0 : Operação solicitada foi executada corretamente
 7 : Não existe OBJ corrente
 8 : Não existe REL corrente
 9 : Não existe GRA corrente

pal2 PNGRA (ValorGRA Valor, coditipo CTA);

Retorno:

0 : Operação solicitada foi executada corretamente
 5 : Código de tipo especificado e inválido
 7 : Não existe OBJ corrente
 8 : Não existe REL corrente
 12: Não existe esse atributo no OBJ/REL corrente

pal2 BDGRA (ValorGRA *Valor, coditipo *CTA);

Retorno:

- 0 : Operação solicitada foi executada corretamente**
- 7 : Não existe OBJ corrente**
- 8 : Não existe REL corrente**
- 9 : Não existe GRA corrente**

pal2 BNGRA (pal2 *TV);

Retorno:

- 0 : Operação solicitada foi executada corretamente**
- 5 : Código de tipo especificado e inválido**
- 7 : Não existe OBJ corrente**
- 8 : Não existe REL corrente**
- 9 : Não existe GRA corrente**

pal2 TAGRA (coditipo CTA);

Retorno:

- 0 : Operação solicitada foi executada corretamente**
- 5 : Código de tipo especificado e inválido**
- 7 : Não existe OBJ corrente**
- 8 : Não existe REL corrente**
- 36: OBJ/REL corrente não tem esse tipo de gráfico**

pal2 VNGRA (void);

Retorno:

- 0 : Operação solicitada foi executada corretamente**
- 7 : Não existe OBJ corrente**
- 8 : Não existe REL corrente**
- 9 : Não existe GRA corrente**

pal2 GTGRA (cadeia valores, ValorGRA *valor);

Retorno:

- 3:Fim da cadeia**
- 62:Separador inválido**
- 80:Argumento inválido**

pal2 FTGRA (ValorGRA *valor, cadeia str);

Retorno:

- 3:Tipo do valor inválido**

7.5 - Testes

A estratégia adotada para a realização dos testes consistiu de duas etapas. A primeira envolveu apenas as rotinas de inclusão e leitura conjuntamente; posteriormente, a rotina de remoção foi testada em conjunto com as demais.

Para que os testes pudessem ser realizados segundo essa estratégia foi criada, para a primeira etapa, uma extensão para um programa já existente denominado GEO_INC. Este programa permite, através de uma linguagem simplificada, realizar inserções e consultas à base de dados. Para a segunda etapa, foi criada uma aplicação onde a manipulação dos atributos gráficos fosse bastante intensa. Esta aplicação será apresentada no capítulo 9.

A extensão feita para que o GEO_INC pudesse manipular Atributos com Características Gráficas consistiu na implementação de dois comandos:

1. Para a inserção: GRÁFICO Tipo_atributo_gráfico Valor_atributo_gráfico
2. Para a consulta: MOSTRA CORRENTE

Para a inserção, foi adicionado, basicamente, o seguinte código:

```
.
.
.
// Verificar se existe objeto corrente
if (BNOBJ(&co) != 0) {
    printk ("\nNao existe objeto corrente.\n");
    return 1;
}

// Verificar se o Tipo_atributo_gráfico está definido no esquema
lexico(str);
opera_string(str, str1);

// Obter o código do Tipo_atributo_gráfico
if ((erro = CodATR (str1, &cta)) != 0) {
    printk ("\n Tipo de grafico desconhecido: %a \n", str1);
    return 1;
}

// Converter a cadeia contendo o Valor_atributo_gráfico para o formato interno
while ((erro=GTGRA (linha, &valor)) == 0) {
    CTGRA (valor, cta); // Insere o Valor_do_atributo_gráfico do tipo cta
}
if (erro!=3) printk ("Erro:%d: Formato de grafico invalido\n", erro);
return 0;
```

Para a consulta, foi adicionado, basicamente, o seguinte código:

```

.
.
.
// Verificar se existe objeto corrente
if (BNOBJ(&co) != 0) {
    printk ("\nNao existe objeto corrente.\n");
    return 1;
}
// mostrar os graficos do objeto
aux=0;
if (TAGRA (Forma) == 0) { // posiciona no primeiro grafico do Obj
    // Obter o tipo e o valor do gráfico corrente
    while (BDGRA (&valgra, &ctg)==0) {
        // Converter o valor gráfico para o formato de cadeia
        FTGRA (&valgra, cadgra);
        if (aux == ctg)
            printk (" %a", cadgra);
        else {
            erro=NomATR (ctg, tipo); // Obter o nome do tipo do atributo gráfico
            printk (";\n %a %a", tipo, cadgra);
            aux=ctg;
        }
    }
    VNGRA(); // Avançar para o próximo gráfico
}
}

```

7.5 - Considerações Finais

A especificação e implementação das rotinas de gerenciamento de atributos com características gráficas, que fazem parte do núcleo do GEO, foram detalhadas acima. Surge, no entanto, uma pergunta: "Qual é a diferença entre manipular as informações gráficas da maneira como foi descrita e manipular as informações gráficas armazenadas como um campo longo ou como uma propriedade do tipo cadeia?". Respondendo a esta pergunta, destaca-se, a seguir, algumas das vantagens obtidas pela abordagem adotada:

- Dado que existe um tipo abstrato de dados específico para manipular atributos gráficos, é possível realizar modificações, com o intuito de fazer melhorias na estrutura de dados ou nos algoritmos, sem afetar quaisquer outras características de atributo;
- Se atributos com características de propriedade fossem utilizados para armazenar informações gráficas, surgiria um pequeno problema conceitual. Os atributos com característica de propriedade podem guardar, por exemplo, valores do tipo inteiro, real e cadeia de caracteres. Uma vez que uma propriedade seja definida no esquema como sendo de um tipo, todas as ocorrências desta propriedade devem ser do tipo

especificado. No entanto, para atributos gráficos, não faz sentido definir no esquema um atributo que permita armazenar apenas um tipo de primitiva, por exemplo, arco. Normalmente, uma aplicação deseja fazer uso de um atributo gráfico para armazenar primitivas gráficas independentemente de seus tipos.

- Torna-se possível efetuar consultas à base de dados cuja resposta independe de transformações gráficas, uma vez que a própria base acompanha as transformações existentes. Por exemplo, suponha que um objeto do tipo prédio seja composto por objetos do tipo janela e que a aparência de cada janela seja um atributo gráfico. Se todas as janelas forem iguais, cada uma conterá um atributo gráfico representado por uma estrutura do tipo **Tipo de Atributos Gráficos** e seus valores indicarão a mesma estrutura do tipo **Valores de Atributos Gráficos**. O objeto prédio conterá seu próprio atributo gráfico, o qual indicará o lugar de cada janela. Nesse caso, se a informação gráfica fosse armazenada como uma propriedade, uma consulta do tipo "Liste as coordenadas dos cantos de todas as janelas do prédio" listaria todas as coordenadas com o mesmo valor. Da forma como foi feito, a mesma consulta listaria todas as coordenadas corretas.

Faltou apenas salientar que, devido ao interesse de se implementar uma aplicação gráfica que fizesse uso destas rotinas, todo o ambiente de programação foi redefinido, isto é, adotou-se o ambiente operacional Windows, mudou-se o compilador de AZTEC 4.1 para Turbo-C++ "for Windows" e foram recriadas todas as bibliotecas já existentes para este novo compilador. Estas mudanças trouxeram alguns benefícios tais como:

1. Descoberta de alguns erros graves não detectados pelo compilador AZTEC 4.1 tais como:
 - Constante fora dos limites na comparação;
 - Código nunca executado, porém devendo ser executado em certas circunstâncias;
 - Passagem de valor por-variável ao invés de por-endereço;
 - Falta fechar comentário;
 - Invasão de memória.
2. Descoberta de alguns problemas não muito graves tais como:
 - Variáveis declaradas mas não utilizadas;
 - Declaração incompatível com o protótipo.
3. Disponibilidade do GEO para o ambiente Windows.

Um único ponto negativo verificado refere-se ao tempo gasto para:

- entender o código existente. Isto foi necessário para que se pudesse corrigir os erros detectados pelo novo compilador. Note-se que isso corresponde a mais de 200 rotinas em 1.4 Mbytes de código fonte.
- aprender o paradigma de programação orientada a objetos [70][17] utilizando o Turbo C++ "for Windows" [61][64];
- aprender a fazer programas para o ambiente operacional Windows utilizando-se as bibliotecas GDI (Graphics Device Interface) do Windows e o OWL (Object Window Library) da Borland [51][55].

Capítulo 8

Especificação da Aplicação

8.1 - Considerações Iniciais

No final do capítulo anterior, comentou-se que o ambiente de programação sofreu alterações para que fosse possível implementar uma aplicação gráfica. O objetivo dessa aplicação consiste em fazer uso das rotinas implementadas no núcleo do GEO com, basicamente, duas finalidades:

1. Demonstrar a utilização das rotinas de manipulação de atributos gráficos;
2. Criar uma ferramenta gráfica para o GEO.

Na realidade, para demonstrar o uso das rotinas de atributos gráficos bastaria criar um pequeno e limitado editor gráfico onde as rotinas pudessem ser testadas. Portanto, a escolha da aplicação gráfica foi influenciada principalmente pelo segundo item. Olhando através deste prisma, a escolha da aplicação foi imediata: um "Editor de Esquemas". Lembrando que o MRO fornece uma representação gráfica para se definir o esquema de uma aplicação, o Editor de Esquemas (EE) foi especificado segundo esta representação.

As dificuldades encontradas para especificar esta aplicação originaram-se da estreita relação existente entre o EE e o GEO, ou seja, ambos manipulam os mesmos conceitos provenientes do MRO. Isto ficará claro na medida em que forem abordadas as alternativas de implementação adotadas neste projeto.

8.2 - As Formas Gráficas do DRO

Uma das primeiras questões levantadas foi quanto à localização da forma dos elementos gráficos que fazem parte do DRO, ou seja, onde armazenar, por exemplo, a forma de um tipo de objeto:

- na aplicação (EE);
- pré-inicializando-se a base; ou
- no meta-esquema (GEO)

A solução adotada foi de armazenar a forma dos elementos gráficos do DRO no meta-esquema. Para justificar esta alternativa, serão feitas a seguir algumas considerações sobre a independência lógica de dados no GEO.

Os SGBDs permitem que se armazene dados de uma aplicação segundo um modelo de dados (esquema). No entanto, na maioria dos casos, os esquemas são definidos estaticamente, e qualquer extensão ou modificação que venham a sofrer poderá acarretar uma drástica reorganização das bases de dados associadas. Desse modo, para facilitar a extensão e/ou mudança do esquema, bem como minimizar a necessidade de reorganização da base de dados, o GEO, desde a sua concepção, permite que o esquema da aplicação seja tratado como uma base de dados, tendo o meta-esquema como seu esquema. O meta-esquema é, portanto, o esquema do próprio Modelo de Representação de Objetos. Verifica-se claramente que um dos benefícios decorrentes desta abordagem é o tratamento homogêneo que o gerenciador pode dar tanto à base de dados intencional (esquema da aplicação) quanto à base extencional (base de dados da aplicação). A abordagem aqui descrita confere ao GEO a característica de sistema meta [65] [5] [9].

O meta-esquema é o esquema da base intencional; assim, a forma dos elementos gráficos que definem o DRO devem estar no meta-esquema — semelhante a uma aplicação de CAD para projeto de engenharia civil, onde encontra-se a forma de uma porta, ou de um bidê definidos em seu esquema. Tendo em mente essas considerações, o meta-esquema foi alterado, acrescentando-se um tipo de atributo gráfico denominado Forma, onde é especificado o formato de cada elemento que compõe o DRO.

Note-se que, nesse caso, houve a necessidade de modificar o meta-esquema apenas para dar suporte ao armazenamento das formas dos elementos que compõe o DRO do EE, e não devido à adição ou à modificações conceituais no meta-esquema. Sendo assim, esta alteração foi realizada apenas na implementação do GEO e não no DRO do meta-esquema.

Atrelado ao atributo do tipo Forma, um conjunto de rotinas foi definido para que o EE pudesse consultar, no meta-esquema, a forma gráfica de um elemento do DRO no momento de sua instanciação.

Tais rotinas são descritas a seguir.

- pal2 MPosPrimPRIM(pal2 tipo): Dado o tipo do elemento gráfico (tipo de objeto, subtipo, tipo de relacionamento, tipo de atributo, etc.), posiciona no primeiro valor gráfico correspondente.
- pal2 MAvancaPRIM(pal2 tipo): Dado o tipo do elemento gráfico, avança para o próximo valor gráfico.
- pal2 METAGRA(pal2 tipo, ValorGRA *valor): Recupera em valor, o valor gráfico correspondente ao tipo do elemento gráfico corrente.

- pal2 FormaGRA(coditipo tgra, coditipo tipo, ValorGRA valor[], int *i):
Dado o tipo do atributo gráfico e o tipo do elemento gráfico, retorna os valores gráficos associados, bem como a sua quantidade.

As três primeiras rotinas realizam consultas específicas da forma gráfica definida no meta-esquema. Já a rotina FormaGRA, pode ser utilizada para realizar consultas dos valores gráficos tanto do meta quanto do esquema. Isto pode ser verificado no seguinte fragmento de código de FormaGRA:

```
.
.
B_leepglobal (META, &como, &longvar); // Obtém o estado corrente do META
if (como==OFF) { // Se meta-desligado, ou seja se esquema ligado
    if (PNOBJ(tipo)!=0) { // Tornar o objeto corrente
        erro=801; // objeto não encontrado
    } else {
        // Obter atributos do objeto corrente
        if (TAGRA(tgra)!=0) { // Posicionar no primeiro gráfico do tipo tgra
            erro = 802; // Não existe este tipo de GRA definido para este objeto
        } else {
            // Obter o valor do primeiro gráfico
            BDGRA(&(valor[0]), &CTA);
            *i = 1;
            while (IVNGRA()) { // Avança para o próximo valor gráfico
                BDGRA(&(valor[*i]), &CTA);
                (*i)++;
            }
        }
    }
}
.
.
} else { // Caso contrário consulta o meta-esquema
    // Se tipo do atributo gráfico igual ao especificado no meta-esquema (Forma)
    if (tgra == Forma) {
        if (MPosPrimPRIM(tipo)!=0) { // Posiciona no primeiro valor grafico
            erro = 802;
        } else {
            METAGRA(tipo, &(valor[0])); // Avança para o próximo valor gráfico
            *i = 1;
            while (!MAvancaPRIM(tipo)) {
                METAGRA(tipo, &(valor[*i]));
                (*i)++;
            }
        }
    }
} else return 802;
}
return erro;
```

Pode ser observado no fragmento acima que existe um teste de igualdade entre *tgra* e *Forma* no caso em que a consulta é realizada no meta-esquema. A variável *tgra* deve indicar, neste caso, o tipo de atributo gráfico que tenha sido definido no meta-esquema. Isso foi feito para que apenas os atributos gráficos do tipo *Forma* pudessem armazenar a forma dos elementos que definem o DRO. Por outro lado, quando o esquema está ativo, a consulta efetuada por *FormaGRA* pode ser sobre qualquer tipo de atributo, desde que esteja definido no esquema da aplicação e tenha a característica gráfica. Retomando-se o exemplo da aplicação CAD para projetos de engenharia civil, o objeto do tipo *Bidê* poderia ter um atributo gráfico do tipo *Perfil* e não precisaria ser, necessariamente, *Forma*.

8.3 - As Instâncias dos Elementos de um DRO

Uma vez solucionado o problema da localização da forma gráfica dos elementos que compõe o DRO, passou-se a considerar o seguinte problema: "Onde armazenar as ocorrências de um elemento gráfico de um DRO específico?". Neste caso, pode-se considerar as seguintes alternativas:

- Modelar o esquema da aplicação, ou seja do Editor de Esquemas e armazenar as ocorrências de cada elemento gráfico que compõe o DRO específico na base de dados.
- Armazenar as ocorrências dos elementos gráficos que compõe o DRO específico no próprio esquema.

A primeira alternativa pode parecer conveniente, no entanto, fazer a modelagem do esquema do EE é como refazer o meta-esquema, uma vez que as informações manipuladas pelo EE é o esquema. Mesmo assim, considerando-se a hipótese de que isto seja feito, a aplicação teria que armazenar as informações gráficas do esquema na base extencional do novo esquema definido, dissociado portanto do próprio esquema. Tal situação seria insustentável pois poderia criar redundâncias, além da dificuldade adicional de ter que criar um gerenciador para evitar inconsistências. Assim, a segunda alternativa foi a adotada. Porém, armazenar as ocorrências dos elementos gráficos que compõe o DRO específico no próprio esquema não é uma tarefa trivial. Necessita-se antes, conhecer a maneira como objetos, relacionamentos e atributos que definem o esquema, são armazenados. Para isso será mostrada a seguir, a correspondência existente entre a visão conceitual do esquema representado pelo DRO hipotético da figura 14 e a forma como ele é efetivamente armazenado, representado pelo DRI da figura 15.

Figura 14: DRO de um esquema hipotético

Esta forma de armazenamento é o reflexo de se estar considerando o meta-esquema, apresentado na figura 12 do capítulo 6, como o esquema da base intencional.

De acordo com os três diagramas considerados (figuras 12, 14 e 15), verifica-se primeiramente que, para armazenar o relacionamento entre Pessoa e Empresa são necessários a instanciação de dois objetos do tipo binário e dois objetos do tipo modalidade1. Os objetos do tipo binário não necessitam de um identificador, uma vez que são utilizados como objetos intermediários para compor os relacionamentos binários; portanto tais objetos somente são acessíveis através de relacionamentos. Os objetos do tipo modalidade1 são portadores da semântica dos relacionamentos binários; desse modo, os identificadores para objetos desse tipo devem ser, respectivamente, Trabalha, e o seu oposto Emprega. Caso fosse especificado um tipo de atributo para o tipos de relacionamentos binário Trabalha e Emprega, seriam as modalidades de primeiro grau (modalidade1) que receberiam este tipo de atributo. A maneira de se fazer isso é semelhante a forma de se especificar um atributo a objetos, ou seja, definindo-se o objeto do tipo atributo, com seu identificador correspondendo ao nome do tipo de atributo sendo especificado, associado por um relacionamento atributos_são.

Figura 15: Maneira como é armazenado o DRO hipotético

8.4 - Representação Interna das Instâncias dos Elementos de um DRO

Entendido como o esquema é armazenado pode-se agora definir a maneira de se armazenar as ocorrências dos elementos gráficos, que compõe um DRO específico, no esquema. É importante lembrar que o EE deve permitir ao projetista da base de dados, visualizar e manipular o esquema que esteja definindo segundo as formas gráficas estabelecidas no meta-esquema.

Inicialmente será mostrado como as ocorrências de Tipos de Objetos foram armazenadas no esquema e, posteriormente, o mesmo será feito para Tipos de Relacionamento Binário e Tipos de Atributo. A representação gráfica de Subtipos e Tipos de Relacionamento Triplo não serão estudadas pois tais conceitos ainda não foram totalmente implementados no GEO.

8.4.1 - Representação Interna de Tipos de Objetos

Considerando novamente as figuras 14 e 15, pode-se guardar a ocorrência gráfica do Tipo de Objeto Pessoa em um atributo gráfico pertencente ao objeto Pessoa. Para que isso possa ser feito, adicionou-se na implementação do meta-esquema um tipo de atributo denominado Val_forma. O conteúdo deste atributo deve ser definido de acordo com a Forma de um Tipo de Objeto definido no meta-esquema, como mostra a figura 16.

Figura 16: Forma de um Tipo de Objeto

Como pode ser verificado, a forma de um Tipo de Objeto é composto de dois tipos de primitivas: ARCO e PLINE. O fragmento de código a seguir detalha como a forma do tipo de objeto foi definida.

```
ValorGRA val_forma[3];
int compoe;
```

```
// Obtém a forma de um Tipo de Objeto
FormaGRA(Forma, Objeto, val_forma, &compoe);
```

Valores inicializados no meta-esquema
Referente ao círculo
val_forma[0].v_tipo = ARCO;
val_forma[0].v_arco.ai = 0; // Ângulo inicial em graus
val_forma[0].v_arco.af = 360; // Ângulo final em graus
val_forma[0].v_arco.r = 30; // Raio em pixels
Referente à reta
val_forma[1].v_tipo = PLINE;
val_forma[1].v_pline.n = 2; // Número de pontos

```
// Valores inicializados no momento da instanciação
// Referente ao círculo
val_forma[0].v_arco.p.x = X;
val_forma[0].v_arco.p.y = Y;
```

```
// Referente à reta
val_forma[1].v_pline.p[0].x = X - 30;
val_forma[1].v_pline.p[0].y = Y;
val_forma[1].v_pline.p[1].x = X + 30;
val_forma[1].v_pline.p[1].y = Y;
```

As variáveis X e Y referem-se às coordenadas do centro da figura no momento de sua instanciação, e a variável **compoe** indica a quantidade de tipos de primitivas gráficas que compõe a figura.

O conteúdo da variável **val_forma** é armazenado no esquema através do atributo **Val_forma** que está associado ao objeto Pessoa, pelo código:

```
// Posiciona no objeto Pessoa
```

```
...
```

```
// Insere no esquema a ocorrência de um Tipo de Objeto  
for (i = 0; i < compoe; i++) CTGRA (val_forma[i], Val_forma);
```

```
.
```

```
.
```

```
.
```

8.4.2 - Representação Interna de Tipos de Relacionamentos

A representação de Tipos de Relacionamentos pode ser armazenada da mesma maneira que a representação de Tipos de Objetos, ou seja, no atributo **Val_forma**. Porém, como pode ser constatado na figura 15, a ocorrência de um Tipo de Relacionamento envolve a instanciação de quatro objetos na base: dois do tipo binário e dois do tipo modalidade1 (ou modalidade de primeiro grau). Assim, em qual desses objetos o atributo **Val_forma** deve ser associado?

Para responder a esta pergunta, considere o DRI da figura 17. A figura ilustra que um Tipo de Relacionamento pode aparecer mais que uma vez em um mesmo diagrama. Para cada nova ocorrência de um Tipo de Relacionamento, são acrescentados apenas mais dois objetos do tipo binário, mantendo-se o número de objetos do tipo modalidade1. Esta correspondência indica que, para cada ocorrência de um Tipo de Relacionamento, existem dois objetos binários associados. Assim, a representação gráfica de um Tipo de Relacionamento deve ser armazenada no atributo **Val_forma** ligado a um dos objetos do tipo binário. É indiferente associar este atributo a um objeto

Figura 17: Duas ocorrências de um mesmo Tipo de Relacionamento

do tipo binário ou à seu oposto; o importante é que apenas um deles possua este atributo, para que não exista o perigo de inconsistência devido a redundância.

Uma outra consideração que deve ser feita é quanto a forma de um Tipo de Relacionamento não ser bem definida, ao contrário do que ocorre com a representação de Tipos de Objetos. Os Tipos de Relacionamentos são representados por arcos que podem ter diversas orientações, dimensões e concavidades. Assim, são necessários mais do que um simples ponto para defini-lo. A solução adotada foi de utilizar a primitiva gráfica Bezier com três pontos de controle; o primeiro e o terceiro ponto para indicar o início e o fim da curva respectivamente e o segundo para definir sua concavidade. Por simplicidade descartou-se, da representação de um Tipo de Relacionamento, o traço que corta o arco.

O fragmento de código abaixo mostra como um Tipo de Relacionamento foi definido:

```
ValorGRA val_forma[1];
int compoe;

// Obtém a forma de um Tipo de Relacionamento
FormaGRA(Forma, Binário, val_forma, &compoe);
```

Valores inicializados no meta-esquema
Referente a curva val_forma[0].v_tipo = BEZIER; val_forma[0].v_bezier.n = 3; // Número de pontos val_forma[0].v_bezier.p[0].x = 0; val_forma[0].v_bezier.p[0].y = 0; val_forma[0].v_bezier.p[1].x = 0; val_forma[0].v_bezier.p[1].y = 0; val_forma[0].v_bezier.p[2].x = 0; val_forma[0].v_bezier.p[2].y = 0;

```
// Valores inicializados no momento da instanciação
val_forma[0].v_bezier.p[0].x = X1;
val_forma[0].v_bezier.p[0].y = Y1;
val_forma[0].v_bezier.p[1].x = X2;
val_forma[0].v_bezier.p[1].y = Y2;
val_forma[0].v_bezier.p[2].x = X3;
val_forma[0].v_bezier.p[2].y = Y3;
```

O conteúdo da variável `val_forma` é armazenado no esquema, através do atributo `Val_forma` que está associado a um dos objetos do tipo binário, pelo código:

```
// Posiciona em um dos objetos do tipo binário
...

// Insere no esquema a ocorrência de um Tipo de Relacionamento
for (i = 0; i < compoe; i++) CTGRA (val_forma[i], Val_forma);
.
.
.
```

8.4.3 - Representação Interna de Tipos de Atributos

Os Tipos de Atributos podem ser especificados tanto para Tipos de Objetos quanto para Tipos de Relacionamentos. Inicialmente será descrita a maneira como os Tipos de Atributos foram associados à Tipos de Relacionamentos e, posteriormente, à Tipos de Objetos.

Como discutido no final da seção 8.3, quem recebem os relacionamentos `Atributos_são` com o objeto do tipo Atributo sendo especificado são os objetos do tipo `modalidade1`. Assim, os Tipos de Atributos para Tipos de Relacionamentos foram implementados seguindo a mesma abordagem que está ilustrada na figura 18.

Conforme a figura considerada, a representação gráfica de um tipo de Atributo foi armazenada em um atributo, vinculado ao objeto do tipo binário, denominado `Val_atributo`. Como o objeto do tipo binário utiliza o atributo `Val_forma` para armazenar a sua forma, ou seja, a representação de um Tipo de Relacionamento, foi necessária a criação deste novo atributo.

Um outro detalhe que pode ser observado na figura, é a indicação do atributo `Val_atributo` para o objeto do tipo atributo (`Data`). Esta indicação, rotulada pela palavra **consiste**, serve para fazer a distinção entre as diversas representações de Tipos de Atributos que um relacionamento pode ter e que são armazenadas no

Figura 18: Tipo de Atributo associado à Tipo de Relacionamento

mesmo atributo Val_atributo. Isto proporciona a realização da operação de remoção da representação gráfica de um Tipo de Atributo, no caso de se removerem os relacionamentos Atributos_são.

A maneira como foi implementada a inserção da representação gráfica de um Tipo de Atributo para relacionamento pode ser observada no seguinte código:

```
ValorGRA val_atr[1];
int compoe;
codiobjeto Cod;

// Obtem a forma de um Tipo de Atributo
FormaGRA(Forma, Atributo, val_atr, &compoe);
```

Valores inicializados no meta-esquema
Referente a curva val_atr.atributo[0].v_tipo = PLINE val_atr.atributo[0].v_pline.n = 3 val_atr.atributo[0].v_pline.p[0].x = -1 val_atr.atributo[0].v_pline.p[0].y = -1 val_atr.atributo[0].v_pline.p[1].x = -1 val_atr.atributo[0].v_pline.p[1].y = -1 val_atr.atributo[0].v_pline.p[2].x = -1 val_atr.atributo[0].v_pline.p[2].y = -1

```
// Valores inicializados no momento da instanciação
val_atr.atributo[0].v_pline.p[0].x = X1;
val_atr.atributo[0].v_pline.p[0].y = Y1;
val_atr.atributo[0].v_pline.p[1].x = X2;
val_atr.atributo[0].v_pline.p[1].y = Y2;
val_atr.atributo[0].v_pline.p[2].x = X3;
val_atr.atributo[0].v_pline.p[2].y = Y4;
```

O conteúdo da variável val_atr é armazenado no esquema através do atributo Val_atributo pelo seguinte código:

```
// Posiciona no objeto do Tipo Atributo
...
// Obtém o código do objeto corrente na variável Cod
...
val_atr.atributo[0].v_consiste = Cod;
// Posiciona no objeto do Tipo Binário que possui o atributo Val_forma
```



```
// Insere no esquema a ocorrência de um Tipo de Atributo
for (i = 0; i < compoe; i++) CTGRA (val_atr[i], Val_atributo);
.
.
.
```

Os Tipos de Atributos vinculados à Tipos de Objetos foram implementados de forma semelhante ao adotado para Tipos de Relacionamentos como pode ser observado na figura 19.

A inicialização da variável `val_atr` é idêntica à inicialização feita para relacionamentos.

Figura 19: Tipo de Atributo associado à Tipo de Objeto

O conteúdo da variável `val_atr` é armazenado no esquema através do atributo `Val_atributo` pelo seguinte código:

```
// Posiciona no objeto do Tipo Atributo
...

// Obtém o código do objeto corrente na variável Cod
...

val_atr.atributo[0].v_consiste =Cod;

// Posiciona no objeto que possui o Tipo de Atributo
...
// Insere no esquema a ocorrência de um Tipo de Atributo
for (i = 0; i < compoe; i++) CTGRA (val_atr[i], Val_atributo);
```

8.5 - A Arquitetura do Programa Editor de Esquemas

De acordo com o que foi discutido no capítulo 1, construiu-se um Subsistema Integrado de Gerenciamento de Gráficos e Base de Dados (SIGGBD) para que o EE pudesse fazer uso dos recursos oferecidos tanto pelo GEO quanto pelo núcleo gráfico, que neste caso é o Windows. Porém, o SIGGBD foi implementado para atender apenas as necessidades específicas do EE.

Figura 20: Principais componentes do Editor de Esquemas

A figura 20 ilustra os principais componentes de uma aplicação, no caso o EE: código específico da Aplicação, SIGGBD, o GEO e o Windows. As linhas que ligam estes componentes indicam a interação existente entre eles.

A interação entre a Aplicação (EE) e o GEO ocorre, nesta implementação, quando o usuário realiza as seguintes operações:

- Abrir, Fechar ou Salvar a base GEO: Estas operações atuam sobre uma determinada base unificada que o usuário deverá escolher entre as disponíveis.
- Ativar ou Desativar o Esquema: Para cada base unificada, existe a possibilidade do usuário escolher um, entre as possíveis alternativas de esquemas para a sua edição. Apesar desta opção estar disponível na interface, ela não foi implementada pois o gerenciamento de múltiplas alternativas de esquemas está sendo desenvolvido em paralelo com este trabalho [9].

De acordo com as operações citadas acima, verifica-se que a interação entre estes módulos, no caso do EE, é muito pequena. O mesmo não se pode dizer das outras interações.

A interação entre a Aplicação (EE) e SIGGBD ocorre quando são executadas as seguintes operações:

- Inserir ou remover tipos de Objetos, Relacionamentos ou Atributos.
- Visualizar um esquema existente.

As operações de inserção e remoção, bem como a operação de visualização dos elementos que compõe o esquema são efetuadas graficamente. Isto implica

que a ativação destas operações, por intermédio da Aplicação (EE), deve provocar chamadas à rotinas tanto do GEO quanto do Windows pelo SIGGBD. Essas chamadas a rotinas constataam o alto grau de interação existente entre o SIGGBD e os componentes GEO e Windows.

A verificação do alto grau de interação entre a Aplicação (EE) e o Windows é imediata, uma vez que a própria interface gráfica da aplicação é implementada utilizando-se os recursos de gerenciamento de interfaces do Windows.

8.5.1 - A Abordagem Adotada para a Implementação da Aplicação (EE) e do SIGGBD

De acordo com o que foi discutido no final do capítulo anterior, a implementação do EE fez uso das bibliotecas OWL. A biblioteca OWL foi escrita para o compilador Turbo C++ "for Windows", a fim de que o usuário pudesse criar aplicações para o ambiente Windows de forma mais simples do que se utilizasse a biblioteca SDK ("Software Development Kit"). O uso dessa biblioteca obrigou a utilização do paradigma de programação orientada a objetos para implementar as componentes Aplicação (EE) e SIGGBD.

8.5.1.1 - A Aplicação (EE)

A Aplicação (EE), além de fornecer os elementos necessários para a interação gráfica aplicativo-usuário, também seleciona as ações que devem ser tomadas de acordo com a mensagem recebida pela janela da aplicação. Estas funcionalidades foram encapsuladas em uma classe denominada TAreaWindow, como mostra o código abaixo:

```
class TAreaWindow : public TWindow {
public:
    TAreaWindow(PWindowsObject pwParent, LPSTR lpszTitle, img *lista_img);
    virtual LPSTR GetClassName(void);
    virtual void GetWindowClass(WNDCLASS&);
    virtual void WMSize(TMessage& msg)           = [WM_SIZE];
    virtual void WMLButtonDown(TMessage& msg)    = [WM_LBUTTONDOWN];
    virtual void WMLButtonUp(TMessage& msg)      = [WM_LBUTTONUP];
    virtual void WMMouseMove(TMessage& msg)      = [WM_MOUSEMOVE];
    void Seleccion_(TMessage& msg)               = [WM_USER + 2];
    virtual void Paint(HDC, PAINTSTRUCT&);
private:
```

A classe `TAreaWindow` é uma classe derivada da classe `TWindow`. `TWindow` é uma classe pré-definida da biblioteca OWL. É a partir dela que todas as janelas da aplicação são definidas. A classe `TAreaWindow` possui uma série de métodos definidos como públicos. O primeiro deles, que possui o mesmo nome da classe, é um método construtor. Este método realiza inicializações tais como definição do estilo da janela, definição das barras de rolagem e inicialização da lista de imagens, no momento em que a classe é instanciada. A lista de imagens é uma estrutura de dados organizada como uma lista linear encadeada onde são armazenados os elementos gráficos que estão sendo manipulados.

Os métodos que possuem a palavra **virtual** redefinem métodos "default" especificados e chamados na classe base, ou seja, na classe `TWindow`. Para começar, o método `GetClassName` retorna o nome da classe utilizada para fornecer uma identificação única para a classe janela. `GetWindowClass` redefine a estrutura `WNDCLASS` para indicar, neste caso, a cor de fundo da janela.

Os métodos que possuem após a sua lista de parâmetros a sequência "`= [WM_XXXX ]`", são métodos que passam a ser executados quando uma mensagem do tipo `WM_XXXX` correspondente for recebida pela janela. São os casos dos métodos: `WMSize`, `WMLButtonDown`, `WMLButtonUp`, `WMMouseMove` e `Selecao_`.

O método `WMSize` redefine as barras de rolagem horizontal e vertical quando a janela for redimensionada. Os métodos `WMLButtonDown` e `WMLButtonUp` tratam de processar as mensagens geradas quando o botão esquerdo do "mouse" for pressionado ou despressionado respectivamente; e `WMMouseMove` é ativado quando o "mouse" é movimentado. A `Selecao_` é um método que processa uma mensagem definida pelo projetista. Ela é responsável pela inicialização do estado do ambiente de edição, permitindo que os métodos `WMLButtonDown`, `WMLButtonUp` e `WMMouseMove` realizem suas operações sobre um determinado objeto de uma determinada classe pertencentes ao SIGGBD. Para esclarecer este ponto, é necessário descrever inicialmente, as opções de edição fornecidas para o usuário e, posteriormente, detalhar como estas opções são executadas pelo SIGGBD.

8.5.1.2 - O SIGGBD

As opções de edição, fornecidas pelo menu do EE são basicamente as seguintes:

A seleção de uma das opções pelo usuário, gera a mensagem definida pelo projetista (`WM_USER + 2`) que, conseqüentemente, ativa o método `Selecao_`. Junto à essa mensagem, está associada a natureza da operação, ou seja, se a operação é de inserir um Tipo de Objeto, Tipo de Relacionamento ou um Tipo de Atributo; ou de Remover. A operação de remoção necessita apenas ser sinalizada pois, a indicação do elemento gráfico que será removido é dependente da escolha do usuário no momento que estiver visualizando o desenho do esquema.

A Aplicação (EE) mantém em memória uma lista das informações gráficas do esquema corrente, chamada Lista de Imagens. Isso é feito para acelerar a manipulação de tela. Devido a isso, sempre que a aplicação é aberta sobre um esquema, todas as suas informações são traduzidas para a Lista de Imagens em memória. As operações de Inserção e Remoção solicitadas pelo usuário promovem alterações simultâneas na Lista de Imagens e no esquema, armazenado na base de dados.

Dado que as operações de edição básicas são quatro, foram também criadas quatro classes de objetos: `Obj`, `Rel`, `Atr` e `Rem` para desempenhar as operações: Inserir um Tipo de Objeto, Inserir um Tipo de Relacionamento, Inserir um Tipo de Atributo e Remover respectivamente. Estas classes definem o que é essencialmente o SIGGBD para o EE.

Pode parecer estranho, a primeira vista, que cada uma das operações sejam realizadas por uma classe de objeto específica. No entanto, isso é necessário para que as mensagens do "mouse" sejam processadas corretamente para cada uma dessas operações.

Considere, como exemplo, a operação de Inserir um Tipo de Objeto. A escolha desta opção gera a mensagem `WM_USER + 2` que é enviada para o método `Selecao_`, juntamente com a natureza da operação. O método `Selecao_`, por sua vez, inicializa o estado do ambiente de edição para operação de Inserir um Tipo de Objeto. A inicialização do estado serve para sincronizar as mensagens provenientes do "mouse" com as ações a serem tomadas. Para que isto fique

claro, considere o seguinte diagrama de estados para a inserção de um Tipo de Objeto.

Neste diagrama são mostrados todos os estados necessários para se realizar a inserção de um Tipo de Objeto. Inicialmente, o método `Selecao_` envia a mensagem `Inicializa` para um objeto da classe `Obj` fazendo com que a máquina de estados fique no estado 0. Neste estado, é feita as seguintes verificações:

1. se não existe o Tipo de Objeto que está querendo se inserir na base intencional e
2. se o Tipo de Colônia indicada para esse Tipo de Objeto existe.

Se ambas as condições são satisfeitas prossegue-se fazendo a leitura da forma de um Tipo de Objeto usando-se a rotina `FormaGRA`.

Estando no estado 0, qualquer mensagem diferente de `WMLButtonDown`, indicado no diagrama pelo símbolo $\Downarrow$, será desprezada a não ser que o usuário faça uma nova seleção através do menu. Note que as mensagens do "mouse" devem ser redirecionadas para o objeto da classe `Obj`. Este redirecionamento é realizado pelos métodos `WMLButtonDown`, `WMLButtonUp` e `WMMouseMove` da classe `TAreaWindow`, de acordo com o conteúdo de uma variável, indicando a natureza da operação, inicializada pelo método `Selecao_`.

Após o recebimento da mensagem `WMLButtonDown`, atualiza-se o estado para 1 e é mostrado na janela o elemento gráfico correspondente ao Tipo de Objeto. Enquanto não ocorrer a mensagem `WMLButtonUp` ($\Uparrow$), o EE permite que o usuário movimente o "mouse" para posicionar a figura na janela. Ocorrendo esta mensagem, é feita a atualização do estado para 2 e é então permitido que o usuário posicione o nome do Tipo do Objeto bem como o nome do Tipo de Colônia correspondente. Este processo é representado, no diagrama, pelas transições associados aos estados 2, 3 e 4. Finalmente, no estado 4, é feita a inserção propriamente dita, de um Tipo de Objeto na base intencional, vinculado ao atributo `Val_forma`. Além disso, é feita a inserção da representação gráfica deste elemento na lista de imagens. Isto é feito para agilizar o processo de regeneração do diagrama na janela no momento em que ela for transladada horizontalmente ou verticalmente através das barras de rolagem.

A inserção de Tipos de Relacionamento e Tipos de Atributos segue basicamente a mesma abordagem utilizada para a inserção da Tipos de Objetos. Ou seja, o método `Selecao_` redireciona as mensagens do "mouse" para um objeto da classe considerada (Obj, Rel ou Atr) sendo que cada classe implementa uma máquina de estados para realizar a inserção apropriada.

A operação de remoção segue também a mesma abordagem adotada para as outras operações; a diferença básica é que ela não necessita de uma máquina de estados pois basta processar a mensagem `WMLButtonDown` para indicar o elemento gráfico que será removido. É certo que, uma vez identificado qual elemento será removido, o objeto da classe `Rem` deverá conter como membro o método de remoção correspondente ao elemento, para que se possa fazer a validação desta operação sobre o elemento gráfico considerado. As possíveis validações são as seguintes:

- Para remover um Tipo de Objeto do esquema verificar:
 - se possui Tipos de Atributos associados,
 - se possui Tipos de Relacionamentos associados.

Se uma destas condições for satisfeita, o Tipo de Objeto não poderá ser removido. Caso contrário é removida da base intencional e da lista de imagens.

- Para remover um Tipo de Relacionamento verificar:
 - se possui Tipos de Atributos associados,
 - se ocorre esse Tipo de Relacionamento mais que uma vez no mesmo diagrama.

Se o Tipo de Relacionamento possuir Tipos de Atributos, ele não poderá ser removido. No caso de ocorrer esse tipo mais que uma vez e não possuir Tipos de Atributos associados, é removida apenas a sua ocorrência do atributo `Val_forma` e da lista de imagens, sem que ele seja efetivamente retirado da base.

- Para remover um Tipo de Atributo verificar se ocorre esse Tipo de Atributo mais que uma vez no mesmo diagrama.

Semelhante à Tipos de Relacionamentos, quando ocorre um Tipo de Atributo mais que uma vez, é removida a sua ocorrência do atributo `Val_atributo` e, também, da lista de imagens, sem que ele seja efetivamente retirado da base.

8.6 - Considerações Finais

Neste capítulo foram discutidos os aspectos de implementação da aplicação (EE). Discutiu-se sobre a localização das formas gráficas do DRO, a maneira de armazenar o esquema, na base intencional, juntamente com sua forma gráfica, bem como a abordagem utilizada para a implementação da Interface Homem/Máquina e do SIGGBD.

O EE foi construído com o objetivo de possibilitar o teste das rotinas implementadas para a manipulação de Atributos com Características Gráficas. Acredita-se que esse objetivo foi alcançado com êxito, pois as operações de inserção e remoção de elementos gráficos no esquema, bem como o salvamento e recuperação de um esquema editado na base intencional, são feitas sem maiores problemas.

Com a implementação do EE, constatou-se que a utilização de uma linguagem de programação orientada a objetos (Turbo C++ "for Windows"), em conjunto com a biblioteca OWL para confeccionar o EE sobre o ambiente Windows foi uma estratégia acertada. O uso destes elementos, facilitou e agilizou o desenvolvimento do EE. A linguagem orientada a objetos contribui para a organização do programa em hierarquias de classes de objetos, permitindo a reutilização de código já existente e facilitando o entendimento da lógica do programa. A biblioteca OWL forneceu mecanismos de alto nível para manipular os recursos existentes no Windows, e o ambiente operacional Windows, com seus recursos de gerenciamento de interface gráfica e memória forneceu a principal sustentação para que o EE pudesse ser implementado. Haja vista que não houve a necessidade de se criar rotinas para tratamento do "mouse", de implementar o gerenciamento de janelas e todo o conjunto de rotinas para traçar primitivas gráficas utilizadas, e tão pouco houve a preocupação em reduzir o código ou em se criar "overlays".

Capítulo 9

Conclusões e Propostas para Futuras Pesquisas

9.1 - Conclusões

A inclusão de Atributos com Características Gráficas no meta-esquema, possibilita a modelagem de informações gráficas utilizando o MRO. Isto antes não era possível, pois o significado de Atributo com Características Gráficas não estava bem definido .

O GEO, fornecendo suporte a atributos gráficos, pode servir como base para a criação de aplicativos que necessitem manipular informações gráficas, bem como servir de elemento de integração para uma coleção de aplicativos.

A utilização do ambiente Windows, bem como do compilador Turbo-C++ e das bibliotecas OWL revelaram-se uma boa opção para construir o EE, propiciando:

- revisar o código fonte já existente do GEO;
- definir algumas operações genéricas para o SIGGBD;
- aventar a hipótese de reescrever o código fonte do GEO utilizando uma linguagem orientada a objetos;
- tornar disponível o GEO para o ambiente Windows.

O EE permitiu, além de testar as rotinas confeccionadas para gerenciar Atributos com Características Gráficas, a manipulação de rotinas que gerenciam a inserção e remoção de Objetos, Relacionamentos e Atributos com Característica de Propriedade. Isto possibilitou a descoberta de um problema ainda não solucionado:

- A remoção do último Tipo de Objeto que mantém um relacionamento **habita** com a Colônia Global causa um "travamento". Para contornar o problema, o EE insere, toda vez que for um novo diagrama, um objeto sem identificador com o relacionamento **habita** com a colônia global, inacessível ao usuário por intermédio do próprio EE, de tal forma que nunca ocorra o problema.

O EE pode ser empregado a longo prazo, como uma ferramenta de apoio à modelagem de dados. No entanto, enquanto o GEO estiver evoluindo, o EE pode também ser utilizado como uma ferramenta de testes, desde que se façam algumas melhorias na implementação atual e, paralelamente, acompanhe as modificações realizadas na implementação do GEO.

9.2 - Propostas para Futuras Pesquisas

Com a definição de Atributos com Características Gráficas para o MRO e a sua implementação no GEO, pode-se abrir frentes para algumas áreas de pesquisa. Entre elas pode-se citar:

- Definição de outras características de atributo tais como de Imagem e Visualização.
- Desenvolvimento de sistemas integrados que necessitem utilizar tanto um gerenciador de base de dados quanto um núcleo gráfico. Como exemplos desses sistemas pode-se citar Ferramentas CASE, CAD e Sistemas de Informação Geográfica.
- Desenvolvimento e implementação de um SIGGBD genérico o suficiente para servir de sustentação à criação de novos aplicativos. Um SIGGBD com esta característica poderia ser utilizado para que os aplicativos que dela se utilizem, possam interagir, pois o formato das informações gráficas passam a ser comum a todos.
- Reescrever o código fonte do GEO utilizando uma linguagem orientada a objetos. Isto pode fazer com que os conceitos do MRO sejam codificados de forma mais natural do que a presente implementação.
- Definir uma interface de programação de Aplicativos Orientada a Objetos.
- Incorporar ao EE outros conceitos do MRO não explorados. Como exemplo, pode-se citar: Subtipo de Objetos, Objetos de Sumarização, Subtipo de Relacionamento, Relacionamento Triplo, as outras características de atributo, Tratamento de Versões e Alternativas, Edição da base de dados extencional, gerenciador gráfico da Hierarquia de Colônias, Tratamento de Estado dos Tipos de Atributo e Tratamento de mais de uma ocorrência de um mesmo objeto num mesmo diagrama.
- Implementação das estruturas de dados utilizando um gerenciador de espaço baseado em registros de tamanho variável. Embora não tenha sido abordado neste trabalho, tal gerenciador foi desenvolvido durante as fases iniciais deste trabalho, restando agora sua validação juntamente com as rotinas de manipulação de atributos gráficos implementadas.

Referências Bibliográficas

- [1] A. Araújo, Técnicas de Realce para Imagem: Filtragem Espacial, IX Congresso da SBC, VIII Jornada de Atualização em Informática, Uberlândia MG, julho 1989.
- [2] M. Astrahan et al., "System R: Relational Approach to Database Management", ACM Transactions on Database Systems, vol. 1, nº 2, pag. 97-137, junho 1976.
- [3] M. Atkinson, F. Bancillon, D. DeWitt, K. Dittrich, D. Maier e S. Zdonik, "The Object-Oriented Database System Manifesto", Rapport Technique Altair 30-89, GIP ALTAIR IN 2 - INRIA - LRI, outubro 1989.
- [4] Autocad Reference Manual - By Autodesk Inc.
- [5] M. Biajiz, MRO* - Uma Atualização do Modelo de Representação de Objetos e Uma Abordagem Formal, Dissertação de mestrado apresentada ao Instituto de Ciências Matemáticas de São Carlos-USP, São Carlos, julho de 1992.
- [6] G. Birtwistle et al., Simula Begin. Berlin, Germany: Studentliteratur and Auerbach, 1973.
- [7] R. Boyce e D. Chamberlin, "SEQUEL: A structured English query language", Proc. ACM SIGMOD Workshop Data Description, Access and Control, New York, pag. 249-264, 1974.
- [8] C. Brown, "PADL-2: A Technical Summary", IEEE Computer Graphics & Applications, vol. 2, nº 2, pag. 69-84, março 1982.
- [9] L. Camolesi, Jr., Suporte a Acesso Multi-usuário em Bases de Dados Orientadas a Objetos através de Esquemas Suplementares, Dissertação de mestrado apresentada ao Instituto de Ciências Matemáticas de São Carlos-USP, São Carlos, outubro de 1992.
- [10] A. Censi, AUTOCAD 10, Guia Prático, Editora Érica, 1989.
- [11] D. Chamberlin et al., "SEQUEL2: A unified approach to data definition, manipulation, and control", IBM Journal Research Development, pag. 560-575, novembro 1976.

- [12] F. Codd, "A Relational Model for Large Shared Data Banks", Communications of ACM, vol. 13, nº 167 6, pag. 377-387, junho 1970.
- [13] F. Codd, "Extending the Relational Database Model to Capture More Meaning", ACM Transactions Database Systems, vol. 4, nº 4, pag. 397-434, dezembro 1979.
- [14] CODASYL Data Base Task Group, Report ACM, New York, abril 1971.
- [15] CODASYL Data Description Language Journal of Development, Material Data Management Branch, Department of Supply and Services, Ottawa, Ontario, 1978.
- [16] P. L. P. Corrêa, Uma Linguagem para o Gerenciador de Objetos Baseada no Modelo de Representação de Objetos, Dissertação de mestrado apresentada ao Instituto de Ciências Matemáticas de São Carlos-USP, São Carlos, maio de 1992.
- [17] B. Cox, Programação Orientada para Objeto, São Paulo, Makron, McGraw-Hill, 1991.
- [18] G. Cunha, A. Beraldo, F. Barreiros e A. Battaiola, Computação Gráfica e suas Aplicações em CAD: Introdução e Padronização, Editora Atlas, São Paulo, 1987.
- [19] F. Dam e S. Hughes, Computer Graphics: Principles and Practices", Addison Wesley, 2ª edição, pag. 285-346, 1990.
- [20] P. Dadam et. al., "A DBMS prototype to support extended NF²-relations: An integrated view on flat tables and hierarchies", Proc. ACM SIGMOD Conference, New York, pag. 376-387, 1986.
- [21] K. Dittrich, "Object-Oriented Database Systems: The Notion and the Issues", Proc. International Workshop on Object-Oriented Database Systems, Pacific Grove, California, IEEE Computer Society Press, setembro 1986.
- [22] K. Dittrich, A. Kotz e J. Mülle, "Database Support for VLSI Design: The DAMASCUS System", CAD-Schnittstellen und Datentransformate im Elektronikbereich, Springer Verlag, Berlin, 1986.
- [23] D. Fishman et al., "Iris: An Object-Oriented Database Management System", ACM Transactions on Office Information Systems, vol. 5, nº 1, janeiro 1987.

- [24]P. Fishwick e C. Blackburn, "Managing Engineering Data Bases: The Relational Approach", Computer in Mechanical Engineering, pag. 8-16, janeiro 1983.
- [25]A. Goldberg e D. Robson, Smalltalk-80: The Language and its Implementation. Reading MA: Addison-Wesley, 1983.
- [26]R. Goldstein e L. Malin, "3D modeling with the Synthavision system", Proc. 1st. Ann. Conference Computer Graphics in CAD/CAM Systems, Cambridge, Mass., pag. 244-247, abril 1979.
- [27]M. Groover e E. Zimmers, CAD/CAM - Computer-Aided Design and Manufacturing, Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1984
- [28]C. Heuser, Modelagem Conceitual de Sistemas: Redes de Petri, versão preliminar publicada para a V Escola Brasileiro-Argentina de Informática, 1991.
- [29]P. Harmon e D. King, Expert Systems: Artificial Intelligence in Business, John Wiley & Sons, Inc., pag. 121-128, 1985.
- [30]R. Haskin e R. Lorie, "On Extending the Functions of a Relational Database System", Proc. ACM SIGMOD Conference, pag. 207-212, junho 1982.
- [31]M. Hornick, S. Zdornik, "A Shared Segmented Memory System for an Object-Oriented Database", ACM Transactions on Office Information Systems, vol. 5, janeiro 1987.
- [32]R. Hull e R. King, "Semantic Database Modeling: Survey, Applications, and Research Issues", ACM Computing Surveys, vol. 19, nº 3, pag 201-259, setembro 1987.
- [33]IBM Personal Computer Graphical Kernel System, Programmer's Guide GKS, 1984.
- [34]G. Jaeschke e H. Schek, "Remarks on the Algebra of Non First Normal Form Relations", Proc. ACM SIGACT-SIGMOD Symposium on Principles of Database Systems, Los Angeles, California, pag. 124-138, março 1982.
- [35]R. Katz, "Managing the Chip Design Databases", IEEE Computer Magazine, vol. 16, nº 12, dezembro 1983.

- [36] R. Katz, E. Chang e R. Bhateja, "Version Modeling Concepts for Computer-Aided Design Databases", ACM SIGMOD Proceedings, Washington DC, pag 379-386, junho 1986.
- [37] A. Kemper e M. Wallrath, "An Analysis of Geometric Modeling in Database Systems", ACM Computing Surveys, vol. 19, nº 1, março 1987.
- [38] A. Kemper, M. Wallrath, e P. Lockemann, "An Object-Oriented System for Engineering Applications", Proc. ACM International Conference on the Management of Data, San Francisco, California, maio 1987.
- [39] W. Kim, "Object-Oriented Databases: Definition and Research Directions", IEEE Transactions on Knowledge and Data Engineering, vol. 2, nº 3, pag. 327-341, setembro 1990.
- [40] W. Kim, J. Banerjee, H. Chou, J. Garza e D. Woelk, "Composite Object Support in an Object-Oriented Database System", Proc. ACM 2nd International Conference on Object-Oriented Programming Systems, Languages, and Applications, Orlando, Florida, pag. 118-125, outubro 1987.
- [41] W. Kim e F. Lochovsky, Object-Oriented Concepts, Databases and Applications, ACM Press, Addison-Wesley Publishing Company, 1989.
- [42] F. Korth, A. Silberschatz, Sistema de Banco de Dados, revisão técnica de Routh Terada, São Paulo, McGraw-Hill, 1989.
- [43] U. Kroszynski, B. Palstroem e E. Trostman, "Geometric Data Transfer Between CAD Systems: Solid Models", IEEE Computer Graphics & Applications, pag. 57-71, setembro 1989.
- [44] R. Lorie e W. Plouffe, "Complex Objects and Their Use in Design Transactions", Proc. ACM Database for Engineering Applications, Database Week, San Jose, pag. 115-121, maio 1983.
- [45] D. Maier, A. Otis e A. Purdy, "Object-Oriented Database Development at Servio Logic", Database Engineering, vol. 18, nº 4, dezembro 1985.
- [46] M. Mäntylä, An Introduction to Solid Modeling, Computer Science Press, Principles of Computer Science Series 13, 1988.
- [47] R. Melo, Bancos de Dados não Convencionais: A tecnologia de BD e suas novas áreas de aplicação, VI Escola de Computação, Campinas, SP, 1988.
- [48] B. Meyer, "Reusability: The Case for Object-Oriented Design", IEEE Software, vol. 4, pag. 50-64, março 1987.

- [49] J. Miller, "Architectural Issues in Solid Modelers", IEEE Computer Graphics & Applications, pag. 72-87, setembro 1989.
- [50] M. Minsky, "A framework for Representing Knowledge", The Psychology of Computer Vision, P.H. Winston, McGraw-Hill, New York, pag. 211-277, 1984.
- [51] P. Norton e P. Yao, Programando em BORLAND C++ para WINDOWS, Série Borland, Berkeley, Rio de Janeiro, 1992.
- [52] ObjectLISP User Manual, LMI, Cambridge MA, 1985.
- [53] J. Peckham e F. Maryanski, "Semantic Data Models", ACM Computing Surveys, vol. 20, nº 3, pag. 153-189, setembro 1988.
- [54] R. C. M. Persiano, "Introdução à Modelagem de Curvas e Superfícies em Computação Gráfica", Apostila do IX Congresso da Sociedade Brasileira de Computação, VIII Jornada de Atualização em Informática, Uberlândia-Minas Gerais, julho 1989.
- [55] C. Petzold, Programando para WINDOWS 3, São Paulo, Makron Books, 1992.
- [56] C. Petzold, "GDI Comes of Age: Exploring the 32-bit Graphics of Windows NT", Microsoft Systems Journal, vol. 7, nº 5, pag. 41-53, setembro 1992.
- [57] P. Pistor e F. Andersen, "Designing a generalized NF² data model with an SQL-type language interface", Proc. of the 12th International Conference on Very Large Databases", (Kyoto, Japão, VLDB Endowment), Saratoga, California, pag. 278-285, 1986.
- [58] R. Price, G. Golendziner, Bancos de Dados para Aplicações não Convencionais, IV Escola Brasileiro-Argentina de Informática, (Termas de Rio Hondo, Argentina), Campinas, janeiro 1989.
- [59] A. Requicha, "Representations for Rigid Solids: Theory, Methods, and Systems", Computer Surveys, vol. 12, nº 4, pag. 437-464, dezembro 1980.
- [60] K. Schmucker, Macintosh Library: Object-Oriented Programming for the Macintosh. Hasbrouck Heights NJ: Hayden, 1986.
- [61] H. Schildt, Turbo-C/C++: The Complete Reference, McGraw-Hill, 1990.

- [62] M. Stonebraker, E. Wong, P. Kreps e G. Held, "The Design and Implementation of INGRES", *ACM Transactions on Database Systems*, vol. 1, nº 3, pag. 189-222, setembro 1976.
- [63] M. Stonebraker e L. Rowe, "The Design of POSTGRES", *Proc. of the 1986 SIGMOD Conference*, Washington DC, maio 1986.
- [64] B. Stroustrup, *The C++ Programming Language*. Reading MA, Addison-Wesley, 1986.
- [65] C. Traina, "GEO-Um Sistema de Gerenciamento de Base de Dados Orientado a Objetos: Estado Atual de Desenvolvimento e Implementação", notas internas do Instituto de Ciências Matemáticas de São Carlos, São Paulo, SP, 1990.
- [66] C. Traina e F. Slaets, "Um Modelo de Representação de Objetos", nos anais do 3º Simpósio Brasileiro de Bases de dados, Recife, PE, março 1988.
- [67] D. Tsichritzis e H. Lochovsky, "Hierarchical Database Management: A Survey", *ACM Computing Surveys*, vol. 8, nº 1, pag. 67-103, março 1976.
- [68] H. Voelcker e A. Requicha, "Geometric Modeling of Mechanical Parts and Processes", *Computer*, pag. 48-57, dezembro 1977.
- [69] T. Venetianer, "Desmitificando a Computação Gráfica", São Paulo, McGraw-Hill, 1988.
- [70] P. Weigner, "Concepts and Paradigms of Object-Oriented Programming", adendo das notas da OOPSLA de outubro de 1989, Brown University, junho, 1990.
- [71] K. Weiler, "Edge-Based Data Structures for Solid Modeling in Curved-Surface Environments", *IEEE Computer Graphics & Applications*, pag. 69-84, março 1982.
- [72] R. Wolfe, M. Wesley, J. Kyle, F. Grace e W. Fitzgerald, "Solid Modeling for Production Design", *IBM Journal Research Development*, vol. 31, nº 3, maio 1987.
- [73] S. Zdonik, "Object Management Systems for Design Environment", *IEEE Database Engineering*, vol. 88, nº 4, 1985.